University of Zagreb

FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Ivan Sikirić

IMPROVING FLEET MANAGEMENT SYSTEMS BY COMPUTER VISION

DOCTORAL THESIS

Zagreb, 2019

University of Zagreb

FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Ivan Sikirić

IMPROVING FLEET MANAGEMENT SYSTEMS BY COMPUTER VISION

DOCTORAL THESIS

Supervisor: Professor Siniša Šegvić, PhD

Zagreb, 2019

Sveučilište u Zagrebu
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Ivan Sikirić

UNAPREĐENJE SUSTAVA ZA UPRAVLJANJE VOZNIM PARKOM PRIMJENOM RAČUNALNOG VIDA

DOKTORSKI RAD

Mentor: Prof. dr. sc. Siniša Šegvić

Zagreb, 2019.

The doctoral thesis was written at the University of Zagreb, Faculty of Electrical Engineering and Computing, Department of Electronics, Microelectronics, Computer and Intelligent Systems.

Supervisor: Professor Siniša Šegvić, PhD

The doctoral thesis has 132 pages.

Doctoral thesis number: _____

About the Supervisor

Siniša Šegvić was born in 1971 in Split, Croatia. He completed elementary school and high school in Zadar, Croatia, with one year abroad in Milano, Italy (1985-86). He received the BS degree in electrical engineering (9 semesters) in 1996, from the Faculty of Electrical Engineering at the University of Zagreb, Croatia. From 1996 to 2005, he has been employed at the Department of Electronics, Microelectronics, Computer and Intelligent Systems of the same Faculty, as a teaching assistant. He is currently employed at the same faculty as a full professor.

Siniša Šegvić participated in several national research projects (2 Croatian and 1 French), one Croatian national technology project and was a leader of an another technology project. He was in charge of the research project MULTICLOD: Multiclass object detection (HrZZ, 2014-2017).

He received MS (2000) and PhD (2004) degrees in computer science from the University of Zagreb, Croatia. In 2005, he started a one-year postdoc position at IRISA, Rennes, France, in the field of appearance-based navigation by monocular computer vision. In 2006, he starts a one-year postdoc position at TU Graz, Austria, in the field of monocular simultaneous localization and mapping, funded by a Marie Curie international incoming fellowship. His research and professional interests include 3D, active and distributed computer vision, especially in the context of the localization and mapping for navigation purposes. He is also interested in image processing, software engineering, and generic and object oriented programming. He is the author or co-author of several papers published in international conference proceedings and reviewed scientific journals.

Siniša Šegvić speaks English and Italian very well, and has basic communication skills in French. He is married and has three children. He is a member of IEEE.

O mentoru

Siniša Šegvić je rođen 1971. godine u Splitu. Osnovnu školu i matematičku gimnaziju završio je u Zadru, osim osmog razreda osnovne škole kojeg je pohađao u Milanu, Italija. Od lipnja 1996. godine do danas, zaposlen je na Zavodu za elektroniku, mikroelektroniku, računalne i inteligentne sustave Fakulteta elektrotehnike i računarstva u Zagrebu. U svibnju 2000. godine obranio je magistarski rad pod naslovom “Uporaba projekcijske geometrije i aktivnog vida u tumačenju scena”. Doktorsku disertaciju pod naslovom “Višeagentsko praćenje objekata aktivnim računalnim vidom” obranio je u lipnju 2004. U srpnju 2006. okončao je jednogodišnje postdoktorsko usavršavanje na institutu IRISA u Rennesu, Francuska, na području primjene računalnog vida u samostalnoj navigaciji vozila u urbanom okruženju. U rujnu 2007. okončao je jednogodišnje postdoktorsko usavršavanje na tehničkom sveučilištu u Grazu, Austrija, na

području analize nesigurnosti procjene geometrije dvaju pogleda.

Siniša Šegvić je sudjelovao u radu više domaćih i inozemnih znanstvenih projekata. Vodio je jedan istraživački projekt u suradnji s gospodarstvom (HrZZ, 2008-2011), jedan projekt primjene informacijske tehnologije (MZT, 2003-2004), te jedan razvojni projekt Sveučilišta u Zagrebu (2012). Bio je suvoditelj jednog bilateralnog austrijsko-hrvatskog projekta (MZOŠ, 2010-2012). Bio je voditelj istraživačkog projekta MULTICLOD (HrZZ, 2014-2017). Njegovi znanstveni, istraživački i profesionalni interesi uključuju računalni vid, obradu slike, programsko inženjerstvo te objektno i generičko programiranje. Samostalno odnosno kao koautor objavio je više članaka u časopisima sa međunarodnom recenzijom te na međunarodnim znanstvenim skupovima. Kao recenzent, sudjelovao je u prosudbi članaka podnesenih za objavljivanje na znanstvenim skupovima i u časopisima.

Tijekom rada na Fakultetu elektrotehnike i računarstva, Siniša Šegvić je održavao predavanja na kolegijima Dinamička analiza scena, Oblikovni obrasci u programiranju, Arhitektura i organizacija računala, Skriptni jezici, Inteligentni sustavi, Duboko učenje i Modeli za predstavljanje slike i videa. Za potrebe nastave, priredio je veći broj didaktičkih tekstova, koji su dostupni na mrežnim stranicama fakulteta. Suautor je knjige Python za znatiželjne. Konačno, sudjelovao je i u radu fakultetskog Odbora za istraživanje i međunarodnu suradnju.

Siniša Šegvić vrlo dobro poznaje engleski i talijanski, a služi se i francuskim jezikom. Član je strukovne udruge IEEE. Oženjen je i ima troje djece.

Abstract

This thesis proposes an image categorization framework to deliver added value to fleet management systems. In order to match the client-server nature of fleet management the framework is conceived around the following two requirements: i) the bandwidth should be used sparingly, and ii) the set of image categories must be open. These requirements can be satisfied by a suitable division of responsibility between the clients and the server. The clients are responsible for representing images with descriptors which are designed to be compact and category-agnostic. The server is responsible for classifying descriptors into an arbitrary set of categories. This organization minimizes the bandwidth requirements due to compactness of the descriptors, and ensures that the set of categories remains open due to clients being oblivious to it.

Several kinds of image descriptors have been considered: handcrafted gradient histograms (GIST, SIFT), spatial Fisher vector embeddings, and convolutional representations trained in an end-to-end fashion (VGG, DenseNet, ResNet, MobileNetV2 and DCGAN). The descriptors are further compressed using PCA and quantization, after which they are classified by SVM. In order to evaluate the considered methods we introduce FM3–a novel image dataset which is specifically designed for fleet management applications. The dataset contains 11448 images which were acquired in different weather conditions and labeled with the following binary attributes: *highway*, *road*, *tunnel*, *tunnel exit*, *settlement*, *overpass*, *booth*, *traffic*. The results indicate that excellent classification results can be achieved with deep convolutional representations trained in a supervised manner. We refrain from fine tuning on the target dataset (although this further improves the results) in order to avoid reducing the descriptor performance on new categories due to catastrophic forgetting. Image descriptors can be as small as 512 bits, while still offering good performance. The proposed framework is able to tolerate adverse weather and poor illumination conditions provided that some such samples are present in the SVM training dataset.

Keywords: computer vision, intelligent vehicles, image classification

Sažetak

Ova doktorska disertacija proučava načine obogaćivanja sustava za upravljanje voznim parkom (engl. *fleet management*) korištenjem računalnog vida. Sustavi za upravljanje voznim parkom detaljno se opisuju, te se ukazuje na moguća poboljšanja koja bi bila omogućena korištenjem podsustava za kategorizaciju slika prometnih scena. Sustavi za upravljanje voznim parkom sastoje se od velikog broja jednostavnih klijenata (uređaja ugrađenih u vozila) koji se spajaju na središnji poslužitelj koji vrši nadzor. Ova disertacija utvrđuje ograničenja i zahtjeve nametnute takvom arhitekturom te predstavlja sustav za kategorizaciju slika koji ih zadovoljava. Razmatraju se suvremene metode računalnog vida potrebne za izgradnju takvog sustava, s glavnim fokusom na metode računanja kratkih opisnika slika. Ova disertacija doprinosi novi skup podataka nazvan FM3 koji se sastoji od 11448 slika prometnih scena snimljenih iz perspektive vozača. Slike su označene s osam binarnih atributa korisnih sustavima za upravljanje voznim parkom: *autocesta*, *cesta*, *tunel*, *izlaz iz tunela*, *naselje*, *nadvožnjak*, *naplatna kućica*, *gust promet*. Skup slika FM3 se koristi za detaljnu eksperimentalnu evaluaciju svih korištenih metoda računalnog vida. Rezultati eksperimenata pokazuju da se prometne scene mogu kategorizirati vrlo pouzdano, poštujući sva ograničenja i zahtjeve sustava za upravljanje voznim parkom. Najbolji rezultati postižu se korištenjem vrlo dubokih modela konvolucijskih neuralnih mreža. Uz korištenje metoda za reduciranje dimenzionalnosti i kvantizaciju moguće je proizvesti opisnik slike velik 512 bitova koji se kategorizira uz prosječnu preciznost 96%.

Poglavlje 1, "Uvod", daje kratak opis područja istraživanja, te ciljeve i doprinose istraživanja.

Poglavlje 2, "Relevantni radovi", daje pregled radova relevantnih sustavima za upravljanje voznim parkom, te srodnih radova iz područja računalnog vida, s naglaskom na kategorizaciju slika. U kontekstu ove disertacije, područja relevantna sustavima za upravljanje voznim parkom su određivanje pozicije vozila na mreži prometnica, te rekonstrukcija ruta. Daje se kratak pregled šesnaest relevantnih radova iz tih područja. Pregled područja kategorizacije slika započinje kratkim opisom ručno krojenih pristupa za opisivanje slika histogramima lokalnih značajki. Slijedi pregled područja dubokog učenja: daje se kratak pregled povijesti neuronskih mreža, definiraju se temeljni pojmovi, te se opisuju arhitekture mreža. Poseban naglasak je na modele korištene u ovoj disertaciji: VGG, ResNet, DenseNet i MobileNet, za koje se daje usporedni pregled broja parametara, složenosti i uspješnosti. Konačno, daje se kratak pregled radova koji se bave kategorizacijom prometnih scena. Učenje prijenosom znanja se primjenjuje u ovom istraživanju, zbog čega slijedi pregled radova koji proučavaju to područje, s naglaskom na problem katastrofične interferencije. Slijedi pregled radova nenadziranog učenja dubokih konvolucijskih modela GAN, DCGAN i WGAN. Poglavlje završava pregledom radova koji se bave metodama za računanje vrlo kratkih opisnika slika, te metodama za skraćivanje zapisnika slika.

Poglavlje 3, “Unaprjeđenje sustava za upravljanje voznim parkom računalnim vidom”, detaljno opisuje namjenu i funkcionalnosti sustava za upravljanje voznim parkom. Daje se detalji pregled klijent/poslužitelj arhitekture tih sustava. Opisuje se na koji način postojeći sustavi prikupljaju informacije o vozilima, te na koji način se prikupljene informacije koriste. Daje se jednostavan i ilustrativan pristup određivanja pozicije vozila na karti, te određivanja rute kojom je vozilo prošlo. Identificiraju se neki problemi postojećih pristupa, te se ilustrira kako bi ih bilo moguće umanjiti poznavanjem kategorija prometnih scena. Poglavlje završava pregledom ostalih potencijalnih upotreba kategorija prometnih scena: unaprjeđenje praćenja u realnom vremenu, alarmiranja te generiranju detaljnijih izvještaja na temelju povijesnih podataka.

Poglavlje 4, “Integriranje komponente za kategorizaciju slika u arhitekturu sustava za upravljanje voznim parkom”, predstavlja podsustav za kategorizaciju slika kakav bi se mogao ugraditi u sustave za upravljanje voznim parkom. Prvo se analiziraju dva naivna pristupa: klasifikacija slika na klijentima i klasifikacija slika na poslužitelju, te se identificiraju njihovi nedostatci. Klasifikacija slika na klijentima nije dovoljno fleksibilna iz dva razloga: i) zahtijeva kompleksne i skupe procedure za promjenu skupa ciljnih kategorija, te ii) onemogućuje rekalkulaciju ciljnih kategorija iz arhivskih podataka. Klasifikacija slika na poslužitelju je vrlo zahtjevna u terminima potrebnog podatkovnog prometa, te prostora za pohranu podataka. Predlaže se bolje rješenje: klijenti računaju i poslužitelju šalju kratak opisnik slike, na temelju kojeg poslužitelj određuje ciljne kategorije prometne scene. Utvrđuju se poželjna svojstva opisnika: deskriptivnost, kratkoća, jednostavnost računanja. Također se utvrđuju poželjna ograničenja na način učenja opisnika. Opisnici ne smiju znati za ciljne kategorije, kako bi se skup ciljnih kategorija mogao što lakše mijenjati u budućnosti. Zbog što bolje generalizacije poželjno je i da se opisnici ne trebaju učiti na ciljnim slikama. Poglavlje završava kratkim razmatranjem sustava klasifikacije i arhiviranja podataka na poslužitelju.

Poglavlje 5, “Metode”, opisuje metode koje su razmatrane za izgradnju predloženog sustava kategorizacije prometnih scena. Prvo se navode dva opisnika temeljena na prostornim Fisherovim vektorima. Jedan se bazira na lokalnim SIFT značajkama i GIST opisniku (nazvan SIFT/SFV+GIST), dok drugi koristi značajke dobivene konvolucijskom mrežom VGG-19 (nazvan VGG/SFV). Zatim je dan pregled opisnika zasnovanih na dubokim konvolucijskim modelima ResNet-50, DenseNet-121 i MobileNetV2. Konačno se razmatra opisnik baziran na nenadziranom učenju, temeljen na DCGAN generativnoj suparničkoj arhitekturi. Za svaki opisnik se analiziraju računalni zahtjevi i složenost, kao i potencijal za postizanje dobre generalizacije. Komentiraju se mogućnosti postizanja niskodimenzionalnih varijanti opisnika odabirom prikladnih hiperparametara. Zatim se opisuju metode za smanjenje dimenzionalnosti i efikasno kodiranje opisnika. Poglavlje se nastavlja opisom SVM klasifikatora te završava pregledom metoda za mjerenje uspješnosti klasifikacije.

Poglavlje 6, “Skup podataka za sustave s upravljanjem voznim parkom”, prezentira skup

podataka prikupljen za potrebe eksperimentalne evaluacije ovog istraživanja. Skup sadrži 11448 slika prometnih scena iz perspektive vozača. Detaljno se opisuje postupak prikupljanja slika. Velika većina slika (98%) je prikupljena iz video snimki vožnji po hrvatskim cestama. Vožnje je snimio autor disertacije kroz period od pet godina, od 2013. do 2018. godine, koristeći kamere pametnih telefona. Manji dio slika (ukupno 205) je preuzeta s internetskog servisa Mapillary.com. Slike preuzete s interneta su rukom odabrane kako bi se povećao broj uzoraka nekih vrsta prometnih scena koje se rijetko pojavljuju. Sve slike su prikupljene za vrijeme dana ili sumraka. Posebna pažnja je posvećena prikupljanju slika s raznim oblicima vizualne degradacije. Podskup od 5035 slika sadrži scene snimljene za vrijeme kiše, snijega, magle, te u periodima nepovoljnih kuteva sunca (pred sumrak i za vrijeme sumraka), dok preostale 6413 slike nisu značajno vizualno degradirane. Uvidom u prikupljene slike uočeno je osam kategorija scena korisnih sustavima za upravljanje voznim parkom: *autocesta*, *cesta*, *tunel*, *izlaz iz tunela*, *naselje*, *nadvožnjak*, *naplatna kućica*, *gust promet*. Za svaku od kategorija opisani su očekivani doprinosi sustavima za upravljanje voznim parkom. Poglavlje završava definiranjem kriterija anotiranja te pregledom distribucije kategorija.

Poglavlje 7, "Eksperimenti", detaljno opisuje postavke i rezultate svih eksperimenata provedenih u ovom istraživanju. Prvo se opisuju detalji sustava za klasifikaciju, jer se isti sustav klasificiranja koristi za evaluiranje svih opisnika. Zatim se precizno i detaljno opisuju hiperparametri svih šest opisnika opisanih u poglavlju 5. Za svaki opisnik se navode postignute prosječne preciznosti klasificiranja na svakoj od osam kategorija. Kao ukupna mjera uspješnosti uzima se aritmetička sredina prosječnih preciznosti svih osam kategorija. Svim opisnicima se zatim smanjuje dimenzionalnost korištenjem PCA, osim MobileNetV2 opisnika, koji je vrlo kratak i bez tog koraka. Opisnici dimenzionalnosti reducirane na potenciju broja dva od 1024 do 16 se zatim klasificiraju linearnim SVM klasifikatorom i SVM klasifikatorom s RBF jezgrom. Ovisnost postignute prosječne preciznosti klasifikacije o duljini opisnika se zatim prikazuje u grafičkom obliku, čime se pokazuje da se opisnici mogu smanjiti do 128 komponenti bez velikih gubitaka performansi, ukoliko se koristi RBF jezgra. Pokazuje se da DenseNet-121 opisnik postiže najbolje rezultate bez obzira na duljinu. Dobiveni opisnici se zatim kvantiziraju koristeći dva pristupa: product quantization (PQ) i naše njemu jednostavnije varijante nazvane component-independent quantization (CQ). Detaljnom serijom eksperimenata se pokazuje da je za vrlo malene kodove opisnika (64 bita i manje) bolje koristiti PQ kvantizaciju, dok je za kodove od 128 bitova i veće bolje koristiti CQ pristup. Opisnici od 128 komponenti kodirani s 512 bitova ne pokazuju gotovo nikakve gubitke u performansama u odnosu na pune verzije opisnika. Daljnji eksperimenti pokazuju da je primjenom algoritama za kompresiju opće namjene na veće skupove opisnika moguće uštedjeti još 20% do 40% prostora za pohranu. Poglavlje završava detaljnom analizom DenseNet-121 opisnika, s naglaskom na testiranje otpornosti na vizualne degradacije. Pokazuje se da se i vizualno degradirane slike mogu uspješno klasificirati,

pod uvjetom da se dio takvih slika uključi u skup za učenje SVM klasifikatora.

Poglavlje 8, "Zaključci i budući rad", rezimira provedeno istraživanje. Dobiveni rezultati se diskutiraju, donose se zaključci te se daje smjernice za buduća istraživanja.

Ova disertacija demonstrira da je moguće izgraditi sustav za kategoriziranje slika koji je koristan sustavima za upravljanje voznim parkom. Ograničenja i zahtjevi takvih sustava su identificirani i zadovoljeni. Razmatrane metode su detaljno evaluirane na novom skupu podataka koji je doprinos ove disertacije. Korištenjem dubokih konvolucijskih modela moguće je dobiti vrlo kompaktne i robusne opisnike slika koje je moguće kategorizirati s velikom razinom preciznosti.

Ključne riječi: računalni vid, inteligentna vozila, kategorizacija slika

Contents

Chapter 1

Introduction

This thesis studies how computer vision can be used to improve fleet management systems.

Fleet management systems monitor fleets of vehicles using a global navigation satellite system (GNSS), and possibly other sources of data. Although use of computer vision in vehicles is on the rise [1], it is being used primarily to implement and improve advanced driver assistance subsystems, not for fleet management.

This thesis shows that computer vision can be used to improve some existing features of fleet management, as well as to add some new features. It discusses the hardware and software requirements, limitations and considerations pertinent to fleet management systems. It proposes an elaborate computer vision framework integration which meets all the requirements and does not exceed the hard limitations. It does so in a way that is feasible at current level of technology, and is also expected to remain relevant in the foreseeable future. It presents a testing framework which verifies the proposed methods and enables their comparison with latest state-of-the-art methods.

1.1 Contributions

One of the contributions of this thesis is improving fleet management systems by providing a vision based subsystem which can be used to augment several aspects of fleet management, as will be discussed in Chapter 3. Due to architectural limitations of fleet management systems, it is not trivial to construct an image categorization subsystem that meets all the requirements. Another contribution of this thesis is exactly that: the organization of a scene categorization subsystem which is technologically suitable for use in fleet management. The proposed framework is presented in Chapter 4. The integration between fleet management and image categorization subsystem is not useful unless the image categorization performs accurately. Design and evaluation of image descriptors on a limited representation budget is presented in Chapter 5, and is another contribution of this thesis. Finally, a detailed experimental evaluation of the pro-

posed methods, and their comparison to latest state-of-the-art methods was done, as presented in Chapter 7. A novel dataset of labeled traffic scene images was collected and contributed in order to make this possible, presented in Chapter 6.

To summarize, the contributions of this thesis are as follows:

1. Including visual information into the vehicle status in order to enable providing a more complete fleet management service.
2. Organizing the subsystem for collecting visual information about the vehicle environment according to requirements and limitations specific to fleet management, such as network bandwidth, processing power of embedded computers and feasible implementation.
3. Designing and evaluating image representation models suitable for describing traffic scenes with descriptors of constrained dimensionality.

Chapter 2

Related work

This thesis explores how fleet management systems might be improved by computer vision methods. This chapter overviews prior research relevant to both fields.

Fleet management systems employ a wide variety of algorithms to implement their many features. The key features of interest in this work are *map matching* and, to lesser extent *route reconstruction*. Their overview is presented in the first section of this chapter.

The second section of this chapter gives detailed overview of relevant computer vision approaches, more specifically of image categorization approaches.

2.1 Map matching and route reconstruction

Map matching is s process of matching the vehicle coordinates with the most likely road segments of a digital road map. *Route reconstruction* is a process of guessing the most probable route the vehicle traveled, based on received coordinates. Since many map matching approaches take road connectivity into account, the route reconstruction is often a byproduct of map matching process.

Quddus et al. [2] categorize map matching approaches into four categories:

- geometric analysis
- topological analysis
- probabilistic algorithm
- other advanced methods

Geometric methods focus only on the geometry of the road map, and ignore connectivity. Bernstein and Kornhauser [3] define three sub-types of geometric methods: *point-to-point* matching, *point-to-curve* matching and *curve-to-curve* matching. In *point-to-point* matching, a point received by the GNSS is matched to the closest point of the road geometry (the road network is represented as a bag of points, not curves or line segments). This process is very simple and fast, but does not provide accurate results. In *point-to-curve* matching, a point received by

the GNSS is matched to the closest curve of the road geometry. This is a useful approach if only a single point of a vehicle trajectory is available. In *curve-to-curve* matching, short segments of vehicle trajectory are matched to the closest curves of the road geometry. White et al. [4] present matching algorithms that rely on both the *point-to-curve* and *curve-to-curve* approaches. Phuyal [5] uses a *curve-to-curve* approach in a matching system that combines GNSS and dead reckoning sensors.

Topological map matching methods take the road connectivity into consideration, as well as geometry. Greenfeld [6] match the initial point to the nearest road segment. Subsequent points are matched to the segment which best matches the distance and orientation, but only taking into consideration road segments that are quickly reachable from the current segment. Blazquez and Vonderhoe [7] present an approach that considers triples of subsequent points. For each of the points, match candidates are collected in the road map neighborhood. Match candidates are considered in turn until feasible shortest paths can be found between the candidates of the first and the second, and second and third points. A shortest path is considered to be feasible if its estimated average speed approximately matches the average speed between received points.

Probabilistic algorithms take the position confidence into consideration. Zhao [8] derives the error region for each position from the error variances received by the GNSS sensor. Several matching candidates are considered within the error region, and the best candidate is selected by examining the distance from the point, the vehicle heading and the connectivity with the previous point. Ochieng et al. [9] present an expert system based on empirical studies. It takes into account the error sources associated with the positioning sensors, the heading and speed information of the vehicle, the historical trajectory of the vehicle, road connectivity and orientation. Similar to approach of Zhao [8], it also considers an error region, but only when the vehicle travels through an intersection, which speeds up the process and reduces errors.

Advanced methods try to refine the matching candidate selection process by using better, more complex models. Yang et al.[10] present an approach based on Dempster–Shafer's (D–S) theory of evidence using rule based logical inference systems. The vehicle positions are smoothed with a Kalman filter. The matching candidates are selected and weighted according to distance obtained by point-to-curve matching, which does not consider the road topology. Syed and Cannon [11] present an approach based on a fuzzy logic model. They use a fuzzy interface system (FIS) to match the vehicle points. The inputs are distance of road segment from the reported point, orientation and distance from previous point. Fu et al. [12] present another approach based on fuzzy logic. The inputs to the FIS of their system are the distance between the road segment and reported point and direction of the road segment. The output is the possibility of matching to that road segment. Newson and Krumm [13] present an approach based on a hidden Markov model (HMM). The HMM states are road segments, and state measurements are the noisy points received from GNSS. Their system matches the points in batch,

after they have all been collected. Another approach based on HMM is presented by Mohamed et al. [14]. Their system is designed to provide map matching for cellular-based positioning. It works in a context of very low input frequency and precision, and assumes the vehicle is driving exclusively on major roads. They use three consecutive filters: *speed filter*, α*-trimmed mean filter* and *direction filter* to detect noisy transitions. Griffin et al. [15] present a system that identifies key waypoints in the vehicle trajectory using a modified Peucker curve reduction algorithm. A driving directions service is then used to find a route containing all of the waypoints. The route is further improved using a filter-and-refine approach to identify incorrect portions of the route and eliminate the waypoints that lead to incorrect matching. Atia et al. [16] develop a lane determination system that fuses accelerometers, gyroscopes and GNSS using an extended Kalman filter. A curve-to-curve matching is done using a HMM by a least-square regression step that estimates the lane. The success rate of achieved lane determination is 97.14%. Taguchi et al. [17] present a system that eliminates the latency inherent in HMM approaches by using a novel probabilistic route prediction model to estimate future GNSS points. The prediction model is trained on historical trajectory data. They achieve equal or greater accuracy than previous HMM models, without any latency.

The author is not aware of any prior work that relies on image categorization specifically to improve the map matching or route reconstruction process.

2.2 Image categorization

Image categorization is the main focus of this thesis. Relevant topics are hand-crafted approaches derived from the bag-of-visual-words idea, deep learning in general, traffic scene classification, transfer and unsupervised learning, and short image descriptors.

2.2.1 Bag-of-visual-words

Before the advent of deep learning methods, most successful image classification approaches have been based on the seminal bag-of-visual words method [18]. The method is inspired by the bag-of-words classification approach in text mining, in which a textual document is represented as a histogram of the words it contains. This approach involves agreeing on a word normalization method (e.g. lemmatisation), and a dictionary which ideally contains all words that might appear in a corpus of documents. For each document, all words are normalized, and number of occurrences of each dictionary word in that document is counted, thus producing the histogram. The histogram is regarded as a feature vector, a representation of that document that can be classified by a general purpose classifier. Note that this approach discards all information about the locations of words inside a document: both absolute word position, as well as relative

word order are completely disregarded.

To adapt the bag-of-words approach to the visual domain, Csurka et al. introduce the concept of *visual words* [18]. First, random sampling of local images patches is performed on all images in the dataset, and their descriptors are calculated. Next, a patch distance measure is defined (e.g. Euclidean, Mahalanobis), and the local image patches are clustered based on closeness. The resulting clusters are termed *visual words*, and thus the dictionary of visual words is formed. Each local patch descriptor is represented by a single visual word: the cluster in which it resides. The image descriptor is obtained by calculating the histogram of visual words. This approach is called bag-of-visual-words. Note that the exact distance of a local patch from the cluster centroid is ignored, as well as the location of visual words inside the image.

The key step in the bag-of-visual-words framework is the calculation of descriptors of local image patches, also know as *local features*. A number of methods have been proposed, such as Scale-invariant feature transform (SIFT) [19, 20], Local binary patterns (LBP) [21], Maximally stable extremal regions (MSER) [22], Speeded up robust features (SURF) [23], Binary robust invariant scalable keypoints (BRISK) [24] and Fast retina keypoint (FREAK) [25]. They were designed with different goals in mind (e.g. blob detection, viewpoint change detection, object tracking, etc.) and as such have different properties (e.g. rotational or scale invariance, resistance to noise or illumination changes, etc.). The SIFT patch descriptor in particular is invariant to scaling, translation and rotation, and partially invariant to illumination changes and affine or 3D projection, and as such is often utilized in bag-of-visual-words approaches.

Since spatial layout of visual words inside an image carries a lot of information for many classes of images, various extensions and modifications of the original bag-of-visual-words method have been proposed over the years [26]. One of the approaches defines a fixed layout of image regions, applying the complete bag-of-visual-words calculation on each region, and concatenating the results. One example of this approach is *spatial pyramid matching* [27], which uses a layout of quad tree. Other approaches use more precise models to describe the local patch spatial distribution. For example, *spatial Fisher vectors* [28] aggregate locations of visual words by leveraging a spatial generative model expressed as a mixture of Gaussians.

An overview of the general bag-of-visual-words descriptor pipeline is shown in Figure 2.1. An image is transformed to a more compact and meaningful representation through a series of steps. The obtained image vector can then be classified using a general-purpose classifier, such as support vector machine (SVM) [29].

2.2.2 Deep learning

The bag-of-visual words classification framework can be regarded as a composition of three non-linear stages: i) extraction of patch descriptors, ii) aggregation of local features, with possible encoding of spatial layout and iii) classification (e.g. using an SVM classifier). Stage i)

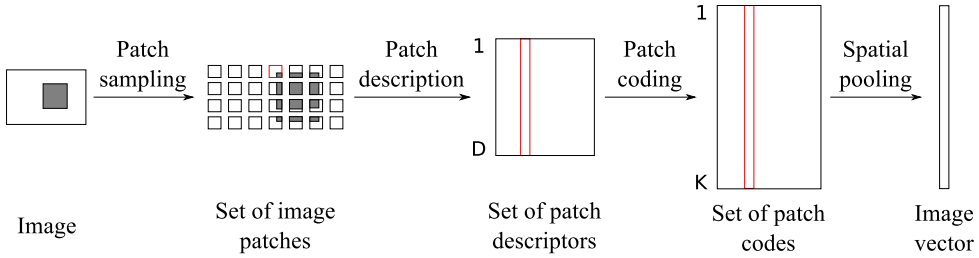


Figure 2.1: The bag-of-visual-words pipeline. Image patches are sampled and described. The patch descriptors are then coded, and the obtained set of patch codes is spatially pooled to produce the image descriptor.

is hard-coded, while the parameters of stages ii) and iii) are learned sequentially. In contrast to the bag-of-visual-words approach, the deep learning models train all stages jointly. In fact, there is no clear separation between the stages, as deep learning models gradually transform the image representation from low-level features to high-level representations through a series of non-linear transformations.

These transformations are often called *layers*, as each one is computed by a non-linear layer of a deep neural network. The front and middle layers usually perform either convolutional mappings followed by a non-linear activation, or aggregation of precedent features through pooling. In the earlier architectures several of the latter layers were fully-connected, while in modern architectures only the last layer is fully-connected, and typically outputs the probability scores for each class label. Gradients of all parameters of the resulting compositional model are determined with the backpropagation algorithm, while the training is usually performed with some variant of the stochastic gradient descent.

A brief history of neural networks

The term *neural network* is used because they were inspired by biological systems, in which a basic processing unit is a neuron. A neuron in artificial neural networks simply computes a weighted sum of its inputs, followed by an *activation function* (also known as a *nonlinearity*).

Neural networks with multiple non-linear layers exist since the early 1960's [30], and they were always capable of modeling very complex mappings, at least in theory. The main problem back then was the same one that remained unsolved for several decades: how to find the best weight parameters of neurons. In theory, the more parameters the network has, the more powerful it is in terms of learning capacity, and is expected to be able to perform better than a simpler network. However, adding more layers to a network actually led to worse performance, as the parameter learning algorithms completely failed to find any parameters that solved the problem at hand [31].

Gradient backpropagation algorithm was proposed in 1986 [32]. It is a mathematically sound way to calculate the gradients of network parameters in the gradient descent step. Though

it is simple and fast, the algorithm by itself is not enough to enable learning of arbitrary networks. Popular neuron activation functions of that time were sigmoid and tanh, which lead to networks that suffered from problems of saturated (dead) neurons, and of vanishing gradient [31]. Dead neurons are the ones for which the gradient becomes very near zero for all samples during training, so it becomes very hard or impossible to change their state in further training iterations. Neurons are effectively dead, stuck in the same state. The problem of vanishing gradient was identified by Hochreiter [33] in 1991 as a prime obstacle in successful application of backpropagation to many-layered neural networks of the time. Sigmoid and tanh functions have gradients in range (0, 1), which are often very small numbers. The backpropagation algorithm uses the chain rule to calculate the gradient of all layers. Each additional layer implies one additional multiplication by a small number, which means the gradient of the front layers is very low (it decreases exponentially with respect to the number of layers). This means it is very hard to successfully learn the correct parameters for the front layers of deep networks, as the learning time rises exponentially [31] with the number of layers.

Despite these problems, neural networks were still able to successfully solve some classification tasks in image processing [32, 34, 35, 36]. LeCun et al. [34] introduced *convolutional neural networks* (CNN) in 1989. They used a network with three hidden layers to classify zip code digits represented as images of resolution 16×16 pixels. The first two layers were convolutional, and used weight sharing as proposed by [32]. The last hidden layer and output layer were fully connected. Rowley et al. [35] used CNNs with two and three hidden layers and 4 types of receptive fields in a face-detection framework. The networks detected the presence of a face in a 20×20 image window with low false-positive rate. In 1998, LeCun et al. [36] presented the network called *LeNet-5*. It consisted of seven layers, three of which were convolutional. The entire network structure is shown in Figure 2.2. The activation function was sigmoid. It was very successful in recognizing hand-written digits of resolution 32×32 pixels, and was used to read several million bank cheques daily [36].

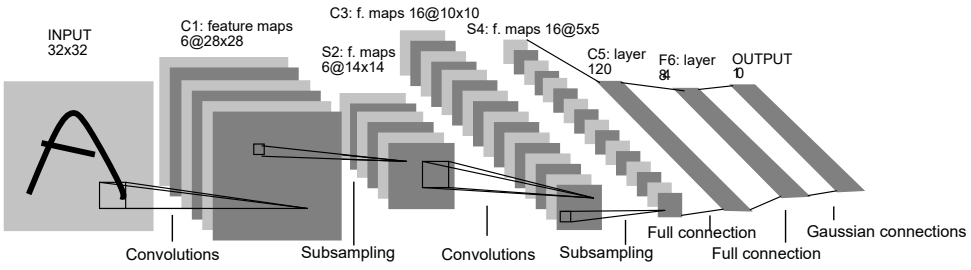


Figure 2.2: Structure of LeNet-5. A 32×32 input is transformed into 1×10 output through 7 different layers. Layers 1, 3 and 5 are convolutional. Layers 2 and 4 perform subsampling (pooling). Layer 6 is fully connected, while the layer 7 is output layer. Figure reproduced from [36].

Deep learning terminology

Before continuing with the overview of deep learning architectures, basic terminology must be defined. A convolutional neural network consists of *layers*. Each layer transforms an *input volume* into an *output volume*. The input and output volumes are tensors with width, height and depth. The width and height are also called *spatial dimensions*, while depth is also called the number of *channels* or *activation/feature maps*. In the context of computer vision, the input volume to the first layer of the network is a tensor $W \times H \times C$, where W and H are the width and height of the input image, and C is the number of image channels ($C = 3$ in case or RGB color images). The input to subsequent layers is the the output of their preceding layer.

A *convolutional* (conv) layer consists of a set of learnable *filters* (also called *kernels*). Each filter has a small receptive field, but extends through the full depth of the input volume. The receptive field slides over the spatial dimensions and applies weighted sum of its inputs followed by a nonlinear activation function, thus producing a single activation or feature map. The *stride* parameter controls how a receptive field slides over the spatial dimensions. A stride vale of k means the field is applied for each k-th input in each k-th row. When an input volume of $W_1 \times H_1 \times C_1$ is fed through a convolutional layer with C_2 filters with perceptive fields of dimensions $M \times M$ and stride k, then the output volume is a tensor $W/k \times H/k \times C_2$. The total number of input weightings (multiplications) is $W/k \cdot H/k \cdot M^2 \cdot C_1 \cdot C_2$. The total number of weight parameter in the layer is $M^2 \cdot C_1 \cdot C_2$.

A *pooling* (pool) layer is a form of non-linear downsampling. A receptive field slides over each feature map of the input volume, and a pooling function is calculated on all inputs in the receptive field. The sliding over the spatial dimensions may be regulated by a stride parameter (which is typically greater than 1), or the pooling may be global (in which case the receptive field covers the entire feature map). Common pooling functions are maximum (max-pooling) and average (avg-pooling). The output has the same depth and lower spatial dimensions. If a $W \times H \times C$ input is pooled by a receptive field of any size and stride k, then the output is a volume $W/k \times H/k \times C$.

A *fully-connected* (FC) layer consists of N neurons, and outputs a vector of size N. Each neuron is connected to every input, and calculates the weighted sum followed by an activation function (usually a simple bias offset).

Common activation functions

A *rectifier* is an activation function given as:

$$f(x) = \max(0, x) \tag{2.1}$$

It was first used by Hahnloser et al. [37] in 2000, with biological motivation (it is one-sided, instead of anti-symmetrical). A unit which employs this activation function is called *rectified linear unit* (ReLU). It is very easy to compute, leads to sparse activation of neurons, and enables efficient gradient propagation. It is a popular choice in modern deep learning frameworks. One potential problem is that is non-differentiable at zero. If necessary, this can be solved by using a differentiable approximation called a *softplus* function:

$$f(x) = \log(1 + e^x) \tag{2.2}$$

Rectifier activation alleviates the problem of dead neurons, as the positive side of the function is linear. However, it is still possible to end up with a large number of dead neurons, as the negative side of the function is constant zero. If the learning rate is too high, neurons can easily enter a state in which they are zero for all possible inputs. This can be mitigated by use of leaky ReLU units:

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ ax, 0 < a < 1 & \text{otherwise} \end{cases} \tag{2.3}$$

AlexNet

In 2012 Krizhevsky et al. [38] presented the *AlexNet*: the first successful deep convolutional neural network applied to image classification. The AlexNet consists of eight layers: first five are convolutional, while the last three are fully-connected (Figure 2.3). Several key improvements over earlier network architectures made successful training possible. They used the rectifier activation function, which is very resilient to the problem of dead neurons, as it can only be saturated from one side. They used *local response normalization* (LRN) to improve generalization. They used a dropout trick[39], which enhances generalization by randomly omitting half of the feature detectors on each training case. The learning algorithm was implemented to run in parallel on powerful GPU hardware, which greatly shortened the learning time. The network was split into two parts (streams), so that each part fits into memory of a single GPU. It was trained using a mini-batch stochastic gradient descent on the dataset of 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [40]. The dataset was very large for that time, containing 1000 classes across over a million images, which was large enough to provide sufficient data to learn the network parameters. Additionally, they used data augmentation to reduce overfitting: the dataset was enlarged by label-preserving transformations such as translations, reflections and pixel intensity alterations. All these improvements together enabled the

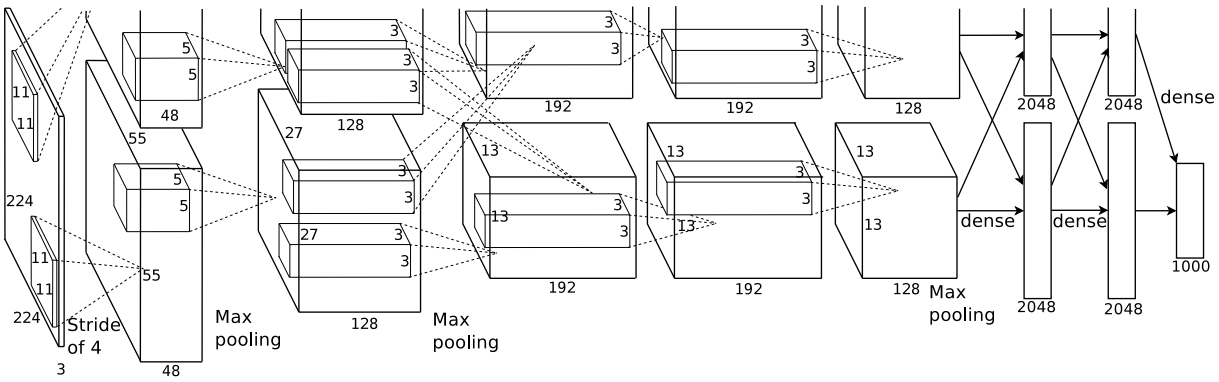


Figure 2.3: Structure of AlexNet. The 224×224 input is transformed into 1×1000 output by passing through eight layers. The first five layers are convolutional, while the last three are fully-connected. The top part of the figure shows the layer parts that run on one GPU, while the other GPU runs the layer-parts at the bottom part of the figure. Figure reproduced from [38].

AlexNet to outperform all contemporaneous approaches and win the 2012 ImageNet ILSVRC by a large margin. They achieved best results with an ensemble of 7 networks.

Incremental network structure improvement

After the success of AlexNet, many researchers strive to reproduce its success, to understand the reasons behind its classification performance and to improve the network structure and optimize the learning algorithm.

Zeiler and Fergus [41] devise a scheme for visualization of the inner workings of convolutional networks. They use a multi-layered transposed convolution network to project the feature activations back to the input space (i.e. pixel space). They also occlude portions of the input image to measure the sensitivity of the classifier output to find the parts of images important for classification. These visualizations lead to better understanding of convolutional networks, and discover some issues with existing models, such as presence of aliasing artifacts caused by large stride in first layer convolutions. Using the visualizations to debug found issues lead to a more successful architecture, with better classification performance than previous models.

Sermanet et al. [42] improve the AlexNet structure in the same manner as Zeiler and Fergus: by reducing the stride and size of receptive window in the first layer. Additionally, they adapt it for object detection and localization. Their network is termed *Overfeat* and it outputs not only object labels but boundaries as well, which are then aggregated to increase detection confidence.

Simonyan and Zisserman present the VGG network [43]. They improve the classification performance of neural networks by increasing their depth while reducing the convolution filter size (3×3 in all layers). During training, the only pre-processing done on the images is subtraction of mean RGB value (computed on the entire training dataset) from each pixel. Unlike [38], they do not use local response normalization. They present several VGG network configurations, referred to by letters A to E. The configurations differ only in their depth: VGG-A

network has 11 layers (8 convolutional and 3 fully-connected), while VGG-E network has 19 layers (16 convolutional and 3 fully-connected). Each of the configurations B to E was created by adding layers to its fully-trained predecessor. Each configuration inherited a set of learned weight parameters from its predecessor, so the convergence of the learning process was faster to achieve than it would be if all the layers were randomly initialized.

Szegedy et al. present the Inception network in 2015 [44]. It is 22 layers deep and uses *inception modules*, which are stacked upon each other at higher layers. In the inception modules the 1×1, 3×3 and 5×5 convolutional features are concatenated, together with a 3×3 max pooling layer. Additional 1×1 convolutions are added to reduce the dimensionality and prevent explosion of number of parameters. The architecture is more sparse than previous models, which results in reduced number of parameters and enables the training of deeper network structure, with better classification performance.

Batch normalization

Ioffe and Szegedy[45] introduce the process of *batch normalization*, in which network activations are normalized to have zero mean and unit variance for each layer in each mini batch in a training iteration. Such normalization eliminates the need for low learning rates and careful parameter initialization[45] (no need to carefully scale the initial parameters with respect to number of inputs and layer depth). Since the batch means and variances fluctuate, this process lowers the risk of overfitting the network, reducing the need for *dropout* regularization. Batch normalization significantly simplifies and speeds up the learning process, while simultaneously improving generalization properties as well. This approach is adopted and used in many future architectures.

For a d-dimensional input $\mathbf{x} = (x^{(1)}...x^{(d)})$ each input dimension is normalized as follows:

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}} \tag{2.4}$$

To preserve the representation power of a layer, parameters $\gamma^{(k)}$ and $\beta^{(k)}$ are added to enable scaling and shifting, respectively:

$$y^{(k)} = \gamma^{(k)}\hat{x}^{(k)} + \beta^{(k)} \tag{2.5}$$

These parameters are learned by back-propagation. During training, expectation and variance are computed separately for each batch of samples from the training dataset. Once the network has been trained, the expectation and variance are computed on the entire population.

Highway networks

Srivastava et al. introduce Highway Networks in 2015 [46]. The highway network architecture allows unimpeded flow of information across several layers on *information highways*, and the regulation of information flow is done by learned gating mechanisms. Such architecture reduces the problem of vanishing gradient to such a degree that is becomes possible to train hundreds of layers by stochastic gradient descent. The gating mechanisms enable the existence of very long paths through which information can flow without attenuation across several layers of neural network.

A single layer of a highway network transforms an input $\mathbf{x}$ to the output $\mathbf{y}$ as follows:

$$\mathbf{y} = H(\mathbf{x}, \mathbf{W}_H) \cdot T(\mathbf{x}, \mathbf{W}_T) + \mathbf{x} \cdot C(\mathbf{x}, \mathbf{W}_C) \tag{2.6}$$

The transformation H is an affine transform followed by a non-linear activation function (same as in plain feed-forward networks). The transformation T is the *transform* gate, and C is the *carry* gate. The T and C transformations express how much of the output is produced by transforming the input and carrying it, respectively. Authors simplify this expression by setting $C = 1/T$, giving:

$$\mathbf{y} = H(\mathbf{x}, \mathbf{W}_H) \cdot T(\mathbf{x}, \mathbf{W}_T) + \mathbf{x} \cdot (1 - T(\mathbf{x}, \mathbf{W}_C)) \tag{2.7}$$

Thus if $T(\mathbf{x}, \mathbf{W}_T) = 0$, then the output is equal to the input, while for $T(\mathbf{x}, \mathbf{W}_T) = 1$ Eq. 2.7 gives a plain feed-forward network layer. By varying the value of $T(\mathbf{x}, \mathbf{W}_T)$ between 0 and 1, the behavior of a layer is smoothly varied between that which simply passes the input and that which transforms it by transform function H. Note that for the Equation 2.7 to be valid, the dimensionality of $\mathbf{x}$, $\mathbf{y}$ and $H(\mathbf{x}, \mathbf{W}_H)$ must be equal. The change of the size of intermediate representations can be achieved by sub-sampling or zero-padding as necessary, or by using a plain layer (which is what authors chose).

Convolutional layers of a highway network use weight-sharing and local receptive fields for both H and T transforms. The transform gate is defined as $T(\mathbf{x}) = \sigma(\mathbf{W}_T^\top \mathbf{x} + \mathbf{b}_T)$.

Residual networks

He et al. [47] introduce ResNet approach in 2015. A ResNet neural network architecture can be considered a special case of Highway networks, in which the transfer and carry transforms are not adaptable, but fixed. Flow of information from front to back layers of the network is preserved explicitly by introduction of *shortcut* connections, which perform identity mapping.

This identity mapping is then element-wise summed with the output of one or more previous layers. An example building block is shown in Figure 2.4 (a). The entire network consists of such building blocks, and the reduction of dimensionality between blocks (if necessary) is done by a linear projection on the shortcut connections to match the dimensions.

The authors consider this simplification relative to the Highway networks to be an advantage, as they add neither extra parameters nor computational complexity. The authors note that by adding a single randomly initialized layer in the middle of a trained network its classification performance is crippled until parameters are re-learned. A lot of iterations of the learning algorithm are required just to achieve performance that is not worse than before the extra layer was added. Even learning an identity mapping (which would produce identical results as before the layer was added) is not simple, and takes many iterations. Authors reason that by making the identity mapping explicit, the adding of new layers does not immediately decrease performance, and any additional learning can only be beneficial. In addition to identity mappings, authors also use *bottleneck* design in their network architecture, which reduces the dimensionality of intermediate representations. For each residual function they use a 1×1 convolutional layer, then a 3×3 convolutional layer, then the 1×1 layer again. The 1×1 layers reduce dimensionality before 3×3 convolution, and then restore it.

In their later work [48], authors evaluate various orderings of layers within a residual block, as shown in Figure 2.4. They find that best results are achieved with *full pre-activation* ordering, shown in Figure 2.4 (e). The ResNet-110 network achieves classification error of 6.61% on the CIFAR-10 [49] dataset if the original ordering is used, and 6.37% if full pre-activation ordering is used.

Authors achieve state-of-the-art results on various classification challenges, with fast learning and better performance per same number of parameters than networks which simply stack layers. They claim the increased depth of the network is responsible for the increase of performance.

Further research disproves the idea that the depth of residual networks is the reason for increased performance. Veit et al. [50] rewrite residual networks as an explicit collection of paths of various depth. They regard the *unraveled* representation of residual blocks (Fig. 2.5), and perform lesion study. They delete individual layers at test time, they delete several modules at test time and they reorder modules at test time. They find that paths do not strongly depend on each other. More importantly, they find that longer paths do not contribute almost any gradient, and that most of the gradient in the residual network comes from paths of short to medium length. For example, for a network with 110 layers, they find most of the gradient comes from paths that are between 10 and 32 layers deep. They conclude the residual network does not behave as a single very deep network, but more as an ensemble of short networks.

Huang et al. [51] use stochastic depth during training of deep networks. For each mini batch

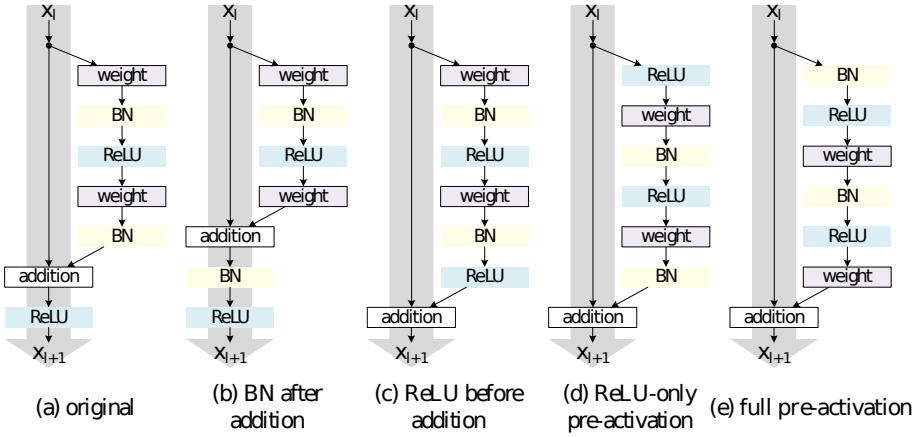


Figure 2.4: Various types of residual blocks. All blocks have the same components, but their ordering is different. The best results are achieved with configuration (e), dubbed full pre-activation. Figure reproduced from [48].

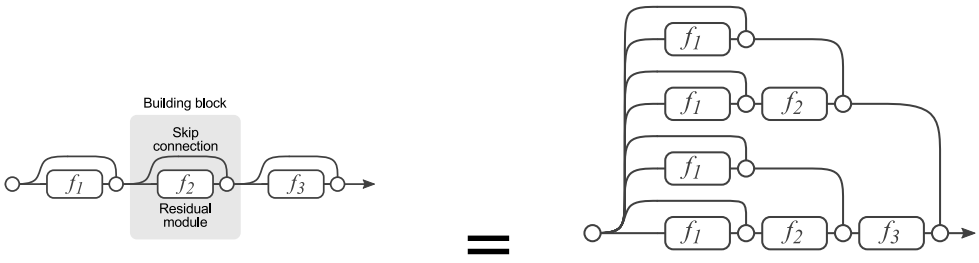


Figure 2.5: Figure on the left shows a conventional representation of a part of residual network. Figure on the right shows an *unraveled view* of the same part of network. Circles represent additions. Figures reproduced from [50].

they randomly exclude some layers of the residual network and replace them with identity connections. They find this achieves better information and gradient flow, acts as a form of regularization, and speeds up the learning process.

DenseNet

The research of Huang et al. [51] indicates that some layers of residual networks contribute very little new information, and mostly forward almost unchanged data. This insight leads Huang et al. [52] to propose the DenseNet architecture. The DenseNet architecture achieves preservation of information flow across layers in a very different way from ResNet and Highway networks. Instead of adding shortcut connections, or allowing single layers to preserve a portion of input information as-is, DenseNet consists of dense building blocks, in which the output of a single layer is directly provided as input to all subsequent layers in the same block. This is illustrated in Figure 2.6. Instead of element-wise summation, the features are combined via concatenation. Bottleneck layers are used to reduce the number of parameters. Each block is connected to the next one through a transition layer, which adapts the dimensionality of the block output to the input of the next block. The transition layer can also reduce the number of feature maps to further reduce the number of parameters. The dense connectivity of this network promotes feature reuse, simplifies the information flow between layers, and enables training of very deep structures. The DenseNet architecture achieves better classification accuracy than previous approaches with same number of parameters, including ResNet approach.

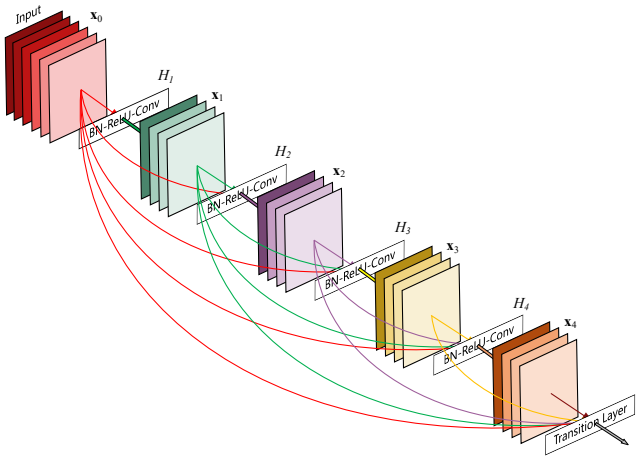


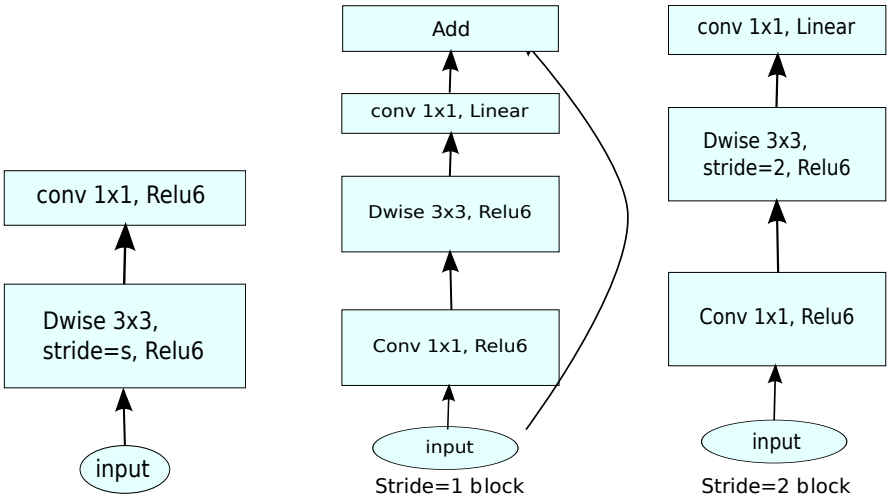
Figure 2.6: A 5-layer dense block. Figure reproduced from [52].

MobileNet

Howard et al. [53] present a CNN designed for computational efficiency called MobileNet. They use depthwise separable convolutions (Figure 2.7 (a)) as basic building blocks to reduce

the number of parameters and computational requirements. The depthwise separable convolution is a factorization of a standard convolution into a depthwise convolution and a pointwise (1×1) convolution. Depthwise convolution applies a single 3×3 filter to each input channel, followed by a batch normalization and ReLU activation, while pointwise convolution combines the output channels of the depthwise convolution, followed by batch normalization and ReLU. The first layer of MobileNet is a full convolution, which is followed by 13 depthwise separable convolutions and concluded by global average pooling and a fully connected layer. Downsampling is done by strided convolutions in the first layer and depthwise layers. This architecture achieves accuracy of 70.6% on ImageNet ILSVRC [40] with 4.2 million parameters.

Sandler et al. [54] present MobileNetV2, which expands on MobileNet ideas by incorporating residual blocks [51]. It achieves slightly better classification accuracy with fewer parameters. The network contains a total of 19 *residual bottleneck* layers (Figure 2.7 (b)). The residual bottleneck layer calculates the residual as follows: it expands an activation tensor to a high-dimensional space by a 1×1 convolution, then filters it with a 3×3 depthwise convolution, and finally projects it back to a low-dimensional space with a linear 1×1 convolution. A shortcut connection is added directly between the bottleneck layers (i.e. between linear convolutions). Downsampling is performed by using a stride 2, in which case a shortcut connection is not added. As in previous MobileNet model, the first layer is fully convolutional. They use ReLU6 [55] activation function because of its robustness when used with low-precision computation [53].



(a) Depthwise separable convolution

(b) Residual bottleneck blocks

Figure 2.7: Building blocks of MobileNet (on the left) and MobileNetV2 (on the right). Figure reproduced from [54].

Comparison of deep convolutional models used in this thesis

This thesis considers VGG-19, ResNet-121, DenseNet-50, and MobileNetV2 architectures in the experimental section. The overview and comparison of these architectures is given in Table 2.1. The table lists the total number of network parameters as well as the number of multiply-add operations required to process an RGB image of resolution 640×480. Finally, the table lists the top-1 and top-5 errors reported on the ImageNet1000 validation dataset.

Table 2.1: Comparison of deep convolutional architectures used in this thesis. The number of multiply-add operations assumes an input RGB image of resolution 640×480.

architecture	number of parameters	multiply-adds (640×480 input)	top-1 error (ImageNet valid.)	top-5 error (ImageNet valid.)
VGG-19	$20 \cdot 10^6$	$120 \cdot 10^9$	24.4 %	7.1 %
ResNet-50	$30 \cdot 10^6$	$25 \cdot 10^9$	20.7 %	5.2 %
DenseNet-121	$6 \cdot 10^6$	$15 \cdot 10^9$	23.6 %	6.7 %
MobileNetV2	$1.2 \cdot 10^6$	$2 \cdot 10^9$	28.1 %	9.5 %

Traffic scene categorization

A number of related works deal with traffic scene categorization specifically, which is of special interest in the context of fleet management. Oeljeklaus et al. [56] present a deep neural network that simultaneously performs traffic scene recognition and segmentation. Not counting the fully connected layer, their network has less than six million parameters, and is computationally efficient. To facilitate the segmentation task, they introduce the *Hadamard* layer which performs element-wise multiplication of input feature maps with a weight matrix of equal dimension. Di et al. [57, 58] research the impact of weather conditions on the classification of traffic scenes. They classify traffic scenes from images taken from the same location, but under different weather or illumination conditions. Their proposed classifier is based on cross-domain dense correspondence, and works by extracting the fine-tuned CNN features and transferring the annotations from the retrieved best matching image. Hussain et al. [59] explore the impact of image resolution on the ability of CNNs to detect and categorize vehicles in traffic scenes. They achieve adequate results even with vehicle regions as small as 90×90 pixels, and find that no significant improvement is achieved when much higher image resolutions are used.

2.2.3 Transfer learning

Convolutional networks have a huge number of parameters, so they usually require a large amount of training data to properly learn those parameters. Unfortunately, not every application

has a readily available training dataset as large as ImageNet. In many such cases it is possible to use some form of transfer learning, i.e. to apply knowledge of one dataset, called *source dataset*, to another, called *target dataset*.

Oquab et al. [60] present a method for learning and transferring mid-level image representations from a dataset with a lot of annotated samples to another with a limited amount of training data. They reuse layers of AlexNet (trained on the images of ImageNet dataset) to compute mid-level image representations for the images in the PASCAL VOC [61] dataset. To solve the problem of different labels (different classes between datasets), they remove the last fully connected layer of AlexNet, i.e. the output layer, and add two fully-connected layers termed *adaptation layers*. They keep the parameters of all the original layers fixed, and only train the two adaptation layers on the PASCAL VOC dataset. Since the images of the PASCAL VOC dataset may contain multiple labeled objects, they employ a sliding window technique to extract around 500 square patches from each image, and classify each patch separately, assigning it a single class label. Patches are spaced at eight different scales, with at least 50% overlap, and each patch is rescaled to 224×224 pixels prior to classification. They assign ground truth labels to patches by examining the patch overlap with ground-truth object locations. They assign a positive class label to a patch if a sufficient area is overlapped (at least 20% of patch area contains the object and at least 60% of the object is contained in the patch), and if the patch does not overlap with other objects. As most patches contain the *background* class, they re-balance the dataset by sampling 10% of the background patches during mini-batch selection in training.

Barat and Ducottet [62] improve classification performance of CNN systems by adding structural representations on top of pretrained CNN features. They represent images as strings of CNN features which are classified by an SVM. To express kernel encoding similarity between pairs of images, they introduce two new edit distance variants.

Cimpoi et al. [63] extract features from deep convolutional layers and use them as filter banks. They aggregate them with a Fisher vector framework to obtain the FV-CNN descriptor, which achieves state-of-the-art performance in texture recognition and classification.

Many other researches obtain image representations by extracting features from layers of various depths from pre-trained CNNs. Gong et al. [64] extract the features from the seventh (fully connected) layer for different scales of the input image and use pooling to obtain an image descriptor which is more robust to geometric variations of the input image. He et al. [65] use the outputs of the final layer of various CNNs in the spatial pyramid pooling scheme to produce an image representation of fixed length regardless of the input image size and scale. Girshick et al. [66] extract features from five convolutional layers and two fully connected layers in a detection scheme which achieves state-of-the-art results on the PASCAL VOC 2012 detection challenge. Athiwaratkun and Kang [67] extract lower-layer features and classify them using Random Forests and an SVM classifier to obtain competitive classification results.

Garcia-Gasulla et al. [68] perform a thourough evaluation of the behavior of different CNN features in transfer learning, for the purpose of feature extraction. They use a VGG16 network (Simonyan and Zisserman [43]) pretrained on a large dataset (ImageNet [40]) to build image representations for eleven alternative datasets. They study the behavior of individual features in all layers in the domain of each target dataset. For each class in a dataset, they evaluate how characteristic each feature in the embedding is in contrast to evaluating the descriptive power of entire groups of features. They use several statistical measures to examine the difference in responses of a single feature for images of a certain class versus the responses for images of other classes. They reach several important conclusions, some of which are listed here:

1. Both presence as well as absence of a feature can be a characteristic of a class.
2. Features from the last convolutional layer and fully connected layers are very specific. They are either characteristic for a class or irrelevant
3. Features from low-level layers are more general and discriminant, and could possibly be used in unsupervised learning.
4. Low and middle level features have a very similar behavior for the dataset they were trained for (source dataset), as for the other target datasets. These features could be used for many datasets without fine-tuning.
5. Features from fully-connected layers are more relevant for target datasets with high similarity to the source, and with a wide variety of classes.
6. If a target dataset has no intersection with the source dataset, then the distribution of Kolmogorov-Smirnov statistic for features from fully-connected layers is similar to that of convolutional features. Both sets of features could be used for knowledge transfer in similar way.
7. They found discriminant features for all classes on all layers.

Nanni et al. [69] build a generic image classification system by combining handcrafted features with features learned by deep convolutional neural networks. They explore three approaches of using learned features: i) remapping the output layer of CNN to classify a different problem, ii) using the output of the penultimate layer of CNN as a feature vector and iii) merging the output of some deep layers, reducing the dimensionality of the merged output, and using that as a feature vector. Besides different deep learning transfer methods, they use Principal Component Analysis Network and Compact Binary Descriptor methods as well. They combine the learned features with several state-of-the-art hand-crafted features. They find that both hand-crafted and learned feature paradigms are capable of extracting the information that the other paradigm ignored.

Catastrophic interference

One of the problems related to transfer learning and neural networks is the problem of *catastrophic interference* (also called catastrophic forgetting), first identified by McCloskey and Cohen [70] in 1989. It is described as the tendency of a neural network to rapidly lose most knowledge of the learned dataset during learning of another dataset. This is inconvenient for applications in which the target dataset grows over time, e.g. because new classes are being added. It would be ideal to just learn the additional classes, instead of re-learning the network on the entire dataset each time a new class is added. It is a problem which affects many types of neural networks, including many modern CNN architectures. When a pre-trained CNN starts learning a different dataset, it will typically un-learn the source dataset in just a few iterations of training. The backpropagation algorithm will change many CNN parameters rapidly and indiscriminately, not knowing which of the active neurons were important for performing the old task. Some methods commonly employed in CNN architectures that help with this problem are *max pooling*, *dropout* and using ReLU activation functions, as all these methods result in slightly reduced number of active neurons [71, 72].

Srivastava et al. [71] introduce the *local winner-take-all* (LWTA) architecture, in which neurons are split into a number of blocks which are then organized into layers. The neurons in a single block compete, and only the winning neurons contribute any information to the next layer. The authors use the simplest competition function: the maximum function, meaning the neuron with the highest activation value wins. They use the identity activation function, and non-linearity in a layer is achieved by turning off the activations of non-winning neurons. Since only winning neurons contribute, they are the only ones which are corrected by backpropagation algorithm during training, and this approach results in reduced catastrophic interference. The LWTA layer is similar to a max-pooling layer, but without the downsampling (the number of features is not reduced, but made more sparse.

Hinton et al. [73] develop a useful technique of *knowledge distillation* in CNN frameworks. It is a technique for transferring knowledge from a trained complex model (e.g. an ensemble of CNNs) to a simpler model which has fewer parameters. Their key observation is that the objective function usually only deals with exact class labels, and that it does not know which classes are easily confused, and which are easy to differentiate. For example, a car is similar to a truck, but not to an apple, but this information is not usually present in the training data. After a complex model is trained, soft probability distribution over classes provided by its softmax layer is used to train the distilled model on the transfer set. More precisely, for a class i with the logit z_i, the class probability q_i is calculated by the softmax output layer of a complex model as:

$$q_i = \frac{\exp\frac{z_i}{T}}{\sum_j \exp\frac{z_j}{T}} \tag{2.8}$$

where T is a temperature parameter, usually set to 1. Using a higher value of $T = 10$ authors produce softer probability distributions over classes. A smaller model is then trained using a cross entropy objective function with the soft targets. If the correct labels are also known for some or all of the transfer set, the cross entropy with the correct labels is also used, and the weighted average of the two is used as the objective function. The smaller model is also trained using a high temperature parameter $T = 10$, but the temperature is reduced back to $T = 1$ during testing phase. The method produces small models that generalize and perform well. While it does not deal with the problem of catastrophic interference directly, it proved to be useful in future frameworks [74, 75] that do address that problem.

Li and Hoiem [74] combine the ideas of knowledge distillation and fine-tuning in their *learning without forgetting* approach. Given a network which was trained to solve a specific task, they introduce additional parameters to enable the network to solve a new task as well. The network parameters are split into three sets: the shared parameters θ_s, the parameters specific for the old task θ_o, and the parameters specific to the new task θ_n. The sets of parameters θ_o and θ_n belong to nodes of the output layer, while all other parameters are shared. Therefore, the added set of parameters θ_n is relatively small. First they record all outputs y_o of the original network on the new data. They then train the network to minimize the loss on the new task. In the first stage of training they freeze the θ_s and θ_o, and train θ_n to convergence. In the second stage they train all the parameters jointly. The Figure 2.8 shows the network structure, and illustrates the learning process for new tasks. The orange part of the network contains the parameters θ_n that will learn to solve the new tasks. The blue parts of output layer contain the parameters θ_o that solve the old tasks, while the remaining parts of the network contain the shared parameters θ_s. While learning the new tasks, they use the distillation loss [73] with temperature parameter set to $T = 2$ to encourage the output probabilities for each image to be close the the pre-recorded output probabilities for old tasks.

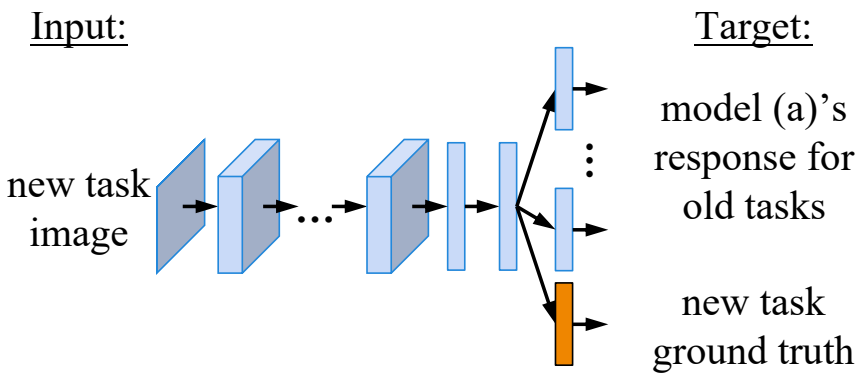


Figure 2.8: Learning a new task in *Learning without forgetting* framework. The orange block contains the parameters that learn to solve the new task. The target is composed of pre-recorded responses of original model on old tasks and of new task ground truth. Figure reproduced from [74].

Shmelkov et al. [75] take a slightly different approach. They propose a framework for

learning new object detectors which consists of two networks, referred to as networks A and B. Network A is the original object detector, trained on the old task, and it remains frozen during training of the new classes. It is used to select proposals corresponding to the old classes, and to compute the distillation loss. Network B is the detector which is adapted for new classes. It is obtained by increasing the number of outputs in the original network, so the network contains outputs for both the old and the new classes. After training, network B is able to perform both old and new detection tasks. During training a diverse set of samples is selected among all samples of the new training dataset by the network A, favoring non-background samples. The responses to those are recorded and used to compute a distillation loss to measure the discrepancy between the two networks during the training of network B, to keep it from catastrophically forgetting the old classes. Authors show the framework can be used to add a single class, multiple classes simultaneously, or even sequentially without significant losses of performance.

2.2.4 Unsupervised learning

Even though unsupervised learning methods cannot compete with supervised methods in terms of classification performance, they offer some other potential benefits that make them worth considering. They can be used to obtain image representations that are input to a stand-alone classifier, which is useful in scenarios where the set of class labels is changing frequently. The cumbersome process of training a feature extractor needs to be done only once, and only the classifier needs to be retrained if the set of classes changes.

Dosovitskiy et al. [76] present an unsupervised CNN framework which aims to facilitate learning on relatively small datasets. They sample a number of 32×32 patches from each input image, in regions with considerable gradients. Each of the patches is considered to be a separate class, and is assigned its own class label. Next, they apply a composition of elementary transformations (e.g. geometric and color transformations) multiple times to each patch, and each transformed version of a patch is assigned the same class label as the original. This increases the size of the dataset and also improves regularization. This dataset of labeled patches is then used to train a CNN. The CNN can then be used to produce image representations for classification by spatially pooling concatenations of features from all network layers (except the final softmax layer). Authors report competitive results: 84% classification accuracy on CIFAR-10 [49] dataset, and 87% classification accuracy on Caltech-101 [77] dataset while using 1884 features.

Goodfellow et al. [78] present *generative adversarial networks* (GANs). It is an unsupervised learning framework in which two parametric functions (called a *generator* and a *discriminator*) are competing in a min-max game. The generator $G(\mathbf{z})$ is modeled by a CNN with ReLU and sigmoid activations, and it maps noisy input $\mathbf{z}$ to data space. The discriminator $D(\mathbf{x})$ is modeled by a CNN with maxout [79] activations, and it outputs the estimated probability of

image data $\mathbf{x}$ being sampled from real data (instead of generated by generator). The objective function of the min-max game is:

$$\mathscr{L}_{\text{GAN}}(G,D) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}(\log D(\mathbf{x})) + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}(\log(1-D(G(\mathbf{z})))) \,. \tag{2.9}$$

The convergence in min-max objective can be interpreted as minimizing the Jensen Shannon (JS) divergence. After convergence is achieved, the generator network is able to produce artificial images which the discriminator cannot differentiate from the source images. One drawback of this method is the possibility of occurrence of mode collapse: generator can learn to generate a very limited set of images (e.g. a single class).

Radford and Metz [80] propose a set of guidelines and constraints on the GAN architecture that make them more stable to train, such as using strided convolutions in the discriminator and fractional-strided convolutions in the generator, using the batch normalization in both, and changing activation functions to ReLU and tanh in the generator and LeakyReLU in the discriminator. They name the improved architecture *deep convolutional generative adversarial networks* (DCGANs). Authors demonstrate that concatenation of spatially pooled convolutional features of the discriminator network serves as an image representation which is very successful in the context of image classification. They achieve a classification accuracy of 82.8% on the CIFAR-10 [] dataset using an SVM classifier. Even though this is lower than the results obtained by [76], the adversarial networks were trained on a different dataset (ImageNet), which demonstrates good generalization properties.

Arjovsky et al. [81] present a solution to mode collapse problems in the form of *Wasserstein generative adversarial networks* (WGANs). They demonstrate that in the context of GANs, *Wasserstein's probability distance* (also known as earth-mover's distance) is a measure preferable to other probability distance and divergence measures, such as total variation distance, Kullback-Leibler divergence and Jensen-Shannon divergence. They propose some adaptations of the network structure, and a learning algorithm which ensures the convergence of the min-max objective corresponds to minimizing the Wasserstein's distance. This results in simplified learning of adversarial networks with meaningful interpretations of learning curves, and without any mode-collapse problems noticed by the authors.

2.2.5 Short image descriptors

Generic image classification systems typically do not prioritize shortness of image descriptors. Most methods for producing very short image representations are motivated by image storage and retrieval systems. Short image representation has a low memory footprint and is used for very fast retrieval of images (e.g. retrieval of images that are visually similar to the query image).

Such representations must capture the visual attributes of the image, but do not necessarily perform well in image classification tasks.

Oliva and Torralba [82, 83] develop the GIST descriptor for scene recognition. It is a very low dimensional representation of the scene that captures perceptual features meaningful to a human observer, such as naturalness, openness, roughness, etc. It is calculated by first subdividing the image into a 4×4 grid, and then concatenating the average energies of responses of 8 orientation filters on 4 scales for each cell. Torralba et al. [84] convert the GIST [82] descriptor to a compact binary code that represents an image with a few hundred bits. This descriptor can be used to perform fast image searches within millions of images on a single personal computer using an approximate nearest neighbor approach, with Hamming distance measure. They use boosting [85] and restricted Boltzmann machines [86] to learn the minimum number of bits necessary for the Hamming distance to still express visual similarity between images.

Torresani et al. [87] introduce a descriptor aimed at object category recognition. It consists of quantized outputs of weakly trained classifiers of various objects, termed *classemes*. They use a two-stage approach: once-only classeme learning; followed by object category-related learning tasks. The stages use distinct training sets. They do not aim for the classifiers to semantically encode meaningful categories (e.g. "water"), but rather measure simple properties (e.g. the presence of a water-like texture in the scene) that add up to describe a more complex scene (e.g. a person swimming). They demonstrate adequate performance can be obtained with small descriptor sizes (as small as 200 bytes).

Bergamo et al. extend the idea of classemes in their PiCoDes [88] and meta-class [89] descriptors. The purpose of both descriptors is efficient image indexing in large databases. PiCoDes size ranges from 16 to 256 Bytes, while meta-class are less than 2 kB.

Instead of crafting a very short descriptor from scratch, it is possible to start with an existing long descriptor and apply dimensionality reduction techniques to find the minimum descriptor size which still offers good descriptive properties. Principal Component Analysis (PCA) [90, 91] is one popular dimensionality reduction method. Product Quantization (PQ) [92, 93] is a more recent method with good results. It decomposes the feature space into a Cartesian product of low-dimensional subspaces, which are then quantized separately. Both approaches are able to identify parts of image representation that are not activated for a particular target dataset [68].

Babenko et al. [94] show that outputs of several high-level layers of CNNs can be used in image retrieval applications, even if the CNNs were trained on an unrelated classification dataset. They also show the PCA can successfully be used in such a scenario to reduce the image representation size, down to 256 components with hardly any degradation.

Zhao et al. [95] connect a hash-function layer to the FCa and FCb layers of [96], and use multi-label image data to learn the parameters of the added layer in a supervised way. This results in a compact image representation suitable for image retrieval systems with multilevel

semantic similarity.

Lin et al. [97] add a hidden layer with sigmoid activation functions to a CNN pre-trained on ImageNet, and then fine-tune the network to target dataset. The outputs of the added layer are binary-like, and can be easily quantized to bits, to obtain a compact image representation suitable for image retrieval.

Liu et al. [98] present the *deep supervised hashing* (DSH) method for producing very short image descriptors with emphasis on preserving similarity. They design a CNN structure that takes pairs of images during training and tries to learn the parameters that would result in discrete values on output (e.g. $+1/-1$). They use supervised information of image pair similarity (obtained from e.g. image labels) in the loss function to maximize the discriminability of the output space. The output consists of 12 to 48 bits, and is suitable for fast image retrieval tasks.

Preliminary research for this thesis [99, 100] explores methods for reducing the size of state-of-the-art hand-crafted image descriptors. The impact of size reduction to the performance of traffic scene classification is measured. A short image descriptor that combines compacted spatial Fisher vectors and GIST descriptor in a lossy encoding scheme is proposed. Classification performance is retained for the descriptor size as low as 48 bytes per image.

The author of this thesis is not aware of any previous work that proposes and evaluates very short image descriptors for classification of traffic scenes.

Chapter 3

Improving fleet management with computer vision

In the broadest sense, fleet management systems are a combination of various hardware and software components which enable the tracking and managing of a fleet of vehicles. Management can include a range of features, such as vehicle telematics, maintenance and financing, driver management, route planning and scheduling, fuel management, cost analysis, driver performance analysis and risk prediction. It usually includes real-time tracking of vehicles using a global navigation satellite system (GNSS). It can also include real-time threat detection and alarming, as well as historical data analysis. It can help reduce a wide variety of risks and improve efficiency. Some fleet management systems only monitor vehicles of a single company, while others manage vehicles of multiple companies, as well as any number of personal vehicles. In some countries (e.g. China, India) GNSS tracking is mandatory for certain classes of commercial vehicles [101, 102, 103]. Some toll road payment systems (e.g. HU-GO E-toll system in Hungary [104]) have open interfaces through which fleet management systems can deliver proof of traveled distance, for automated distance-based payment. A screenshot of an exemplar fleet management system user interface is shown in Figure 3.1, illustrating the use of real-time tracking and historical data analysis features. This chapter will briefly discuss the hardware and software architecture of fleet management systems, as well as present some improvements which can be achieved by introducing a computer vision subsystem.

3.1 Fleet management architecture

The fleet management systems have a client-server architecture, the clients being the tracked vehicles, and the server being a single computer or a cluster of computers behind a common access point.

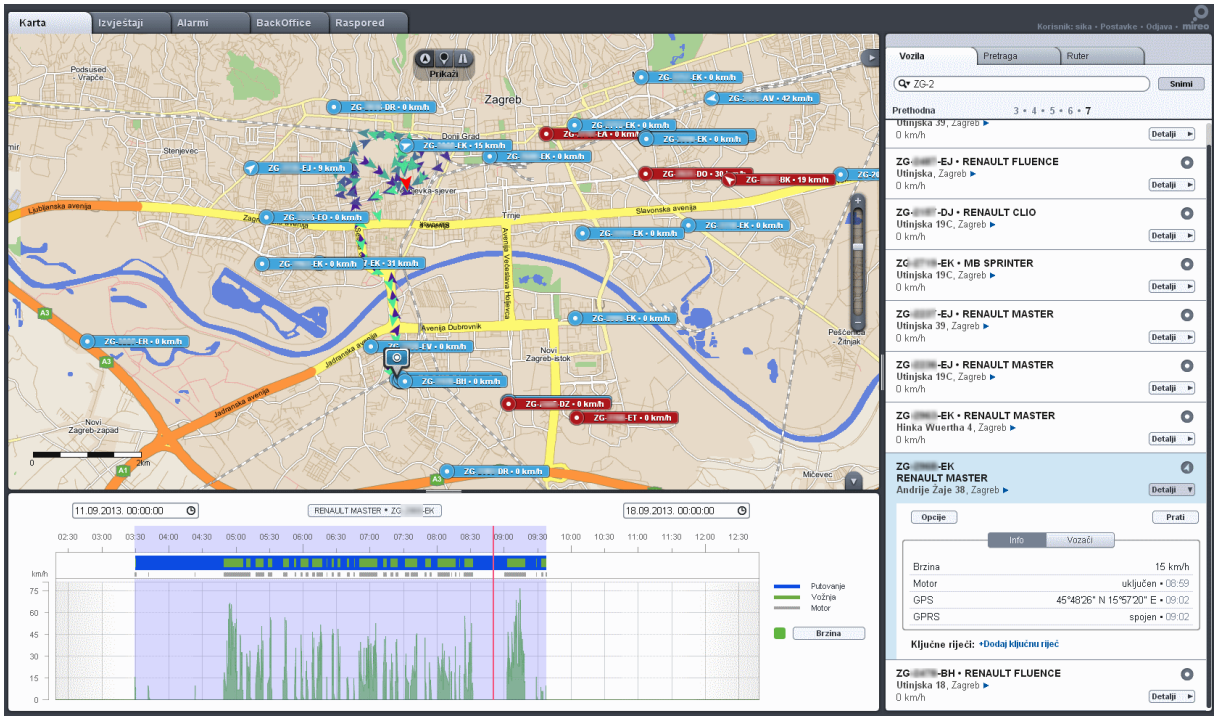


Figure 3.1: A screenshot of a fleet management user interface. The right pane contains a filterable list of vehicles with the addresses of their current locations, speed, and other status information. The locations, speeds and orientations of the vehicles from the list are displayed on the map in real time. The historical data for a selected vehicle is shown on the bottom pane, and the map-matched positions for the same vehicle are shown on the map as well, in the form of blue-green arrows. All vehicle plate numbers have been blurred for privacy.

3.1.1 Clients

Each tracked vehicle is equipped with a device connected to a variety of sensors which collects the relevant data and transmits it to the central server for processing. The collected data almost always includes the vehicle position, velocity and course (bearing), as determined by a global navigation satellite system. It can also include a variety of other information: status of ignition, battery voltage, current fuel level, total fuel consumption, accelerometer readings, multiple temperature sensor readings, current engine RPM, state of acceleration and breaking pedals, driver sign-ins and sign-offs. Some data is obtained from sensors embedded in the device, some from in-car computer (e.g. via CAN bus or OBD), and some from external sensors. Devices installed into specialized vehicles can also include the state of their equipment, such as taxi meter in taxis, plow level and salt dispenser width in snow plows, ambulance siren and lights status in ambulances, etc. Most of data is transmitted as-is to the central server, but some is processed on the device prior sending. For example, the accelerometer sensor is typically polled many times per second, so it is prudent to interpret this data on the device and only send the interpretations (e.g. harsh cornering detected) to conserve the bandwidth. Depending on the device configuration, the data collection can occur in fixed time intervals (e.g. once per minute), as well as when certain thresholds are met (e.g. a vehicle traveled 100 meters, vehicle course changed more than 30 degrees, vehicle ignition was turned off or on). The data is typically transmitted over a cheap low-bandwidth channel, such as GPRS. If the data can not be transmitted (e.g. server down for maintenance, vehicle is in an underground garage), then the data needs to be stored in the permanent memory of the device, to be transmitted at a later time. This behavior is also used if data transmission is prohibitively expensive due to vehicle being in international roaming. The vehicle tracking hardware is typically inexpensive (depending on the size and value of the fleet). It usually draws power from the vehicle battery, so power-efficiency is important, especially while the engine is turned off. The tracking device software is easy to configure and ideally does not require frequent updates, as updates increase data costs. Software updates are also cumbersome to install, as some vehicles might be in locations with no data connection (e.g. underground garages) for long periods of time (e.g. weeks or months).

3.1.2 Server

The data is collected, processed, analyzed and stored in a centralized system. The centralized system can be single server (suitable for smaller fleets of vehicles, up to several thousand), or a cluster of computers. The software suite consists of data receivers, streaming data analyzers, storage subsystems (persistency, replication, indexing), business analytics engine, real-time monitoring and alarming subsystems. The cost of the hardware components is typically not an issue, the hardware can be as powerful as it needs to be. Compared to the client software, server

software is much easier to update, modify and reconfigure. If necessary, server can easily communicate with many different types and versions of client protocols, and server software often implements workarounds for various client bugs and design flaws. The amount of persistent storage required for the server depends on factors such as the number of the monitored clients, the length of the required data keeping time period and the average frequency of client data sampling. Typically it is in the order of magnitude of several megabytes per vehicle per month.

3.2 Proposed vision-based improvements

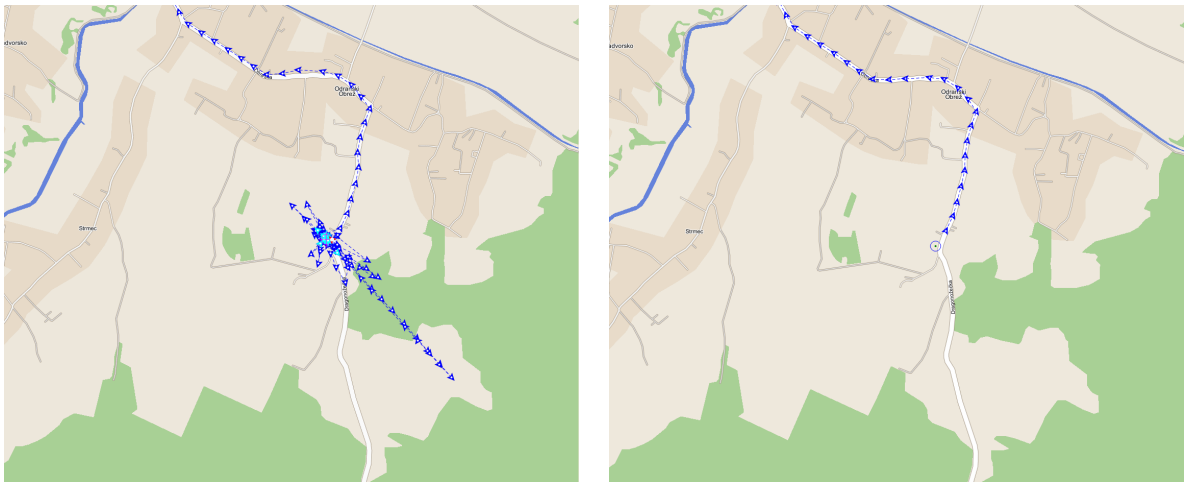
The proposal of this thesis is to use inexpensive dashboard-mounted cameras as additional sensors attached to the vehicle tracking device. Note that popularity of consumer dashboard-mounted cameras has increased in recent years [105], as they are being used to prevent fraudulent claims after traffic accidents. Additionally, front-facing cameras are standard equipment in some vehicles with advanced driver assistance systems. The visual information can be used to improve several aspects of fleet management systems, as will be discussed shortly. The proposal of this thesis is to use image categorization to determine the type of traffic environment the vehicle is currently in, as well as to detect interesting events and traffic scenarios. Such a feature would enable the improvement of the fleet management systems as follows:

- The position filtering, map-matching and route reconstruction could be improved by giving a prior on the road type.
- The position filtering, map-matching and route reconstruction could be improved by giving an additional prior on the GNSS accuracy.
- The real-time monitoring, alarming and historical data reporting could be improved by detecting some important events or traffic scenarios that are hard to detect without use of vision.

3.2.1 Improving position filtering, map-matching and route reconstruction

The map-matching and route reconstruction can not always be done in an unambiguous way. Sometimes the received coordinates are such that several almost equally plausible explanations exist. The usage of visual information can help in those cases, by giving additional priors to type of road the vehicle is thought to be on. Additionally, sometimes the received GNSS data is in direct conflict with the known map data, and there are no plausible explanations. This can happen because of incorrect map data, because of GNSS errors, or because of violation of traffic rules, such as driving the wrong way in a one-way street. The use of visual cues can help in two ways to resolve such cases. Firstly, it can provide priors on the type of the

matched road. Secondly, it can provide priors on the probability of GNSS errors. The typical GNSS sensor already provides some information about its perceived precision and reliability, such as the HDOP value (horizontal dilution of precision), and the number of visible satellites. Unfortunately, in some cases the sensor can report perfect accuracy with great number of visible satellites, while still delivering incorrect coordinates. One example of such errors is the *multipath error* [106], in which the GNSS signal does not travel directly from the satellite to the received, but is instead bounced off of objects in local environment. This often happens near tall objects (e.g. buildings), or if the receiver is under objects such as overpasses, tunnels, toll booths. Consider the example in Figure 3.2. The Figure 3.2a shows the set of raw GNSS readings. Note that it appears the vehicle has driven through a forest. Figure 3.2b shows the map-matched set of coordinates for the same vehicle, after most of the incorrect readings have been excluded by a heuristic designed to detect multipath errors. If the incorrect positions were positioned on an existing road, then the heuristic would fail to detect them, and the vehicle owner would be automatically notified that the vehicle is being towed away.



(a) The readings as received by a GNSS sensor.

(b) After heuristic filtering and map matching.

Figure 3.2: A series of vehicle positions and orientations is shown on a map as a series of blue arrows connected by a dashed line. The figure on the left shows the original readings, as received by a GNSS sensor. Note that many readings fall outside of known street network, and it appears the vehicle moved through a forest. A heuristic algorithm guesses which positions are incorrect GNSS readings, and filters them out. The figure on the right shows the positions which remain after map-matching and filtering have been performed.

A potential use of visual cues is providing priors to the type of vehicle environment, especially the type of the road, which helps GNSS map matching disambiguation and improves vehicle route reconstruction quality. One of more frequent problems is determining which of the two parallel roads the vehicle traveled: fast one or a slow one, as illustrated in Figure 3.3. In this case the differentiation between a fast road and a slow road should be useful. Less frequent are other types of ambiguities, but they do exist. For example, in the Croatian city or Rijeka there are several parallel roads located on a mountain slope. Several tunnels on those roads often

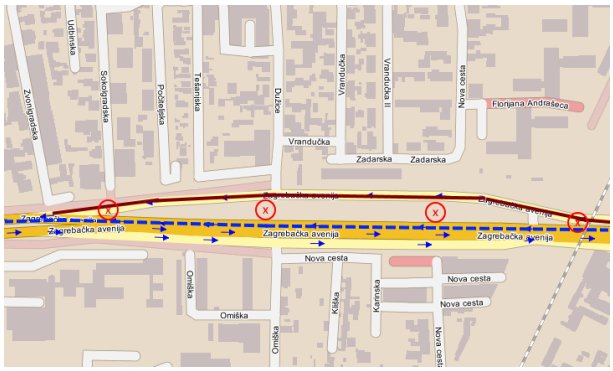


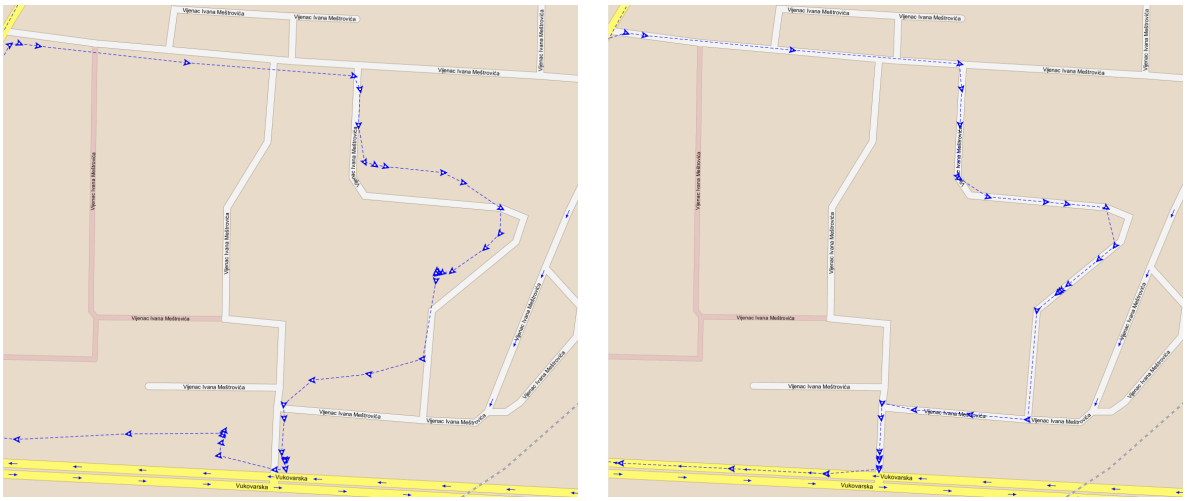
Figure 3.3: Ambiguous route reconstruction due to poor GNSS precision. GNSS readings are marked with circled X-marks. It is equally plausible that the vehicle traveled the fast road (in dashed blue) and the slow road (in solid red).

cause the loss of GNSS signal and cause poor GNSS accuracy upon exiting the tunnels. That confuses the map matching algorithm, which could benefit from knowing whether the vehicle is still in the tunnel, near the tunnel exit, or outside the tunnel.

Another way to improve the map matching and route reconstruction is by providing priors on the probability of GNSS accuracy loss. The vehicle being in a tunnel indicates a very high probability of loss of the GNSS precision, and often causes a complete loss of GNSS signal. The vehicle being near the tunnel exit predicts the return of lost GNSS signal, followed by a period of low GNSS precision, as it takes some time to re-establish the GNSS fix. The presence of an overpass and toll booths indicate a moderate probability of GNSS accuracy loss, due to possibility of multipath errors. The vehicle being on an open road indicates a very low probability of multipath errors. The vehicle being in a settlement indicates neither low nor high probability of multipath errors, since there is a possibility of being near tall objects (e.g. buildings), but they are not necessarily in the field of view of a dashboard camera.

Map-matching

One of the features of quality fleet management systems is the automatic map-matching, i.e. the aligning of coordinates received from a tracking device onto a road map. The coordinates received by a tracking device are rarely perfectly accurate, and aligning them with known road geometry can increase the accuracy of measured distance, and the correctness of addresses obtained by reverse geo-coding. This can be very useful for vehicles that do not often engage in off-road activities. Most tracking devices report some measure of accuracy of sent coordinates (e.g. rms, CEP, SEP, R95), which can be used to estimate the coordinate probability distribution. By assuming the vehicle is located on a road, a more likely vehicle position can be determined. Consider the example shown in Figure 3.4: the coordinates received by the tracking device (Fig. 3.4a) are actually mean values of the probability distributions, and are misaligned from the road network. A map-matching algorithm examines more likely candidates for each received



(a) Before map matching.

(b) After map matching.

Figure 3.4: A series of vehicle positions and orientations is shown on a map as a series of blue arrows connected by a dashed line. The figure on the left shows the original readings, as received by a GNSS sensor. Note the positions do not align with the streets. The figure on the right shows the corrected positions, obtained by applying a map-matching algorithm.

position, and obtains a more convincing reconstruction of vehicle movement, shown in Figure 3.4b.

A simple map-matching algorithm is the one that only considers and matches a single GNSS reading, disregarding the previous readings. It examines the nearby road network, finds all plausible position candidates, calculates their likelihoods and picks the most likely one. The assumption is that the actual vehicle coordinates lie on the road, and that the vehicle course approximately matches the orientation of the road. Consider the example in Figure 3.5. The coordinates and course received by a tracking device are labeled as *reading*. The dotted circumference indicates the 99% confidence radius for this reading. Each road is approximated by a series of line segments, called road segments. The road segments lying within the 99% confidence radius are labeled as s_1 to s_{10}, in the order of increasing distance from the mean. The point on the road segment closest to the mean is a good candidate for the map-matched value. Since the course needs to be taken into account, each road segment contributes up to two candidates: one candidate for each permitted travel direction. The candidates are scored according to the distance from the mean and the difference in course, and the one with the highest score is the output of the map-matching algorithm. In more advanced systems, several candidates with high enough scores are the output, and the subsequent route reconstruction algorithm decides which candidate results in the most plausible route.

There are many ways to score the candidates, but the basic idea is simple. Let $status = (x^s, y^s, c^s, v^s, acc^s)$ be a status information sent by the tracking device, where (x^s, y^s) are the mean coordinates, c^s is the course, v^s is the speed, and acc^s is the accuracy measure. Let $candidate = (x^c, y^c, c^c)$ be a map-matching candidate, where (x^c, y^c) are its coordinates and c^s

Figure 3.5: Map-matching a single GNSS reading. A total of 10 nearby road segments have been collected, marked with labels from s_1 to s_{10} in the order of increasing distance from the GNSS reading. The dotted circumference depicts the road segment collection radius. For each collected segment, the point closest to the GNSS reading is considered, and one map-matching candidate is contributed for each of the permitted traffic directions of that segment. A total of 19 candidates are considered in this example, labeled from c_1 to c_{19}. Candidates c_{2k-1} and c_{2k} are contributed by segment s_k. For example, candidates c_3 and c_4 are contributed by segment s_2, while candidates c_7 and c_8 are contributed by segment s_4. Note that segment s_{10} only permits one-way traffic, so it only contributes one candidate, c_{19}. Even though the candidates c_1 and c_2 are the closest to the GNSS reading, the best candidate in this example is c_3, due to much better course match.

is the course. A simple scoring function p_c is given by the Equation 3.1.

$$p_c(candidate, status) = \frac{p_{dist} + p_{cdiff} \cdot p_{cc}}{1 + p_{cc}} \tag{3.1}$$

Where p_{dist} is the candidate score based on the distance from the mean, p_{cdiff} is the candidate score based on the course difference, and p_{cc} is the probability of course information being correct. Although this scoring function is a very simple ad-hoc formula, it is easy to compute and works well in practice. The distance based score p_{dist} is usually calculated by assuming a multivariate normal distribution, and depends on the distance from the mean, as well as on the accuracy measure acc^s. The course difference score p_{cdiff} depends on the c^s and cc only. The p_{cc} value depends on both v^s and acc^s values, as the course information is derived from successive GNSS readings. This means it is less accurate the slower the vehicle is moving.

The visual-based improvements can be achieved if the status message is extended to contain visual information, and if additional road attributes are considered when scoring the map-matching candidates. Let $status = (x^s, y^s, c^s, v^s, acc^s, cl^s, p^s_{clc})$ be the extended status, where cl^s is a class label assigned to image captured by a dashboard camera, and p^s_{clc} is the probability of class label being correct (given by the classifier). The probability of correctness of the class label can be determined a posteriori (it can be estimated by classifying a dataset with known ground truths). Detection of out-of-distribution classifier input [107] can be used to prevent overly optimistic estimates of p^s_{clc}. Let $candidate = (x^c, y^c, c^c, ra^c)$ be the extended map-matching candidate, where ra^c is the set of candidate road attributes, such as road type, speed limit, etc. Let $p_{err}(cl^s)$ be a function that assigns the probabilities of multipath errors to class labels. Then a simple way to calculate the score of GNSS reading reliability is given by the Equation 3.2.

$$p_{pc} = 1 - p_{err}(cl^s) \cdot p^s_{clc} \tag{3.2}$$

The improved candidate scoring is given by the Equation 3.3.

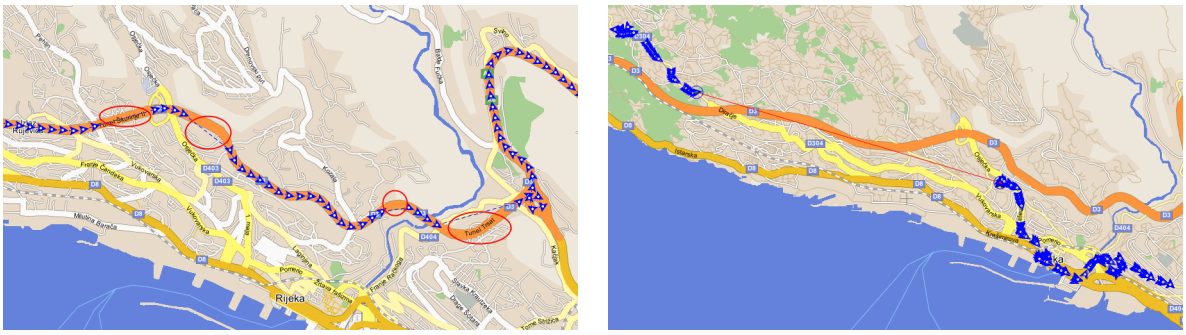
$$p_{cv}(candidate, status) = \frac{p_{dist} \cdot p_{pc} + p_{cdiff} \cdot p_{cc} \cdot p_{pc} + p_{rt} \cdot p^s_{clc}}{p_{pc} + p_{cc} \cdot p_{pc} + p^s_{clc}} \tag{3.3}$$

Where p_{rt} is the score based on the matching of road attributes with the visual evidence, and depends on the road attributes ra^c and status class label cl^s.

Improving route reconstruction

A route reconstruction is a process of reconstructing or guessing the exact route the vehicle traveled, as evidenced by all available sensor inputs. It heavily depends on the quality of map-matching algorithm, so using vision to improve map-matching already improves the route re-

construction quality. But there is an additional way in which visual information can be useful. Route reconstruction implies the reconstruction of parts of route between each two consecutive GNSS readings. This can be done in a variety of ways, e.g. by assuming the fastest path, or by finding several plausible routes and choosing the one that best matches the time duration between the points. This is easy when the distance between readings is small, but becomes harder as the distance increases. The most interesting cases are the ones where the loss of GNSS signal occurs for prolonged periods of time. Two examples of GNSS signal losses in the Croatian city of Rijeka are shown in Figure 3.6. The first example, in Figure 3.6a, shows a case in which the GNSS signal is lost upon tunnel entry, and fix was quickly re-established after each tunnel exit. That case can be easily solved, provided the improved map matching ensures the low-accuracy coordinates upon tunnel exit are properly matched. The second example, in Figure 3.6b, shows a more serious case, in which the loss of GNSS signal lasted for 10 minutes.



(a) Intermittent loss of GNSS fix, caused by driving through tunnels.

(b) GNSS fix lost for ten minutes, cause unknown.

Figure 3.6: A series of vehicle positions and orientations is shown on a map as a series of blue arrows connected by a dashed line. The figure on the left shows the case in which an intermittent loss of GNSS fix is caused by driving through four tunnels. The locations of the tunnels are marked by red ellipses. The figure on the right shows a case in which a GNSS fix is lost due to an unknown cause, and then re-established 10 minutes later.

If the vehicle tracking device continues to send status reports even though the GNSS is not available, then the visual information can be used to more precisely select between the several plausible route reconstruction candidates. Let R be a route reconstruction candidate for a set of vehicle statuses S, and let $s(R,S)$ be a route candidate scoring function which does not use visual information. Additionally, let p^s_{ra} be a score of a match between road attributes at time of status s with class label of status s and let p^s_{clc} be the probability of class label of status s being correct. Then a simple route candidate scoring function s_v, which uses visual information, is given by Equation 3.4.

$$s_v(R,S) = \frac{1}{1+\alpha}\left(s(R,S) + \alpha \cdot \frac{\sum_{s\in S} p^s_{rt} \cdot p^s_{clc}}{\sum_{s\in S} p^s_{clc}}\right) \tag{3.4}$$

Where α is score weighting factor used to regulate the degree to which visual information

impacts the route scoring.

3.2.2 Improving the real-time monitoring, alarming, and historical data reporting

Besides improving the quality of vehicle route reconstruction, the visual information can be used to detect some interesting events, or to better explain them. For example, a human operator monitoring the fleet of vehicles in real-time sees a list of vehicles which have lost the GNSS signal, and wants to investigate the reasons behind GNSS signal disappearance for each of those vehicles. It might be useful to indicate which of these vehicles have entered a tunnel, so the operator can focus on other cases. The operator also sees the current velocity of each vehicle. If a vehicle has stopped outside of settlement, or if the current vehicle speed is much lower than the speed limit for the current road, this might indicate a problem. Using visual cues to identify dense traffic scenarios can help the operator more easily identify the reason behind sudden vehicle stopping or slow driving. Other explanation of sudden stopping could be encountering a toll booth.

The set of interesting events and traffic scenarios partially depends on the types of monitored vehicles and businesses they are conducting. For example, a highway maintenance crew might want to know which of their vehicles are currently driving on shoulder lanes, which can be done by visual scene categorization. The driving on the shoulder lane would be very hard to detect by relying on GNSS and map data only, as neither consumer GNSS nor map data are typically precise enough to do this reliably. A taxi company might want to detect if their vehicles are stopping on forbidden locations, such as municipal bus stops. A transport company might want to measure how often their trucks are driving behind other vehicles, and how close. Some companies might care about detecting gas stations, toll booths, railroad crossings and ferry embarkations, others might not, and instead wish to know how often their vehicles get stuck in dense traffic. It would be very hard to accurately predict all interesting traffic scene categories ahead of time, so a useful feature of the proposed image categorization system would be to have the set of classes remain open, and that it can be easily changed in the future.

Chapter 4

Integrating an image categorization component into the fleet management architecture

This chapter outlines the practical implementation problems encountered when an image categorization framework is integrated into a fleet management system. The problems are illustrated by analyzing two naive approaches before proposing the architecture which solves all important issues in a satisfactory way, and enables most desired features with little compromise.

4.1 Server-side classification

The simplest and most straightforward way to implement the image categorization subsystem is to configure the mobile clients to simply send the entire image captured by a dashboard mounted camera along with all other sensor readings. There are several issues with this approach. First issue is that collecting and storing full images of traffic scenes raises potential privacy concerns. Depending on the local laws of each country in which a fleet management is used, a special effort might have to be made to limit the access to images and anonymize the privacy-sensitive information. Another problem with sending entire images is the drastic increase in data traffic. A single 640×480 full-color bitmap is around 900 kB in size. It can be compressed to around 200 to 300 kB by using a lossless PNG compression, or down to 20 to 50 kB by using a lossy JPEG compression. A typical data packet containing sensor readings is around 100 B in size, which is at least two orders of magnitude smaller than a compressed image. If a tracking device sends one image every 60 seconds, it will transmit between 30 MB and 300 MB of data daily, instead of the usual 150 kB, which may significantly increase the costs of data traffic. The third problem is the increase in storage capacity. While the tracking device is not able to connect to the fleet server, it needs to store all unsent data. During a week long international trip with

no data roaming, the device would need an extra 385 MB for storing 640×480 JPEG images once per minute, or extra 140 GB for storing 640×480 PNG images once per second. Once the central server receives the images, it may wish to store them for long periods of time, so that it can re-evaluate the data if new image categories are added, or if a better classifier is trained. Storing the images of resolution 640×480 for a fleet of 10000 vehicles would require extra storage starting from 5 TB up to 30 PB per year, depending on the compression rate and the frequency of image collection.

A tabular overview of the required yearly storage for 1000 vehicles, depending on the format of the transmitted image and the frequency of status updates is given in Table 4.1. Note the realistic average update intervals are between 2 and 60 seconds, while those of 10 minutes are extreme and are included in the table for completeness. Also note that there no reason to ever use a bitmap format, as PNG format is much smaller with equal (lossless) quality. The images of resolution of 160×120 could be considered to be too small for the human to properly discern some important details of the scene. However, they were also included in the table, to illustrate that even images with small resolution and strong lossy compression would still have a significant impact on the storage requirements.

One benefit from the approach in which images are stored on the server would be that the collection of traffic scene images would be steadily increasing. An annotator could periodically label the new images and thus produce a larger training dataset. With bigger datasets it is often possible to construct and train better performing image classifiers, especially for rarely occurring image classes. It would be possible to periodically retrain the existing classifier using the new image data, and then even re-evaluate the historic data.

4.2 Client-side classification

A different approach would be to perform the classification on the client tracking devices, and only send the class labels to the server. This would solve all bandwidth and storage issues, but would introduce different problems. The first problem is that the re-examination of historic data becomes impossible: the images are lost, and it is not possible to re-classify them after new class labels are added, or if a better version of a classifier is trained. The second problem is that after every addition of new class labels it becomes necessary to update every client in order to make them aware of the new labels. This will not only increase the data transfer costs, but can also cause the set of classes across clients to be out of sync while the update is being rolled out. Some clients might be in areas with no data connection for weeks (e.g. international trips). Until they connect to the server and receive the update, they will continue to use the obsolete set of labels, which will then be sent to the server. The synchronization issues could be somewhat alleviated by introducing a significant delay before the downloaded update becomes active, so

Table 4.1: Approximate data traffic generated by a 1000 tracking devices in a single year, depending on the format of transmitted image and the frequency of status updates.

	average time between status updates			
image included in the status packet	2 seconds	10 seconds	1 minute	10 minutes
640×480 bitmap	12 PB	2 PB	440 TB	44 TB
640×480 PNG	3 PB	717 TB	119 TB	11 TB
640×480 high quality JPEG	1 PB	329 TB	54 TB	5 TB
640×480 medium quality JPEG	574 TB	114 TB	19 TB	1 TB
640×480 low quality JPEG	144 TB	28 TB	4 TB	491 GB
160×120 bitmap	826 TB	165 TB	27 TB	2 TB
160×120 PNG	473 TB	94 TB	15 TB	1 TB
160×120 high quality JPEG	215 TB	43 TB	7 TB	736 GB
160×120 medium quality JPEG	86 TB	17 TB	2 TB	296 GB
160×120 low quality JPEG	43 TB	8 TB	1 TB	149 GB
no image	734 GB	146 GB	24 GB	2 GB

that all clients start using the same set of labels at a preset time in the future. Another way to handle the syncing problems it is to allow the clients to send both the new and old set of labels until server confirms that all the clients have been updated. In any case, the addition of new class labels in this setup is an expensive, slow and complicated process.

4.3 Proposed framework

To summarize: the server-side classification approach suffers from bandwidth and storage limitations, while the client-side classification approach is too rigid, it complicates and delays the future improvements while also preventing the reevaluation of historical data. The framework proposed by this work solves all of the problems above. The proposal is that the clients calculate and transmit very short image representations, called *descriptors*, which can then be classified and stored on the server side. The overview of the framework is shown in Figure 4.1.

4.3.1 Desirable descriptor properties

The desirable properties of the *descriptor* are: *descriptiveness*, *compactness* and *computational simplicity*.

Descriptiveness is the most important property of the descriptor. The descriptor needs to cap-

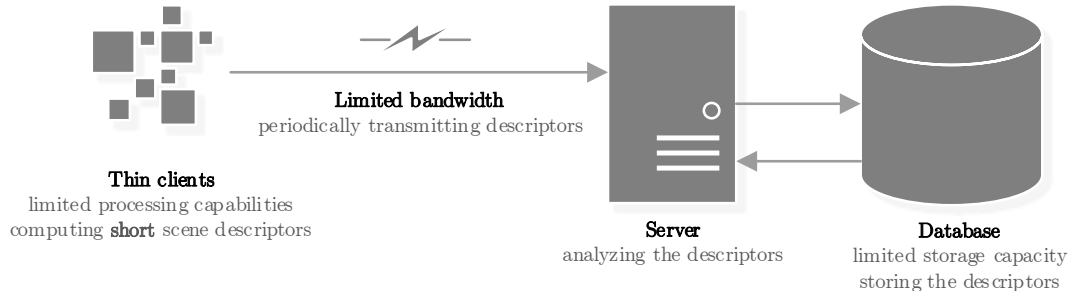


Figure 4.1: The architecture of the proposed image classification framework. A number of thin clients periodically calculate a compact image descriptor and transmit it over a limited bandwidth channel to the central server. The server then classifies the descriptors to determine the set of class labels belonging to each described image.

ture the difference between the images of various classes well, so that good classification accuracy can be achieved by the classifier used on the central server. The descriptiveness can be experimentally estimated by measuring the classification accuracy, or other classification performance measure. In the experimental section of this thesis the classifier *mean average precision* will be used as a proxy for descriptiveness, as will be discussed later. If the classifier is not able to classify images adequately, the whole classification subsystem becomes worthless. The better the classification performance, the greater the contribution to the overall quality of fleet management will be made by the visual subsystem. All other descriptor properties only influence the costs of the bandwidth, storage, and computational hardware. That being said, small compromises are acceptable, a 1% drop in classification accuracy might be acceptable trade-off for $10\times$ reduction in descriptor size.

Compactness is the property of the descriptor to be expressible using a low number of bits. The descriptor needs to be compact not only to reduce the data transmission costs, but also to limit the impact to both the client and server storage requirements. There are several approaches to achieving compactness, which can be used independently:

- Using a method that was designed with compactness in mind.
- Starting with a large descriptor, and reducing its dimensionality (e.g. PCA).
- Using an efficient quantization, or other compression scheme.

Computational simplicity is not as important as descriptiveness or compactness, but is still very desirable. The computation of the descriptor must be simple enough to run on realistic client hardware, and not to waste too much power. Expensive mobile devices are often equipped with dedicated neural processing units (NPUs). The computational power of NPUs in mobile devices often exceeds 500 GFLOPs, and for some it exceeds 3 TFLOPs [108]. Movidius Myriad 2 chip is capable of processing up to two trillion 16-bit operations per second while using only 500 mW of power [109, 110], and is available in the form of a USB stick at a retail price under 80 USD [111]. Note that proposed fleet

management visual framework does not require real-time image analysis, processing a single image every several seconds is sufficient. However, if the descriptor computation is fast and does not require expensive hardware, then a wider range of consumer mobile devices can be used, which may significantly reduce the cost of the entire system.

Note that there is no requirement for the ability to reconstruct the original image from the descriptor. This suggest that even shorter representations should be achieved than what is offered by many state-of-the-art image and video compression schemes. Many image retrieval systems offer very short image representations, but require the representations to adhere to a distance metric so that the distances between descriptors of visually similar images are low. Such a requirement is not useful in the scenario of traffic scene classification, so it is better to trade it off in favor of better image descriptiveness.

4.3.2 Descriptor training restrictions

Many descriptor algorithms have parameters that must be learned. Once learned and deployed to the clients, it would be cumbersome to update them at a later time. Ideally, once trained, the descriptor should not need to be re-trained as more data becomes available. An ideal descriptor would not have any parameters and no learning process involved, but such descriptors usually do not achieve high descriptiveness. Some descriptors require the knowledge of the target images dataset, as well as the class labels. Since the set of class labels needs to remain open, such descriptors are highly undesirable. Some descriptors might require only the image data, without class labels. Such descriptors could be useful if they generalize well across different image sets. If they do not generalize well, then they might need re-training if they are to be used on images of traffic scenes from different parts of the world (that were not part of the original training dataset). Thus the additional desirable properties of the descriptor are:

Simplicity of training. The training of the descriptor should not depend on the target class labels, so that it does not need to be updated when the labels change.

Generalization. The descriptor should provide good descriptiveness on images of traffic scenes from different geographical areas, even those that were not present in the training dataset.

4.3.3 Classifier

The classification happens on the central server, which is typically equipped with powerful hardware, and can even be a distributed system consisting of many computers. Therefore, the main requirement on the classifier is good classification accuracy. It should be able to handle several thousand classifications per second, but that is typically not an issue. The training process can be as computationally expensive as necessary. While it is still conceivable that the descriptor algorithm would need to be updated in the future, in this proposed framework the

updating of clients would not be as cumbersome as in the client-side classification setup. This is because during the client update period the server can easily keep two copies of the classifier, one for each version of the descriptor. The addition or removal of the class labels does not necessarily require a client update, and can easily be done instantaneously for all clients by simply changing the classifier.

4.3.4 Discussion

The proposed framework has two drawbacks in comparison to server-side and client-side classification approaches. First, it requires more data transmission and storage than the client-based classification approach. This could be alleviated by making the size of the descriptor be configurable and adaptive. If a client is nearing its storage capacity or data transmission quota, it could sacrifice some descriptiveness and use a more compact image representation. A trivial example of this would be to reduce the number of bits of precision for floating point numbers in the descriptor encoding. Second, the collection of new images is not as simple as in server-side classification approach, so it is harder to enlarge the training dataset if the need arises. This could also be alleviated by using a scheme in which certain clients occasionally do send an entire image, thus causing a slight increase in data transmission costs. In this scheme clients would keep the last several images for which the descriptor was sent. The server would decide if it requires the entire image to be sent, according to a series of flexible rules. For example, an out-of-distribution input detector [107] could be used to find images that are unusual, and would benefit the dataset. The images could also be requested from clients that are known to be in previously unseen geographical locations. The costs could be managed if known data transmission quotas are used, by only requesting images from devices that are not near the limit. If a client company requires a development of a new fleet management functionality that relies on recognizing a new type of traffic environment, then only their vehicles can be used to collect the images of the new type of traffic environment.

The storage requirements of the proposed architecture depend on the size of the descriptor and the average frequency of status updates. The size of the descriptor depends on the number of vector components and the number of bits required to encode each component. Storage required by several plausible descriptor encodings with respect to frequency of status updates is shown in Table 4.2. The realistic average time between status updates is between 2 and 60 seconds, while the 10 minute intervals are included for completeness. Note that 32 bits is enough to encode an IEEE 754 single precision floating point number, which should be a sufficiently precise representation for components of any descriptor. Prior research [100] has shown that some descriptors can be encoded with 4 bits per component without any loss in classification performance. The impact on the required storage is still significant when compared to plain status update packets with no image descriptors, but much less so than if entire images were to be

transmitted. Additional descriptor compression algorithms might be applicable, depending on the type of image descriptor. For example, if a descriptor is sparse, then it can be very efficiently transmitted by using a sparse encoding scheme. General-purpose compression algorithms such as Deflate [112], LZMA [113] or BZip2 [114] do not seem to be useful for compression of a single descriptor, but might be useful for the purposes of compressing a large number of descriptors during archiving and backup.

The feasibility of the proposed framework hinges on the existence of descriptor algorithms which satisfy the desirable properties and training restrictions. Several such descriptors will be presented in the next Chapter, along with all other methods which can be used to build the proposed framework.

Table 4.2: Approximate data traffic generated by 1000 tracking devices in a single year, depending on the size of transmitted descriptor and the frequency of status updates. Average size of the status packet without visual information is assumed to be 50 bytes (shown in the last row).

	average time between status updates			
image descriptor encoding	2 seconds	10 seconds	1 minute	10 minutes
2048 32-bit components	118 TB	23 TB	3 TB	403 GB
2048 16-bit components	59 TB	11 TB	1 TB	202 GB
2048 4-bit components	15 TB	3 TB	525 GB	52 GB
256 32-bit components	15 TB	3 TB	525 GB	52 GB
256 16-bit components	8 TB	1 TB	275 GB	27 GB
256 4-bit components	2 TB	522 GB	87 GB	8 GB
32 32-bit components	2 TB	522 GB	87 GB	8 GB
32 16-bit components	1 TB	334 GB	55 GB	5 GB
32 4-bit components	969 GB	193 GB	32 GB	3 GB
no descriptor (status only)	734 GB	146 GB	24 GB	2 GB

Chapter 5

Methods

This chapter details all methods required to build the proposed fleet management visual categorization framework. While Chapter 2 already gives a general overview of the used methods, this chapter focuses on specifics. It explains why the presented methods were chosen, what makes them applicable in the proposed fleet management framework, and how they are expected to perform.

The chapter starts with the presentation of several descriptor methods which satisfy all requirements outlined in Chapter 4. The chapter continues with the presentation of several vector quantization approaches, and is concluded by an overview of classification methods and performance measures.

The descriptors to be used in the proposed fleet management framework need to be descriptive, yet compact. They may not be overly computationally intensive, or require large amounts of working memory. The descriptors should generalize well to unseen traffic images, or traffic images from different areas of the world. Finally, if they have any trainable parameters, their learning should not depend heavily on the target imagery and labels. The reason for the last point is to avoid frequent client updates since the client update is an expensive and cumbersome process, as explained in Chapter 4. If the parameters depend heavily on the target data, then they need to be re-learned when new data becomes available. If instead they can be adequately estimated from a small sample of target data, then the clients may, but do not have to be updated as more data becomes available. Since the set of target labels is expected to change, and must remain open-ended, none of the proposed descriptors use target labels during their training. Some, but not all of the proposed descriptors use target imagery during training.

5.1 Descriptors based on spatial Fisher vector framework

Fisher vector [115] characterizes the samples X as a gradient vector $\bigtriangledown_{\Theta} \ln p(X|\Theta)$ where p is the probability density function of samples given the parameters Θ of a suitable generative model

of the data X. This gradient vector can be classified by a discriminative classifier. Besides being able to handle variable-length signals, Fisher vectors are also useful in image classification [28, 116].

Spatial Fisher vector (SFV) framework [28] is an extension of bag-of-words approach which uses Fisher vector representation to encode both the appearance and spatial layout of image patches with respect to visual words. It was an image descriptor method which enabled state-of-the-art image classification results until it was overtaken by deep learning approaches. Fisher vectors are still used in recent works [63, 117, 118] in combination with deep learning frameworks.

Let us now overview the dimensionality of the spatial Fisher vector descriptors. In the plain bag-of-words approach [18], each D-dimensional image patch descriptor is hard-assigned to its nearest visual words (cluster centroid), which results in the K-dimensional histogram, where K is the number of visual words. In the Fisher vector extension [116], the visual words are represented as a Gaussian mixture model (GMM), and each patch is soft-assigned to its nearby visual words (Gaussians). Additionally, the first and the second order moments of the patches assigned to each visual word are stored as well, meaning that the dimension of the resulting vector is $K(1+2D)$. This extension by itself encodes the appearances of the patch descriptors, but not their locations. Using a spatial encoding scheme such as *spatial pyramid matching* (SPM) [27] with C cells would result in a vector of length $CK(1+2D)$. The spatial Fisher vector approach does not use the SPM, but instead uses the Fisher kernel principle to encode the spatial information as well as appearance information. The location (in pixel space, not in local feature space) of image regions assigned to visual words is modeled by a GMM with C components. The location, first and second moment are stored, meaning that an additional $CK(1+2d)$ vector components are used to store spatial information, where $d=2$ for a 2-dimensional signal (an image). Thus the total SFV image descriptor is of length $K(1+2D)+KC(1+2d)$, which is significantly shorter than $CK(1+2D)$ if $d \ll D$. The dimensionality thus depends on the number of appearance and spatial components, and the dimensionality of the local patch descriptor. Compact descriptors can be obtained by carefully setting the hyper-parameters, and/or by applying a dimensionality reduction technique (e.g. PCA).

Note that the GMM models can be learned in a supervised way [116], which is not done here, as the requirement of the framework is to avoid any knowledge of the target class labels.

Two descriptors based on the SFV will now be presented. One is handcrafted, while the other uses local features learned by a deep learning framework.

5.1.1 SIFT/SFV + GIST descriptor

This descriptor is handcrafted. It is a concatenation of two descriptors: the spatial Fisher vectors (SFV) [28] and the GIST scene descriptor [82, 83]. The dense SIFT algorithm [19, 20] was

used as a source of local features for the spatial Fisher vectors. Knowledge of target imagery is required for the SFV descriptor, as it needs to learn a vocabulary of visual words. The GIST scene descriptor is completely handcrafted, and has no training requirements. It was designed for scene categorization, and by itself does not perform well for general purpose classification tasks [82, 83]. However, appending the GIST descriptor to the SFV descriptor has shown to be beneficial to the classification performance on traffic scenes, especially if the descriptors sizes are very small. The sizes of both the SFV and GIST descriptors can be regulated by selecting the appropriate hyper-parameters, such as number of spatial components or the dimensionality of local features for the SFV descriptor, or the number of scales or grid size for the GIST descriptor. If we consider a short M-dimensional SFV descriptor (e.g. 64 components, regulated by SFV hyper-parameters K and C), then it is more beneficial to append a 64-dimensional GIST descriptor than to modify the SFV hyper-parameters to produce a $(M+64)$-dimensional SFV descriptor. Another way to obtain short image representations is to produce a large descriptor first, and then reduce the dimensionality by using Principal Component Analysis (PCA). This descriptor can be made to be compact and is simple to compute. Its parameters are learned by examining the target dataset imagery. It does not require knowledge of target class labels. If the size regulation is achieved by using the PCA instead of hyper-parameter configuration, then a sample of descriptors instances computed on the target dataset is needed in order to learn the principal components.

5.1.2 VGG/SFV descriptor

This descriptor is very similar to the previously introduced one. It also relies on the spatial Fisher vector framework (SFV), but it is not combined with the GIST descriptor. Instead of SIFT features (which are hand-crafted), it uses the responses of a convolutional neural network (which were obtained by end-to-end learning) as local features. More precisely, it uses the responses of `conv5_4` layer of VGG-19 [43] convolutional neural network, trained on the ImageNet1000 dataset and not fine-tuned to the target dataset. Not fine-tuning was a deliberate choice, which aims to improve the simplicity of training and generalization properties of this descriptor. If the convolutional features are not learned on the particular set of traffic images, then they are equally likely to generalize well to traffic images from different geographical areas. The architecture of the relevant section of the network is shown in Table 5.1. The input sizes listed in the table assume the resolution of input images is 640×480, which is the resolution of images of the dataset used in this research. A spatial padding of 1 pixel is used for the 3×3 convolutional layers so that the spatial resolution remains unchanged after the convolution. The only preprocessing of the input is subtraction of the mean RGB value (calculated on ImageNet) from each pixel. The used part of the network has around 20 million parameters, and produces 1200 features of size 512, which are used as a local feature input to the spatial Fisher vector

framework. Approximately 120 billion multiply-add operations are required to feed an image of resolution 640×480 through the used part of the network. The size of this descriptor can be regulated by selecting appropriate SFV hyper-parameter values, or by using a PCA to compress a longer feature vector. As the SFV framework learns the distribution of the local features on the target dataset, this descriptor also requires access to the target imagery.

5.2 Descriptors based on supervised transfer learning

Recent convolutional architectures [47, 52, 54] use a simple global pooling in favor of fully connected layers, which enables them to outperform the older approaches (such as VGG) despite having fewer parameters. Their pooled features have a relatively low dimensionality, which means they can be directly used as compact image descriptors. They are simple to compute. If publicly available parameterizations are used (learned on the ImageNet), and no fine-tuning is performed, then they require neither knowledge of target class labels nor target dataset imagery. If they perform well without knowledge of the target dataset, then they are expected to generalize well, and minimum future updates are expected. In the pursuit of good generalization and training simplification, the Fisher vector aggregations were not used with these features, even though it might offer improved performance.

Three specific architectures have been selected for experimental evaluation in this research: ResNet [47], DenseNet [52] and MobileNetV2 [54]. All three architectures have already been discussed in Chapter 2. We now go into more detail, with more focus on the implementational considerations. We specify the exact models that were used in this research, and examine them with focus on the dimensionality of obtained features, the number of parameters and computational complexity.

5.2.1 ResNet descriptor

The ResNet-50 network architecture is used in the experimental section of this thesis. It has 50 layers, and the average-pooled responses of the layer `conv5_9` are used as the image descriptor, which is 2048-dimensional. The architecture of the network is shown in Table 5.2. The fully connected and softmax layers are not included. A spatial padding is used as necessary for the convolutional layers so that the spatial resolution remains unchanged after the convolution. The only preprocessing of the input is subtraction of the mean RGB value (calculated on ImageNet) from each pixel. Batch normalization is used after each convolution (except last), before activation. Note that we do not use pre-activation residual blocks presented in [48], as they did not lead to noticeable improvements of classification accuracy on our dataset.

The architecture consists of 16 residual building blocks. In each block, immediately before

Table 5.1: The used section of the VGG-19 architecture, starting from the input and ending with the extracted features.

input size	layer type	parameters
$480 \times 640 \times 3$	3×3 conv. ReLU, stride 1, 64 channels	1728
$480 \times 640 \times 64$	3×3 conv. ReLU, stride 1, 64 channels	36864
$480 \times 640 \times 64$	2×2 max-pool, stride 2	0
$240 \times 320 \times 64$	3×3 conv. ReLU, stride 1, 128 channels	73728
$240 \times 320 \times 128$	3×3 conv. ReLU, stride 1, 128 channels	147456
$240 \times 320 \times 128$	2×2 max-pool, stride 2	0
$120 \times 160 \times 128$	3×3 conv. ReLU, stride 1, 256 channels	294912
$120 \times 160 \times 256$	3×3 conv. ReLU, stride 1, 256 channels	589824
$120 \times 160 \times 256$	3×3 conv. ReLU, stride 1, 256 channels	589824
$120 \times 160 \times 256$	3×3 conv. ReLU, stride 1, 256 channels	589824
$120 \times 160 \times 256$	2×2 max-pool, stride 2	0
$60 \times 80 \times 256$	3×3 conv. ReLU, stride 1, 512 channels	1179648
$60 \times 80 \times 512$	3×3 conv. ReLU, stride 1, 512 channels	2359296
$60 \times 80 \times 512$	3×3 conv. ReLU, stride 1, 512 channels	2359296
$60 \times 80 \times 512$	3×3 conv. ReLU, stride 1, 512 channels	2359296
$60 \times 80 \times 512$	2×2 max-pool, stride 2	0
$30 \times 40 \times 512$	3×3 conv. ReLU, stride 1, 512 channels	2359296
$30 \times 40 \times 512$	3×3 conv. ReLU, stride 1, 512 channels	2359296
$30 \times 40 \times 512$	3×3 conv. ReLU, stride 1, 512 channels	2359296
$30 \times 40 \times 512$	3×3 conv. ReLU, stride 1, 512 channels	2359296

Table 5.2: The relevant part of ResNet-50 architecture, starting from the input and ending with the extracted features. Residual blocks are shown in brackets. The factor beside the brackets indicates how many times a building block is repeated (stacked). Batch normalization is used after each convolution, before activation. Layers conv3_1, conv4_1 and conv5_1 perform downsampling with a stride of 2, while other layers in residual blocks have a stride of 1.

layer name	layer type	output size
conv1	7×7 conv. ReLU, stride 2, 64 channels	320×240
conv2_1	3×3 max-pool, stride 2	160×120
conv2_x	$\begin{bmatrix} 1 \times 1 \text{ conv. ReLU, } 64 \text{ channels} \\ 3 \times 3 \text{ conv. ReLU, } 64 \text{ channels} \\ 1 \times 1 \text{ conv. ReLU, } 256 \text{ channels} \end{bmatrix} \times 3$	160×120
conv3_x	$\begin{bmatrix} 1 \times 1 \text{ conv. ReLU, } 128 \text{ channels} \\ 3 \times 3 \text{ conv. ReLU, } 128 \text{ channels} \\ 1 \times 1 \text{ conv. ReLU, } 512 \text{ channels} \end{bmatrix} \times 4$	80×60
conv4_x	$\begin{bmatrix} 1 \times 1 \text{ conv. ReLU, } 256 \text{ channels} \\ 3 \times 3 \text{ conv. ReLU, } 256 \text{ channels} \\ 1 \times 1 \text{ conv. ReLU, } 1024 \text{ channels} \end{bmatrix} \times 6$	40×30
conv5_x	$\begin{bmatrix} 1 \times 1 \text{ conv. ReLU, } 512 \text{ channels} \\ 3 \times 3 \text{ conv. ReLU, } 512 \text{ channels} \\ 1 \times 1 \text{ conv. ReLU, } 2048 \text{ channels} \end{bmatrix} \times 3$	20×15
output	20×15 global average pool	1×1

the last nonlinearity, an element-wise summation of linearly projected block inputs with the output of the last weight layer is performed. Inputs are linearly projected using 1×1 convolutions to match the number of feature maps. A stride of 2 is used when the size of output feature maps is smaller than the size of input feature maps. If sizes and number of feature maps match, then no linear projection is necessary, and the inputs are elementwise summed directly. The network has around 30 million parameters. Approximately 25 billion multiply-add operations are required to feed a 640×480 image through this network.

5.2.2 DenseNet descriptor

The DenseNet-BC-121 network architecture is used in the experimental section of this thesis. It has 121 layers, and the average-pooled responses of the last convolutional layer are used as the image descriptor, which is 1024-dimensional. The architecture of the network is shown in Table 5.3. The fully connected and softmax layers are not included. A spatial padding is used as

necessary for the convolutional layers so that the spatial resolution remains unchanged after the convolution. The only preprocessing of the input is subtraction of the mean RGB value from each pixel.

The architecture consists of four dense building blocks, connected by three transition layers. Before entering the first dense block, a convolution with 64 output channels is performed on the input images. Each layer in a dense block performs a batch normalization, followed by a ReLU activation, followed by a convolution. The dense block consists of L pairs of layers. The l-th pair receives the feature maps of all the preceding $l-1$ pairs in the block as its input. The feature maps produced by the l-th pair are then included into input of all successive $L-l$ pairs in the block. Growth rate is $k = 32$, meaning that each pair of layers produces k feature maps, so that l-th pair has $k_0 + k(l-1)$ input feature maps, where k_0 is the number of feature maps in the block input. The first layer in a pair serves as a bottleneck that reduces the number of feature maps fed into the second layer to exactly $4k$. The transition layers perform a 1×1 convolution to halve the number of feature maps, followed by average pooling to reduce the feature map size. The network has around 6 million parameters. Approximately 15 billion multiply-add operations are required to feed a 640×480 image through this network.

5.2.3 MobileNetV2

The main feature of MobileNetV2 model is its reduced computational cost in comparison to more powerful models such as ResNet and DenseNet. Authors provide a series of network architectures pre-trained on the ImageNet dataset. The basic architecture is the same, the difference is in the hyper-parameter α, called a *width multiplier*. It changes the base network so that each bottleneck block produces αC output channels instead of C channels. We use the base network, i.e. $\alpha = 1$. The relevant part of MobileNetV2 architecture is shown in Table 5.4. The table assumes input images of resolution 640×480. The first layer is a 3×3 convolution, after which follow 17 bottleneck residual blocks. A single bottleneck block with an expansion factor k is composed of a 1×1 ReLU6 convolution that transforms an input tensor $W \times H \times C$ to $W \times H \times t \cdot C$, followed by a 3×3 depthwise ReLU6 convolution that does not change the number of channels, followed by a 1×1 linear convolution that changes the number of channels to a fixed value (as shown in the fourth column of Table 5.4). The input of each block is element-wise summed to the output. Batch normalization is used in all layers except the last layer of the block. The stride of depthwise convolutions is occasionally set to 2 to halve the spatial resolution (as indicated in the sixth column of Table 5.4), otherwise is it set to one. Note that the output of the discussed network section is $15 \times 20 \times 320$. We add a global average pooling layer to obtain a compact descriptor of length 320. Total number of parameters is around 1.2 million. Feeding a 640×480 image through the network requires approximately 2.3 billion multiply-add operations, which can be computed in under a minute even without dedicated

Table 5.3: The relevant part of DenseNet-BC-121 architecture, starting from the input and ending with the extracted features. Dense blocks are constructed by stacking the layers inside brackets the indicated number of times. Growth rate is $k = 32$.

layer name	layer type	output size
convolution	7×7 BN-ReLU-conv., stride 2	320×240
pooling	3×3 max-pool, stride 2	160×120
dense block 1	$\begin{bmatrix} 1 \times 1 \text{ BN-ReLU-conv.} \\ 3 \times 3 \text{ BN-ReLU-conv.} \end{bmatrix} \times 6$	160×120
transition layer 1	1×1 BN-ReLU-conv.	160×120
	2×2 average pool, stride 2	80×60
dense block 2	$\begin{bmatrix} 1 \times 1 \text{ BN-ReLU-conv.} \\ 3 \times 3 \text{ BN-ReLU-conv.} \end{bmatrix} \times 12$	80×60
transition layer 2	1×1 BN-ReLU-conv.	80×60
	2×2 average pool, stride 2	40×30
dense block 3	$\begin{bmatrix} 1 \times 1 \text{ BN-ReLU-conv.} \\ 3 \times 3 \text{ BN-ReLU-conv.} \end{bmatrix} \times 24$	40×30
transition layer 3	1×1 BN-ReLU-conv.	40×30
	2×2 average pool, stride 2	20×15
dense block 4	$\begin{bmatrix} 1 \times 1 \text{ conv. BN-ReLU-conv.} \\ 3 \times 3 \text{ conv. BN-ReLU-conv.} \end{bmatrix} \times 16$	20×15
output	20×15 global average pool	1×1

Table 5.4: Relevant section of MobileNetV2 network.

input size	block type	expansion factor	output channels	blocks in sequence	stride of first block
$480 \times 640 \times 3$	conv2d		32	1	2
$240 \times 320 \times 32$	bottleneck	1	16	1	1
$240 \times 320 \times 16$	bottleneck	6	24	2	2
$120 \times 160 \times 24$	bottleneck	6	32	3	2
$60 \times 80 \times 32$	bottleneck	6	64	4	2
$30 \times 40 \times 64$	bottleneck	6	96	3	1
$30 \times 40 \times 96$	bottleneck	6	160	3	2
$15 \times 20 \times 160$	bottleneck	6	320	1	1

neural hardware. Depending on the CPU, it could be done in several seconds. For slower CPUs the reduction of input image resolution is required. For example, only 576 million multiply-add operations are required to feed an input image of resolution 320×240.

5.3 Descriptors based on unsupervised learning

Note that the VGG/SFV, DenseNet and ResNet descriptors are trained in a supervised way, using ImageNet images and class labels. Some ImageNet class labels are somewhat traffic-related and might overlap with fleet management class labels (e.g. *traffic light*, *fire truck*, *tractor*). It is possible that fleet management classes that do overlap or partially overlap with ImageNet classes might be easier to classify than other fleet management classes. Conversely, it is possible that at some point in the future a new class label would be introduced into the fleet management visual categorization, and that it would not be easy to classify due to its large difference in appearance from all other ImageNet classes. One way to ensure the balance between the classes is to use completely unsupervised training methods. Unsupervised methods do not rely on labeled data during training, they learn from the data samples only.

5.3.1 DCGAN descriptor

The state-of-the-art unsupervised convolutional models are Generative Adversarial Networks (GAN) [78], and their extensions Deep Convolutional GAN (DCGAN) [80] and Wasserstein GAN (WGAN) [81]. They all consist of a pair of adversarial networks, the discriminator and the generator, which are involved in a min-max game and trained jointly on a dataset of images.

The generator tries to generate artificial images that match the distribution of images in the dataset, while the discriminator tries to differentiate between the artificial images and images that are actual members of the dataset. It was shown that the discriminator network learns image representation hierarchies which can be used for image classification [80]. By max-pooling the convolutional features from discriminator layers over a grid and concatenating the responses it is possible to obtain an image representation that can be classifier by an SVM classifier. This approach can even be used to successfully classify images from a completely different dataset [80], which means the obtained descriptor generalizes well and achieves good knowledge transfer. The descriptor presented in the original paper [80] is relatively long (28672 components), but the length can be reduced by using a smaller pooling grid.

5.4 Generalization potential

In total, this thesis examines six descriptor approaches that might be suitable for the fleet management target framework. In this thesis they are referred to as *SIFT/SFV+GIST*, *VGG/SFV*, *ResNet*, *DenseNet*, *MobileNet* and *DCGAN* descriptors. They have been named after their core components, in the interest of brevity.

The proposed descriptors differ in how much knowledge of the target dataset they require in order to successfully learn their parameters. Some of them can learn the parameters on an unrelated dataset and still be successfully applied via transfer learning. The ImageNet Large Scale Visual Recognition Competition (ILSVRC) [40] dataset was used as an unrelated dataset of choice, as it has a large number of images with wide variety of class labels, so it is likely many useful image representations can be learned on this dataset. Finally, the learning process can be supervised or unsupervised. The supervised approaches usually achieve better performance at the risk of not generalizing well to unseen or unrelated classes. A complete overview of the proposed descriptors with respect to data used to train their parameters is shown in Table 5.5. Note that SIFT/SFV+GIST is the only descriptor that does not use the ImageNet data in any way. The VGG/SFV descriptor is the only one that uses the knowledge of both the ImageNet dataset as well as target dataset, as the VGG network was trained on the ImageNet1000, while the spatial Fisher vector framework was trained on the target dataset imagery. No descriptors use the target labels during training, and the DCGAN descriptor does not use any labels whatsoever.

5.5 Dimensionality reduction

In this research, *Principal component analysis* (PCA) [90, 91] is used to achieve dimensionality reduction of image feature vectors (descriptors). PCA is an orthogonal linear transformation that maps the input data to a new coordinate system such that the variance of the projection of

Table 5.5: An overview of the proposed descriptors with respect to the extent of received training. More training leads to better results but increases the risk of overfitting.

proposed descriptor	SIFT/SFV +GIST	VGG/SFV	DenseNet, ResNet and MobileNet	DCGAN
has learnable parameters	yes	yes	yes	yes
trained using target imagery	yes	yes	no	no
trained using target labels	no	no	no	no
trained using ImageNet imagery	no	yes	yes	yes
trained using ImageNet labels	no	yes	yes	no

the data to the first coordinate axis is the greatest possible, the projection of the data to the next coordinate axis is the next greatest possible, and so on. The coordinate axes are called *principal components*.

Let $\mathbf{X}$ be a $N \times D$ matrix of input data, $N \geq D$, organized so that the rows represent the N D-dimensional input data samples $\mathbf{x}_i$, and columns represent sample features. Let the mean of all columns of $\mathbf{X}$ be zero. Then, the first principal component $\mathbf{w}_1$ is a unit vector such that the variance of sample projections in that direction is maximized:

$$\mathbf{w}_1 = \underset{||\mathbf{w}_1||=1}{\arg\max} \left\{ \sum_{i=1}^{N} (\mathbf{x}_i \cdot \mathbf{x}_1)^2 \right\} = \underset{||\mathbf{w}_1||=1}{\arg\max} \left\{ \mathbf{w}_1^T \mathbf{X}^T \mathbf{X} \mathbf{w}_1 \right\} \tag{5.1}$$

Since $\mathbf{w}_1$ is a unit vector, this can be expressed as

$$\mathbf{w}_1 = \arg\max \left\{ \frac{\mathbf{w}_1^T \mathbf{X}^T \mathbf{X} \mathbf{w}_1}{\mathbf{w}_1^T \mathbf{w}_1} \right\} = \arg\max \left\{ R(\mathbf{X}^T \mathbf{X}, \mathbf{w}_1) \right\} \tag{5.2}$$

where $R(\mathbf{M}, \mathbf{w})$ is the Rayleigh quotient. For a positive semidefinite matrix $\mathbf{X}^T\mathbf{X}$, Rayleigh quotient achieves maximum value when $\mathbf{w}_1$ is equal to the eigenvector of $\mathbf{X}^T\mathbf{X}$ with the largest eigenvalue. By subtracting the found principal components from $\mathbf{X}$ and repeating the same analysis it follows that all other principal components are eigenvectors of $\mathbf{X}^T\mathbf{X}$, and that their order is in descending order of their corresponding eigenvalues.

After the principal components have been found, the sample vectors are projected using only first d principal components, thus reducing the dimensionality from D to d.

5.6 Quantization

All descriptor methods described above generate image representations that are vectors of real numbers, $\mathbf{x} \in \mathbb{R}^D$, where D is the dimensionality of a vector. Since computers are machines

with finite memory, they are not able to represent arbitrary real numbers with perfect accuracy. Instead, computer programs typically use the IEEE 754 floating point format [119] to approximate real numbers. The two commonly used IEEE 754 floating point formats are *binary32* and *binary64*. The *binary32* format, also known as *single precision* encodes numbers in 32 bits and offers precision of 24 binary digits (or 7.22 decimal digits). The *binary64* format, also known as *double precision* encodes numbers in 64 bits and offers precision of 53 binary digits (or 15.95 decimal digits). Since most modern computational hardware provides optimized calculation of commonly used arithmetic operations involving these two formats, they are widely adopted and used in many machine learning systems.

In our proposed fleet management framework, however, minimizing the size of the transmitted and stored descriptors is a priority. Encoding the numbers using 32 or 64 bits should be avoided unless it is absolutely necessary. Instead, numbers should be encoded using the minimum required precision for the classification to achieve adequate accuracy. The process of mapping values from a large set (possibly even continuous set) to a smaller set (which must be discrete) is called *quantization*. The quantization mapping is such that a range of input values is approximated by a single value (called a quantum value). An image descriptor should be quantized before it is stored or transmitted. Several quantization approaches shall now be discussed.

5.6.1 Component-independent quantization

Component-independent quantization (CQ) is an approach proposed by the author of this thesis in prior research [100]. It quantizes each component of a D-dimensional descriptor $\mathbf{x} \in \mathbb{R}^D$ independently. Value of i-th component x_i is first clipped to the interval $[l_i, r_i]$:

$$x_i^c = \min(r_i, \max(l_i, x_i)) \tag{5.3}$$

The interval $[l_i, r_i]$ is split into $d_i = 2^{q_i}$ bins (intervals) of equal width, and all the values in the same interval are approximated by the interval midpoint. The value x_i^c is encoded as the zero-indexed ordinal number of the bin it falls into, c_i, thus using only q_i bits. More precisely:

$$c_i(x_i^c) = \begin{cases} \left\lfloor d_i \frac{x_i^c - l_i}{r_i - l_i} \right\rfloor & \text{if } x_i^c < r_i \\ d_i - 1 & \text{if } x_i^c = r_i \end{cases} \tag{5.4}$$

The quantized value x_i^q of a component x_i can be calculated from its code c_i as follows:

$$x_i^q = l_i + (c_i + 0.5)\frac{r_i - l_i}{d_i} \tag{5.5}$$

Finding the appropriate clipping intervals $[l_i, r_i]$, and adequate number of bits per component q_i is the only prerequisite of using this approach. A reasonable approach is using the clipping intervals $[\mu_i - k\sigma_i, \mu_i + k\sigma_i]$, where μ_i and σ_i are the estimated mean and variance of the component i, and k is a parameter that scales the width of the interval. Another reasonable assumption is that the same number of bits should be used to encode all components, $q_i = \text{bpc}$, where bpc stands for *bits for component*. Under these assumptions the equations 5.3, 5.4 and 5.5 assume the form:

$$x_i^c = \min(\mu_i + k\sigma_i, \max(\mu_i - k\sigma_i, x_i)) \tag{5.6}$$

$$c_i(x_i^c) = \begin{cases} \left\lfloor \frac{x_i^c - \mu_i + k\sigma_i}{2k\sigma_i} 2^{\text{bpc}} \right\rfloor & \text{if } x_i^c < \mu_i + k\sigma_i \\ 2^{\text{bpc}} - 1 & \text{if } x_i^c = \mu_i + k\sigma_i \end{cases} \tag{5.7}$$

$$x_i^q = \mu_i - k\sigma_i + (c_i + 0.5)\frac{2k\sigma_i}{2^{\text{bpc}}} \tag{5.8}$$

Since μ_i and σ_i can be estimated by using a small sample of values, only the parameters k and bpc need to be optimized experimentally.

5.6.2 Clustering quantization

A clustering scheme can be used to quantize a set of vectors. In order to encode a vector using n bits, apply a clustering algorithm (e.g. k-means [120]) to find 2^n clusters. Each vector can then be approximated by its nearest cluster centroid. The cluster centroids are used as a codebook, and each centroid is assigned a zero-based index. The code of a single vector is simply the index of its nearest cluster centroid. The code size is thus n bits, as centroid indexes are from the set $\{0, \ldots, 2^n - 1\}$. This method can be used even if only a sample of vectors from the set is available, as long as the sample adequately represents the set distribution. A major drawback of this approach is that it can only be used to quantize vectors into a very small number of bits, as the number of clusters rises exponentially with the number of bits.

5.6.3 Product quantization

The Product quantization (PQ) [92, 93] is an extension of the clustering quantization approach. It solves the exponential rise of the codebook size by subdividing the problem space into smaller subspaces. It requires a representative sample of the vector set in order to build a codebook, same as clustering quantization approach. The D-dimensional vectors $\mathbf{x} \in \mathbb{R}^D$ are regarded as concatenations of M subvectors $\mathbf{x} = [\mathbf{x}^1, \ldots, \mathbf{x}^M]$. Each subvector sample is quantized separately into n_i bits, using the clustering quantization approach. This is feasible if all n_i are small enough. The quantization of subvector samples results in M codebooks $\{C^i\}, |C^i| = 2^{n_i}, i = 1, \ldots, M$.

The codebooks are used to produce subvector codes $\mathbf{c}^1, \ldots, \mathbf{c}^M, \mathbf{c}^i \in C^i$. The produced codes are then concatenated to obtain the code of the entire vector $\mathbf{c}(\mathbf{x}) = [\mathbf{c}^1, \ldots, \mathbf{c}^M]$. Note that since $\mathbf{c} \in C = C^1 \times \cdots \times C^M$, the Cartesian product of the subvector codebooks C^i can be considered the codebook for the full vector. Although its size rises exponentially with M (assuming $n_1 = \cdots = n_M$ and $|x^1| = \cdots = |x^M|$), it only requires M times more memory and computation time than a single subvector codebook C_i.

5.7 Classification

A number of works have demonstrated successful image categorization by using a support vector machine (SVM) [29] classifier. Preliminary research for this thesis [99, 100, 121] showed that the SVM classifier outperformed the random forest [122] classifier in almost all cases.

5.7.1 The support vector machine classifier

The support vector machine (SVM) [29] is a binary classifier. Given a training dataset $\{(\mathbf{x}_i, y_i)\}$ of N linearly separable samples $\mathbf{x}_i$ with labels $y_i \in \{-1, +1\}$, it finds a maximum-margin hyperplane $\mathbf{w} \cdot \mathbf{x} - b = 0$ that divides the samples with label -1 from the samples with label $+1$. The maximum-margin means that the distance of the hyperplane from the nearest sample is maximized. This is known as a *hard-margin* SVM.

If the samples are not completely linearly separable (e.g. if there are some outliers), then a *soft-margin* variant is used instead. A *hinge loss* function is defined as:

$$l(\mathbf{x}) = \max(0, 1 - y_i(\mathbf{w} \cdot \mathbf{x}_i - b) \tag{5.9}$$

The hinge loss function is equal to zero for samples x_i which are correctly classified. For incorrectly classified samples, it is proportional to the distance from the dividing hyperplane. The soft-margin SVM minimizes the following cost function:

$$\frac{1}{N} \sum_{i=1}^{N} l(\mathbf{x}_i) + C||\mathbf{w}||^2 \tag{5.10}$$

The C is a parameter that determines the tradeoff between maximizing the size of the margin, and maximizing the number of correctly classified samples.

Kernel function

In some cases the samples are not linearly separable at all. They might become linearly separable if they are mapped into a higher-dimensional space by a carefully selected function $\phi(\mathbf{x})$.

The explicit mapping into a higher dimensional space is not strictly necessary, if a *kernel function k* can be found such that $k(\mathbf{x}_1, \mathbf{x}_2) = \phi(\mathbf{x}_1) \cdot \phi(\mathbf{x}_2)$. In this research a *radial basis function* (RBF) is used as an SVM kernel function, defined as:

$$k(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma ||\mathbf{x}_i - \mathbf{x}_k||^2}, \gamma > 0 \tag{5.11}$$

Label weighting

The hinge loss function 5.9 assigns equal importance to both the positive and negative samples. As a result, the number of incorrectly classified positive and negative samples will likely be roughly equal. If the number of positive and negative samples is not balanced, then the relative ratios of positive and negative misclassified samples are also not balanced. This means that the label with larger number of samples is more likely to be correctly classified, which can be undesirable. To correct this, a label-dependent weighting factor can be introduced to the hinge loss function, to make the misclassified samples with a certain label contribute more to the overall loss, thus balancing the ratios of misclassified samples. A reasonable choice is using a factor inversely proportional to the frequency of the label occurrences in the dataset.

5.7.2 Multi-label classification

The proposed classification system shall be multi-label, as it is expected that traffic scene images shall often simultaneously depict multiple categories of interest to fleet management. For example, an image might depict a scene of dense traffic on a highway, and both *highway* and *dense traffic* could be classes of interest in fleet management.

There are two common ways to implement a multi-label classifier using only binary classifiers: *label powerset* method and *binary relevance* method.

Label powerset is a transformation method that trains one binary classifier for each combination of labels encountered in the dataset. One problem with this approach is that if the labels are independent, then the number of their combinations can rise exponentially, with maximum number of combinations of N labels being 2^N. Another problem is that some of the combinations might occur very rarely in the dataset, which would hinder training. The benefit of this approach is that it is able to learn dependence between labels.

Binary relevance is an approach in which a single binary classifier is trained for each label. The classifiers are trained independently, which means that this approach cannot learn the dependence between labels. The benefit of this approach is its simplicity and low computational cost.

Since the set of class labels needs to remain open, we use the binary relevance method. This approach is very flexible, as removal and addition of class labels can be done in a way that does

not interfere with other class labels. It is computationally feasible as long as the number of classes is not very large. A different approach might be required if the number of class labels exceeds several hundreds, but that scenario is not expected in this application.

5.8 Classification performance measures

Several classification performance measures will now be discussed. Before continuing, some basic terms must be defined. Consider a set of labeled samples $\{(\mathbf{x}_i, y_i)\}$, where $\mathbf{x}_i \in X$ are the samples, and $y_i \in \{\text{false}, \text{true}\}$ are the labels indicating whether a sample belongs to a certain class or not. Also consider a binary classifier $f : X \to \{\text{false}, \text{true}\}$. A classifier output for a single sample $f(\mathbf{x}_i)$ is called a *prediction*. The prediction *true* is also called a *positive*, while the prediction *false* is also called a *negative*. If the prediction is equal to the sample label, then it is called a *true positive/negative*, otherwise it is call a *false positive/negative*. This is overviewed in Table 5.6.

5.8.1 Simple measures

Accuracy is defined as the proportion of correctly classified samples among all samples:

$$\text{accuracy} = \frac{|\{\text{true positives}\}| + |\{\text{true negatives}\}|}{|X|} \tag{5.12}$$

which is equal to 1 if all samples are correctly classified, or 0 if none of the samples are correctly classified. It is a simple measure of classification performance that reasonably well describes the quality of the classifier. Other measures can be used to better characterize the classifier.

Precision is a measure of classification performance defined as:

$$\text{precision} = \frac{|\{\text{true positives}\}|}{|\{\text{true positives}\}| + |\{\text{false positives}\}|} \tag{5.13}$$

Note that the sum in the denominator is equal to the total number of positives. The *precision* measures the proportion of true positives among all predicted positives.

Table 5.6: Terminology of classifier predictions, with regard to their value and correctness.

	label	
prediction	true	false
true	true positive	false positive
false	false negative	true negative

Recall is a measure of classification performance defined as:

$$\text{recall} = \frac{|\{\text{true positives}\}|}{|\{\text{true positives}\}| + |\{\text{false negatives}\}|} \tag{5.14}$$

Note that the sum in the denominator is equal to the total number of samples labeled *true*. The *recall* measures the proportion of correctly classified samples labeled *true*. For this reason it is also called *true positive rate* (TPR). The *false positive rate* (FPR) is defined as:

$$\text{FPR} = \frac{|\{\text{false positives}\}|}{|\{\text{false positives}\}| + |\{\text{true negatives}\}|} \tag{5.15}$$

Note that neither precision nor recall alone are adequate classification measures. It is possible for a useless classifier to achieve very high recall, or very high precision. For example, a constant classifier $f(\mathbf{x}) = \text{true}$ achieves 100% recall. A classifier which only outputs one positive prediction, and happens to guess correctly would have a precision of 100%. Precision and recall are often expressed together to describe a classifier performance. A good classifier must have both recall and precision as close to 100% as possible. A simple way to encode both measures into a single number would be to use a F_1 score, defined as a harmonic mean of precision and recall:

$$F_1 = 2\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{5.16}$$

If precision and recall are not equal, then the value of F_1 score is closer to the $\min(\text{precision}, \text{recall})$ than to $\max(\text{precision}, \text{recall})$.

5.8.2 Receiver operating characteristic curve

Some binary classifiers (such as SVM) do not map the samples directly to the predictions $\{\text{true}, \text{false}\}$. Instead, they first calculate a score: $s : X \to \mathbb{R}$, and then use a selected threshold t to determine the label:

$$f(\mathbf{x}) = \begin{cases} true & \text{if } s(\mathbf{x}) \geq t \\ false & \text{if } s(\mathbf{x}) < t \end{cases} \tag{5.17}$$

Not all classification applications consider the true positive rate, false negative rate, and other measures to be equally important. For example, a medical test for a rare disease, or a forensic test should both have a very low probability of a false positive. By varying the threshold t, it is possible to regulate the trade-off between the number of false positives and false negatives. A *receiver operating characteristic curve* (ROC curve) is created by plotting the true positive rate against the false positive rate at various threshold values.

5.8.3 Average precision

An alternative to ROC curve is a precision-recall curve, which is the plot of precision against the recall at various threshold values. Average precision (AP) is the area under this curve:

$$\mathrm{AP} = \int_0^1 p(r)dr \tag{5.18}$$

were $p(r)$ is the precision expressed as a function of recall r. In practice this is calculated differently: first, the $N = |X|$ samples are sorted into a sequence in decreasing order of their scores $s(\mathbf{x})$. Then the average precision is calculated as:

$$\mathrm{AP} = \sum_{k=1}^{N} P(k)(R(k) - R(k-1)) \tag{5.19}$$

where $P(i)$ and $R(i)$ are the precision and recall (respectively) achieved for threshold set to the value of the score of i-th sample in ordered sequence if $i > 0$, and $R(0) = 0$.

It is possible for two classifiers to have the same accuracy (for the same threshold), but different average precision. The classifiers for which the scores of the incorrectly classified samples are relatively closer to the threshold (in relation to other samples) will measure the higher AP value. In this regard the AP is superior to accuracy measure. When dealing with binary classifiers on imbalanced datasets (which is the case in this thesis), the precision-recall curve is more informative than the ROC curve [123, 124].

The *mean average precision* (mAP) is the mean of AP values for all classes in a dataset. The AP and mAP are the classification performance measures most often used in this thesis. Since a great number of classifiers will be examined in the experimental section, it would be unpractical to include the ROC or precision-recall curves for all of them. It is much more convenient to reduce the estimation of classification quality to a single number, and AP values are adequate in this regard.

Chapter 6

Fleet Management Dataset

A novel dataset was compiled in order to experimentally evaluate the proposed methods. It is called The Fleet Management Dataset, or FM dataset for short. It grew iteratively over the years of research, and is currently in its third iteration, called the FM3 dataset[125]. The dataset aims to accurately represent the set of traffic scenes that might be encountered in a fleet management system.

The FM3 dataset contains 11448 labeled images of traffic scenes in Croatia. The images are further split into two subsets. The first subset is called the *main set*, or FM3m for short, and contains 6413 images captured in the good visibility conditions, with little or no visual degradation of the scene caused by bad weather or poor illumination. The second subset is called the *appendix*, or FM3a for short, and contains 5035 images with various types of visual degradation caused by bad weather or illumination conditions.

6.1 Data acquisition

All images are of the resolution 640×480, and were captured by a dashboard mounted in-car camera, so that the traffic scene is seen from the perspective of the vehicle's driver. A vast majority of images (a total of 11243 of them) were obtained by capturing video while driving over various locations in Croatia, and then selecting representative video frames. Additional 572 images were hand-picked from the on-line service Mapillary [126]. This brings the number of images in the dataset to the total of 11448.

6.1.1 Videos captured by the author

These videos were acquired in various locations in Croatia at several different times of a year, over the period of five years, from year 2013 to year 2018. The drives were filmed by a dashboard mounted camera, resulting in more than forty videos of various lengths. All drives were

done during daytime, from early morning until dusk (including twilight period). In the early stages of data collection, a Samsung Galaxy S2 smartphone built-in camera was used, while in the latter stages the built-in camera of an Apple iPhone 6 smartphone was used. The camera setup and the exemplar captured video frame are shown in Figure 6.1. Note that most captured images contain visual artefacts such as small parts of smartphone holder suction cup, parts of vehicle interior (both directly visible and reflected off the windshield glass), as well as occasional specks of dirt on the windshield glass.

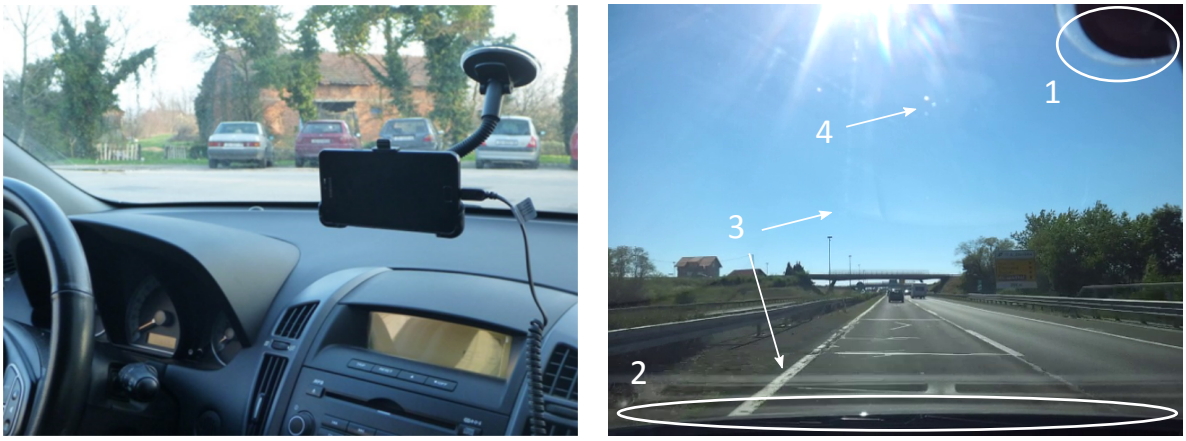


Figure 6.1: Camera setup. The image on the left shows the location of the dashboard mounted camera, in this case an integrated camera of a Samsung Galaxy S2 smartphone. The image on the right is an example frame from a video captured by the camera, with annotated visual obstructions or artefacts. Note the visibility of camera mount (annotation 1), parts of vehicle interior (annotation 2), the reflections of vehicle interior (annotation 3), and the specks of dirt (annotation 4).

Video sampling method

The captured videos were sampled at a rate of one frame for every two seconds of video. As previously discussed, vehicle tracking devices used in fleet management typically send a vehicle state snapshot every 2 to 60 seconds, depending on the current speed of the vehicle. If a vehicle is moving at a speed of 100 km/h, then it travels a distance of 55.6 meters in 2 seconds. It is not likely that any important type of traffic scenario would be missed by this sampling rate. The sampled images were then manually inspected to remove near-duplicate images, as well as images that did not contain a visible traffic scene. Near-duplicate images are caused by vehicle remaining stationary while the traffic scene remains constant (for example, waiting on a traffic light at an empty intersection). The images that do not contain a visible traffic scene usually happen directly after entering or exiting a tunnel, which can result in completely dark or completely white (over-exposed) images due to sudden lighting changes. This image collection criteria approximates the expected use case for fleet management systems well.

6.1.2 Images downloaded from Mapillary

It quickly became apparent that the images obtained by sampling captured videos contain very few samples of certain types of traffic scenes. To improve the number of samples of such scenes, a total of 205 images were hand-picked and downloaded from the on-line street view crowd-sourcing service Mapillary [126]. They were manually cropped and resized as necessary to obtain an image from the approximately same point and field of view as other images in the FM3 dataset. The resolution of images was 640×480 after cropping and resizing. The downloaded images were captured by many different cameras, and suffer from different kinds of visual artefacts. Some show small parts of vehicle interior, others do not. Some images show distortions that look like a consequence of a rolling-shutter effect of a wobbly camera, as shown in Figure 6.2.

Figure 6.2: An example of distorted image obtained from Mapillary. Note the wobbly appearance of the street lights.

6.2 Visual degradation

As was mentioned earlier, the FM3 dataset is split into two subsets of images. The first subset is the *main* set, denoted as FM3m. It contains 6413 images with no significant visual degradation. Images from the FM3m set were captured during good weather, with occasional clouds but no precipitation. The position of the sun was high enough in the sky that every type of open environment was well illuminated. The second set of images is denoted as FM3a, and contains 5035 visually degraded images. The images from FM3a set either have poor illumination conditions caused by the low sun angles, or were captured during bad weather, with at least some precipitation. Three types of bad weather conditions were captured: rain, snow and fog. Note that FM3m and FM3a sets were not separated by examining the visual evidence. The FM3m images were collected by taking care to drive during clear weather and good visibility

conditions. Conversely, the FM3a images were collected by deliberately driving during adverse conditions and capturing the footage of the drives. If during any drive the adverse conditions ceased (e.g. the rain stopped falling for a few minutes), the frames sampled during those periods were manually excluded. This means that exact causes of each visual degradation are known, not guessed.

6.2.1 Low sun

Several drives were made during low sun angles. The position of the sun was at least low enough to cause serious illumination problems if the camera was pointed in the direction of the sun. Some footage was captured during twilight, but no footage was captured after dusk. Due to camera's limited dynamic range, capturing footage while looking at the sun causes one portion of the scene to be very bright, while other portions remain extremely dark. During early pre-twilight period the scene appears normal whenever the camera is not pointed towards the sun. Such portions of the captured footage were excluded manually, only visually degraded images were collected. As the sun continues to set, the appearance of the entire scene becomes darker, even if the camera is not pointed towards the sun. After the sun has set, there is still a decent amount of sunlight during twilight, and the scene remains visible even without use of headlights. The examples of traffic scenes degraded in appearance by low sun angles are shown in Figure 6.3. Note that street lights are turned on in some scenes, but the scene would have been visible even without them.

Figure 6.3: Examples of traffic scenes visually degraded by poor illumination conditions, captured during pre-twilight and twilight.

6.2.2 Rain

Images captured during falling rain often contain raindrops on the windshield glass, which may blur, distort or occlude parts of the scene. The camera occasionally focuses onto the

raindrops on the windshield instead on the traffic scene, in which case the entire scene is slightly blurred. The visibility is usually low and the illumination poorer. Portions of the scene are often occluded by windshield wipers. The wet road may contain reflections. Some images of traffic scenes captured during falling rain are shown in Figure 6.4.

Figure 6.4: Examples of traffic scenes captured during falling rain.

6.2.3 Snow

Images captured during falling snow often appear similar to images captured during rain: the visibility is low and the illumination is poor. There are however, some differences. The most obvious difference is that the objects beside the road are covered with snow, altering the appearance of the scene. Snowflakes are less transparent than raindrops, so the scene appears more noisy than distorted. The road contains fewer and less noticeable reflections, depending on how fast the snow melts. The FM3a contains no footage of drives during very heavy snowfall. There are no images in which snow covers large portions of the road surface. The road lane markings are visible on all images in this category. The Figure 6.5 shows some examples of road scenes captured while driving during falling snow.

6.2.4 Fog

The scenes captured while driving through fog have severely degraded visibility. Distant parts of the scene are often completely covered in fog, leaving only nearby objects as visual evidence about the type of the environment. The fog slowly condensates into water droplets on the windshield glass, which cause the same issues as in case of rain: parts of the scene appear blurred, distorted or occluded. Camera sometimes focuses on the water droplets covering the windshield glass, instead on the traffic scene. The windshield wipers occasionally cover parts of the scene. Some examples of images captured in fog are shown in Figure 6.6.

Figure 6.5: Examples of traffic scenes captured during falling snow. The snow covers parts of the scene beside the road, but the road surface is clearly visible.

Figure 6.6: Examples of traffic scenes captured while driving through fog.

6.2.5 Overview

The number of images captured for each major type of visual degradation is shown in Table 6.1. Note that there is some overlap between the categories: 12 images were captured during both low sun and rain conditions. Windshield wipers occlude parts of the scene in a total of 785 images. The majority of visually degraded images were captured during low sun and falling rain conditions, while there are far fewer samples of snow and fog. The low sun conditions occur every day of the year, the rain occurs often throughout the year, and snow and fog are much rarer events in comparison. Since snow and fog have a high impact on the traffic throughput and safety, they are of great importance to fleet management systems. It would be worthwhile to invest additional time to collect more samples of snow and fog traffic scenes in the future.

Table 6.1: Distribution of visually degraded images in the FM3 dataset with respect to cause of degradation.

type of degradation	low sun	rain	snow	fog
number of images	1990	2428	514	115

6.3 Class labels

After the images were collected, a set of eight classes of interest for fleet management was identified: *highway*, *road*, *tunnel*, *exit*, *settlement*, *overpass*, *booth* and *traffic*. Each of them will now be closely described and their importance in the context of fleet management discussed. This will be followed by a brief overview and then continued by an explanation of the labeling process and precise definition of the annotation criteria. Finally, the distribution of class labels across subsets of the FM3 dataset will be presented and discussed.

6.3.1 Highway

The *highway* class represents traffic scenes of a fast, wide and open road. As was discussed in Chapter 3, differentiating between different types of roads can help fleet management perform higher quality map-matching and route reconstruction. By knowing that a vehicle is in an open environment, the fleet management can infer a low probability of encountering GNSS errors, meaning the GNSS data will be treated as very reliable. Additionally, high-speed travel is expected, so sudden vehicle stops in such an environment should probably trigger an automatic alarm. Examples of this class are shown in Figure 6.7. Most highway images are very similar in appearance, as the scene is usually dominated by large asphalt surfaces and open sky.

Figure 6.7: Examples of traffic scenes labeled as *highway*.

6.3.2 Road

The *road* class represents traffic scenes of an open road that is not a highway, meaning the road is slower and less wide. It is useful for map-matching and route reconstruction augmentations, as well as for indicating a low probability of encountering GNSS errors. Additionally, if a vehicle is driving at very high speeds (e.g. 130 km/h), then it is probably violating the traffic rules, which should be a cause for alarm. Example images for this class are shown in Figure 6.8. The *road* images are more varied in appearance than *highway* images, as there are many types of regional roads that fall into this category. Some of them have two lanes per direction, some just one. Some are well maintained, some not. The objects beside the road are more varied as well (e.g. tall trees could overhang over a narrow road, while the same is not possible in case of highways).

Figure 6.8: Examples of traffic scenes labeled as *road*.

6.3.3 Tunnel

The *tunnel* class represents the scene directly before, on inside a tunnel, but not the very exit of a tunnel. The reason is that the intended usage of tunnel entry and tunnel exit information by fleet management is different. By entering a tunnel a loss of GNSS precision becomes significant. Several consecutive images sent from a tunnel indicate almost certain complete loss of GNSS signal. Encountering the tunnel exit signals the probable restoration of GNSS signal, and for that reason the tunnel exit is considered a separate class in this context. Driving through tunnels can potentially be dangerous, which is an additional reason to detect and report the tunnel entry in a fleet management system. Some samples of this class are shown in Figure 6.9. Most samples are dominated by orange hues caused by tunnel lighting, although some shorter tunnels are poorly lit.

Figure 6.9: Examples of traffic scenes labeled as *tunnel*.

6.3.4 Exit

The *exit* class represents the tunnel exit, which indicates a likely return of GNSS signal if it was lost, or restoration of GNSS precision in case the signal was not completely lost (which is possible in shorter tunnels). Since driving through tunnels can be more dangerous than driving through open areas, exiting the tunnel is an interesting event which should be noted by a fleet management system. Some samples of this class are shown in Figure 6.10. There are few samples of this class, and many samples contain large blobs of bleeding white light caused by overexposure and slow adaptation of camera to sudden illumination changes.

Figure 6.10: Examples of traffic scenes labeled as *exit*.

6.3.5 Settlement

The *settlement* class represents traffic scenes inside settlements. The loss of GNSS accuracy is more likely than in case of open road, as there is a possibility of tall objects (e.g. buildings) being near the vehicle. Since it is impossible to tell for sure whether or not a tall object is near the vehicle, this class by itself does not contribute any information about the probability

of encountering GNSS errors. It can, however, be used as a location prior, to improve the map-matching. If a vehicle encounters a tunnel in a settlement, then the differentiation between the tunnel scene and a settlement scene can improve the quality of map-matching upon exiting the tunnel. Additionally, the settlement scene indicates that the speed limits are likely to be lower than on open roads, and that complete vehicle stops are much more likely to happen. Examples of this class are shown in Figure 6.11. There is a great variability in appearance of samples of this class, much greater than for scenes of open roads or tunnels.

Figure 6.11: Examples of traffic scenes labeled as *settlement*.

6.3.6 Overpass

The *overpass* class represents a scene directly in front of, or under an overpass. Driving under an overpass can cause undetected GNSS multipath errors, which is the main reason for including this class. Detecting an overpass can serve as a GNSS error probability prior. It is also possible it could be used as a location prior, if the map data is detailed enough to contain all types of overpasses (road, railway, etc.). Some samples of this class are shown in Figure 6.12.

Figure 6.12: Examples of traffic scenes labeled as *overpass*.

6.3.7 Booth

The *booth* class represents scenes directly in front of, or under a toll booth. Being under a toll booth can cause undetected multipath errors, so this class provides not only a location prior, but a GNSS error prior as well. More importantly, detecting toll booths is very useful for purposes of reporting (e.g. for keeping track of monthly expenses), and for explaining the very slow speeds and full stops that can precede the toll booth location. Although the locations of toll booths are often present in the map data, they can be incorrectly placed, or missing. Coupled with the fact that the GNSS errors are likely, it is beneficial to be able to detect them visually as well. Some samples of this class are shown in Figure 6.13.

Figure 6.13: Examples of traffic scenes labeled as *booth*.

6.3.8 Traffic

The *traffic* class represents the scenes of dense traffic. Being in a dense traffic is mostly useful for explaining the sudden vehicle stops, and to explain why the perceived vehicle speed is unexpectedly low. A human operator monitoring the vehicle fleet in realtime would expect to be automatically notified of any sudden vehicle stops on highways. It would be beneficial to offer explanation of sudden stops if possible, which makes detecting dense traffic scenarios a very useful feature. If a large vehicle is directly in front of the tracked vehicle, then it can completely occlude the scene. Such a scene can not be assigned any other class label, so it is labeled as *traffic*, to indicate the reason of scene occlusion. Complete scene occlusions by other vehicles should only happen during very low vehicle speeds, as greater distance between vehicles should be kept during high speeds. Samples of this class vary greatly in their appearance. Example images for this class are shown in Figure 6.14.

Figure 6.14: Examples of traffic scenes labeled as *traffic*.

6.3.9 Contributions to fleet management

Each of the described eight FM3 class labels has its own set of potential applications in fleet management systems. Some of them represent the location of a vehicle, others describe an important attribute of the scene. Some are mutually exclusive, while others may co-exist in a single image. Four useful applications that are common across several classes have been identified:

- location disambiguation
- estimating the probability of encountering GNSS errors
- estimating the range of expected speeds
- event reporting

The location disambiguation and the estimation of probability of encountering GNSS errors can be used to improve map-matching and route reconstruction, as was discussed in Chapter 3. Additional usefulness is the ability to explain the reason behind the loss of GNSS to the human operator who might be wondering what's wrong.

The estimation of likely vehicle speeds can be used to raise alarms when an unlikely vehicle behavior is detected (e.g. a sudden stop on a highway, driving very fast through a settlement). Although some maps may contain the information about the exact speed limit for each road segment, a visual approach is still useful even in those cases. Consider a case of construction works on a segment of a highway: maps are not aware of the road works and the reduced speed limit, but a visual inspection could determine that the current scene is not of a typical open highway.

Some classes represent interesting events, such as entering and exiting the tunnels, encountering and driving through dense traffic, and stopping at a toll booth. As such, they should be detected by a fleet management system and presented to the human operator, either in real-time monitoring or in monthly reports. The class *booth* is especially useful for this purpose, as it helps monitor the expenses.

Table 6.2: Usefulness of the FM3 class labels.

	useful for			
label	location disambiguation	estimating the probability of encountering GNSS errors	estimating the range of expected speeds	event reporting
highway	yes	yes	yes	no
road	yes	yes	yes	no
tunnel	yes	yes	yes	yes
exit	yes	yes	no	yes
settlement	yes	no	yes	no
overpass	yes	yes	no	no
booth	yes	yes	yes	yes
traffic	no	no	yes	yes

The usefulness of each class label in these four described applications is listed in Table 6.2. Note that every class except *traffic* can be used for location disambiguation. The usage of *overpass* labels for disambiguation relies on having detailed enough maps (e.g. the railroad network must be known to detect railroad overpasses).

Of course, many other types of traffic scenes could potentially represent interesting classes, but this depends on the type of business the monitored vehicles are conducting. Additionally, not enough samples of such scenarios are present in the images of the FM3 dataset. For example, detecting drives over a shoulder lane could be useful for vehicles of highway maintenance crew, but there are no such samples in the FM3 dataset. Detecting ferry boardings, railroad crossings or driving over a bus-only lane could also be interesting events, but there almost no samples of such scenes in the FM3 dataset either. Detecting the weather conditions (e.g. fog) could also be very useful, but it would require obtaining a large number of samples for each weather type in each type of environment, which is outside the scope of this research.

6.3.10 Annotation process

The following criteria was used to assign class labels to images of the FM3 dataset:

Highway is assigned to images in which the vehicle is on a wide and open road. More precisely, the road is not inside of a tunnel, and there is no definite evidence of vehicle being in a settlement. The road must have at least two lanes and a shoulder lane, or at least three lanes dedicated to the currently driven direction.

Road is assigned to images in which the vehicle is on an open road, outside of settlement, but

not on a highway. The criteria is the same as for the label *highway*, except the number of lanes dedicated to the currently driven direction is less than two, or exactly two but without a shoulder lane.

Tunnel is assigned to images in which the vehicle is directly in front of, or inside a tunnel, but not at the very exit from a tunnel.

Exit is assigned to images in which the vehicle at the very exit from a tunnel.

Settlement is assigned to images in which visible evidence of vehicle being in a settlement is present. Evidence may include visible houses, skyscrapers, playgrounds, or any other visual artifact that does not normally occur outside of settlements. The vehicle is not in a tunnel.

Overpass is assigned to images in which a vehicle is directly in front of, or under an overpass.

Booth is assigned to images in which a vehicle is directly in front of, or at the toll booth. If this label is assigned to an image, then no other labels may be assigned.

Traffic is assigned to images of very dense traffic, or images in which major parts of the scene are occluded by another vehicle. The traffic must be dense in the currently traveled direction, and there must not be a large gap in traffic in front of the vehicle (the currently traveled lane must be occupied by other vehicles).

The above criteria were chosen in order to improve the consistency of the labeling process. In reality, the highways are not defined in terms of number of lanes per direction, or by presence of the shoulder lane, but by the lane width. Fast roads typically have wider lanes than slower roads. The definition of a highway above is used so as to avoid possible inconsistencies that might arise if the annotator was required to estimate the lane width in order to determine if a road was a highway or a local road. Additionally, this also eliminates the cases in which the annotator recognizes the road in question and uses non-visual knowledge to infer the type of the road. Note that the ambiguous term "directly in front of" is used in definition of several labels. The guideline was to allow for up to 30 meters of visually estimated distance from the annotated object. That is, up to 30 meters away from the overpass, toll booth, tunnel entrance or tunnel exit. An inspection of the images containing scenes with toll booths showed that there was a great deal of inconsistency regarding other labels. For example, it is very hard to objectively determine if the traffic at the toll booth is dense or not. Also, some buildings are frequently present in these scenes, and it is hard to objectively determine if those buildings are an evidence of a settlement, or a part of toll booth complex. For that reason the label *booth* is defined as mutually exclusive with the other labels. If in fact the traffic is dense, and the vehicle is in a settlement, it will become apparent after the toll booths are cleared, or was already apparent even before the vehicle arrived to the toll booths.

Some images have multiple labels assigned. The classes *highway*, *road*, *tunnel*, *exit*, *settlement* and *booth* describe a vehicle location, and are defined as mutually exclusive, while the

Table 6.3: The distribution of class labels in the FM3 dataset and its subsets.

class	FM3 total	FM3m total	FM3a total	FM3a (low sun)	FM3a (rain)	FM3a (snow)	FM3a (fog)
highway	6459	4646	1813	1375	312	12	114
road	1020	390	630	79	401	150	0
tunnel	621	616	5	2	3	0	0
exit	80	73	6	4	1	0	1
settlement	3155	583	2572	525	1707	352	0
overpass	242	152	90	42	32	14	2
booth	106	101	5	5	0	0	0
traffic	483	132	351	56	257	38	0
total	11448	6413	5035	1990	2428	514	115

classes *overpass* and *traffic* describe an attribute of the scene, and can coexist with the other labels. Thus the maximum number of labels an image can be assigned is three (e.g. *road, overpass, traffic*). In some countries it might be logical to allow simultaneous assignment of labels *settlement* and *highway*, to detect highways going through large cities, but such a scheme would not be very useful in Croatia. Note that in some cases another vehicle occludes major parts of the scene, so it is possible that none of the six location labels (*highway*, *road*, *tunnel*, *exit*, *settlement* and *booth*) will be assigned. In such cases, only the label *traffic* shall be assigned.

All images of the dataset were permuted, and then a single annotator assigned the class labels relying on the described criteria. No outside knowledge of recognized locations was used to infer the class labels.

The distribution of class labels across all subsets of FM3 dataset is shown in Table 6.3. It is clear that by far the most represented classes are *highway*, *settlement* and *road*, which is of no surprise as those environments are encountered far more frequently than tunnels, overpasses and toll booths. Vehicles can spend a lot of time in slow traffic, but they do not usually cover great distances through dense traffic, which explains the low number of samples of dense traffic scenes. Although video frames were sampled once every two seconds, the near-duplicate images that occur by vehicle standing still in traffic were removed manually from the dataset.

Note that the FM3a set of images has very few samples of classes *tunnel*, *exit* and *booth* each. It is very dangerous, and also improbable to encounter bad weather conditions inside tunnels, so the visually degraded scenes were only encountered upon tunnel entry and exit. The class *booth* is a very rare event. Many samples of toll booth scenes were downloaded from Mapillary [126], where images with bad visibility are not acceptable.

Also note that some types of visual degradation have a very poor balance across classes. Namely, there are very few highway images during falling snow, while images captured during fog contain almost exclusively highway scenes. Severe class disbalance limits the usability of such image subsets in many types of experimental evaluation frameworks, so this should be a priority for improvement in future iterations of the dataset.

Chapter 7

Experiments

This chapter presents a detailed experimental evaluation of the methods described in Chapter 5 using the contributed dataset FM3, which was presented in Chapter 6. The classification setup will be described first, as the same classifier is used to evaluate all descriptor methods. The classifier parameters, data preprocessing steps, and the dataset split will be specified. After that, the classification performance of each descriptor will be evaluated. The descriptor parameters will be discussed, and their achieved per-class classification performance will be presented using the same performance measure for all descriptors. Next, the classification performance will be explored as a function of descriptor length, to see which methods perform better under restricted image representation sizes. This will be followed by evaluation of descriptor compression methods, including two quantization methods: Component-independent quantization (CQ) and Product quantization (PQ). Classification performance will be measured for a range of quantization parameters to see how much information loss is an acceptable trade-off in fleet management applications. A brief exploration of generic compression methods will also be presented. The chapter is concluded by an in-depth discussion of the results, and additional analysis of the best performing descriptor, including the measurement of its resistance to various types of visual degradation.

7.1 The classifier

The SVM classifier was selected, for reasons explained in Chapter 5. The LibSVM implementation [127] was used. The *binary relevance* problem transformation method was applied to achieve multi-label classification. This means that one binary SVM classifier was trained for each of the eight classes of the FM3 dataset. We counter class imbalance by weighing SVM scores with inverse frequency of class samples in the training set. Both linear SVM and SVM with RBF kernel function were tested. The RBF kernel offered better results, especially for very small image representations.

7.1.1 The dataset split

The FM3 dataset consists of two separate sets: FM3m, containing the images in good visibility conditions, and the FM3a, containing visually degraded images. The FM3m set contains the adequate number of instances for all eight classes of interest: *highway*, *road*, *tunnel*, *exit*, *settlement*, *overpass*, *booth* and *traffic*. The FM3a only contains the adequate number of instances for five classes of interest: *highway*, *road*, *settlement*, *overpass* and *booth*. Additionally, the FM3a is poorly balanced across various types of visual degradation. For these reasons, the FM3m set was used in the majority of the experiments, while the FM3a was only used to verify the resistance of the best performing methods to visual degradation of the images.

The FM3m set was split into training, validation and testing subsets as follows. For each class, 25% of instances were used for training, and another 25% of instances were used for validation. The remaining 50% of class instances were used for testing. To find the optimal SVM parameters, each binary classifier was trained on the training subset, and tested on the validation subset. The SVM was then re-trained on the union of the training and validation subsets, using parameters that showed best results on the validation subset. This re-trained classifier was then evaluated on the testing subset, and the achieved results are reported. It is important to note that the same dataset split was used in all experiments in this chapter.

7.1.2 Data preprocessing

Prior to classification, all columns of descriptor vectors were centered and normalized to have zero mean and unit variance. The mean and the variance were estimated on the training subset. In some experiments, PCA was used to reduce the descriptor dimensionality. The centering and normalization step was done before PCA.

7.1.3 Performance measure

As discussed in Chapter 5, *average precision* is selected as a classification performance measure. The average precision for all classes is reported in some experiments, while in others only the mean average precision (mAP) of all classes is reported, for brevity. The mAP is the cost function for selection of all hyperparameters, e.g. the SVM parameters and the quantization parameters.

7.2 SIFT/SFV + GIST descriptor

The SIFT/SFV+GIST descriptor is a concatenation of SIFT/SFV and GIST descriptors. Dense SIFT features were projected down from 128 to $D = 80$ dimensions via PCA. In the SFV framework 16 appearance components were used ($K = 16$) and 1 spatial component ($C = 1$), which

produces a vector of length $K(1+2D)+KC(1+2d)=2656$, where $d=2$ for two-dimensional signals (images). For GIST the implementation provided by the authors [83] was used, without any modifications, which produces a vector of length 512. The final size of the descriptor is thus $2656+512=3168$. PCA was used to further compress the descriptor to lengths 1024, 512 and every other power of two, down to 16. The compressed descriptors were classified with an SVM classifier with the RBF kernel, and the results are shown in Table 7.1.

Table 7.1: Average precision (%) of SIFT/SFV+GIST descriptor on FM3m dataset (SVM with RBF kernel)

	descriptor length						
class	1024	512	256	128	64	32	16
highway	99.73	99.81	99.83	99.85	99.81	99.7	99.34
road	92.10	92.73	92.51	92.02	90.92	87.77	83.00
tunnel	99.92	99.90	99.84	99.65	99.60	99.54	99.83
exit	95.58	95.65	95.04	95.13	92.81	92.09	91.67
settlement	96.10	96.64	96.38	96.44	95.79	94.24	92.39
overpass	96.49	95.99	96.04	94.70	92.48	86.72	87.05
booth	86.43	87.04	85.76	86.51	84.69	78.56	69.67
traffic	82.69	82.42	82.55	81.56	77.53	75.41	63.15
mean	93.63	93.77	93.49	93.23	91.70	89.25	85.76

7.3 VGG/SFV descriptor

The VGG/SFV descriptor uses convolutional features of the VGG network instead of hand-crafted SIFT features. More precisely, it uses the responses of `conv5_4` layer of the VGG-19 [43] convolutional neural network. The network was pre-trained on the ImageNet dataset, and was not additionally trained on the FM3 dataset. The resolution of images in the FM3 dataset is 640×480, so the shape of local features is $30\times40\times512$ (see Table 5.1). In total, there are 1200 features of size $D=512$. PCA was not used to reduce dimensionality of these features. As in SIFT/SFV+GIST descriptor, 16 appearance components were used ($K=16$) and 1 spatial component ($C=1$), thus obtaining a vector of length $K(1+2D+5)=16480$, which was then reduced via PCA to lengths of power of two, from 1024 to 16. The compressed descriptors were classified with an SVM classifier with the RBF kernel, and the results are shown in Table 7.2.

Even though the hand-crafted SIFT/SFV+GIST approach achieves decent results, the deep

Table 7.2: Average precision (%) of VGG/SFV descriptor on FM3m dataset (SVM with RBF kernel)

	descriptor length						
class	1024	512	256	128	64	32	16
highway	99.78	99.76	99.75	99.75	99.69	99.6	99.52
road	91.99	91.30	91.44	90.73	89.28	87.31	85.96
tunnel	99.93	99.94	99.95	99.95	99.96	99.92	99.91
exit	96.74	96.85	97.00	96.41	96.46	95.44	94.17
settlem.	97.49	97.46	97.48	97.38	96.87	96.31	95.28
overpass	97.55	96.95	96.11	95.96	96.49	96.03	90.22
booth	99.58	99.31	98.68	99.17	98.60	97.10	89.77
traffic	85.03	84.63	85.60	84.47	82.87	77.97	74.22
mean	96.01	95.78	95.75	95.48	95.03	93.71	91.13

learning based approach VGG/SFV is clearly better. With only 64 components it outperforms the hand-crafted descriptor of length 1024 (95.9% vs 94.94%).

7.4 ResNet, DenseNet and MobileNetV2 descriptors

Publicly available parameterizations [128, 129, 130] of ResNet-50 [47], DenseNet-BC-121 [52] and MobileNetV2 [54] architectures were used. In case of ResNet and DenseNet, the activations in the second-to-last layer of both networks were extracted (see Tables 5.2 and 5.3), and used as descriptors. In case of MobileNetV2, the globally average pooled outputs of the last bottleneck layer were used (see Table 5.4). For brevity, these descriptors will be referred to as simply ResNet, DenseNet and MobileNet descriptors. The networks were pre-trained on the ImageNet ILSVRC dataset [40], and were not fine-tuned to the FM3 dataset. The input images were neither cropped nor resized. Instead, the spatial dimensions of activation tensors were increased as necessary to accommodate the input of resolution 640×480. In case of ResNet and DenseNet, the second-to-last layer performs global pooling, so the descriptor length is equal to the number of feature maps in that layer, which is 2048 for the ResNet-50 network, and 1024 for the DenseNet-BC-121 network. In case of MobileNet, global average pool was added after the output of the last bottleneck layer to make the descriptor length equal to the number of feature maps in that layer, which is 320.

The descriptors were classified by an SVM classifier with the RBF kernel, and the results are shown in Table 7.3. The classification results show that the DenseNet descriptor performs slightly better than ResNet, despite being shorter. As the ResNet-50 performs better on Ima-

Table 7.3: Average precision (%) of ResNet, DenseNet and MobileNet descriptors on FM3m dataset (SVM with RBF kernel)

class	ResNet (length 2048)	DenseNet (length 1024)	MobileNet (length 320)
highway	99.84	93.68	99.41
road	91.13	99.97	84.50
tunnel	99.94	97.96	99.90
exit	97.70	98.33	96.47
settlement	98.33	97.86	96.28
overpass	97.15	98.81	91.81
booth	98.75	87.85	93.32
traffic	86.75	96.80	75.90
mean	96.20	96.41	92.20

geNet than DenseNet-BC-121 [52], this would suggest that the DenseNet-BC-121 achieves a better knowledge transfer to our dataset. This was verified by fine-tuning both networks on the FM3 dataset, after which ResNet-50 showed a better performance (98.0% vs 97.1%, using a linear SVM classifier). MobileNet shows the least successful classification results, as was expected from a model that sacrifices classification accuracy in favor of computational efficiency.

7.5 The DCGAN descriptor

A public DCGAN [80] parameterization [131] was applied to input images downsampled to a resolution of 32×32 pixels. In the first experiment, the convolutional features of the discriminator network were max-pooled over a 4×4 grid, and concatenated into a descriptor of length 28672. In the subsequent experiments, the descriptor was shortened by reducing the size of the max-pooling grid to 1×1, which results in descriptors of length 1792. Additionally, we also extracted the features obtained by an untrained discriminator. This was done to test how much knowledge transfer happens from ImageNet to FM3, and how much descriptiveness is due to network structure itself. An untrained discriminator was randomly initialized 1000 times, and mean results with standard deviations are reported. All these results are shown in Table 7.4.

Note that some performance is lost when 4×4 max-pooling is replaced with 1×1 max-pooling. The reduction of descriptor size by a factor of 16 makes this performance drop an acceptable trade-off in the context of usage in fleet management. Also note that the random-initialized discriminator achieves a mAP of 85%. This suggests that much of the descriptiveness

Table 7.4: Average precision (%) of descriptors based on DCGAN discriminator on FM3m dataset (SVM classifier)

trained on	ImageNet	ImageNet	ImageNet	untrained
kernel	linear	RBF	linear	linear
max-pool grid	4×4	1×1	1×1	1×1
length	28672	1792	1792	1792
highway	99.71	99.86	99.37	99.47±0.11
road	93.63	92.10	81.32	84.79±1.72
tunnel	99.07	99.33	99.48	99.20±0.11
exit	98.53	93.67	98.14	97.73±0.61
settlement	96.74	95.43	92.91	89.31±1.33
overpass	81.67	83.85	83.08	78.44±3.56
booth	88.59	86.53	86.24	71.42±3.23
traffic	78.53	66.47	65.28	62.92±3.29
mean	92.06	89.66	88.23	85.41±0.81

comes from the convolutional structure itself.

7.6 Evaluation of performance with respect to length

This section evaluates the dependency of classification average precision on descriptor length. Since not all descriptors discussed so far are of the same length, PCA was used to obtain the descriptor variants of various lengths, as necessary. All evaluated descriptors lengths were powers of two, starting from 1024, down to 16. The SIFT/SFV+GIST and VGG/SFV descriptors already use PCA as a part of their framework, and their full results are shown in Tables 7.1 and 7.2. PCA was applied to ResNet, DenseNet and DCGAN 4×4 descriptors as well. Since the full length of DenseNet descriptor is 1024, PCA was only used to produce its variants of lengths from 512 to 16. Using PCA on vectors calculated on FM3 dataset means that all descriptors obtained in this way have some knowledge of the FM3 image data. MobileNet is not included in this experiment because its maximum length is only 320, and because both descriptors from the same category (DenseNet and ResNet) outperform it.

To simplify further text, a numerical suffix N will be added to the name of the descriptor method to refer to the version of that descriptor of length N. For example, a DenseNet descriptor of length 128 will be referred to as DenseNet 128 descriptor.

All descriptor variants were classified with both the linear SVM classifier and the SVM with

RBF kernel. The summary of the results is best seen in graph form, in Figure 7.1 (a, b), which shows the mean average precision of all classes for each of the methods, with respect to the length of the representation. The Figure 7.1 (a) shows the results obtained by a linear SVM classifier, while the Figure 7.1 (b) shows the results for the SVM classifier with the RBF kernel. The RBF kernel shows strictly better results, but it is not as noticeable for large representations as it is for very small ones. This indicates that the RBF kernel is very advantageous when the representations are very small. However, the linear SVM may prove the only applicable solution for large training datasets.

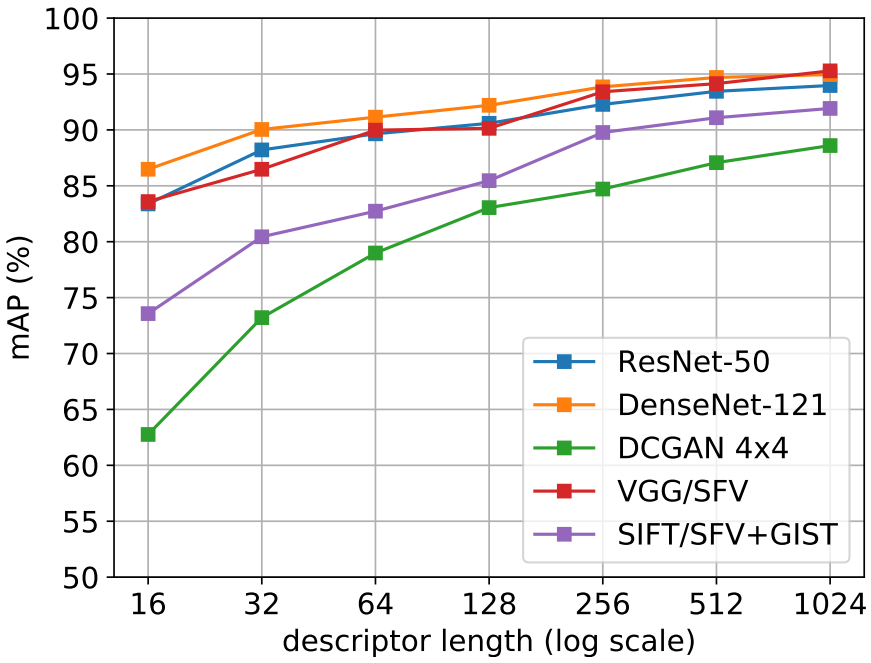
Note that descriptor length reduction does not change the relative order of methods. The DenseNet is a clear winner, followed by ResNet and VGG/SFV, followed by SIFT/SFV+GIST (which relies on hand-crafted features) and finally the DCGAN 4×4 (which is trained in an unsupervised manner). Another thing to note is that the size reduction from 1024 down to 128 barely causes any changes in the mAP for most methods, if RBF kernel is used. The drop becomes more noticeable for lengths 64 and 32, and is significant for length 16.

Average precision across all classes is arguably not the measure for which the system should be optimized. Ideally, every class in the dataset should be classified at acceptable levels of error, so perhaps the minimum AP across all classes is a better choice. Since all tested descriptors perform the worst on the *traffic* class, the results for that class are also included, in Figure 7.2. A considerable drop with respect to the mean performance can be observed for all representation budgets. Note that AP axis starts from 50%.

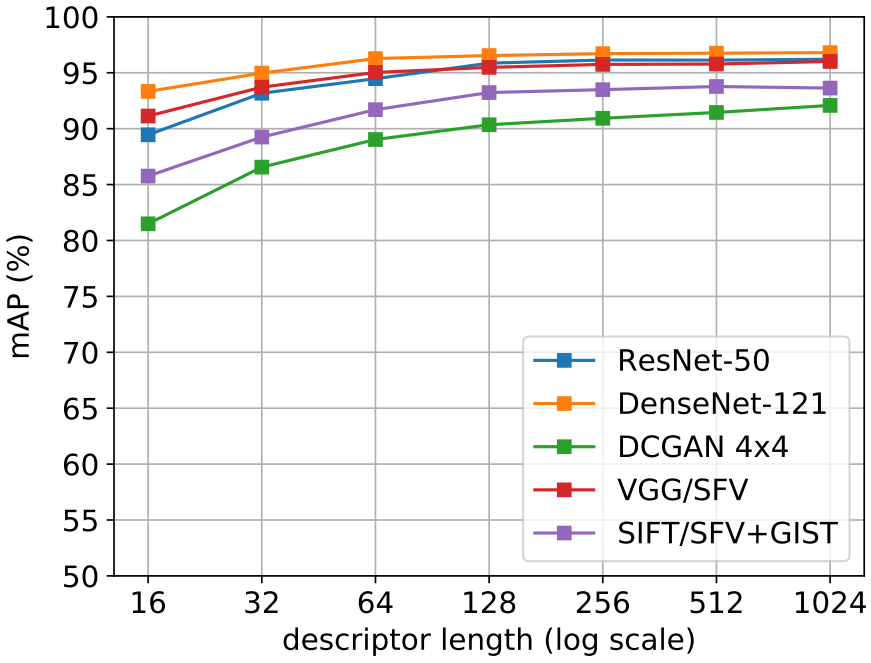
7.7 Analysis at the class level

The results indicate that some classes of the FM3 dataset are much easier to classify than others. The relative order of measured AP values for each class stays approximately the same across different descriptors. The easiest class to classify was *highway*, which is the most represented class in the dataset. Good results are also achieved with class *tunnel*, which is drastically different in appearance from most other classes. Most of the tunnel images are dominated by black and orange colors. Class *exit* is somewhat similar in appearance, and some instances of class *overpass* are dominated by dark colors, but most other images are drastically different and can easily be differentiated from tunnel images even with simpler descriptors. The hardest class is *traffic*, which includes the scenes of dense traffic as well as images with major occlusions of scene by other vehicles. Note that scenes of dense traffic can happen in any kind of location, which increases the visual variability of this class. An interesting thing to note is that the class *booth* is only hard to SIFT/SFV+GIST and DCGAN descriptors, while other methods have excellent performance on this class.

DCGAN showed the worst results overall. However, it could probably be improved if

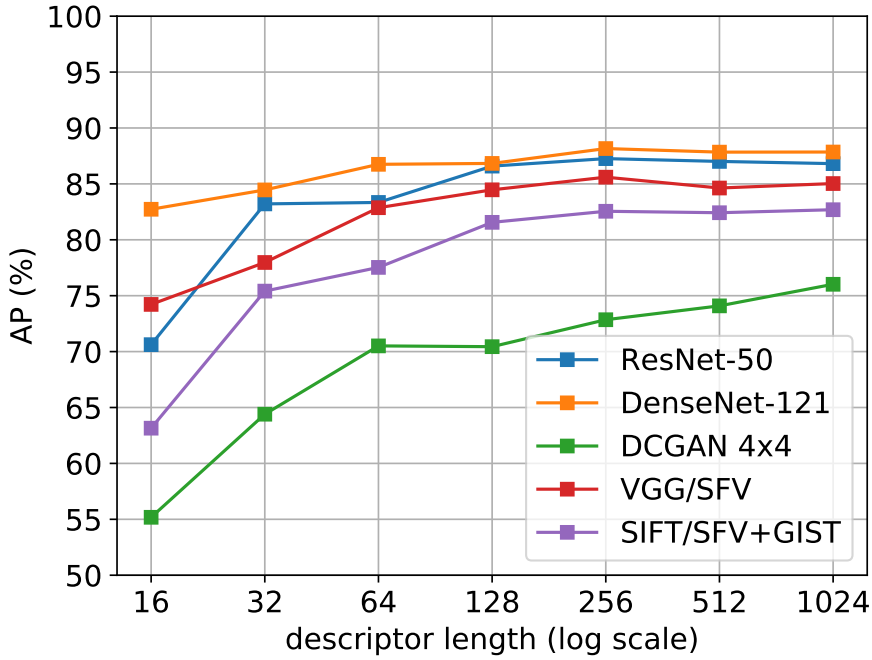


(a) linear SVM, all classes



(b) SVM with RBF kernel, all classes

Figure 7.1: Mean average precision (%) of selected descriptors on FM3m dataset with respect to representation budget. Particular representations are obtained via PCA as necessary. Results for the linear SVM classifier are shown on the left, while results for the SVM classifier with the RBF kernel are shown on the right. Best viewed in colour.



(a) SVM with RBF kernel, class traffic

Figure 7.2: Average precision (%) of selected descriptors for the class *traffic* in the FM3m dataset, with respect to representation budget. Particular representations are obtained via PCA as necessary. An SVM classifier with the RBF kernel was used. Best viewed in colour.

trained on traffic scene images acquired world-wide. Such training would not depend on the class labels, which is a very desirable feature in the proposed fleet management framework. The DenseNet descriptor showed the best results overall, even though it was only trained on the ImageNet image data, and had knowledge of neither the FM3 image data nor the FM3 class labels. The DenseNet 128 shows the classification performance almost equal to that of full-length DenseNet (1024 components), while being 8 times more compact. Therefore, the DenseNet 128 is considered the be the strongest candidate for use in the proposed fleet management framework.

7.8 Descriptor encoding

The exact size of the descriptor in computer memory depends not only on the number of the vector components, but also on the vector encoding. In this section, two vector quantization schemes presented in Chapter 5 will be evaluated: Component-independent quantization and Product quantization [92, 93]. MobileNet and DCGAN descriptors will not be considered, as they showed the worst performance.

7.8.1 Component-independent quantization

A general form of the Component-independent quantization (CQ) approach has three parameters per descriptor component: the endpoints l_i and r_i of the clipping interval $[l_i, r_i]$, and the number of encoding bits q_i, which determines the number of bins $d_i = 2^{q_i}$ into which the interval $[l_i, r_i]$ is split. To simplify the the evaluation, we assume the same number of bits should be used for each component $q_i = \text{bpc}$, and that optimal clipping intervals are $[l_i, r_i] = [\mu_i - k\sigma_i, \mu_i + k\sigma_i]$. This means the simpler equations 5.3, 5.4 and 5.5 can be used. Additionally, since the mean of the descriptors is known to be zero, the μ_i can be removed from the equations. The σ_i is calculated from the corresponding x_i values of the FM3m training set.

Thus only two parameters (bpc and k) are considered when quantizing each proposed descriptor, and the size of the encoded (quantized) D-dimensional descriptor is $D \cdot \text{bpc}$ bits. The classification experimental setup is nearly exactly the same as in section 7.6: the SVM classifier with RBF kernel is used to measure the performance on the FM3m dataset. The only difference is that prior to SVM training and evaluation, the descriptor samples are encoded and then decoded using the component-independent quantization scheme. The SIFT/SFV+GIST, VGG/SFV, ResNet and DenseNet descriptors are considered, and they are reduced to lengths of 16, 32, 64, 128, 256, 512 and 1024, using PCA as necessary. In cases where PCA is used, it is applied prior to vector quantization step. All parameter combinations (bpc, k) from the set $\{2,3,4,8\} \times \{3,4,5,6,7\}$ are tested, and the resulting classification mAP value is reported.

The entire set of experimental results is rather extensive, and is fully included in the Appendix. Some illustrative examples will now be presented and discussed.

The experimental results for the DenseNet 128 descriptor are shown in Table 7.5. Without quantization, this descriptor achieved the classification mAP of 96.54%. If the descriptor is component-independently quantized to 8 bits per component, then the performance drops to $mAP = 96.45\%$, a very minor decrease. The same holds true for all other descriptors: quantization to 8 bits per component shows a very minor drop in classification performance. The quantization to 4 bits per component results in a bit more noticeable drop in classification performance: $mAP = 96.00\%$, about half a percent. Further reduction to 3 and then 2 bits shows a more steep drop in performance, and finally using only a single bit per component significantly reduces the performance. The per-class performance is shown in the Table 7.6. Note that class *traffic* seems to be the most sensitive to the loss of information caused by the quantization, while classes *highway* and *tunnel* are barely affected, as one bit per component seems to be sufficient for their discrimination.

The results indicate that there is little point in encoding descriptor components as IEEE-754 32-bit or 64-bit floating point numbers. For most purposes using 8 bits per component should be sufficient, as they show very little loss in classification performance. Using 4, or even 3 bits per component might be acceptable in systems with more severe bandwidth or storage limitations.

Table 7.5: Classification performance (mAP values, %) of CQ quantized DenseNet 128 descriptor. SVM with RBF kernel on the FM3m dataset. Without quantization, the mAP value of 96.54% is achieved. The maximum values in each row are shown in bold font.

	clipping interval				
bpc	$\pm 3\sigma$	$\pm 4\sigma$	$\pm 5\sigma$	$\pm 6\sigma$	$\pm 7\sigma$
1	88.22	**89.00**	86.81	85.14	83.87
2	**94.08**	93.73	91.90	91.25	90.15
3	94.98	**95.31**	94.57	94.93	94.37
4	95.23	**96.00**	95.73	95.92	95.71
8	95.41	96.32	96.06	96.19	**96.45**

Table 7.6: Impact of Component-independent quantization on per-class classification performance. Several quantized descriptors are compared to their unquantized counterparts. In all examples 4 bits per component (bpc=4) with clipping intervals of $\pm 4\sigma$ ($k = 4$) were used. The average precision values (AP, %) on the FM3m test dataset are reported, SVM classifier with RBF kernel.

	DenseNet 128		VGG/SFV 64		ResNet 256	
class	original	CQ	original	CQ	original	CQ
highway	99.87	99.85	99.69	99.68	99.82	99.82
road	93.88	93.41	89.28	89.09	90.65	89.85
tunnel	99.97	99.96	99.96	99.94	99.93	99.90
exit	97.62	96.66	96.46	96.46	97.62	96.10
settlement	98.27	98.10	96.87	95.99	98.23	97.96
overpass	97.28	96.53	96.49	95.13	96.96	96.32
booth	98.58	98.96	98.60	95.37	98.63	98.23
traffic	86.83	84.54	82.87	77.07	87.26	86.13
mean	96.54	96.00	95.03	93.59	96.14	95.54

Table 7.7 show the best performing descriptors quantized to 128 bits. Table 7.8 lists the single best performing CQ configuration for each of the four tested descriptors, for each of the quantized sizes of 32, 64, 128, 256, 512, 1024 and 2048 bits. The DenseNet descriptor performs the best, while the SIFT/SFV+GIST descriptor shows the worst performance, regardless of the quantized descriptor size. The ResNet performs better than VGG/SFV at all sizes except at 32 and 128 bits.

Table 7.7: Best performing descriptors quantized to 128 bits using CQ method. Results are given in descending average precision values (AP, %) on the FM3m dataset. Classification was performed by an SVM classifier with RBF kernel.

descriptor	components	bpc	clipping	mAP (%)
DenseNet	32	4	$\pm 4\sigma$	94.53
DenseNet	16	8	$\pm 4\sigma$	93.22
DenseNet	64	2	$\pm 3\sigma$	93.15
VGG/SFV	64	2	$\pm 4\sigma$	92.03
ResNet	32	4	$\pm 6\sigma$	91.95
VGG/SFV	32	4	$\pm 7\sigma$	91.75
ResNet	64	2	$\pm 4\sigma$	91.18
VGG/SFV	16	8	$\pm 6\sigma$	89.95
ResNet	16	8	$\pm 7\sigma$	89.68
SIFT/SFV+GIST	32	4	$\pm 6\sigma$	88.41
SIFT/SFV+GIST	64	2	$\pm 3\sigma$	87.85
SIFT/SFV+GIST	16	8	$\pm 7\sigma$	86.43

7.8.2 Product quantization

An exemplar MATLAB implementation of product quantization (PQ) [92, 93] method is provided by the authors [132]. The source code was modified in three ways: i) the number of iterations of *k*-means clustering algorithm was increased to 100 thousand, to insure convergence, ii) the number of clusters was made variable, instead of constant 256 and iii) *k*-means is done on the training subset of the FM3m set.

The DenseNet, ResNet, VGG/SFV and SIFT/SFV+GIST descriptors were reduced via PCA as necessary to lengths of powers of two from 1024 down to 16, and then quantized. The parameters of the PQ method are number of components per code, and number of clusters per code. The number of components per code was set to values of powers of two starting from 1 to descriptor length, while the number of clusters was set to values 16 and 256, thus requiring 4 and

Table 7.8: Best performing CQ configuration for a given number of bits for each of the following descriptor methods: DenseNet, ResNet, VGG/SFV and SIFT/SFV+GIST. Mean average precision (mAP, %) on the FM3m dataset is reported, as achieved by the SVM classifier with RBF kernel.

descriptor	total size (bits)	components	bpc	clipping	mAP (%)
DenseNet	2048	256	8	$\pm7\sigma$	96.69
ResNet	2048	256	8	$\pm6\sigma$	96.07
VGG/SFV	2048	512	4	$\pm6\sigma$	95.54
SIFT/SFV+GIST	2048	256	8	$\pm6\sigma$	93.59
DenseNet	1024	128	8	$\pm7\sigma$	96.45
ResNet	1024	128	8	$\pm7\sigma$	95.71
VGG/SFV	1024	256	4	$\pm6\sigma$	95.08
SIFT/SFV+GIST	1024	256	4	$\pm6\sigma$	93.33
DenseNet	512	64	8	$\pm7\sigma$	96.09
ResNet	512	128	4	$\pm7\sigma$	95.20
VGG/SFV	512	128	4	$\pm7\sigma$	94.62
SIFT/SFV+GIST	512	128	4	$\pm7\sigma$	93.00
DenseNet	256	64	4	$\pm4\sigma$	95.56
ResNet	256	64	4	$\pm4\sigma$	94.20
VGG/SFV	256	64	4	$\pm7\sigma$	93.61
SIFT/SFV+GIST	256	64	4	$\pm6\sigma$	91.25
DenseNet	128	32	4	$\pm4\sigma$	94.53
VGG/SFV	128	64	2	$\pm4\sigma$	92.03
ResNet	128	32	4	$\pm6\sigma$	91.95
SIFT/SFV+GIST	128	32	4	$\pm6\sigma$	88.41
DenseNet	64	16	4	$\pm6\sigma$	91.82
ResNet	64	32	2	$\pm4\sigma$	89.52
VGG/SFV	64	32	2	$\pm3\sigma$	88.13
SIFT/SFV+GIST	64	16	4	$\pm4\sigma$	84.57
DenseNet	32	16	2	$\pm3\sigma$	85.42
VGG/SFV	32	16	2	$\pm3\sigma$	80.34
ResNet	32	16	2	$\pm3\sigma$	78.84
SIFT/SFV+GIST	32	16	2	$\pm3\sigma$	74.13

8 bits to encode, respectively. The quantized vectors were then classified using an SVM with RBF kernel, and mean average precision (mAP) values are used as a measure of classification performance.

The performance of all configurations resulting in descriptor sizes of 1024 bits is shown in Table 7.9. The best performing quantized descriptors of size 64 bits are shown in Table 7.10. For complete results, refer to the Appendix. For any fixed size of quantized descriptors, DenseNet shows the best results, followed by ResNet, then VGG/SFV and finally SIFT/SFV+GIST.

Table 7.11 shows the best results obtained via CQ and PQ method for every fixed quantized descriptor size from 32 to 2048 bits (powers of two only). For sizes of 32 and 64 bits, the PQ with 256 clusters per code (i.e. 8-bit codes) shows noticeably better performance. For sizes of 128 bits and larger, the CQ and PQ approaches show similar performance (regardless of number of bits per single PQ code). Note that CQ sometimes slightly outperforms PQ method. Since CQ bins the descriptor component values uniformly, while the PQ bins them according to cluster centroids, this suggest that cluster centroids do not necessarily represent the most discriminative values. Further research is necessary to determine if that is the case.

7.8.3 General-purpose compression

As discussed in Chapter 4, a general-purpose compression might be used to archive a large number of descriptors, thus reducing the storage requirements. The FM3 dataset contains 11448 images, which would require 89 MB of memory if each image would be represented as an 1024-dimensional vector of 64-bit floating point numbers. That number of descriptors could be sent by a single vehicle in 8 days, or by a fleet of 1000 vehicles in 11 minutes, assuming a rate of one status report per minute.

DenseNet representations of length 1024 were computed for all images in the FM3 dataset. They were encoded as arrays of 64-bit and 32-bit floating point numbers, and also quantized using CQ and PQ approaches. For CQ encoding 8 bits per component (bpc=8) and 4 bits per component were used (bpc=4), resulting in representation sizes of 1024 B and 512 B. For PQ encoding representation size was set to 128 B in two different ways: first by using 8 components per code (cpc=8) and 8 bits per code (bpc=8), then by using 4 components per code (cpc=4) and 4 bits per code (bpc=4). For each encoding, the entire dataset was dumped to a binary file with no headers, and no padding for alignment, so the file size is equal to the dataset representation in memory. Each file was then compressed using 7Zip v16.04 software [133] using 7z archive format, LZMA2 compression method, and maximum compression setting: compression level ultra, dictionary size 1536 MB, word size 273, solid block.

The results are shown in Table 7.12. The least compressible encodings are the ones that are also the most wasteful: the floating point arrays. Quantized descriptors can be compressed to save an additional 20% to 38% of storage, depending on the quantization method.

Table 7.9: Classification performance of descriptors reduced to 1024 bits via PQ. The SVM classifier with RBF kernel was used, and mAP values on the FM3m dataset are reported, in descending order.

descriptor	components total	components per code	code size (bits)	mAP (%)
DenseNet	128	1	8	96.41
DenseNet	1024	4	4	96.27
DenseNet	256	1	4	96.22
DenseNet	512	4	8	96.15
DenseNet	256	2	8	96.14
ResNet	256	1	4	96.08
ResNet	256	2	8	96.04
DenseNet	1024	8	8	95.99
ResNet	128	1	8	95.81
DenseNet	512	2	4	95.46
VGG/SFV	256	1	4	95.36
ResNet	512	4	8	95.27
VGG/SFV	512	2	4	95.24
ResNet	512	2	4	95.24
VGG/SFV	256	2	8	95.11
VGG/SFV	128	1	8	94.80
VGG/SFV	512	4	8	94.37
SIFT/SFV+GIST	256	2	8	93.37
SIFT/SFV+GIST	256	1	4	93.27
SIFT/SFV+GIST	128	1	8	93.20
SIFT/SFV+GIST	512	4	8	92.98
VGG/SFV	1024	8	8	92.93
ResNet	1024	8	8	92.84
SIFT/SFV+GIST	512	2	4	92.56
VGG/SFV	1024	4	4	92.24
ResNet	1024	4	4	91.81
SIFT/SFV+GIST	1024	8	8	90.68
SIFT/SFV+GIST	1024	4	4	89.40

Table 7.10: Classification performance of descriptors reduced to 64 bits via PQ. The SVM classifier with RBF kernel was used, and mAP values on the FM3m dataset are reported, in descending order. Only combinations achieving mAP greater than 90% are listed.

descriptor	components total	components per code	code size (bits)	mAP (%)
DenseNet	32	4	8	93.28
DenseNet	128	16	8	91.96
DenseNet	64	8	8	91.95
DenseNet	32	2	4	91.85
ResNet	32	4	8	91.78
VGG/SFV	32	2	4	91.41
DenseNet	1024	128	8	91.40
DenseNet	16	1	4	91.02
DenseNet	16	2	8	90.94
VGG/SFV	64	8	8	90.83
ResNet	32	2	4	90.63
ResNet	64	8	8	90.61
VGG/SFV	32	4	8	90.18

Table 7.11: Mean average precision (mAP, %) of best performing CQ and PQ quantized descriptors for several fixed sizes, as achieved by SVM classifier with RBF kernel on the FM3m dataset.

descriptor size (bits)	descriptor method	CQ quantization	PQ (4-bit codes)	PQ (8-bit codes)
2048	DenseNet	**96.69**	96.68	96.66
1024	DenseNet	**96.45**	96.27	96.41
512	DenseNet	96.09	**96.29**	96.21
256	DenseNet	**95.56**	95.45	95.50
128	DenseNet	**94.53**	94.38	94.31
64	DenseNet	91.82	91.85	**93.28**
32	DenseNet	85.42	87.76	**90.50**

7.9 Analysis of DenseNet 128 descriptor

The confusion matrix of the DenseNet 128 descriptor is shown in Table 7.13. Note that only six classes that are defined as mutually exclusive are included. The ROC curves for all classes are shown in Figure 7.3. Some examples of mispredicted classes are shown in Figure 7.4.

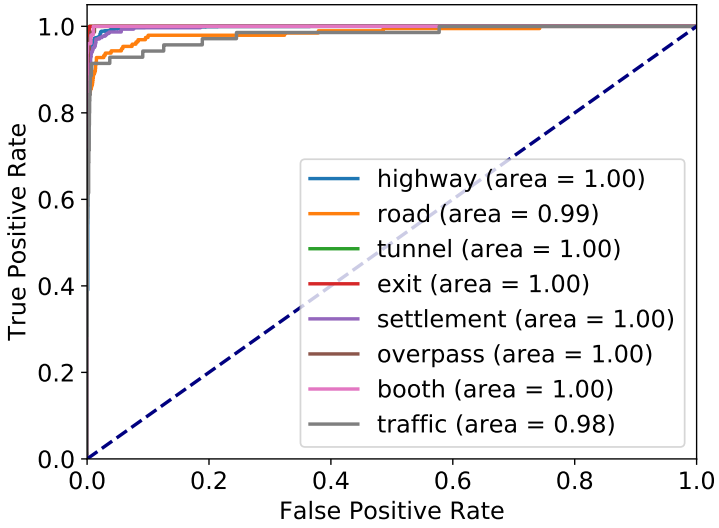


Figure 7.3: ROC curves for the SVM classifier with RBF kernel trained on the DenseNet 128. Measured on the FM3m dataset. Best viewed in color.

7.9.1 Resilience to visual degradation

All the experiments so far were done on clear images with good visibility and little or no visual degradation. To test the resilience of the DenseNet 128 descriptor to various types of visual degradation, we performed a series of experiments involving the FM3a set of images. The first experiment uses the SVM classifier trained on FM3m train to classify the images from the

Table 7.12: Original and compressed sizes of FM3 dataset with images represented as various encodings of the DenseNet 1024 descriptors

encoding	original size	compressed size	space savings (%)
unquantized, 64-bit floating point	91584 kB	83781 kB	8.52
unquantized, 32-bit floating point	45792 kB	41015 kB	10.43
CQ, k=7, bpc=8	11448 kB	8252 kB	27.91
CQ, k=4, bpc=4	5724 kB	3532 kB	38.28
PQ, cpc=8, bpc=8	1431 kB	1133 kB	20.84
PQ, cpc=4, bpc=4	1431 kB	1035 kB	27.68

Table 7.13: Confusion matrix for DenseNet 128 descriptor, SVM classifier with RBF kernel

	predicted class					
actual class	highway	road	tunnel	exit	settl.	booth
highway	2298	2	0	0	6	0
road	28	153	0	0	5	0
tunnel	0	0	302	3	0	0
exit	2	0	1	37	0	0
settlement	16	4	0	0	276	0
booth	1	0	0	0	0	44

(a) road as highway

(b) road as highway

(c) settlement as highway

(d) road as settlement

(e) traffic false positive

(f) traffic false negative

(g) road as settlement (rain)

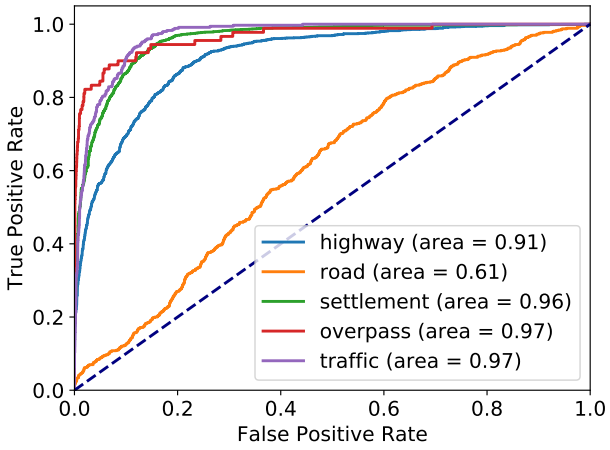
(h) highway as road (fog)

Figure 7.4: Examples of mispredictions for the SVM classifier with RBF kernel on DenseNet 128 descriptors. Examples (a) to (f) are from the FM3m testing set, while examples (g) and (h) are from the FM3a set. In all examples the SVM was trained on the FM3m training set.

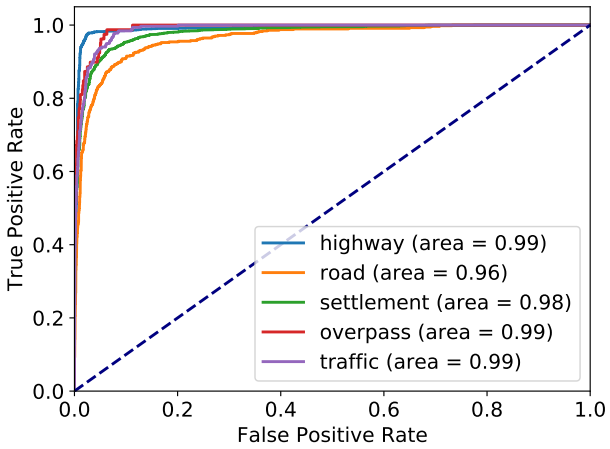
FM3a set. In the second experiment, the classifier was trained on the training set of FM3m and 10% of FM3a images, and evaluated on the remaining 90% of the FM3a images. In the third experiment, the classifier was trained on FM3m train and an additional 25% of FM3a images, and evaluated on the remaining 75% of the FM3a images. The measured AP values for all three experiments is shown in Table 7.14. Since only the classes *highway*, *road*, *settlement*, *overpass* and *traffic* are adequately represented in the FM3a set, only results for those five classes are listed. Note that even without any training on the degraded images, the system was able to classify all the classes surprisingly well, except class *road*. In this case, the class *road* was most frequently confused by classes *settlement* (93 times) and *highway* (82 times). Two examples of these mispredictions are shown in Figure 7.4. After adding some degraded images to the training set, the mean average precision quickly rises up to 94.46%, which is very close to mean average precision value for the same set of classes on non-degraded images (which is 95.22%). The ROC curves for the five evaluated classes across all three experiments are shown in Figure 7.5. Note that in Figure 7.5 (a) the ROC curves for classes *settlement*, *overpass* and *traffic* are visibly better than the ROC curve for the class *highway*, which is confused with the worst-performing class *road*. In Figures 7.5 (b) and (c) the ROC curve for the class *highway* is better than ROC curves of other classes. The ROC curve for the class *road* is still the worst, but it is much less noticeable, as in Figure 7.5 (c) all curves look similar.

Table 7.14: Average precision (%) of the DenseNet 128 descriptor trained on increasing portions of the FM3a dataset (SVM with RBF kernel)

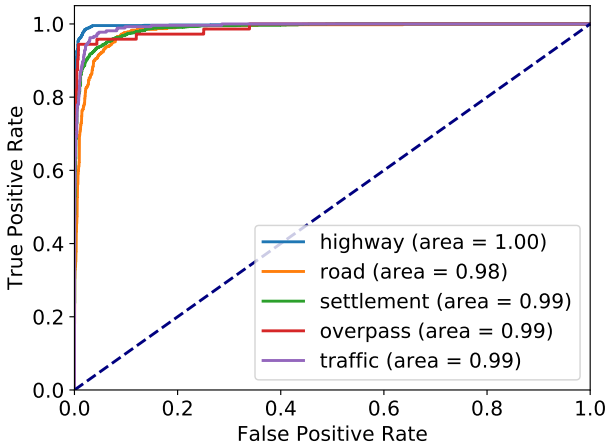
class	not trained on FM3a images	trained on 10% of FM3a images	trained on 25% of FM3a images
highway	85.25	98.42	99.55
road	18.87	83.52	90.30
settlement	95.73	98.04	98.85
overpass	72.08	76.97	90.07
traffic	71.70	86.96	93.53
mean	68.73	88.78	94.46



(a) training set contains no FM3a samples



(b) training set contains 10% of FM3a samples



(c) training set contains 25% of FM3a samples

Figure 7.5: ROC curves for the SVM classifier with RBF kernel trained on the DenseNet 128 descriptor. Training data contains 50% of FM3m samples, and incrementally increasing amounts of FM3a samples. Tested on the rest of FM3a images. Best viewed in color.

Chapter 8

Conclusion and outlook

This thesis demonstrates that it is possible to build an image categorization system that is applicable (and useful) in fleet management. A typical fleet management system consists of thousands of clients and the central server. The clients capture the images, while the central server determines image categories (and exploits them) to deliver added value to the customers. This thesis identifies the following requirements: i) the bandwidth should be used sparingly in order to reduce the costs, and ii) the set of image categories has to be open (i.e. not known by clients) in order to support flexibility of the business logic. This thesis satisfies the requirements by calculating compact image representations on the clients, and performing the classification on the server.

The proposed solution achieves the compact image representation by applying an efficient (possibly lossy) coding scheme to a short image descriptor. Several coding schemes and short image descriptors are studied. To ensure that the set of image categories remains open, we consider only image descriptors that can be trained without the knowledge of the categories. This restriction has an added benefit of reducing the risk of overfitting. To further reduce the risk of overfitting to target dataset, this thesis also studies the descriptors that can be trained without an application specific dataset. Here we consider several general purpose descriptors trained on ImageNet. Our experiments show that these descriptors can deliver very useful classification performance in scenes which are specific to fleet management even without fine tuning. Finally, a completely unsupervised learning approach is also explored, in order to evaluate the feasibility of training on extremely large unlabeled datasets.

The evaluated descriptors include handcrafted gradient histograms (GIST, SIFT), non-linear embeddings with respect to placement and appearance distribution of local image features (spatial Fisher vectors), and convolutional representations trained in an end-to-end fashion (VGG, DenseNet, ResNet, MobileNetV2 and DCGAN). The descriptors dimensionality was reduced using PCA down to as low as 16 components. Subsequently, the descriptors were quantized by product quantization (PQ) and its simpler variant which we call component-independent quan-

tization (CQ). Finally, the descriptors are classified with SVM. The effect of general purpose compression algorithms was considered as well.

This thesis introduces FM3–a novel traffic scene dataset containing 11448 images labeled with eight classes of interest to fleet management* This dataset has been extensively used in the experiments which perform experimental evaluation of all described methods in terms of generalization accuracy at different representation budgets. Best performance is achieved by two deep convolutional models trained in a supervised manner: DenseNet-121 and ResNet-50. They achieve mean average classification precision (mAP) above 96% without using any knowledge of the target dataset. Another way to achieve nearly the same classification performance is by using target images to learn spatial Fisher vector embeddings of features obtained by VGG trained on ImageNet. As expected from a model designed for computational efficiency, the MobileNetV2 network trained on ImageNet achieves much lower classification performance (mAP=92%), on par with performance shown by completely handcrafted model based on SIFT and GIST (mAP=93%). Finally, the approach based on DCGAN shows the worst results (mAP=89%), as was expected of completely unsupervised approach.

Experiments with PCA show that descriptors can be reduced to as few as 128 components without sacrificing much classification performance. In some cases even descriptors reduced to 32 components might be considered useful. For such short image representations the best results are obtained by using an SVM classifier with RBF kernel, while for longer image representations the linear SVM produces equally good results. Both PQ and CQ quantization approaches prove to be useful for further reducing the image representations. Mean average classification precision (mAP) above 96% has been achieved with image representations of only 512 bits, while mAP above 93% has been achieved with only 64 bits. The simpler CQ approach is slightly better for representations of 128 bits and above, while PQ is better for representations of 64 bits and shorter.

Experiments with general purpose compression show that further savings of between 20% and 40% are possible, depending on the descriptor encoding. General purpose compression typically involves a fixed space overhead, which makes it unsuitable for compressing a single short descriptor. However, it can be used for compress large bulks of descriptors, for long-term archiving.

The final set of experiments explores the sensitivity of the proposed pipeline to adverse imaging conditions. In order to achieve that goal we have collected and labeled a dataset of 5035 traffic images with various types of severe visual degradation, caused by bad weather and low sun angles. Experiments indicate that the best performing descriptor (DenseNet-121) performs very well on such images, provided that some portion of them is added to the training

*In order to facilitate further research in the field, we release the FM3 dataset to the research community at the address below:
`http://www.zemris.fer.hr/~ssegvic/datasets/unizg-fer-fm3am.tar.gz`

data.

In conclusion, using a deep convolutional model trained in a supervised manner in combination with a lossy quantization scheme is sufficient to obtain very robust and short image representations. These representations can later be categorized into an open set of classes, which is useful for systems with limited bandwidth requirements, such as fleet management. Future work should investigate whether this can be achieved in other domains as well, for datasets unrelated to traffic scenes, and for more diverse sets of classes.

Future work should also include measuring the robustness of the system to yet unseen traffic scenes. Ideally, encountering scenes not found in the dataset (e.g. underground garage, traffic accident scene, camera malfunctions and severe scene occlusions) should not result in predicting the wrong class. Instead, the system should not output any prediction at all.

Generalization capabilities of this approach should also be investigated, by testing the classifier on traffic images from different countries. Further increasing the number of classes and the number and difficulty of degraded images would enable measuring the influence of the task difficulty on the image representation size requirements. Another potentially interesting future work would be further compression of convolutional descriptors by fine-tuning on ImageNet. Besides models trained on the ImageNet dataset, semantic segmentation datasets (e.g. Vistas, WildDash, CityScapes) could also be considered.

Additional experimental results

This appendix lists some experimental results that were too extensive to include directly into the main text.

8.1 Quantization

This section lists the complete set of experimental results regarding the Component-independent quantization and Product quantization. The results are grouped by the size of quantized descriptor (expressed in bits), and sorted in the descending order of classification performance. For each descriptor we include the values of optimal hyperparameters.

8.1.1 Component-independent quantization

size of quantized descriptor	descriptor method	number of components	bits per component	clipping interval	mAP (%)
8192 bits					
	DenseNet	1024	8	$\pm 7\sigma$	96.79
	ResNet	1024	8	$\pm 7\sigma$	96.10
	VGG/SFV	1024	8	$\pm 7\sigma$	96.01
	SIFT/SFV+GIST	1024	8	$\pm 6\sigma$	93.55
4096 bits					
	DenseNet	512	8	$\pm 7\sigma$	96.78
	DenseNet	1024	4	$\pm 4\sigma$	96.49
	ResNet	512	8	$\pm 7\sigma$	96.07
	VGG/SFV	1024	4	$\pm 6\sigma$	95.96
	ResNet	1024	4	$\pm 7\sigma$	95.77
	VGG/SFV	512	8	$\pm 7\sigma$	95.76
	SIFT/SFV+GIST	512	8	$\pm 6\sigma$	93.78
	SIFT/SFV+GIST	1024	4	$\pm 7\sigma$	93.19
2048 bits					
	DenseNet	256	8	$\pm 7\sigma$	96.69
	DenseNet	512	4	$\pm 4\sigma$	96.43
	ResNet	256	8	$\pm 6\sigma$	96.07
	ResNet	512	4	$\pm 7\sigma$	95.60
	VGG/SFV	512	4	$\pm 6\sigma$	95.54
	VGG/SFV	256	8	$\pm 7\sigma$	95.41
	DenseNet	1024	2	$\pm 3\sigma$	94.96
	VGG/SFV	1024	2	$\pm 4\sigma$	94.36

	SIFT/SFV+GIST	256	8	±6σ	93.59
	ResNet	1024	2	±4σ	93.53
	SIFT/SFV+GIST	512	4	±7σ	93.51
	SIFT/SFV+GIST	1024	2	±4σ	90.92
1024 bits					
	DenseNet	128	8	±7σ	96.45
	DenseNet	256	4	±4σ	96.35
	ResNet	128	8	±7σ	95.71
	ResNet	256	4	±7σ	95.60
	VGG/SFV	256	4	±6σ	95.08
	VGG/SFV	128	8	±5σ	94.84
	DenseNet	512	2	±3σ	94.83
	VGG/SFV	512	2	±4σ	94.28
	SIFT/SFV+GIST	256	4	±6σ	93.33
	SIFT/SFV+GIST	128	8	±6σ	93.21
	ResNet	512	2	±4σ	93.11
	SIFT/SFV+GIST	512	2	±4σ	91.31
512 bits					
	DenseNet	64	8	±7σ	96.09
	DenseNet	128	4	±4σ	96.00
	ResNet	128	4	±7σ	95.20
	DenseNet	256	2	±4σ	94.77
	ResNet	64	8	±7σ	94.71
	VGG/SFV	128	4	±7σ	94.62
	VGG/SFV	64	8	±7σ	94.15
	VGG/SFV	256	2	±4σ	93.36
	ResNet	256	2	±4σ	93.05
	SIFT/SFV+GIST	128	4	±7σ	93.00
	SIFT/SFV+GIST	64	8	±6σ	92.00
	SIFT/SFV+GIST	256	2	±4σ	90.81
256 bits					
	DenseNet	64	4	±4σ	95.56
	DenseNet	32	8	±4σ	94.95
	ResNet	64	4	±4σ	94.20
	DenseNet	128	2	±3σ	94.08
	VGG/SFV	64	4	±7σ	93.61
	ResNet	32	8	±6σ	92.85
	ResNet	128	2	±4σ	92.70
	VGG/SFV	128	2	±4σ	92.64
	VGG/SFV	32	8	±5σ	92.60
	SIFT/SFV+GIST	64	4	±6σ	91.25
	SIFT/SFV+GIST	128	2	±3σ	90.08
	SIFT/SFV+GIST	32	8	±6σ	89.31
128 bits					
	DenseNet	32	4	±4σ	94.53
	DenseNet	16	8	±4σ	93.22
	DenseNet	64	2	±3σ	93.15
	VGG/SFV	64	2	±4σ	92.03
	ResNet	32	4	±6σ	91.95
	VGG/SFV	32	4	±7σ	91.75
	ResNet	64	2	±4σ	91.18

	VGG/SFV	16	8	$\pm 6\sigma$	89.95
	ResNet	16	8	$\pm 7\sigma$	89.68
	SIFT/SFV+GIST	32	4	$\pm 6\sigma$	88.41
	SIFT/SFV+GIST	64	2	$\pm 3\sigma$	87.85
	SIFT/SFV+GIST	16	8	$\pm 7\sigma$	86.43
64 bits					
	DenseNet	16	4	$\pm 6\sigma$	91.82
	DenseNet	32	2	$\pm 3\sigma$	90.66
	ResNet	32	2	$\pm 4\sigma$	89.52
	VGG/SFV	32	2	$\pm 3\sigma$	88.13
	VGG/SFV	16	4	$\pm 4\sigma$	88.07
	ResNet	16	4	$\pm 4\sigma$	87.99
	SIFT/SFV+GIST	16	4	$\pm 4\sigma$	84.57
	SIFT/SFV+GIST	32	2	$\pm 3\sigma$	83.50
32 bits					
	DenseNet	16	2	$\pm 3\sigma$	85.42
	VGG/SFV	16	2	$\pm 3\sigma$	80.34
	ResNet	16	2	$\pm 3\sigma$	78.84
	SIFT/SFV+GIST	16	2	$\pm 3\sigma$	74.13

8.1.2 Product quantization

size of quantized descriptor	descriptor method	components (total)	components per code	code size (bits)	mAP (%)
2048 bits					
	DenseNet	512	1	4	96.68
	DenseNet	1024	4	8	96.66
	DenseNet	1024	2	4	96.50
	DenseNet	512	2	8	96.36
	DenseNet	256	1	8	96.28
	ResNet	512	2	8	96.17
	ResNet	256	1	8	96.14
	ResNet	512	1	4	96.08
	VGG/SFV	512	1	4	95.74
	VGG/SFV	1024	2	4	95.70
	VGG/SFV	512	2	8	95.61
	ResNet	1024	4	8	95.46
	VGG/SFV	256	1	8	95.28
	VGG/SFV	1024	4	8	95.12
	ResNet	1024	2	4	95.12
	SIFT/SFV+GIST	512	2	8	93.82
	SIFT/SFV+GIST	512	1	4	93.42
	SIFT/SFV+GIST	256	1	8	93.33
	SIFT/SFV+GIST	1024	2	4	91.77
	SIFT/SFV+GIST	1024	4	8	91.76
1024 bits					
	DenseNet	128	1	8	96.41
	DenseNet	1024	4	4	96.27
	DenseNet	256	1	4	96.22
	DenseNet	512	4	8	96.15
	DenseNet	256	2	8	96.14

	ResNet	256	1	4	96.08
	ResNet	256	2	8	96.04
	DenseNet	1024	8	8	95.99
	ResNet	128	1	8	95.81
	DenseNet	512	2	4	95.46
	VGG/SFV	256	1	4	95.36
	ResNet	512	4	8	95.27
	VGG/SFV	512	2	4	95.24
	ResNet	512	2	4	95.24
	VGG/SFV	256	2	8	95.11
	VGG/SFV	128	1	8	94.80
	VGG/SFV	512	4	8	94.37
	SIFT/SFV+GIST	256	2	8	93.37
	SIFT/SFV+GIST	256	1	4	93.27
	SIFT/SFV+GIST	128	1	8	93.20
	SIFT/SFV+GIST	512	4	8	92.98
	VGG/SFV	1024	8	8	92.93
	ResNet	1024	8	8	92.84
	SIFT/SFV+GIST	512	2	4	92.56
	VGG/SFV	1024	4	4	92.24
	ResNet	1024	4	4	91.81
	SIFT/SFV+GIST	1024	8	8	90.68
	SIFT/SFV+GIST	1024	4	4	89.40
512 bits					
	DenseNet	128	1	4	96.29
	DenseNet	128	2	8	96.21
	ResNet	128	1	4	95.73
	ResNet	128	2	8	95.65
	DenseNet	64	1	8	95.47
	ResNet	256	4	8	95.40
	DenseNet	1024	16	8	95.25
	DenseNet	256	4	8	95.13
	DenseNet	256	2	4	95.02
	ResNet	256	2	4	94.96
	DenseNet	1024	8	4	94.82
	ResNet	64	1	8	94.80
	VGG/SFV	128	2	8	94.52
	VGG/SFV	128	1	4	94.34
	DenseNet	512	8	8	94.29
	VGG/SFV	256	2	4	94.20
	VGG/SFV	64	1	8	93.95
	VGG/SFV	256	4	8	93.70
	SIFT/SFV+GIST	128	2	8	93.25
	SIFT/SFV+GIST	128	1	4	93.16
	ResNet	512	8	8	93.16
	DenseNet	512	4	4	92.95
	SIFT/SFV+GIST	256	4	8	92.23
	ResNet	512	4	4	92.09
	VGG/SFV	512	4	4	92.04
	SIFT/SFV+GIST	256	2	4	91.97
	SIFT/SFV+GIST	64	1	8	91.80

	ResNet	1024	16	8	91.74
	VGG/SFV	512	8	8	91.70
	SIFT/SFV+GIST	512	8	8	90.77
	VGG/SFV	1024	16	8	90.61
	SIFT/SFV+GIST	512	4	4	88.87
	VGG/SFV	1024	8	4	88.48
	SIFT/SFV+GIST	1024	16	8	88.23
	ResNet	1024	8	4	86.90
	SIFT/SFV+GIST	1024	8	4	86.56
256 bits					
	DenseNet	128	4	8	95.50
	DenseNet	64	1	4	95.45
	DenseNet	64	2	8	95.01
	VGG/SFV	128	2	4	94.81
	DenseNet	1024	32	8	94.76
	DenseNet	128	2	4	94.74
	DenseNet	32	1	8	94.74
	ResNet	64	2	8	94.67
	ResNet	64	1	4	94.59
	ResNet	128	4	8	94.31
	ResNet	128	2	4	94.30
	VGG/SFV	64	1	4	93.91
	ResNet	256	8	8	93.88
	VGG/SFV	64	2	8	93.86
	DenseNet	256	8	8	93.52
	VGG/SFV	128	4	8	93.51
	DenseNet	1024	16	4	93.28
	ResNet	32	1	8	92.89
	VGG/SFV	32	1	8	92.75
	ResNet	256	4	4	92.63
	DenseNet	256	4	4	92.63
	SIFT/SFV+GIST	128	2	4	92.25
	DenseNet	512	16	8	92.03
	SIFT/SFV+GIST	128	4	8	91.99
	SIFT/SFV+GIST	64	2	8	91.81
	SIFT/SFV+GIST	64	1	4	91.63
	VGG/SFV	256	8	8	91.39
	VGG/SFV	256	4	4	90.75
	DenseNet	512	8	4	90.48
	VGG/SFV	512	16	8	90.35
	ResNet	512	16	8	89.50
	SIFT/SFV+GIST	32	1	8	89.20
	SIFT/SFV+GIST	256	4	4	89.12
	SIFT/SFV+GIST	512	16	8	89.09
	SIFT/SFV+GIST	256	8	8	88.97
	VGG/SFV	1024	32	8	88.96
	VGG/SFV	512	8	4	87.46
	ResNet	1024	32	8	87.18
	ResNet	512	8	4	87.14
	SIFT/SFV+GIST	512	8	4	86.35
	SIFT/SFV+GIST	1024	32	8	85.14

	VGG/SFV	1024	16	4	85.08
	ResNet	1024	16	4	81.28
	SIFT/SFV+GIST	1024	16	4	80.57
128 bits					
	DenseNet	32	1	4	94.38
	DenseNet	32	2	8	94.31
	DenseNet	128	8	8	94.03
	DenseNet	64	4	8	94.03
	DenseNet	1024	64	8	93.63
	ResNet	32	2	8	93.29
	ResNet	64	2	4	93.15
	DenseNet	64	2	4	93.15
	ResNet	32	1	4	92.65
	VGG/SFV	64	4	8	92.60
	VGG/SFV	64	2	4	92.40
	ResNet	64	4	8	92.33
	VGG/SFV	32	2	8	92.30
	VGG/SFV	32	1	4	91.88
	DenseNet	16	1	8	91.80
	DenseNet	128	4	4	91.70
	VGG/SFV	128	8	8	91.62
	ResNet	128	8	8	91.58
	ResNet	128	4	4	91.49
	ResNet	256	16	8	91.05
	SIFT/SFV+GIST	64	2	4	90.87
	DenseNet	512	32	8	90.68
	VGG/SFV	128	4	4	90.58
	DenseNet	256	16	8	90.48
	SIFT/SFV+GIST	64	4	8	90.29
	VGG/SFV	16	1	8	90.02
	SIFT/SFV+GIST	128	4	4	89.92
	ResNet	16	1	8	89.50
	DenseNet	256	8	4	89.30
	SIFT/SFV+GIST	32	2	8	89.18
	SIFT/SFV+GIST	32	1	4	89.09
	SIFT/SFV+GIST	128	8	8	88.89
	DenseNet	1024	32	4	88.88
	VGG/SFV	256	16	8	88.21
	ResNet	512	32	8	87.49
	VGG/SFV	512	32	8	87.25
	SIFT/SFV+GIST	256	16	8	87.01
	VGG/SFV	256	8	4	86.89
	ResNet	1024	64	8	86.25
	SIFT/SFV+GIST	256	8	4	86.23
	SIFT/SFV+GIST	16	1	8	86.21
	ResNet	256	8	4	86.11
	SIFT/SFV+GIST	512	32	8	85.27
	VGG/SFV	1024	64	8	85.16
	DenseNet	512	16	4	85.00
	VGG/SFV	512	16	4	83.40
	SIFT/SFV+GIST	512	16	4	82.88

	SIFT/SFV+GIST	1024	64	8	82.51
	ResNet	512	16	4	80.74
	VGG/SFV	1024	32	4	75.64
	SIFT/SFV+GIST	1024	32	4	73.29
	ResNet	1024	32	4	71.63
64 bits					
	DenseNet	32	4	8	93.28
	DenseNet	128	16	8	91.96
	DenseNet	64	8	8	91.95
	DenseNet	32	2	4	91.85
	ResNet	32	4	8	91.78
	VGG/SFV	32	2	4	91.41
	DenseNet	1024	128	8	91.40
	DenseNet	16	1	4	91.02
	DenseNet	16	2	8	90.94
	VGG/SFV	64	8	8	90.83
	ResNet	32	2	4	90.63
	ResNet	64	8	8	90.61
	VGG/SFV	32	4	8	90.18
	DenseNet	64	4	4	89.95
	VGG/SFV	16	2	8	89.91
	ResNet	64	4	4	89.83
	ResNet	128	16	8	89.76
	ResNet	16	2	8	89.10
	VGG/SFV	16	1	4	88.99
	ResNet	16	1	4	88.95
	VGG/SFV	64	4	4	88.67
	DenseNet	256	32	8	88.38
	ResNet	256	32	8	88.20
	VGG/SFV	128	16	8	88.12
	SIFT/SFV+GIST	32	4	8	87.90
	SIFT/SFV+GIST	32	2	4	87.76
	DenseNet	512	64	8	87.47
	DenseNet	128	8	4	87.31
	VGG/SFV	256	32	8	86.91
	SIFT/SFV+GIST	64	8	8	86.84
	SIFT/SFV+GIST	64	4	4	86.22
	ResNet	512	64	8	86.03
	ResNet	128	8	4	85.88
	SIFT/SFV+GIST	16	2	8	85.72
	SIFT/SFV+GIST	128	8	4	85.68
	VGG/SFV	512	64	8	85.64
	SIFT/SFV+GIST	128	16	8	85.45
	VGG/SFV	128	8	4	85.18
	SIFT/SFV+GIST	16	1	4	85.00
	SIFT/SFV+GIST	256	32	8	84.77
	DenseNet	256	16	4	84.77
	ResNet	1024	128	8	83.51
	DenseNet	512	32	4	83.26
	DenseNet	1024	64	4	82.89
	SIFT/SFV+GIST	512	64	8	79.91

	SIFT/SFV+GIST	256	16	4	79.86
	VGG/SFV	256	16	4	79.53
	ResNet	256	16	4	79.43
	VGG/SFV	1024	128	8	79.40
	SIFT/SFV+GIST	512	32	4	75.19
	SIFT/SFV+GIST	1024	128	8	73.10
	VGG/SFV	512	32	4	72.14
	ResNet	512	32	4	67.33
	ResNet	1024	64	4	60.34
	VGG/SFV	1024	64	4	53.75
	SIFT/SFV+GIST	1024	64	4	53.25
32 bits					
	DenseNet	64	16	8	90.50
	DenseNet	32	8	8	88.77
	DenseNet	1024	256	8	88.54
	DenseNet	16	4	8	88.41
	ResNet	32	8	8	88.13
	VGG/SFV	32	8	8	87.89
	DenseNet	16	2	4	87.76
	DenseNet	32	4	4	87.50
	ResNet	64	16	8	87.29
	VGG/SFV	16	4	8	87.13
	ResNet	16	4	8	86.90
	ResNet	32	4	4	86.67
	DenseNet	256	64	8	86.54
	VGG/SFV	64	16	8	86.31
	DenseNet	128	32	8	86.28
	ResNet	256	64	8	86.00
	ResNet	16	2	4	85.96
	DenseNet	512	128	8	85.75
	ResNet	128	32	8	85.42
	VGG/SFV	64	8	4	85.08
	VGG/SFV	32	4	4	84.83
	DenseNet	128	16	4	84.74
	DenseNet	64	8	4	84.44
	VGG/SFV	16	2	4	84.37
	VGG/SFV	128	32	8	83.53
	ResNet	512	128	8	83.24
	SIFT/SFV+GIST	32	4	4	83.06
	VGG/SFV	256	64	8	82.98
	ResNet	64	8	4	82.77
	SIFT/SFV+GIST	16	4	8	82.15
	SIFT/SFV+GIST	128	32	8	81.76
	SIFT/SFV+GIST	64	8	4	81.09
	SIFT/SFV+GIST	64	16	8	81.07
	DenseNet	256	32	4	81.03
	SIFT/SFV+GIST	16	2	4	79.86
	SIFT/SFV+GIST	512	128	8	79.82
	SIFT/SFV+GIST	32	8	8	79.29
	ResNet	1024	256	8	78.90
	SIFT/SFV+GIST	256	64	8	77.98

DenseNet	1024	128	4	77.27
SIFT/SFV+GIST	128	16	4	76.65
VGG/SFV	512	128	8	76.57
VGG/SFV	128	16	4	75.84
ResNet	128	16	4	74.97
DenseNet	512	64	4	73.22
VGG/SFV	1024	256	8	71.96
VGG/SFV	256	32	4	70.29
SIFT/SFV+GIST	1024	256	8	69.51
ResNet	256	32	4	68.41
SIFT/SFV+GIST	256	32	4	64.74
ResNet	512	64	4	60.54
SIFT/SFV+GIST	512	64	4	56.72
ResNet	1024	128	4	55.07
VGG/SFV	512	64	4	52.67
VGG/SFV	1024	128	4	51.71
SIFT/SFV+GIST	1024	128	4	46.95

Bibliography

[1] Ranft, B., Stiller, C., “The role of machine vision for intelligent vehicles”, IEEE Transactions on Intelligent Vehicles, 2016.

[2] Quddus, M. A., Ochieng, W. Y., Noland, R. B., “Current map-matching algorithms for transport applications: State-of-the art and future research directions”, Transportation Research Part C: Emerging Technologies, Vol. 15, 2007.

[3] Bernstein, D., Kornhauser, A. L., “An introduction to map matching for personal navigation assistants”, New Jersey TIDE Center, 1996.

[4] White, C. E., Bernstein, D., Kornhauser, A. L., “Some map matching algorithms for personal navigation assistants”, Transportation Research Part C: Emerging Technologies, Vol. 8, no. 1, 2000, pp. 91 - 108, available at: http://www.sciencedirect.com/science/article/pii/S0968090X00000267

[5] Phuyal, B. P., “Method and use of aggregated dead reckoning sensor and GPS data for map matching”, Proceedings of the 15th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GPS 2002), 2002, pp. 430 - 437.

[6] Greenfeld, J. S., “Matching GPS observations to locations on a digital map”, in Proceedings of the 81st Annual Meeting of the Transportation Research Board, 2002.

[7] Blazquez, C., Vonderohe, A., “Simple map-matching algorithm applied to intelligent winter maintenance vehicle data”, Transportation Research Record, Vol. 1935, 01 2005, pp. 68-76.

[8] Zhao, Y., Vehicle Location and Navigation Systems. Artech House, Inc., 1997.

[9] Ochieng, W. Y., Quddus, M. A., Robert, N. B., “Map-matching in complex urban road networks”, Brazilian Journal of Cartography, Vol. 55, 2013, pp. 1 - 14.

[10] Yang, M.-S., Moon, W. M., “Decision level fusion of multi-frequency polarimetric SAR and optical data with Dempster-Shafer evidence theory”, in Proceedings of 2003 IEEE

International Geoscience and Remote Sensing Symposium (IGARSS), Vol. 6, July 2003, pp. 3668-3670 vol.6.

[11] Syed, S., Cannon, M. E., “Fuzzy logic based-map matching algorithm for vehicle navigation system in urban canyons”, in Proceedings of ION NTM, 2004.

[12] Fu, M., Li, J., Wang, M., “A hybrid map matching algorithm based on fuzzy comprehensive judgment”, in Proceedings of The 7th International IEEE Conference on Intelligent Transportation Systems, Oct 2004, pp. 613-617.

[13] Newson, P., Krumm, J., “Hidden Markov map matching through noise and sparseness”, in Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, 2009, pp. 336–343, available at: http://doi.acm.org/10.1145/1653771.1653818

[14] Mohamed, R., Aly, H., Youssef, M., “Accurate and efficient map matching for challenging environments”, in Proceedings of the 22Nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, 2014, pp. 401–404, available at: http://doi.acm.org/10.1145/2666310.2666429

[15] Griffin, T., Huang, Y., Seals, S., “Routing-based map matching for extracting routes from GPS trajectories”, in Proceedings of the 2Nd International Conference on Computing for Geospatial Research & Applications, 2011, pp. 24:1–24:6, available at: http://doi.acm.org/10.1145/1999320.1999344

[16] Atia, M. M., Hilal, A. R., Stellings, C., Hartwell, E., Toonstra, J., Miners, W. B., Basir, O. A., “A low-cost lane-determination system using GNSS/IMU fusion and HMM-based multistage map matching”, IEEE Transactions on Intelligent Transportation Systems, Vol. 18, no. 11, Nov 2017, pp. 3027-3037.

[17] Taguchi, S., Koide, S., Yoshimura, T., “Online map matching with route prediction”, IEEE Transactions on Intelligent Transportation Systems, 2018, pp. 1-10.

[18] Csurka, G., Dance, C. R., Fan, L., Willamowski, J., Bray, C., “Visual categorization with bags of keypoints”, in Workshop on Statistical Learning in Computer Vision, ECCV, 2004.

[19] Lowe, D. G., “Object recognition from local scale-invariant features”, in Proceedings of the Seventh IEEE International Conference on Computer Vision, Vol. 2, Sep. 1999, pp. 1150-1157 vol.2.

[20] Lowe, D. G., “Distinctive image features from scale-invariant keypoints”, Int. J. Comput. Vision, Vol. 60, no. 2, Nov. 2004, pp. 91–110.

[21] Ojala, T., Pietikainen, M., Maenpaa, T., “Multiresolution gray-scale and rotation invariant texture classification with local binary patterns”, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 24, no. 7, July 2002, pp. 971-987.

[22] Matas, J., Chum, O., Urban, M., Pajdla, T., “Robust wide baseline stereo from maximally stable extremal regions.”, in BMVC, Rosin, P. L., Marshall, A. D., Eds. British Machine Vision Association, 2002, available at: http://dblp.uni-trier.de/db/conf/bmvc/bmvc2002.html#MatasCUP02

[23] Bay, H., Ess, A., Tuytelaars, T., Van Gool, L., “Speeded-up robust features (SURF)”, Comput. Vis. Image Underst., Vol. 110, no. 3, Jun. 2008, pp. 346–359, available at: http://dx.doi.org/10.1016/j.cviu.2007.09.014

[24] Leutenegger, S., Chli, M., Siegwart, R. Y., “BRISK: Binary robust invariant scalable keypoints”, in 2011 International Conference on Computer Vision, Nov 2011, pp. 2548-2555.

[25] Ortiz, R., “FREAK: Fast retina keypoint”, in Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), ser. CVPR ’12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 510–517, available at: http://dl.acm.org/citation.cfm?id=2354409.2354903

[26] Chatfield, K., Lempitsky, V., Vedaldi, A., Zisserman, A., “The devil is in the details: an evaluation of recent feature encoding methods”, in Proc. BMVC, 2011.

[27] Lazebnik, S., Schmid, C., Ponce, J., “Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories”, in Proc. CVPR. Washington, DC, USA: IEEE Computer Society, 2006, pp. 2169–2178.

[28] Krapac, J., Verbeek, J. J., Jurie, F., “Modeling spatial layout with Fisher vectors for image categorization”, in Proc. ICCV, 2011.

[29] Cortes, C., Vapnik, V., “Support-vector networks”, Mach. Learn., Vol. 20, no. 3, Sep. 1995, pp. 273–297, available at: https://doi.org/10.1023/A:1022627411411

[30] Widrow, B., Lehr, M., “30 years of adaptive neural networks: perceptron, Madaline and backpropagation”, Proceedings of the IEEE, Vol. 78, no. 9, 1990, pp. 1415–1442.

[31] Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J., “Gradient flow in recurrent nets: the difficulty of learning long-term dependencies”, in A Field Guide to Dynamical Recurrent Neural Networks. IEEE Press, 2001.

[32] Rumelhart, D. E., Hinton, G. E., Williams, R. J., “Learning representations by back-propagating errors”, Nature, Vol. 323, no. 6088, Oct. 1986, pp. 533–536.

[33] Hochreiter, S., “Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München”, 1991.

[34] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., Jackel, L. D., “Backpropagation applied to handwritten zip code recognition”, Neural Computation, Vol. 1, no. 4, Dec 1989, pp. 541-551.

[35] Rowley, H. A., Baluja, S., Kanade, T., “Neural network-based face detection”, in Proceedings CVPR IEEE Computer Society Conference on Computer Vision and Pattern Recognition, June 1996, pp. 203-208.

[36] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., “Gradient-based learning applied to document recognition”, Proceedings of the IEEE, Vol. 86, no. 11, November 1998, pp. 2278-2324.

[37] Hahnloser, R. H. R., Sarpeshkar, R., Mahowald, M. A., Douglas, R. J., Seung, H. S., “Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit.”, Nature, Vol. 405 6789, 2000, pp. 947-51.

[38] Krizhevsky, A., Sutskever, I., Hinton, G. E., “Imagenet classification with deep convolutional neural networks”, in Advances in Neural Information Processing Systems 25. Curran Associates, Inc., 2012, pp. 1097–1105.

[39] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., Salakhutdinov, R., “Improving neural networks by preventing co-adaptation of feature detectors”, CoRR, Vol. abs/1207.0580, 2012.

[40] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., Fei-Fei, L., “ImageNet large scale visual recognition challenge”, International Journal of Computer Vision (IJCV), Vol. 115, no. 3, 2015, pp. 211-252.

[41] Zeiler, M. D., Fergus, R., “Visualizing and understanding convolutional networks”, in Computer Vision – ECCV 2014. Cham: Springer International Publishing, 2014, pp. 818–833.

[42] Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., Lecun, Y., “Overfeat: Integrated recognition, localization and detection using convolutional networks”, International Conference on Learning Representations (ICLR) (Banff), 12 2013.

[43] Simonyan, K., Zisserman, A., “Very deep convolutional networks for large-scale image recognition”, CoRR, Vol. abs/1409.1556, 2014.

[44] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A., “Going deeper with convolutions”, in Computer Vision and Pattern Recognition (CVPR), 2015, available at: http://arxiv.org/abs/1409.4842

[45] Ioffe, S., Szegedy, C., “Batch normalization: Accelerating deep network training by reducing internal covariate shift”, in Proceedings of the 32nd International Conference on Machine Learning (ICML-15). JMLR Workshop and Conference Proceedings, 2015, pp. 448-456.

[46] Srivastava, R. K., Greff, K., Schmidhuber, J., “Training very deep networks”, in Proceedings of the 28th International Conference on Neural Information Processing Systems, ser. NIPS’15. Cambridge, MA, USA: MIT Press, 2015, pp. 2377–2385.

[47] He, K., Zhang, X., Ren, S., Sun, J., “Deep residual learning for image recognition”, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016.

[48] He, K., Zhang, X., Ren, S., Sun, J., “Identity mappings in deep residual networks”, arXiv preprint arXiv:1603.05027, 2016.

[49] Krizhevsky, A., “Learning multiple layers of features from tiny images”, Canadian Institute for Advanced Research, Tech. Rep., 2009.

[50] Veit, A., Wilber, M. J., Belongie, S. J., “Residual networks behave like ensembles of relatively shallow networks”, in Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems, 2016, pp. 550–558.

[51] Huang, G., Sun, Y., Liu, Z., Sedra, D., Weinberger, K. Q., “Deep networks with stochastic depth.”, in ECCV (4), ser. Lecture Notes in Computer Science, Vol. 9908. Springer, 2016, pp. 646-661.

[52] Huang, G., Liu, Z., van der Maaten, L., Weinberger, K. Q., “Densely connected convolutional networks”, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017.

[53] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H., “MobileNets: Efficient convolutional neural networks for mobile vision applications”, CoRR, Vol. abs/1704.04861, 2017, available at: http://arxiv.org/abs/1704.04861

[54] Sandler, M., Howard, A. G., Zhu, M., Zhmoginov, A., Chen, L., "Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation", CoRR, Vol. abs/1801.04381, 2018, available at: http://arxiv.org/abs/1801.04381

[55] Krizhevsky, A., "Convolutional deep belief networks on CIFAR-10", Unpublished manuscript, 2010.

[56] Oeljeklaus, M., Hoffmann, F., Bertram, T., "A combined recognition and segmentation model for urban traffic scene understanding", in ITSC, 2017.

[57] Di, S., Zhang, H., Mei, X., Prokhorov, D., Ling, H., "A benchmark for cross-weather traffic scene understanding", in ITSC, 2016.

[58] Di, S., Zhang, H., Li, C. G., Mei, X., Prokhorov, D., Ling, H., "Cross-domain traffic scene understanding: A dense correspondence-based transfer learning approach", IEEE Transactions on Intelligent Transportation Systems, 2018.

[59] Hussain, K. F., Afifi, M., Moussa, G., "A comprehensive study of the effect of spatial resolution and color of digital images on vehicle classification", IEEE Transactions on Intelligent Transportation Systems, 2018.

[60] Oquab, M., Bottou, L., Laptev, I., Sivic, J., "Learning and transferring mid-level image representations using convolutional neural networks", in Proceedings of the Computer Vision and Pattern Recognition (CVPR). IEEE, 2014.

[61] Everingham, M., Gool, L., Williams, C. K., Winn, J., Zisserman, A., "The PASCAL visual object classes (VOC) challenge", Int. J. Comput. Vision, Vol. 88, no. 2, Jun. 2010, pp. 303–338.

[62] Barat, C., Ducottet, C., "String representations and distances in deep convolutional neural networks for image classification", Pattern Recognition, Vol. 54, 2016, pp. 104-115, available at: https://hal-ujm.archives-ouvertes.fr/ujm-01274675

[63] Cimpoi, M., Maji, S., Kokkinos, I., Vedaldi, A., "Deep filter banks for texture recognition, description, and segmentation", International Journal of Computer Vision, Vol. 118, no. 1, May 2016, pp. 65–94.

[64] Gong, Y., Wang, L., Guo, R., Lazebnik, S., "Multi-scale orderless pooling of deep convolutional activation features", in Computer Vision – ECCV 2014. Cham: Springer International Publishing, 2014, pp. 392–407.

[65] He, K., Zhang, X., Ren, S., Sun, J., "Spatial pyramid pooling in deep convolutional networks for visual recognition", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 37, no. 9, Sept 2015, pp. 1904-1916.

[66] Girshick, R., Donahue, J., Darrell, T., Malik, J., "Rich feature hierarchies for accurate object detection and semantic segmentation", in Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 580–587, available at: http://dx.doi.org/10.1109/CVPR.2014.81

[67] Athiwaratkun, B., Kang, K., "Feature representation in convolutional neural networks", CoRR, Vol. abs/1507.02313, 2015.

[68] Garcia-Gasulla, D., Parés, F., Vilalta, A., Moreno, J., Ayguadé, E., Labarta, J., Cortés, U., Suzumura, T., "On the behavior of convolutional nets for feature extraction", CoRR, Vol. abs/1703.01127, 2017, available at: http://arxiv.org/abs/1703.01127

[69] Nanni, L., Ghidoni, S., Brahnam, S., "Handcrafted vs non-handcrafted features for computer vision classification", Pattern Recognition, 2017.

[70] McCloskey, M., Cohen, N., "Catastrophic interference in connectionist networks: The sequential learning problem", Psychology of Learning and Motivation - Advances in Research and Theory, Vol. 24, no. C, 1 1989, pp. 109–165.

[71] Srivastava, R. K., Masci, J., Kazerounian, S., Gomez, F., Schmidhuber, J., "Compete to compute", in Advances in Neural Information Processing Systems 26. Curran Associates, Inc., 2013, pp. 2310–2318, available at: http://papers.nips.cc/paper/5059-compete-to-compute.pdf

[72] Goodfellow, I. J., Mirza, M., Xiao, D., Courville, A., Bengio, Y., "An empirical investigation of catastrophic forgetting in gradient-based neural networks", CoRR, 2013.

[73] Hinton, G., Vinyals, O., Dean, J., "Distilling the knowledge in a neural network", in NIPS Deep Learning and Representation Learning Workshop, 2015, available at: http://arxiv.org/abs/1503.02531

[74] Li, Z., Hoiem, D., "Learning without forgetting", CoRR, Vol. abs/1606.09282, 2016.

[75] Shmelkov, K., Schmid, C., Alahari, K., "Incremental learning of object detectors without catastrophic forgetting", in ICCV. IEEE Computer Society, 2017, pp. 3420–3429.

[76] Dosovitskiy, A., Springenberg, J. T., Riedmiller, M., Brox, T., "Discriminative unsupervised feature learning with convolutional neural networks", in Advances in Neural Information Processing Systems 27. Curran Associates, Inc., 2014, pp. 766–774,

available at: http://papers.nips.cc/paper/5548-discriminative-unsupervised-feature-learning-with-convolutional-neural-networks.pdf

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

[78] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y., "Generative adversarial nets", in Advances in Neural Information Processing Systems 27. Curran Associates, Inc., 2014, pp. 2672–2680.

[79] Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A. C., Bengio, Y., "Maxout networks", in ICML (3), ser. JMLR Workshop and Conference Proceedings, Vol. 28. JMLR.org, 2013, pp. 1319–1327.

[80] Radford, A., Metz, L., Chintala, S., "Unsupervised representation learning with deep convolutional generative adversarial networks", CoRR, Vol. abs/1511.06434, 2015, available at: http://arxiv.org/abs/1511.06434

[81] Arjovsky, M., Chintala, S., Bottou, L., "Wasserstein generative adversarial networks", in ICML, ser. Proceedings of Machine Learning Research, Vol. 70. PMLR, 2017, pp. 214–223.

[82] Oliva, A., Torralba, A., "Modeling the shape of the scene: A holistic representation of the spatial envelope", Int. J. Comput. Vision, Vol. 42, no. 3, May 2001, pp. 145–175.

[83] Oliva, A., Torralba, A. B., "Scene-centered description from spatial envelope properties", in Proc. BMCV. London, UK, UK: Springer-Verlag, 2002, pp. 263–272.

[84] Torralba, A., Fergus, R., Weiss, Y., "Small codes and large image databases for recognition", in Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on, June 2008, pp. 1-8.

[85] Shakhnarovich, G., Viola, P., Darrell, T., "Fast pose estimation with parameter-sensitive hashing", in Proceedings of the Ninth IEEE International Conference on Computer Vision - vol. 2, 2003, pp. 750–.

[86] Hinton, G., Salakhutdinov, R., "Reducing the dimensionality of data with neural networks", Science, Vol. 313, no. 5786, 2006, pp. 504 - 507.

[87] Torresani, L., Szummer, M., Fitzgibbon, A., Maragos, P., Paragios, N., Efficient Object Category Recognition Using Classemes. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 776–789.

[88] Bergamo, A., Torresani, L., Fitzgibbon, A. W., "PiCoDes: Learning a compact code for novel-category recognition", in Proc. NIPS, 2011, pp. 2088-2096.

[89] Bergamo, A., Torresani, L., "Meta-class features for large-scale object categorization on a budget", in Proc. CVPR, 2012.

[90] Pearson, K., "On lines and planes of closest fit to systems of points in space", The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, Vol. 2, no. 11, 1901, pp. 559-572.

[91] Jolliffe, I., Principal Component Analysis. Springer Verlag, 1986.

[92] Jegou, H., Douze, M., Schmid, C., "Product quantization for nearest neighbor search", IEEE Trans. Pattern Anal. Mach. Intell., Vol. 33, no. 1, Jan. 2011, pp. 117–128.

[93] Ge, T., He, K., Ke, Q., Sun, J., "Optimized product quantization", IEEE Trans. Pattern Anal. Mach. Intell., Vol. 36, no. 4, 2014, pp. 744–755.

[94] Babenko, A., Slesarev, A., Chigorin, A., Lempitsky, V., "Neural codes for image retrieval", in Computer Vision – ECCV 2014. Cham: Springer International Publishing, 2014, pp. 584–599.

[95] Zhao, F., Huang, Y., Wang, L., Tan, T., "Deep semantic ranking based hashing for multi-label image retrieval", CoRR, Vol. abs/1501.06272, 2015, available at: http://arxiv.org/abs/1501.06272

[96] Krizhevsky, A., "One weird trick for parallelizing convolutional neural networks", CoRR, Vol. abs/1404.5997, 2014, available at: http://arxiv.org/abs/1404.5997

[97] Lin, K., Yang, H. F., Hsiao, J. H., Chen, C. S., "Deep learning of binary hash codes for fast image retrieval", in 2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), June 2015, pp. 27-35.

[98] Liu, H., Wang, R., Shan, S., Chen, X., "Deep supervised hashing for fast image retrieval", in The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2016.

[99] Sikirić, I., Brkić, K., Krapac, J., Šegvić, S., "Image representations on a budget: Traffic scene classification in a restricted bandwidth scenario", Proc. IEEE Intelligent Vehicles Symposium, 2014.

[100] Sikirić, I., Brkić, K., Krapac, J., Šegvić, S., "Robust traffic scene recognition with a limited descriptor length", in Proc. CVPR Workshop on Visual Place Recognition in Changing Environments, 2015.

[101] Lahiri, T., “Every new car in China must have a tracking chip starting next year”, 2018, Online, available at: https://qz.com/1305240/every-new-car-in-china-must-have-a-tracking-chip-from-2019/, accessed: 2018-11-28.

[102] Philips, T., “China orders GPS tracking of every car in troubled region”, 2017, Online, available at: https://www.theguardian.com/world/2017/feb/21/china-orders-gps-tracking-of-every-car-in-troubled-region/, accessed: 2018-11-28.

[103] Dash, D. K., “GPS, panic buttons must on public transport vehicles by Apr 1”, 2018, Online, available at: https://timesofindia.indiatimes.com/india/gps-panic-buttons-must-on-public-transport-vehicles-by-apr-1/articleshow/62546882.cms, accessed: 2018-11-28.

[104] “HU-GO electronic toll system - frequently asked questions”, 2018, Online, available at: https://www.hu-go.hu/documents/document/faq, accessed: 2018-11-28.

[105] Wood, D., “Dash cam sales grow faster than tablets and digital cameras in 2015”, 2016, Online, available at: http://www.businesscar.co.uk/news/2016/dash-cam-sales-grow-faster-than-tablets-and-digital-cameras-in-2015, accessed: 2018-11-28.

[106] Luo, A., Chen, S., Xv, B., “Enhanced map-matching algorithm with a hidden Markov model for mobile phone positioning”, ISPRS Int. J. Geo-Information, Vol. 6, 2017, p. 327.

[107] Bevandić, P., Krešo, I., Oršić, M., Šegvić, S., “Discriminative out-of-distribution detection for semantic segmentation”, CoRR, Vol. abs/1808.07703, 2018, available at: http://arxiv.org/abs/1808.07703

[108] Ignatov, A., Timofte, R., Szczepaniak, P., Chou, W., Wang, K., Wu, M., Hartley, T., Van Gool, L., “AI benchmark: Running deep neural networks on Android smartphones”, arXiv preprint arXiv:1810.01109, 2018.

[109] Moloney, D., Barry, B., Richmond, R., Connor, F., Brick, C., Donohoe, D., “Myriad 2: Eye of the computational vision storm”, in 2014 IEEE Hot Chips 26 Symposium (HCS), Aug 2014, pp. 1-18.

[110] “Myriad 2 vision processor product brief”, 2014, Online, available at: http://uploads.movidius.com/1441734401-Myriad-2-product-brief.pdf, accessed: 2018-12-02.

[111] “Intel NCSM2450.DK1 Movidius neural compute stick”, https://www.amazon.com/Intel-NCSM2450-DK1-Movidius-Neural-Compute/dp/B076751BN8, accessed: 2018-12-02.

[112] Deutsch, P., “DEFLATE compressed data format specification version 1.3”, Tech. Rep., 1996, available at: https://www.ietf.org/rfc/rfc1951.txt

[113] Pavlov, I., “LZMA software development kit”, available at: https://www.7-zip.org/sdk.html 2013.

[114] Seward, J., “Bzip2 and libbzip2: a program and library for data compression”, available at: http://sources.redhat.com/bzip2 1998.

[115] Jaakkola, T. S., Haussler, D., “Exploiting generative models in discriminative classifiers”, in Advances in Neural Information Processing Systems 11, (NIPS), 1998, pp. 487–493, available at: http://papers.nips.cc/paper/1520-exploiting-generative-models-in-discriminative-classifiers

[116] Perronnin, F., Dance, C. R., “Fisher kernels on visual vocabularies for image categorization”, in CVPR, 2007.

[117] Song, Y., Hong, X., McLoughlin, I., Dai, L., “Image classification with CNN-based Fisher vector coding”, in 2016 Visual Communications and Image Processing (VCIP), Nov 2016, pp. 1-4.

[118] Perronnin, F., Larlus, D., “Fisher vectors meet neural networks: A hybrid classification architecture”, in 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2015, pp. 3743-3752.

[119] “IEEE standard for floating-point arithmetic”, IEEE Std 754-2008, Aug 2008, pp. 1-70.

[120] Lloyd, S., “Least squares quantization in PCM”, IEEE Transactions on Information Theory, Vol. 28, no. 2, March 1982, pp. 129-137.

[121] Sikirić, I., Brkić, K., Šegvić, S., “Classifying traffic scenes using the GIST image descriptor”, in Proc. CCVW, September 2013, pp. 1-6.

[122] Liaw, A., Wiener, M., “Classification and regression by randomForest”, R News, Vol. 2, no. 3, 2002, pp. 18-22, available at: https://CRAN.R-project.org/doc/Rnews/

[123] Davis, J., Goadrich, M., “The relationship between precision-recall and ROC curves”, in Proceedings of the 23rd International Conference on Machine Learning, ser. ICML ’06. New York, NY, USA: ACM, 2006, pp. 233–240, available at: http://doi.acm.org/10.1145/1143844.1143874

[124] Saito, T., Rehmsmeier, M., “The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets”, PLOS ONE, Vol. 10, no. 3, 03 2015, pp. 1-21, available at: https://doi.org/10.1371/journal.pone.0118432

[125] Sikirić, I., "The FM3 dataset", 2018, Online, available at: http://www.zemris.fer.hr/~ssegvic/datasets/unizg-fer-fm3am.tar.gz, accessed: 2018-12-02.

[126] "Map data at scale from street-level imagery", https://www.mapillary.com, accessed: 2018-12-02.

[127] Chang, C.-C., Lin, C.-J., "LIBSVM: A library for support vector machines", ACM Transactions on Intelligent Systems and Technology, Vol. 2, 2011, pp. 27:1–27:27, software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[128] He, K., Zhang, X., Ren, S., Sun, J., "Deep residual networks (implementation)", 2016, Online, available at: https://github.com/KaimingHe/deep-residual-networks, accessed: 2018-12-02.

[129] Huang, G., Liu, Z., van der Maaten, L., Weinberger, K. Q., "Densely connected convolutional networks (implementation)", 2018, Online, available at: https://github.com/liuzhuang13/DenseNet, accessed: 2018-12-02.

[130] Sandler, M., Howard, A. G., Zhu, M., Zhmoginov, A., Chen, L., "MobileNetV2 (implementation)", 2018, Online, available at: https://github.com/tensorflow/models/tree/master/research/slim/nets/mobilenet, accessed: 2018-12-02.

[131] Radford, A., Metz, L., Chintala, S., "Unsupervised representation learning with deep convolutional generative adversarial networks (implementation)", 2016, Online, available at: https://github.com/Newmu/dcgan_code/tree/master/, accessed: 2018-12-02.

[132] Ge, T., He, K., Ke, Q., Sun, J., "Optimized product quantization (implementation)", 2013, Online, available at: http://kaiminghe.com/cvpr13/matlab_OPQ_release_v1.1.rar, accessed: 2018-12-02.

[133] Pavlov, I., "7-Zip file archiver", available at: https://www.7-zip.org/ Accessed: 2018-12-02.

List of Figures

List of Tables

Biography

Ivan Sikirić was born in Zadar, Croatia in 1984. His primary education was completed at Stjepan Radić elementary school in Bibinje, while his high school education was completed at Gymnasium Franjo Petrić in Zadar. During his high school education he participated in international olympiads in informatics, as well as mathematics, winning a total of 1 bronze and 4 silver medals. He received the Zadar City Council award in 2002. In 2002 he enrolled in the undergraduate study program in computing at the Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia. He graduated in 2007, receiving the title of Master of Engineering in Computer Science (dipl. ing.), with a diploma thesis titled "Collocation extraction optimization using evolution programming". Since 2008 he is employed at Mireo, where he works as a software developer, primarily on the Mireo fleet management system. The same year he enrolled in the postgraduate program in computing at the Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia. During the course of his postgraduate study, he co-authored one paper published in an international scientific journal, one accepted for publishing in an international scientific journal, and ten papers in the proceedings of international conferences. His professional and scientific interests include computer vision, fleet management, navigation and intelligent transportation systems.

List of publications

1. Sikirić, I., Brkić, K., Bevandić, P., Krešo, I., Krapac, J., Siniša, Š., "Traffic Scene Classification on a Representation Budget", IEEE Transactions on Intelligent Transportation Systems (accepted for publication)
2. Brkić, K., Hrkać, T., Kalafatić, Z., Sikirić, I., "Face, Hairstyle and Clothing Colour De-identification in Video Sequences", IET signal processing, 2017, pp. 1062-1068
3. Brkić, K., Sikirić, I., Hrkać, T., Kalafatić, Z., "I Know that Person: Generative Full Body and Face De-identification of People in Images", Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2017, pp. 1319-1328
4. Brkić, K., Sikirić, I., Hrkiać, T., Kalafatić, Z., "De-identifying People in Videos Using Neural Art", Sixth International Conference on Image Processing Theory, Tools and Ap-

plications (IPTA), 2016, pp. 1-6

5. Brkić, K., Hrkać, T., Sikirić, I., Kalafatić, Z., "Towards Neural Art-based Face De-identification in Video Data", First International Workshop on Sensing, Processing and Learning for Intelligent Machines (SPLINE), IEEE, 2016, pp. 11-15
6. Sikirić, I., Brkić, K., Krapac, J., Šegvić, S., "Robust Traffic Scene Recognition with a Limited Descriptor Length", Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2015, pp. 1-9
7. Sikirić, I., Brkić, K., Horvatin, I., Šegvić, S., "Multi-Label Classification of Traffic Scenes", Croatian Computer Vision Workshop (CCVW), 2014, pp. 9-14
8. Sikirić, I., Brkić, K., Krapac, J., Šegvić, S., "Image Representations on a Budget: Traffic Scene Classification in a Restricted Bandwidth Scenario", IEEE Intelligent Vehicles Symposium (IV), 2014, pp. 845-852
9. Sikirić, I., Brkić, K., Šegvić, S., "Classifying Traffic Scenes Using The GIST Image Descriptor", Croatian Computer Vision Workshop (CCVW), 2013, pp. 1-6
10. Sikirić, I., Majić, A., Šegvić, S., "Using GPS Positioning to Recover a Comprehensive Road Appearance Mosaic", International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2011, pp. 270-275
11. Brkić, K., Šegvić, S., Kalafatić, Z., Sikirić, I., Pinz, A., "Generative Modeling of Spatio-temporal Traffic Sign Trajectories", Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2010, pp. 25-31
12. Sikirić, I., Brkić, K., Šegvić, S., "Recovering a Comprehensive Road Appearance Mosaic From Video", 33rd International Convention of Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2010, pp. 242-246
13. Šnajder, J., Dalbelo Bašić, B., Petrović, S., Sikirić, I., "Evolving New Lexical Association Measures Using Genetic Programming", ACL 2008, 46th Annual Meeting of the Association for Computational Linguistics, 2008, pp. 181-184

Životopis

Ivan Sikirić rođen je u Zadru 1984. godine. Osnovnu školu Stjepana Radića pohađa u Bibinju, a Gimnaziju Franje Petrića u Zadru. U sklopu srednjoškolskog obrazovanja sudjeluje na međunarodnim olimpijadama iz informatike i matematike, gdje je osvojio ukupno jednu brončanu i četiri srebrne medalje. 2002. godine mu je dodijeljena nagrada Grada Zadra. Diplomski studij upisao je 2002. godine na Fakultetu elektrotehnike i računarstva Sveučilišta u Zagrebu. Diplomirao je 2007. godine te stekao zvanje diplomiranog inženjera računarstva. Diplomski rad mu nosi naslov "Primjena evolucijskog programiranja na nalaženje optimalnih mjera za ekstrakciju kolokacija iz teksta". Od 2008. godine zaposlenik je tvrtke Mireo, gdje radi kao softverski inženjer, primarno na razvoju sustava za upravljanje voznim parkom. Iste godine upisao je poslijediplomski doktorski studij računarstva na Fakultetu elektrotehnike i računarstva. Tijekom poslijediplomskog studija objavio je jedan rad u međunarodnom časopisu, jedan rad prihvaćen za publikaciju u međunarodnom časopisu, te deset radova u zbornicima međunarodnih znanstvenih skupova. Njegovi stručni i znanstveni interesi uključuju računalni vid, sustave za upravljanje voznim parkom, navigaciju i inteligentne transportne sustave.