Óbuda University

PhD Thesis

Multi-Directional Image Projections with Fixed Resolution for Object Recognition and Matching

Gábor Kertész

Supervisors:

Dr. Zoltán Vámossy

Dr. habil. Sándor Szénási

Doctoral School of Applied Informatics and Applied Mathematics

Budapest, 2019.

Date of Public Defense:
October 17th, 2019

Declaration

I, the undersigned, **Gábor Kertész**, hereby state and declare that this Ph.D. thesis represents my own work and is the result of my own original research. I only used the sources listed in the references. All parts taken from other works, either as word for word citation or rewritten keeping the original meaning, have been unambiguously marked, and reference to the source was included.

Nyilatkozat

Alulírott **Kertész Gábor** kijelentem, hogy ez a doktori értekezés a saját munkámat mutatja be, és a saját eredeti kutatásom eredménye. Csak a hivatkozásokban felsorolt forrásokat használtam fel, minden más munkából származó rész, a szó szerinti, vagy az eredeti jelentést megtartó átiratok egyértelműen jelölve, és forráshivatkozva lettek.

Gábor Kertész

Contents

Acknowledgments

First of all, I would like to thank my doctoral supervisors, *Dr. Zoltán Vámossy* and *Dr. habil. Sándor Szénási* for their valuable, ongoing support during my doctoral studies. As a result of their professional leadership and the many years of their dedicated work I became the researcher and lecturer that I am today. I am immensely grateful for the guidance and encouragement they have given me along this path.

I am thankful to the members of the committee, especially *Prof. Dr. Péter Nagy*, *Dr. Kálmán Palágyi*, *Dr. László Csink*, and *Dr. Adrienn Dineva*, for sacrificing time and providing feedback during the review of the dissertation.

Special thanks to the staff of the John von Neumann Faculty of Informatics; in particular *Prof. Dr. Dezső Sima*, *Dr. habil. András Molnár*, *Dr. habil. Imre Felde* and *Dr. Szabolcs Sergyán* for providing professional background and an inspiring environment. Thanks to my colleague *Dániel Kiss* for his regular professional help.

Thanks to the Doctoral School of Applied Informatics and Applied Mathematics, especially *Prof. Dr. Aurél Galántai* and *Prof. Dr. László Horváth* for providing regular professional feedback and the opportunity to attend international conferences.

I am grateful to my colleagues at MTA SZTAKI for their patience when I finalized the dissertation.

I am thankful to *Dr. Éva Nagyné Hajnal* and *Dr. Márta Seebauer*, for considering me capable, and starting me on this path.

I would like to thank my family, especially my parents, for their support and encouragement in helping me achieve my goals throughout my life. Without their selfless financial and spiritual support, my dissertation would not have been possible. Thank you for the patience of my friends during the writing of the dissertation.

Balázs, *Csaba*, *István* and *Zsolt* provided much useful advice during our professional discussions. Thank you for the inspiration.

I am grateful to my current and former students for their hard work, which resulted in presented and awarded studies at the National Scientific Students' Associations Conference.

I am grateful to *János*, who, many years ago, directed me to the teaching profession – a turning point in my life.

Finally, I owe special thanks to *Gina* – this dissertation would not have been completed without her understanding, patience, and support.

Köszönetnyilvánítás

Szeretnék köszönetet mondani doktori témavezetőimnek, *Dr. Vámossy Zoltán*nak és *Dr. habil. Szénási Sándor*nak a tanulmányaim során nyújtott folyamatos, értékes segítségükért. Szakmai vezetésük, több éves áldozatos tevékenységük eredményeként váltam azzá a kutatóvá és oktatóvá, aki ma vagyok. Mérhetetlenül hálás vagyok az iránymutatásért és ösztönzésért, amellyel végigsegítettek ezen az úton.

Köszönöm a bizottság tagjainak, kiemelten *Prof. Dr. Nagy Péter*nek, *Dr. Palágyi Kálmán*nak, *Dr. Csink László*nak és *Dr. Dineva Adrienn*-nek hogy idejüket áldozták és tanácsokkal láttak el a disszertáció áttekintése során.

Külön köszönettel tartozom munkatársaimnak az Óbudai Egyetem Neumann János Informatikai Karán, elsősorban *Prof. Dr. Sima Dezső*nek, *Dr. habil. Molnár András*nak, *Dr. habil. Felde Imré*nek és *Dr. Sergyán Szabolcs*nak köszönöm a professzionális szakmai háttér biztosítását és az inspiráló környezetet. Köszönöm *Kiss Dániel* kollégámnak a rendszeres szakmai segítséget.

Köszönet az Alkalmazott Informatikai és Matematikai Doktori Iskolának, különösen *Prof. Dr. Galántai Aurél*nak és *Prof. Dr. Horváth László*nak a rendszeres szakmai visszajelzést és a nemzetközi konferenciákon való részvételi lehetőség biztosítását.

Hálás vagyok az MTA SZTAKInál dolgozó kollégáimnak a türelmükért, amelyet a disszertáció befejezésekor tanúsítottak.

Köszönöm *Dr. Nagyné Hajnal Évá*nak és *Dr. Seebauer Mártá*nak, hogy évekkel ezelőtt rátermettnek tartottak, és elindítottak ezen az úton.

Köszönettel tartozom családomnak, elsősorban a szüleimnek a támogatásukért és ösztönzésükért, amellyel egész életemben segítettek a céljaim elérésében. Önzetlen anyagi és lelki támogatásuk nélkül a dolgozatom nem jöhetett volna létre. Köszönöm barátaim türelmét, amelyet a disszertáció megírása során tanúsítottak.

Számtalan hasznos tanáccsal látott el szakmai diszkusszióink során *Balázs*, *Csaba*, *István* és *Zsolt*, köszönöm az inspirációt.

Hálás vagyok a jelenlegi és egykori hallgatóimnak a lelkiismeretes munkájukért, melynek eredményeképp Tudományos Diákköri Konferencián bemutatott és díjazott pályamunkák, valamint közös publikációk születtek.

Hálával tartozom *János*nak, aki hosszú évekkel ezelőtt az oktatói pályára terelt, amely fordulópont volt az életemben.

Végül különleges köszönettel tartozom *Ginának* – ez a disszertáció nem készült volna el megértése, türelme és támogatása nélkül.

Multi-Directional Image Projections with Fixed Resolution for Object Recognition and Matching

Gábor Kertész

Abstract

The use of computer vision and image processing in traffic analysis and control has shown significant growth in recent years. In addition to traditional solutions (such as traffic counting, accident detection), multi-camera applications based on vehicle identification and tracking have also emerged.

Innovation is supported by the explosive growth of image classification efficiency in recent years, which is the result of machine learning, in particular, the rise of deep learning.

This dissertation introduces solutions for multi-camera vehicle tracking to recognize and match vehicles through multiple with low-resolution, low-quality camera images.

The first group of theses deal with multi-directional image projections. I have developed a method for mapping two-dimensional image functions to one-dimensional projection vectors similar to the Radon transformation. The novelty of the solution is the fixed bin number, which is independent of the projection angle and specified as the input to the procedure. The memory cost of the resulting projection matrix is independent of the size of the input image.

I have designed and implemented a parallel version of the method in a GPU environment, taking into account the specific architecture and properties of the graphics processors. Runtime is linear and speedup is proportional to the number of available processing units.

I have compared the defined multi-directional image projection method with lower dimensional procedures in case of object matching. After evaluating the results, I have found that by applying the method, accuracy increased significantly and by using a fixed number of bins, the efficiency improved as well.

In the second group of theses, I present my results related to object matching based on machine learning solutions. In accordance with the state-of-the-art solutions, I have designed a method based on a Siamese convolutional neural network, which is capable of handling image projections as input. For a comprehensive measurement, it was first necessary to review possible network architectures and develop a method that generates multiple Convolutional neural network architectures based on input matrix dimensions. Based on the known convolutional design patterns, I have developed an optimization method based on backtracking search for Neural Architecture Generation. Based on the given input size and the number of hidden layers, the algorithm makes suggestions for the architecture of the head parts of the Siamese structured neural network, having the memory cost of the parameters taken into account.

The resulting architectures can be trained in parallel on a cluster of multiple workstations with GPUs. For the implementation, I designed and implemented a Master/Worker method, where the scheduler optimizes parallel efficiency based on

the parameters of the architectures. Based on the measurement results, I found that the acceleration is approximately equal to the number of workstations involved.

During the investigations, I analyzed the effects of the application of different projection methods as input to the Siamese networks and compared the results to the classic image input based comparators. When evaluating the results, I concluded that the projection methods are suitable for object matching and the method based on the fixed number of bins is Pareto optimal in terms of pairing efficiency and memory cost.

Többirányú, rögzített felbontású képi vetületek objektumok felismerésére és párosítására

Kertész Gábor

Kivonat

A gépi látás és képfeldolgozás használata a forgalomelemzés és -irányítás területén az elmúlt években jelentős növekedésnek indult. A hagyományos megoldások (például forgalomszámlálás, balesetészlelés) mellett a járművek azonosításán és követésén alapuló több-kamerás alkalmazások is megjelentek.

Az innovációt támogatja a képi klasszifikáció hatékonyságának elmúlt években tapasztalt robbanásszerű növekedése, amely a gépi tanulás, azon belül is a mélytanulás megjelenésének eredménye.

Jelen disszertáció több-kamerás járműkövetés témakörében mutat be megoldásokat a gépjárművek azonosítására és több képen keresztüli párosítására alacsony felbontású, gyenge minőségű kameraképek esetén.

Az első téziscsoport a többirányú képi vetületek témakörével foglalkozik. Kidolgoztam egy módszert, amely a Radon transzformációhoz hasonlóan alkalmas kétdimenziós képi függvények egydimenziós vetületi vektorokra való leképezésére. A megoldás újdonsága a vetületi szögtől független fix rekeszszám, amely az eljárás bemeneteként megadható. Az így készített vetületi mátrix memóriaköltsége független a bemeneti kép méretétől.

Megterveztem és implementáltam a módszer párhuzamos változatát GPU környezetben, figyelembe véve a grafikus processzorok sajátos felépítését és tulajdonságait. A futásidő lineáris, a gyorsulás az elérhető műveletvégzők számával arányos.

A kidolgozott többirányú képi vetületi módszert összevetettem alacsonyabb dimenziójú eljárásokkal egyezésvizsgálat esetén. Az eredmények kiértékelése után megállapítottam hogy a módszer alkalmazásával jelentősen nőtt a pontosság, illetve a fix rekeszszámnak köszönhetően a hatékonyság is is javult.

A második téziscsoportban az egyezésvizsgálat gépi tanuláson alapuló megoldásával kapcsolatos eredményeim mutatom be. A korszerű megoldásoknak megfelelően, egy sziámi konvolúciós neurális hálózatra épülő módszert terveztem, amely alkalmas akár képi vetületet bemeneteként fogadni. Az átfogó méréshez elöször szükséges volt áttekinteni a lehetséges hálózati architektúrákat, és kidolgozni egy módszert amely többféle konvolúciós neurális hálózati architektúrát generál a bemeneti mátrix dimenzióihoz igazodva. Az ismert konvolúciós tervezési minták alapján kidolgoztam egy visszalépéses keresésen alapuló optimalizációs módszert neurális architektúra generálásra. Az algoritmus az adott bemeneti méret és a rejtett rétegek száma alapján tesz javaslatokat a sziámi szerkezetű neurális hálózat fej-részeinek architektúrájára, figyelembe véve a paraméterek memória-költségét.

Az így előállított architektúrák tanítása párhuzamosan is történhet, több, GPU-val felszerelt munkaállomásból álló klaszterben. A megvalósításhoz egy Master/Worker elvű módszert terveztem és valósítottam meg, ahol az ütemező az architektúrák paraméterei alapján optimalizálja a párhuzamos hatékonyságot. A mérési eredmények alapján megállapítottam hogy a gyorsítás megközelítőleg egyezik a bevont munkaállomások számával.

A vizsgálatok során különféle vetületi módszerek alkalmazásának hatásait vizsgáltam a sziámi hálózatok bemeneteként, és az eredményeket összehasonlítottam a hagyományos képi bemenetű komparátorokra. Az eredmények kiértékelésekor megállapítottam, hogy a vetületi módszerek alkalmasak objektum párosításra, valamint a fix rekesszámú módszer Pareto optimális a párosítás hatékonysága és memória költsége szempontjából.

Abbreviations & notations

In this study the following notations and abbreviations were used:

Abbrevations

PCC	Pearson Correlation Coefficient
CNN	Convolutional Neural Network
SNN	Siamese Neural Network
GPU	Graphical Processing Unit
FCN	Fully Convolutional Network
FC	Fully Connected (layers)
MDIPFL	Multi-directional Image Projections with Fixed Length
LPT	Longest Processing Times

Projections

$\mathbf{I}$	image matrix
$\mathbf{I}_{i,j}$	pixel value of image on coordinate (i,j)
$N \times M$	matrix size
$\boldsymbol{\pi}_H$, $\boldsymbol{\pi}_V$, $\boldsymbol{\pi}_D$, $\boldsymbol{\pi}_A$	horizontal, vertical, diagonal, antidiagonal projection vectors
$\mathbf{S_2}, \mathbf{S_4}$	two- and four-dimensional projection-based signatures
σ	standard deviation
$\mathrm{cov}(\cdot,\cdot)$	covariance between two vectors
$\rho_H, \rho_V, \rho_D, \rho_A$	correlation coefficient of horizontal, vertical, diagonal vectors
$r(\cdot)$	rectified value of scalar
μ	similarity
$\check{f}(p,\phi) = \mathcal{R}f$	Radon transform of function f
p	line of summarization
α, ϕ	rotational angle
LL, HL	projection segments around point P
S	number of bins
res	resolution
$start$, end	position of projection on segment $LL + HL$
P_H^+	set of ρ_H correlation coefficients for true pairs
P_H^-	set of ρ_H correlation coefficients for false pairs

Neural networks

O_W, O_H	width and height of output
I_W, I_H	width and height of input
F_W, F_H	width and height of filter
P_W, P_H	horizontal and vertical padding
S_W, S_H	horizontal and vertical stride
F_{num}	number of filters
$F_{num}^{(i)}$	number of filters in layer i
L_{num}	number of convolutional and pooling layer pairs
N_{num}	number of neurons
$N_{num}^{(i)}$	number of neurons in layer i
$x \in X$	observations
$c \in C$	categories
$f(x) = c$	mapping of observations to categories
$S(\cdot, \cdot)$	predicted similarity of inputs
$m \in M$	models
$f_{err}(\cdot)$	error rate of model
$f_{memory}(\cdot)$	parameter number of model
$m_1 \succ m_2$	m_1 Pareto dominates m_2

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Equations are just the boring part of mathematics.
I attempt to see things in terms of geometry.

— Stephen Hawking

Closed-circuit television cameras – also known as surveillance cameras – are often applied to monitor traffic. Based on the use cases, several applications of these traffic cameras are known: congestion-detection and accident-detection systems are popular, speed cameras, safety and various enforcement solutions as well [1]. Most of these solutions require the device to be able to identify or track the vehicle, which could be challenging depending on the brightness and weather conditions. Advanced devices have high-resolution cameras with infrared LEDs for night vision [2], also PTZ[1] cameras can be used to track object movement.

On distant locations like public roads, highways, bridges and tunnels simple static cameras are placed with non-overlapping fields of view. These camera-networks are mostly used to measure traffic after crossroads and calculate the average speed of vehicles based on the distance between the cameras and the time of observations [3].

Recognizing a vehicle by using the plate number is not always feasible [4]. In tunnels, where natural light is rare and colors are hard to detect, the usage of such high-level devices is not cost-efficient, giving similar low-quality images as other, less expensive cameras. The changes in lighting and vehicle movement can cause difficulties when matching the image representations, as well as different camera settings could reflect in errors.

[1]pan-tilt-zoom

In computer vision, there are simple methods to find objects with specific colors or shapes [5]. Most methods are based on lines or corners, or other keypoint-based descriptors extracted from template images. In the case of noisy and low-quality pictures, low-level techniques can be applied, for example, template matching [K1] and histogram- or image projection-based comparison [6].

The improvement of the projection-based method can be done by increasing the number of projection angles. There are some well-known methods that provide a mapping from two-dimensional data to one-dimensional projections, like the Radon transform [7] [8]. This, however, in the case of a rectangular structure, gives different projection lengths for different angles [K2], affecting in the result matrix of the Radon transform (the sinogram) as blank areas.

In Chapter 2, a method that generates the multi-directional projections of a squared input is introduced. This transform has a similar result as the application of the Radon transform; however, the negative effects are removed with a fixed vector length. To demonstrate the robustness of this algorithm, a data-parallel implementation is also described and evaluated in Chapter 2, Section 2.2.

In the following section (Section 2.3), this method is used as an input for object matching in the case of vehicle images. The precision rate, memory efficiency and computation cost of the method is compared with the rates given by previously studied techniques.

As a second thesis group, in Chapter 3, a machine learning-based solution is designed and developed for the same problem, based on Siamese-architectured convolutional neural networks. To analyze the applicability of the projection-based neural comparison, a novel method to generate convolutional neural network (CNN) architectures is proposed (Section 3.2). A distributed environment with high parallel efficiency is designed and implemented (Section 3.3), where the networks with the generated architectures are trained and evaluated (Section 3.4).

1.1 Object matching

Richard Szeliski [9] declares object recognition as one of the most challenging tasks of computer vision. By defining the actual task, the problem can be object detection, instance or category recognition. If the size and visual representation of the sought object or objects are known, and the task is to locate these on the input image, the task is object detection. The most popular applications for object detection can be observed in digital cameras, smartphones or even on social network sites for face detection on images.

In the case of re-recognizing a previously examined object, the task is referred to as instance recognition. The process of determining that two visual objects represent the same entity is object matching [9].

To match object representations, a number of methods were introduced during the history of computer vision. The early approaches are summarized in the paper of Joseph L. Mundy [10], concluding that the basic detection methods, such as template matching and basic signal processing, were used to detect and match objects. Mundy also states that some of the early ideas were revisited later in the 1990s.

Modern approaches for object matching were done using geometric object descriptors, lines, edges, corners and other interesting keypoints.

In the following sections some of these methods are presented.

1.1.1 Color-based segmentation and matching

A simple solution for object segmentation is based on colors. If the foreground color of the object is different from the colors of the environment and the background, object boundaries can be easily marked.

These color-based segmentation methods [5] can be implemented based on the RGB[2] representation of colors; however, the HSI[3] intensity-based model is a proper choice for these tasks.

After the object is separated, multiple procedures can follow to match the object of interest with a reference. Based on the color information, the color histograms of the two can be compared [11]. It is worth mentioning, that although the results for true pairs from the same collection of images are satisfactory, changes in lightning or small color changes can significantly change the measured distance between the two histograms. Another disadvantage of this matching method is that the shape and texture of the object is ignored [12], and as different images can have very similar color histograms, in these cases the defined similarity would result in a false positive match.

It is notable, that segmentation could also be done based on intensity, texture [13] or edges [14]. Modern approaches are based on trainable convolutional neural networks. Examples for semantic segmentation are fully-convolutional networks [15] or to handle the computational expenses the U-Net structure [16] is often used.

If segmentation is done as a preprocessing step, beyond color data, distance functions based on the shape of the segmented object can be applied to match the representation with others. Such a method is given as the Edge Histogram Descriptor [17] where the relative distribution of different edge-types was used as a signature of the object.

1.1.2 Keypoints and feature descriptors

While color information is important, the human eye uses more information to recognize objects [18]. Segmentation, categorization and instance detection is based on spatial information, as well as relationships with adjacent objects and the surrounding environment.

In computer vision algorithms the problem space is narrowed down, reducing the visual representation as possible. According to a survey by Mundy [10], the paradigm of detection switched from basic geometric descriptions to appearance features.

Image features are based on keypoints and descriptors. A keypoint is a point of interest that is invariant to rotation, translation, scale and intensity transformations. The descriptor gives information about the region around this point.

Points of interest could be corners, line endings, or points on a curve where the curvature is locally maximal. Historically, the first corner detection method was presented by Hans P. Moravec [19] in 1977. The idea was that corners (and edges) could be detected locally using a small window: if the shifting of the window to any direction results in a large change in the intensity sum, a corner is detected. As only a 45-degree shift is considered, any edges not horizontal, vertical or diagonal

[2]Red-Green-Blue

[3]Hue-Saturation-Intensity

are incorrectly marked as corner points.

The Moravec detector is not isotropic, the response is not invariant to rotation.

An improved solution was presented by Christopher G. Harris and Mike Stephens [20] in 1988. To deal with the noise and possible rotations, a Gaussian window function was introduced. Using this window, all possible shifts should be examined; however, this step would have been computationally intense. So, instead, the first-order Taylor series expansion was used to approximate the value.

In 1994, Jianbo Shi and Carlo Tomasi [21] proposed a method based on the Harris detector, with a small and effective change on the scoring formula.

It is important to point out that both the Harris corner detector and the Shi-Tomasi detector are invariant to rotation; however, both are affected by input scaling. The scaling issue could be handled by storing external information about the region around the point of interest, for example, the patch size [22]. Other descriptors based on derivatives are often used as descriptors, like the Laplacian of Gaussians (LoG) or Difference of Gaussians (DoG) [23].

Scale-invariant feature transform (SIFT) was published by David G. Lowe in 1999 [24]. Inspired by the Harris detection, the procedure of feature extraction was based on the application of Gaussian kernels to obtain multiple scale representations of the same image, which was used to discard points of interest with low contrast.

The SIFT method is sensitive to lightning changes and blur, but the most important drawback of the application is that it is computationally heavy and real-time applications are not feasible.

The Speeded up robust features (SURF) [25] technique is using integral images resulting in a faster detection with similar precision [26]. It is worth mentioning, that the SURF method itself is not real-time; however, parallelization is possible.

If the task is matching instances of similar objects of the same category with similar appearance on low-quality images, the application of feature-based descriptors could result in low performance. While scale-invariant feature detection is a computationally intense process, noise of low contrast, blur and lightning changes affect the features, and therefore the result of matching as well.

For the vehicle-matching problem several solutions were introduced. For aerial object tracking, [27] introduced an extended line segment-based approach to detect and warp observations for matching. The authors proposed another approach in the following year based on 3D model approximation [28]. A similar method for vehicle matching of observations in different poses was introduced in [29], where the estimation is supported by the reflected light. In [30], it was pointed out that temporal information could also be used for tracking in a multi-camera network.

1.1.3 Template matching

To deal with the matching of low-quality images, classic template matching [31] can be used. Finding the visual representation of an object on a reference image is based on a basic, pixel-level comparison of the reference pixels and the template. The matching process is comparing the pixels of the template image with the reference image, using a sliding window, and selecting the best fit. The window is a template-sized patch, which is moved over the reference image.

While moving the window over the reference image, the fitting of the corresponding pixels is measured. There are several methods to evaluate: the simple ways are to calculate the sum of squared differences or the cross-correlation. However, the normalized versions of these methods usually provide better results.

Depending on the chosen method, the minimum or maximum value represents the best match on the image. Multiple matches can be handled as well, if a threshold is applied instead of the minimum-maximum selection.

Template matching is sensitive to rotational or scalable changes on the reference image. A possible solution to solve the problems raised by different sizes is to create multiple images resizing the original template, match each of them on the reference, and, finally, select the best match of the results. These multi-scale template-matching methods are highly parallelizable [K3]. It is notable that this method could still fail detecting rotated or tilted objects.

1.1.4 Haar-like features

The processing of intensity values on an image patch is computationally expensive, and although parallelizational methods exist, the effectiveness is questionable. An alternative to working with intensities is based on integral images.

In 1997, Constantine P. Papageorgiou et al. presented a method [32, 33] which was based on the idea of Haar wavelets. After a statistical analysis of multiple objects from the same class, the trainable method was able to detect objects, for example, pedestrians or faces.

In 2001, Paul Viola and Michael Jones [34] published a machine learning approach to detect visual representations rapidly. Motivated by Papageorgiou et al. the system was not working directly with intensities, initially an integral image representation was computed from the source, storing the sum of pixel intensities for every point on the image.

The authors also defined Haar-like features [35] as differences in the sum of intensities in rectangular regions in an image. Using the integral image, the intensity sum of a rectangular region can be calculated in constant time, producing a fixed step number for Haar-like feature computation.

The Viola-Jones detector is trainable and uses an AdaBoost-based technique to select the important features and to order them in a cascade structure to increase the speed of the detection. The method was very popular and widely used from face detection to vehicle detection.

After the detection, the matching could be done based on the same features used for detection [36], without the need of unnecessary computation. Reyes Rios-Cabrera et al. [4] presented a complex system for vehicle detection, tracking and identification. In the identification method, a so-called vehicle fingerprint was used, which is based on the Haar-features used for detecting the vehicles.

The speed and accuracy of a system based on reusing Haar-like features are satisfactory; however, the condition of necessary pre-training is difficult to meet in real-life applications. The training set of images and the test set should be acquired on the same environmental conditions. Also, the set needs to be large enough to include multiple environments, conditions (lightning, weather, etc.).

1.2 Projection features

Similar low-level features could be constructed based on the sum of intensities for every column and row of the patch. These sums, the horizontal and vertical projections, are one-dimensional vectors mapped from the 2D image.

Applications of row and column sum vectors are first described by Herbert J. Ryser in 1957 [37], which is referred as one of the first discrete tomography [38, 39] methods: based on the two orthogonal projections, the original binary matrix is reconstructable. Reconstruction from a small number of projections can yield multiple valid results. It is notable that, in the case of a non-binary domain of values, the number of possible solutions of the reconstruction method can be lowered by more projections.

While reconstruction is important in medical applications [40], for object image matching the projection functions themselves could be used. Margrit Betke et al. presented a method for vehicle detection and tracking [41], where projection vectors are calculated of an edge map and are used to adjust the position during tracking.

In other applications, projections can be used for gait analysis for walking people [42], where the video sequence is transformed into a spatio-temporal 2D representation. Patterns in this so-called Frieze-group [43] representation can be analyzed using horizontal and vertical projections. As an extension, a variation of this method could be developed based only on the shape information [44], resulting in a method that was not affected by body appearance.

The idea of using a mapping of the vehicle observation for further processing also appeared in [45], where the distance calculation between observations were based on edge maps.

1.2.1 4D signature calculation

Vedran Jelača et al. [46] presented a solution based on projections for vehicle matching in tunnels, where object signatures are calculated from projection profiles. A possible interpretation of this method is presented here in detail.

After the region of interest is selected, the area is completed to a square and cropped. As color data are irrelevant in dark, artificially lighted areas such as tunnels, the images are grayscaled, meaning that the information is simplified from a RGB structure to a single intensity value. In the case of 8-bit grayscale images, the intensity information of one pixel is stored in one single byte.

Each image can be handled as matrix $\mathbf{I} \in \mathbb{N}^{N \times N}$ where $\mathbf{I}_{i,j} = [0, 1, \dots 255]$ denotes the element of the matrix at $(i; j)$. The horizontal ($\boldsymbol{\pi}_H$) and vertical projections ($\boldsymbol{\pi}_V$) for a squared $N \times N$ matrix result in vectors with the same length:

$$\dim \boldsymbol{\pi}_H = \dim \boldsymbol{\pi}_V = N. \tag{1.1}$$

These projections are the averaged sums of the rows and columns of the matrix, normalized to $[0, 1]$ by the value of maximal possible intensity:

$$\begin{aligned} \boldsymbol{\pi}_H(i) &= \frac{1}{255N} \sum_{j=1}^{N} \mathbf{I}_{i,j}, \\ \boldsymbol{\pi}_V(j) &= \frac{1}{255N} \sum_{i=1}^{N} \mathbf{I}_{i,j}, \end{aligned} \tag{1.2}$$

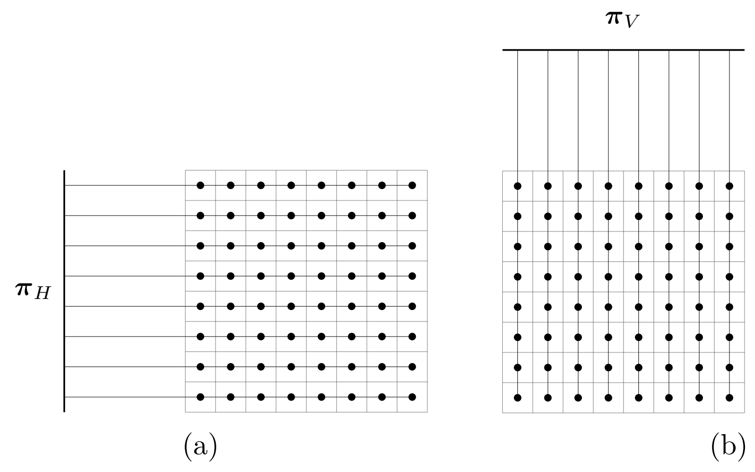


Figure 1.1. Horizontal (a) and vertical (b) image projections are calculated by summarizing the row and column values in the matrix.

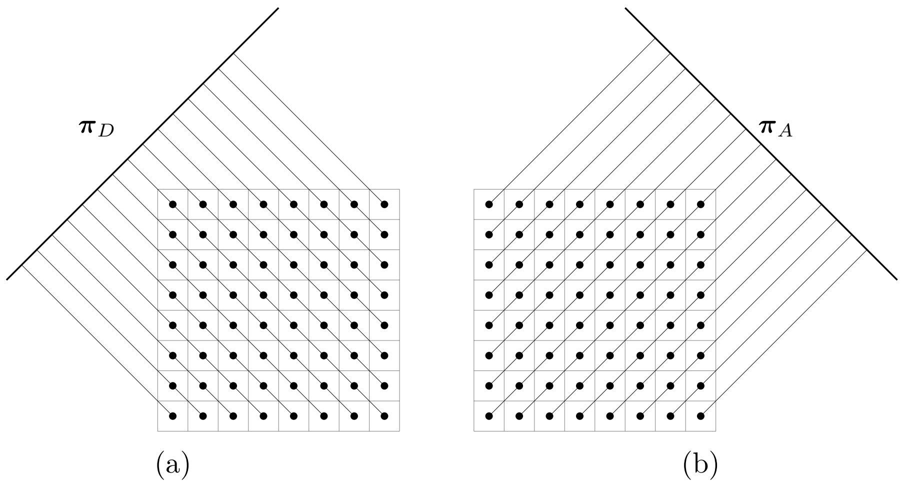


Figure 1.2. Diagonal (a) and antidiagonal (b) image projections of a squared image matrix.

visualized in Figure 1.1.

$\boldsymbol{\pi}_H, \boldsymbol{\pi}_V$, therefore, defines the two-dimensional projection signature of the object:

$$\mathbf{S_2} = (\boldsymbol{\pi}_H, \boldsymbol{\pi}_V). \tag{1.3}$$

The diagonal and antidiagonal projections can be calculated likewise, but it is important to point out that the number of elements for each projected value differs (Figure 1.2).

The length of the diagonal projection vectors are:

$$\dim \boldsymbol{\pi}_D = \dim \boldsymbol{\pi}_A = 2N - 1, \tag{1.4}$$

the number of elements in each summarization is based on the distance from the main diagonal:

$$ElemNum(i) = \begin{cases} i & \text{if } i \leq N \\ 2N - i & \text{otherwise} \end{cases} \tag{1.5}$$

where i is the index of an element in a diagonal projection, having $i \leq 2N - 1$.

The calculation of the diagonal projections $\boldsymbol{\pi}_D$, $\boldsymbol{\pi}_A$ is formalized as:

$$\boldsymbol{\pi}_D(i) = \begin{cases} \frac{1}{255 \cdot ElemNum(i)} \sum_{j=1}^{i} \mathbf{I}_{j,N-(i-j)} & \text{if } i \leq N \\ \frac{1}{255 \cdot ElemNum(i)} \sum_{j=1}^{i-N} \mathbf{I}_{j+(i-N),j} & \text{otherwise} \end{cases} \tag{1.6}$$

$$\boldsymbol{\pi}_A(i) = \begin{cases} \frac{1}{255 \cdot ElemNum(i)} \sum_{j=1}^{i} \mathbf{I}_{j,(i-j)+1} & \text{if } i \leq N \\ \frac{1}{255 \cdot ElemNum(i)} \sum_{j=1}^{i-N} \mathbf{I}_{j+(i-N),N-(j-1)} & \text{otherwise} \end{cases}$$

These projection vectors together provide the 4D projection signature of the object:

$$\mathbf{S_4} = (\boldsymbol{\pi}_H, \boldsymbol{\pi}_V, \boldsymbol{\pi}_D, \boldsymbol{\pi}_A). \tag{1.7}$$

After normalizing the values using the number of elements that make up the sum, multiplied with the maximum intensity value 255, the domain of the projection function is $[0; 1]$. The 4D signature of a vehicle observation is visualized in Fig. 1.3.

1.2.2 Projection-based object matching

As the size of the input images could be different, the length of projection functions is different as well. To be able to match these functions properly a similarity measurement method must be defined.

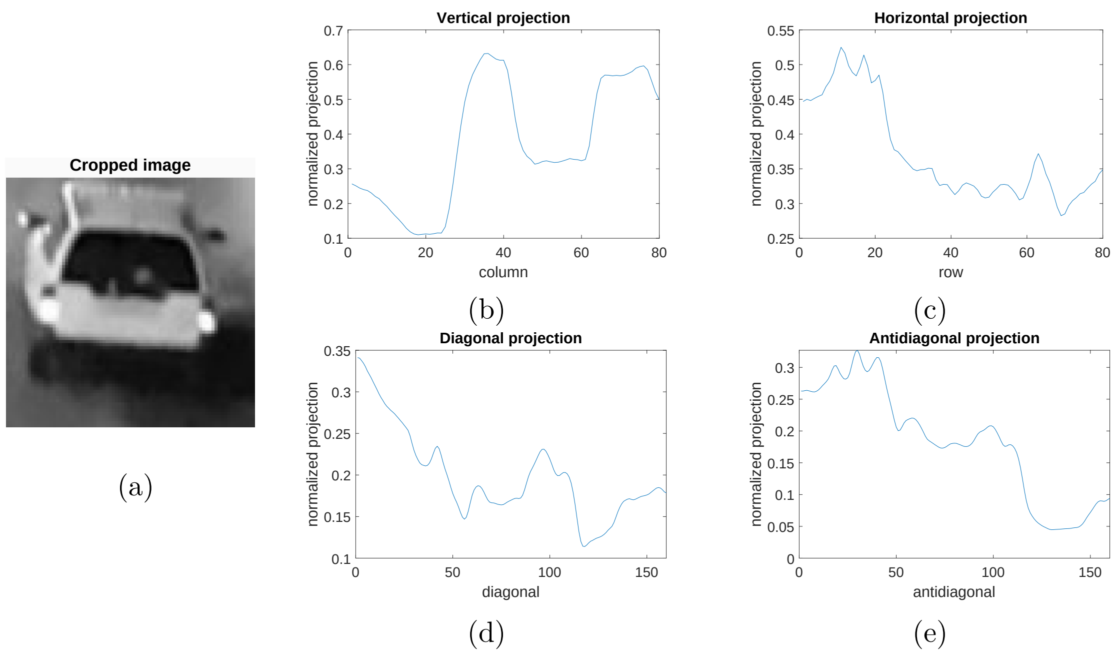


Figure 1.3. The visualized projection signature of a vehicle observation. Subfigure (a) shows the squared image of a rear-viewed vehicle. In diagrams (b), (c), (d) and (e), the horizontal, vertical, diagonal and antidiagonal projections are visualized ($\boldsymbol{\pi}_H$, $\boldsymbol{\pi}_V$, $\boldsymbol{\pi}_D$ and $\boldsymbol{\pi}_A$, respectively).

To calculate the alignment of the functions, the method suggested in [46] is to align the projection functions globally, and then fine-tune with a "local alignment" using a method similar to the Iterative Closest Point [47, 48].

First, the signature vectors are rescaled based on the camera settings and the position of the object in the image, so the effects of zoom and camera settings are corrected. After the signatures are resized, it is still necessary to align the signatures because of shifting and length differences.

The functions are compared based on the sliding window technique: the shorter function is moved over the longer function, and each correlation coefficient is calculated using the Pearson correlation coefficient (PCC) formula:

$$\rho_l(\mathbf{x}, \mathbf{y}, s) = \frac{\operatorname{cov}(\mathbf{x}, \mathbf{y}_s)}{\sigma(\mathbf{x})\sigma(\mathbf{y}_s)}, \tag{1.8}$$

where $\mathbf{x}, \mathbf{y}$ are projection vectors, where $\dim \mathbf{x} \leq \dim \mathbf{y}$. $\mathbf{y}_s$ represents the part of vector $\mathbf{y}$, which is shifted by s and overlaps $\mathbf{x}$.

Basically, the $\rho_l(\mathbf{x}, \mathbf{y}, s)$ correlation coefficients are calculated for each step s, where the number of steps is $\dim \mathbf{y} - \dim \mathbf{x}$. $\operatorname{cov}(\mathbf{x}, \mathbf{y}_s)$ means the covariance between the two vectors and σ indicates the standard deviation.

The highest value $\max_s \rho_l(\mathbf{x}, \mathbf{y}, s)$ is selected as ρ, defining the similarity of $\mathbf{x}$ and $\mathbf{y}$. For horizontal, vertical, diagonal and antidiagonal projection functions, notations ρ_H, ρ_V, ρ_D and ρ_A are used, respectively.

The local alignment suggested in [46] was based on signature smoothing followed by curve alignment: step-by-step iteratively removing the extremas caused by noise and finding the best fit for two functions.

The range of the values are mapped to $[-1; 1]$, which could be easily handled: the higher the coefficient, the better the match. After all similarity values are calculated for the projections, the result values are filtered with a rectifier, setting all negative values to zero:

$$r(v) = \begin{cases} v & \text{if } v > 0 \\ 0 & \text{otherwise} \end{cases} = \max(0, v) \tag{1.9}$$

Negative correlation values mean that the changes of one function affect an opposite change on the other function, meaning that the relationship between the two is inverse. So, in this case, the penalization of the negative correlation is necessary, because projection inverses should not be relevant at all.

The suggestion in [46] is to equally handle each dimension of the data, by using the Euclidean norm (L2 norm). A single similarity value μ is calculated from the 4D signature as

$$\mu = \frac{\sqrt{r(\rho_H)^2 + r(\rho_V)^2 + r(\rho_D)^2 + r(\rho_A)^2}}{2} \tag{1.10}$$

where 2 is the square root of the dimension number, therefore, normalizing the norm.

1.3 Goal of the research

The research described in the following chapters was inspired by this four-dimensional projection signature technique. The motivation for further research was based on the study of multiple projection directions for object descriptors.

The main goal is to analyze the effectiveness of multi-directional projection signatures for object matching. As the four-dimensional projection signatures are applicable for instance recognition, it is presumed that multiple directions would increase the accuracy of the method.

The length of a projection vector depends on the size of the image and on the projection angle. If the calculated similarity of different length functions is based on the best fitting, the resulting values result in high scores even in the case of false pairs.

My goal is to develop a multi-directional projection mapping method, where the number of bins is independent from the rotational angle or the image size.

The properties of the designed transformation should be analyzed and the performance of object matching should be compared to other similar techniques.

The possibilities of parallel implementation should also be examined, as modern computer architectures provide great support for data-parallel execution.

For different object types, projections could have different significance in terms of object matching. To analyze the weight of each angle, statistical or machine learning approaches could be done.

The secondary goal of this dissertation is to study the modern machine learning approaches of object matching and to analyze the applicability of multi-directional projections for object descriptors for the same task.

A comprehensive experiment is necessary to determine the performance of such methods. In large experiments, the selective environment and the broad input data are key factors. In this dissertation, a complex experiment is designed and implemented for object image matching using neural networks.

As a part of this task, a neural network architecture generation method is developed and a distributed approach for processing the models is designed.

Chapter 2

Multi-directional image projections

Good approximations often lead to better ones.
— George Pólya,
Mathematical Methods in Science

While projection-based matching methods can be simply described and implemented, the noise of surrounding objects and lighting changes result in noise, which affects the precision of the solution. In works presented earlier [4, 45, 46], these drawbacks are handled by complex appearance modeling and multiple levels of alignment during the process.

It is clear that the computational cost of the projection transformation is low and data-parallel solutions could be given to further reduce the runtime. However, the complexity of the matching algorithm alongside the rescaling, noise removal and alignment steps significantly affect the overall performance.

The idea of increasing the number of projections was motivated by the mechanism of computer-assisted tomography machines: the more input data is given, the more precise the reconstructed 3D image is going to be. More accurately, there is a tradeoff between accuracy and computational cost, and the latter is directly affected by the number of projection angles.

In the following sections, the well-known Radon transform is introduced and analyzed, followed by the definition of a novel method for multi-directional image projection calculation in Section 2.2. To demonstrate the real-time capabilities of the method, a data-parallel solution is given in Section 2.3. The method is implemented and results for object matching are evaluated in Section 2.4.

2.1 Introduction

For horizontal and vertical projections on rectangular patches, the summarization of rows and columns is a correct approach. In case the diagonal and antidiagonal projections are also required in a squared matrix, the method of summarizing elements by traversing through the matrix as given in Section 1.2 is similarly simple.

For other angles (i.e. other than 0, 45, 90, 135 degrees), this discrete, element-based method is not applicable without direct loss of information or ambiguity.

The solution for this problem can be based on the work of mathematician Johann Radon, who defined a mathematical framework to create a transformation between an object and its projections [40].

2.1.1 The Radon-transform

The original mapping of 2D objects to 1D projection profiles was first studied by Johann Radon [7] in 1917; a translation of his paper was published in 1986 [8]. The formula and the inversion formula became popular [40] as they were used with CT[1], PET[2], MRI[3] and SPECT[4] scanners to reconstruct images from the obtained data. Current reconstruction algorithms are based or inspired by the inverse Radon-transformation [40]. Other application fields include electron microscopy [49], astronomy [50], and geophysics [51].

To formally define the method, define f as a function on the two-dimensional Euclidean space $\mathbb{R}^2$. Let (x, y) designate coordinates of points of the plane. The Radon transform of f results in $\check{f}$ by integrating f over all L lines in the plane.

So the Radon transform is usually defined as

$$\check{f} = \mathcal{R}f = \int_L f(x, y)\mathrm{d}s, \tag{2.1}$$

The projection or line integral is based on line $L \in \mathbb{R}^2$, and ds is an increment along L [40]. It is notable, that there are multiple generalized forms of the transform, where the domain is extended to higher dimensions (such as $\mathbb{R}^2$ or $\mathbb{R}^n$) [52, 53].

A line could be given in normal form as

$$p = x\cos(\phi) + y\sin(\phi), \tag{2.2}$$

where ϕ is the angle of rotation. From p and ϕ, the Radon transform of $f(x, y)$ can be given as

$$\check{f}(p, \phi) = \mathcal{R}f = \int_L f(x, y)\mathrm{d}s. \tag{2.3}$$

A rotated coordinate system with axes p and s could be presented by applying the rotation matrix $Rot(\phi)$ as

[1]Computerized Tomography
[2]Positron Emission Tomography
[3]Magnetic Resonance Imaging
[4]Single-Photon Emission Computed Tomography

$$\begin{aligned}\begin{bmatrix}p\\ s\end{bmatrix} &= Rot(\phi)\begin{bmatrix}x\\ y\end{bmatrix}\\ &= \begin{bmatrix}\cos\phi & -\sin\phi\\ \sin\phi & \cos\phi\end{bmatrix}\begin{bmatrix}x\\ y\end{bmatrix}\\ &= \begin{bmatrix}x\cos(\phi) - y\sin(\phi)\\ y\cos(\phi) + x\sin(\phi)\end{bmatrix}.\end{aligned} \tag{2.4}$$

Having axes p and s of the new coordinate system, then

$$\begin{aligned}x &= p\cos(\phi) - s\sin(\phi)\\ y &= p\sin(\phi) + s\cos(\phi),\end{aligned} \tag{2.5}$$

resulting with the form of the Radon transform in

$$\check{f}(p,\phi) = \int_{-\infty}^{\infty} f(p\cos(\phi) - s\sin(\phi), p\sin(\phi) + s\cos(\phi))\mathrm{d}s. \tag{2.6}$$

Important to point out that values of ϕ are defined on $[0; 2\pi)$; the values of trigonometric functions outside this range are cyclically repeating.

J. Radon also defined [7, 8] the cyclic recurrence, by elegantly stating that

$$\check{f}(p,\phi) = \check{f}(-p,\phi+\pi). \tag{2.7}$$

Other properties regarding the rotational angle are further discussed in Section 2.1.2.

2.1.2 Properties of the Radon-transform

The visual representation of the Radon transform is often referred to as a sinogram, where p projection sums are presented for each ϕ angle. A sample of the sinogram representation is in Figure 2.1.

Sinusoid

The name sinogram comes from the sinusoid representation of the transformed points. The reason of the sinus wave could be simply proven: the representation of point $P(x_0, y_0)$ in the Radon space can be given from (2.2) as

$$p = x_0\cos(\phi) + y_0\sin(\phi), \tag{2.8}$$

which can be reducted to a sinus function using

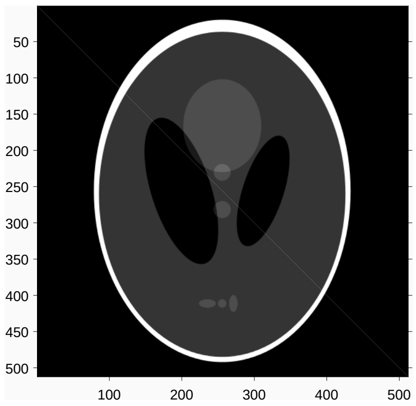
$$\sin(\alpha + \arctan(z)) = \frac{z\cos(\alpha) + \sin(\alpha)}{\sqrt{1+z^2}}, \tag{2.9}$$

which is concluded having

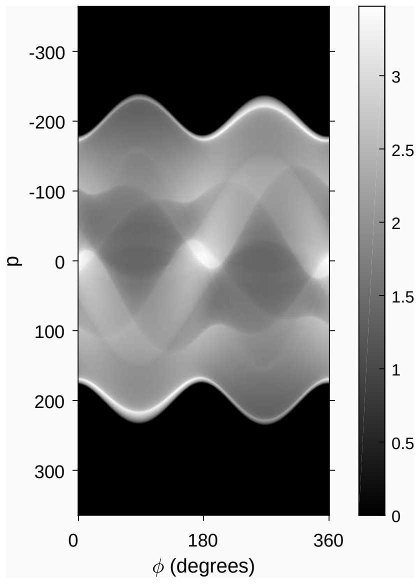
$$\sin(\alpha+\beta) = \sin\alpha\cos\beta + \cos\alpha\sin\beta, \tag{2.10a}$$

where

$$\beta = \arctan(z), \tag{2.10b}$$



(a)



(b)

Figure 2.1. The Shepp-Logan [54] phantom is a standard test image used in the testing of image reconstruction algorithms: it consists of 10 differently sized, rotated ellipses with different intensity levels inside the squared area.

$$\sin(\arctan(z)) = \frac{z}{\sqrt{1+z^2}}, \tag{2.10c}$$

$$\cos(\arctan(z)) = \frac{1}{\sqrt{1+z^2}}. \tag{2.10d}$$

Using (2.9), (2.8) can be rearranged as

$$\frac{p}{y_0} = \frac{x_0}{y_0}\cos(\phi) + \sin(\phi) \tag{2.11a}$$

$$\frac{\dfrac{p}{y_0}}{\sqrt{1+\left(\dfrac{x_0}{y_0}\right)^2}} = \frac{\dfrac{x_0}{y_0}\cos(\phi) + \sin(\phi)}{\sqrt{1+\left(\dfrac{x_0}{y_0}\right)^2}} \tag{2.11b}$$

$$\frac{\dfrac{p}{y_0}}{\sqrt{1+\left(\dfrac{x_0}{y_0}\right)^2}} = \sin(\phi + \arctan(\frac{x_0}{y_0})) \tag{2.11c}$$

$$\frac{p}{y_0\sqrt{1+\left(\dfrac{x_0}{y_0}\right)^2}} = \sin(\phi + \arctan(\frac{x_0}{y_0})) \tag{2.11d}$$

$$p = y_0\sqrt{1+\left(\dfrac{x_0}{y_0}\right)^2}\sin(\phi + \arctan(\frac{x_0}{y_0})) \tag{2.11e}$$

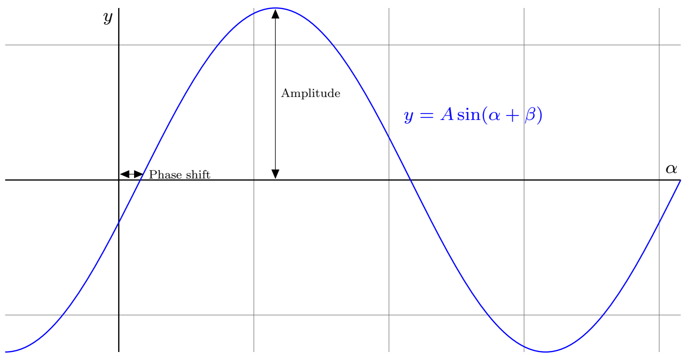


Figure 2.2. The transformation of the sinus wave is formalized as $A\sin(\alpha+\beta)$, where A stands for the amplitude and β is the phase shift.

$$p = \sqrt{x_0^2 + y_0^2}\sin(\phi + \arctan(\frac{x_0}{y_0})). \tag{2.11f}$$

Equation (2.11f) defines a sinus wave [55] with amplitude $A = \sqrt{x_0^2 + y_0^2}$ and phase shift $\theta = \arctan(\frac{x_0}{y_0})$, visible in Figure 2.2.

Translation

The horizontal and vertical translation of an object by $\Delta x, \Delta y$ on input I could be denoted as

$$p_s(\phi) = \Delta x\cos(\phi) + \Delta y\sin(\phi). \tag{2.12}$$

Based on (2.1), the Radon transform of a translated object is given as

$$\mathcal{R}f = \check{f}(p - p_s(\phi), \phi). \tag{2.13}$$

As visualized in Figure 2.3, the horizontal translation of the object results in amplitude and phase changes in the sinusoids, which is formulated in (2.11f).

Vertical translation or scaling have similar effects on the sinogram.

Mirror-effect

As Johann Radon elegantly pointed out in (2.7) [7, 8], $\check{f}(p,\phi) = \check{f}(-p,\phi+\pi)$, meaning that the projections between $[0;\pi)$ and $[\pi;2\pi)$ are the same but inverted.

This results in the fact that the Radon transform of an image is only relevant on the range of $[0;\pi)$, projections for every following angle can be constructed from it.

Shifting

The most trivial property of the Radon transform is that the rotation of input image $I = f(x,y)$ effects on $\check{f}(p,\phi)$ sinogram as a linear shifting.

As a result, the rotation of the object of interest does not affect the projection-based recognition, as it leads to a circular shift in the projection space.

In Figure 2.4, the circular shifting phenomenon is presented by applying the Radon inversion formula. The inversion formula is used to reconstruct the original image from its projections. The basic formula is given in the original work of Radon

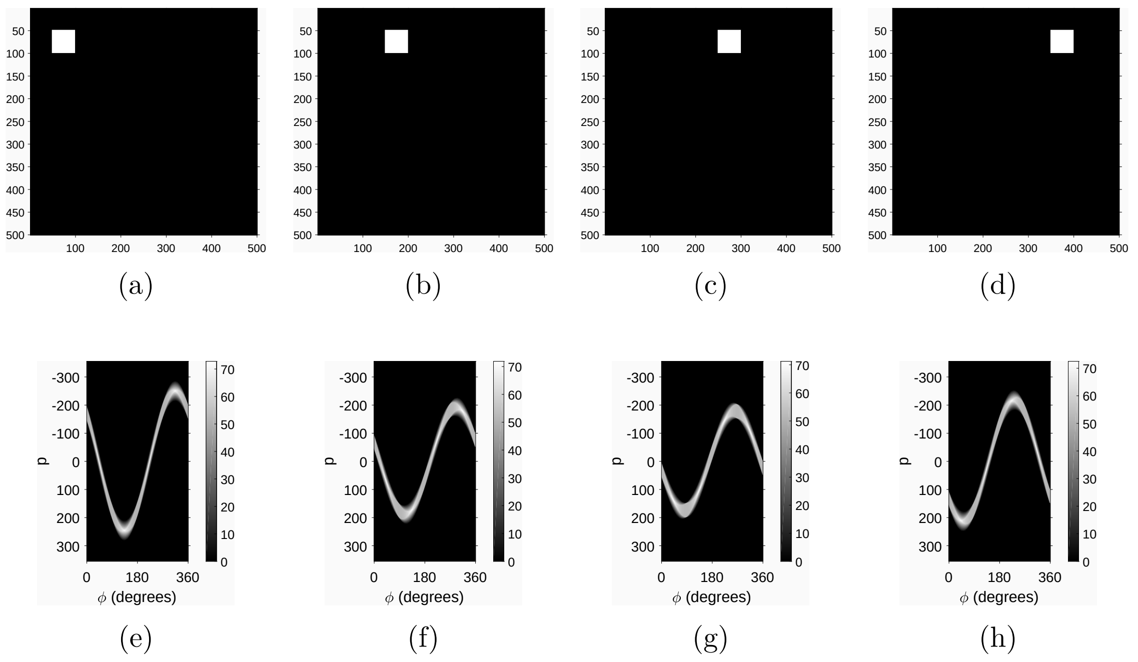


Figure 2.3. The results of horizontal translation on the sinogram: subfigures (a), (b), (c) and (d) show the input image, where the object is shifted to the right. Subfigures (e), (f), (g) and (h) present the sinograms for each input image [K2]. It is notable, that the changes of the amplitude are directly affected by the distance changes from the center of the image.

[7] [8]; however, there are multiple approaches for reconstruction, based on the Fourier slice theorem [56] or on filtered backprojection [57].

An interesting fact is that if the object is mirrored, the resulting sinogram is rotated by 180 degrees, which is the same as considering the reflection from left to right and then upside-down.

Other properties

When using the Radon transform on a 2D image, some properties of the form can be generalized, or simplified. First of all, having image as an $N \times N$ matrix I, the integrals calculated for each line are the sums of the affected pixel intensities.

The size of the result matrix needs to be defined according to the longest projection of the image, which is the diagonal. For a squared $N \times N$ image, it is given by $\sqrt{2}N$.

The sampling rate of ϕ is a key factor: increasing the step size causes less computation; however, larger step sizes result in projection data loss. The range of the value of ϕ is $[0, 2\pi)$. However, according to the previously seen mirror-effect, the range of $[\pi; 2\pi)$ is the same as the values in $[0; \pi)$, only flipped around p axis.

Because of this phenomena, it is only necessary to define the projection sums p of I image in the range $[0, \pi)$ of ϕ. For a discrete representation, such as an image made of pixel intensities, a step size for rotation has to be defined. The optimal step-size selection depends on multiple assumptions, mostly on matrix size N, so it will be referred to as $Step(N)$.

As the sinogram size is defined as an $\mathbf{R_f}$ matrix $\left\lceil \sqrt{2}N \right\rceil \times \left\lceil \dfrac{\pi}{Step(N)} \right\rceil$, and the

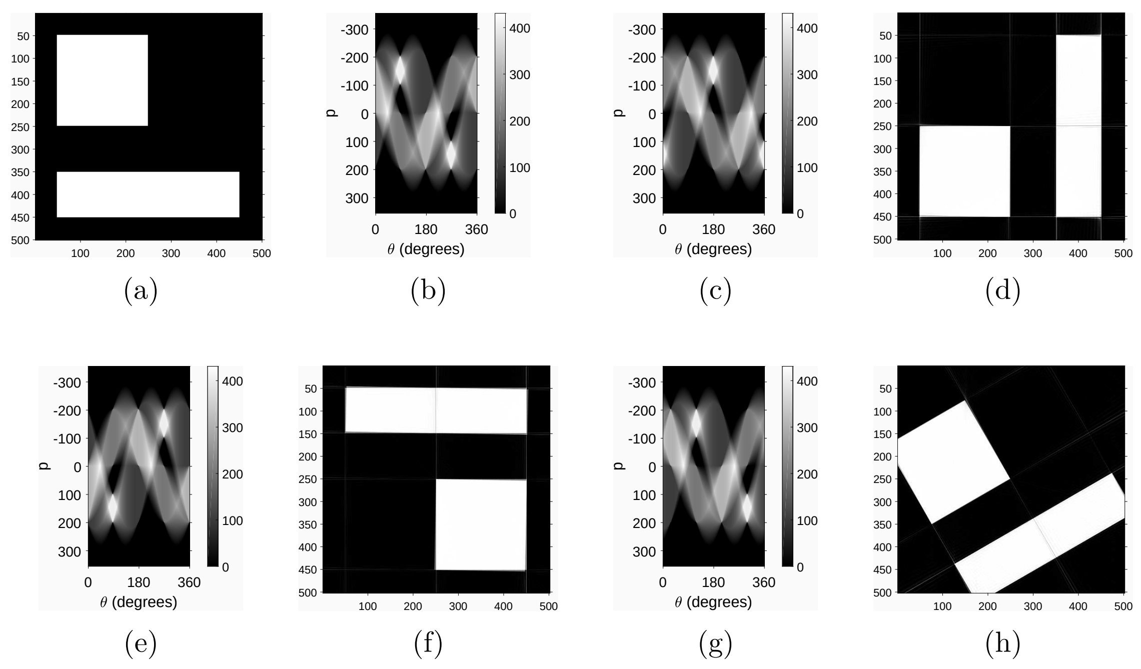


Figure 2.4. The effect of shifting in Radon space results in rotation. The original image (in subfigure (a)), and the sinogram of its projection sums (in subfigure (e)). To illustrate the effect of the circular shifting, the resulting matrix of the Radon transform is shifted by $\phi = \pi/2$, π and $\pi/6$, presented in subfigures (b), (c) and (d), respectively. Subfigures (f), (g) and (h) show the result of the reconstruction after shifting [K2].

input is an $N \times N$ matrix referred as I, a few specialties could be noticed.

The first interesting property of the application of the Radon transform on squared input is that the R_f result matrix will have regions, where the sum is always zero, caused by a lack of crossing pixels.

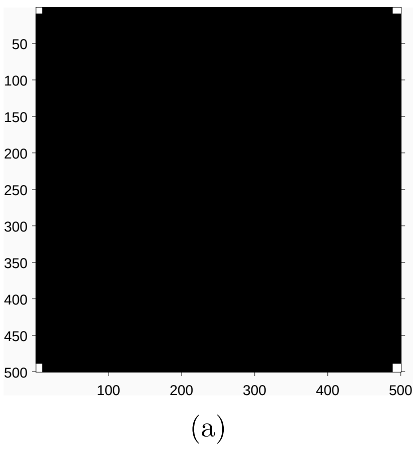
In Figure 2.5, the sample binary matrix is defined with zero values, whereas the corner points have unit intensities as

$$\mathbf{I_1} = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 1 \\ 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & 0 & 0 & \cdots & 1 \end{bmatrix}. \tag{2.14}$$

As visible, the regions over and below these boundaries are insignificant. As in Figure 2.6, where the input matrix is a unit matrix, all values are equal to one, as

$$\mathbf{I_2} = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & 1 & 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & 1 & \cdots & 1 \end{bmatrix}, \tag{2.15}$$

the same effect can be noticed.



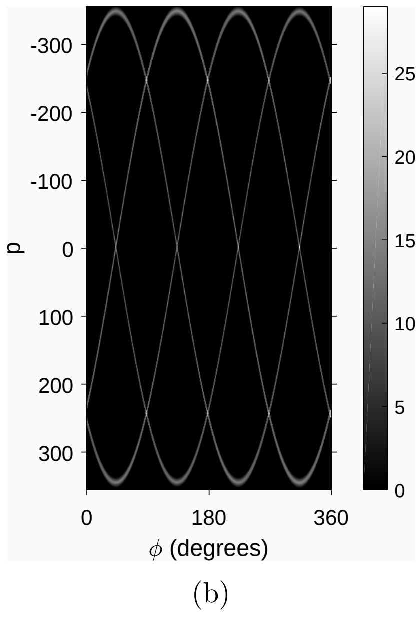
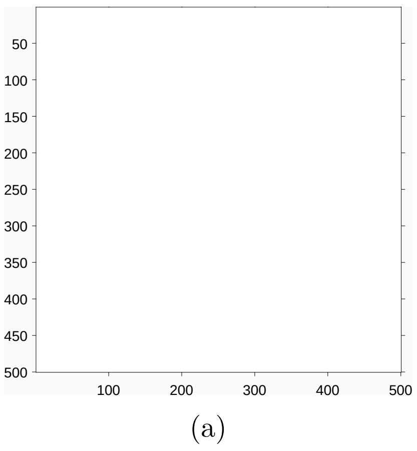


Figure 2.5. Insignificant regions in the sinogram. In subfigure (a) is the image representation of the matrix defined in (2.14). In subfigure (b) is the sinogram of the Radon transform of the image. The regions above and below the projection of the corner points are insignificant [K2].



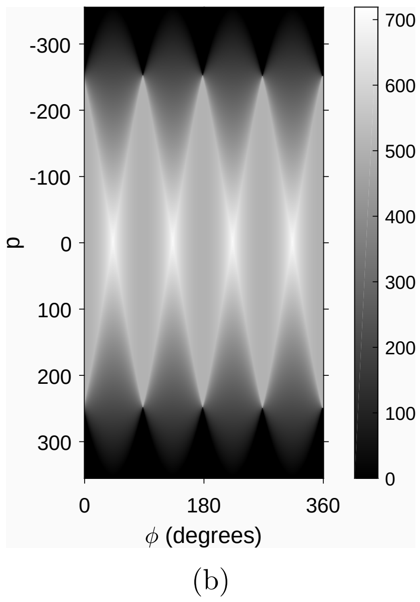


Figure 2.6. The Radon transform of a unit matrix. In subfigure (a) is the image representation of the unit matrix defined in (2.15). Subfigure (b) shows the sinogram of the Radon transform of this image. Note, that the intensity is affected by the number of pixels summarized: the brightest points are in the centers of the diagonal and antidiagonal projections at 45, 135, 225 and 315 degrees [K2].

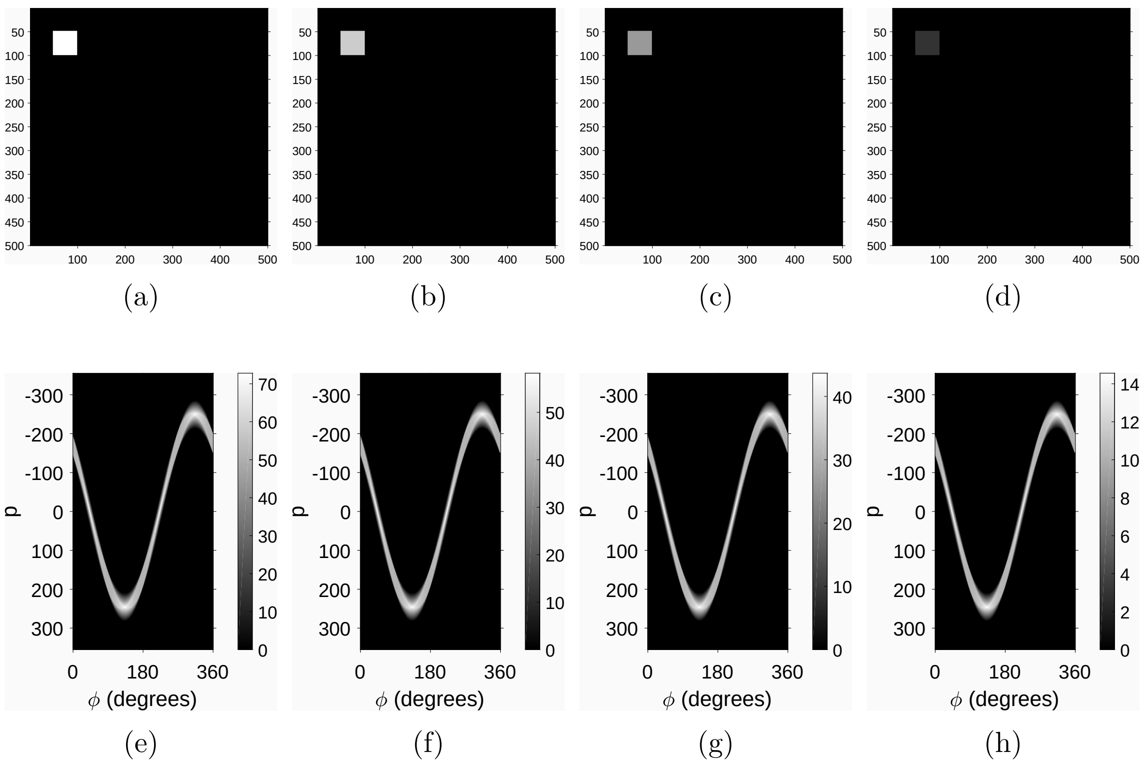


Figure 2.7. The effect of intensity changes on the sinogram. Subfigures (a), (b), (c) and (d) show the input image, where the object intensity is 1, 0.8, 0.6 and 0.2, respectively, and subfigures (e), (f), (g) and (h) presents the sinograms for the input images [K2].
While the forms of the sinusoids are the same, the intensity levels (visible on the bars) are different.

Another trivial property of the Radon transform is that if the intensity of the visual representation of the object changes, it affects the projection sums; however, the form of the wave is unchanged. In Figure 2.7 this phenomenon is visualized by all sinograms presenting the very same form. Please note that the intensity levels of each sinogram differ. In this special case, the normalization of the sinograms result in equivalent matrices.

In computer vision and image processing, noise removal is an important procedure, with multiple different approaches [9, 14]. The effects of noise on the projection sums of an image are very significant (Figure 2.8 a, b).

There are multiple filters that could be applied to reduce the noise on a 2D image. However, recent studies [58] show, instead of denoising the input image (as seen on 2.8 c), noise reduction filters could be used in the Radon space.

Please note that the filtered backprojection itself comes with minor data loss: by using the inverse Radon formula, the reconstructed image will lose some relevant data besides the noise (Figure 2.8 d).

To illustrate the effects of applying filtering in the Radon space, Figure 2.8 e shows the denoised sinogram, where each projection function was filtered using a simple moving average filter. After noise removal from each projection, the reconstructed image is shown in Figure 2.8 f.

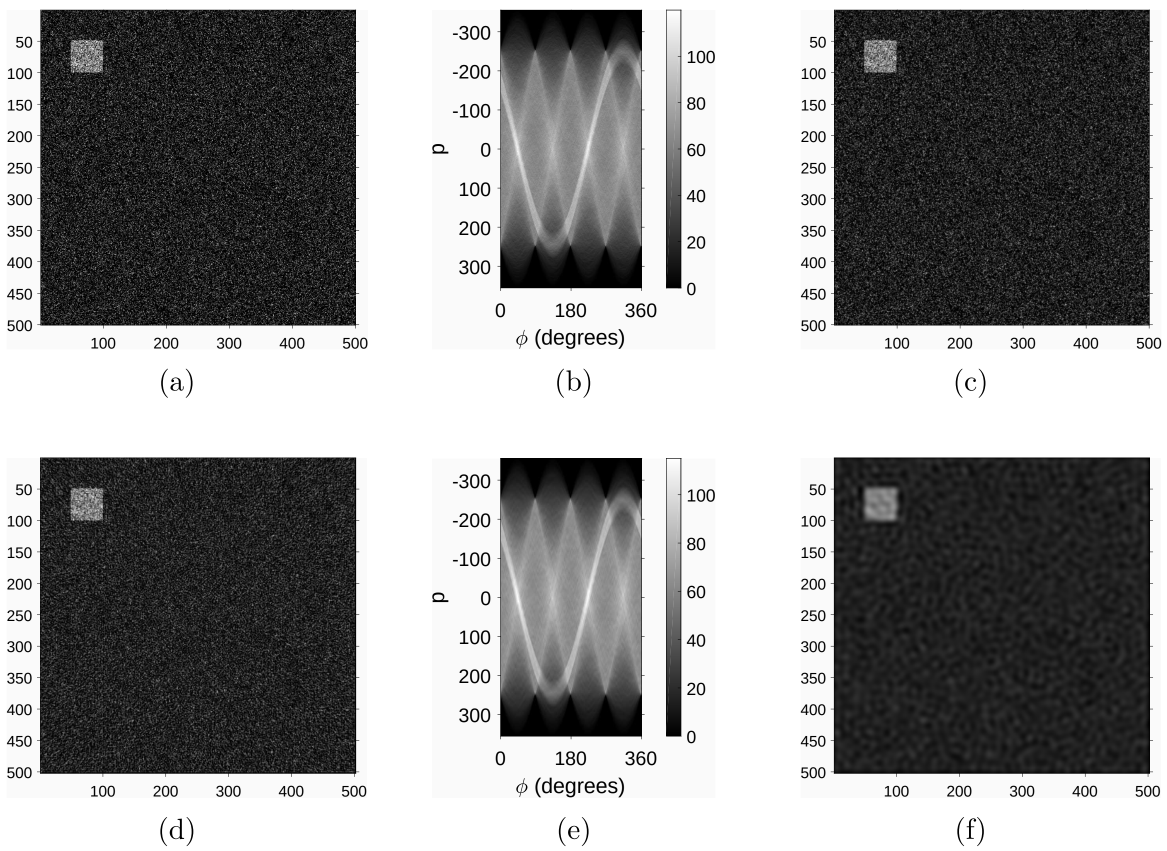


Figure 2.8. The effect of noise in the Radon space. Subfigure (a) shows the original image with a zero mean, 0.1 variance Gaussian noise. In (b), the sinogram of the original image is visualized. In (c) is the Gauss filtered original image. Subfigure (d) gives the reconstructed image. Note, that the reconstruction removes some noise. Subfigure (e) shows the sinogram after filtering each column with a moving average filter, and in subfigure (f) is the reconstructed image of this method [K2].

2.1.3 Other related transformations

The Trace transform defined by Alexander Kadyrov and Maria Petrou is a generalization of the Radon transform. As given in [59], the Trace transform calculates functional T along the corresponding line L. If functional T is the integral of the function, then this definition fits the previously defined Radon transform.

The Trace transform defines a mapping of the image as a function (p, ϕ), similarly to the Radon transform; however, the properties could differ, based on the T functionals applied.

The Hough transform is named after Paul V. C. Hough, who described a method in a US patent [60] to detect straight lines in black-and-white images. The generalized version of this technique, the Hough transform is a mapping from image space to parameter space, similarly as the Radon transform [61]. The Hough transform is generally used to detect complex line patterns, parametrized curves or ellipsoids on images, which method was first described in [62].

The relationship between the Radon and Hough transforms are largely discussed in the literature [40, 61], with different approaches. Despite the fact that Radon published the results 45 years before the patent of Hough, it is a fact that the Radon transform was not discovered until years later [63]. The patent of Hough describes a method with a very similar result, but the idea is independent from the original paper of Radon.

There is a fundamental mathematical connection between the Radon transform and the Fourier transform. Specifically, the two-dimensional Radon transform can be obtained by a Radon transform, followed by a one-dimensional Fourier transform [40].

2.2 Multi-directional projections with fixed bin number

As previously described, the result of Radon transform – when applied to an image (Figure 2.9) – could be used for object matching. While it is highly sensitive to noise, noise reduction filters could be applied in Radon space.

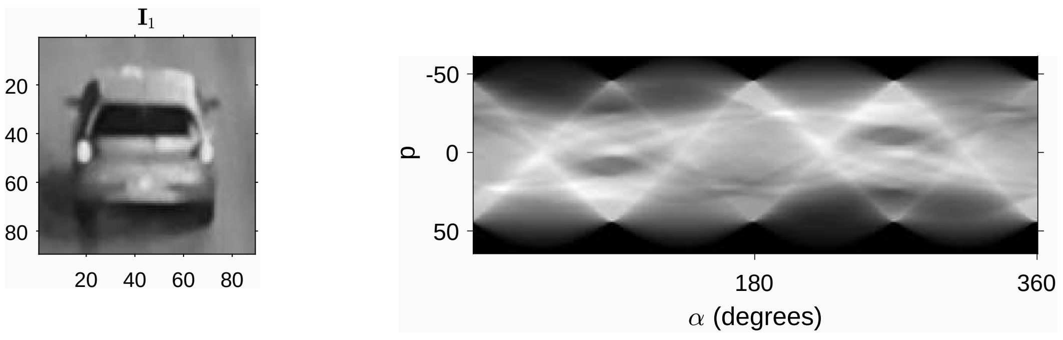
However, few disadvantages of the method are visible: the blank areas on the sinogram could hold up memory space in a possible implementation. To resolve this issue, the size of the result matrix should be fixed, and the input values should be standardized, using fixed scales, so the number of significant elements of the projection vector does not depend on the angle of projection.

As each pixel is technically a small square, and they are the smallest visualizable elements, it is easily concluded that projections of an image result in a set of vectors where the length of each result vector differs and for each length $l \in \mathbb{R}$.

For further calculations, it is realized that a standardized scale should be used, where the length of each vector is equal and $l \in \mathbb{Z}$. This scale, referred as resolution, is defined by dividing the projection line into a given number of pieces.

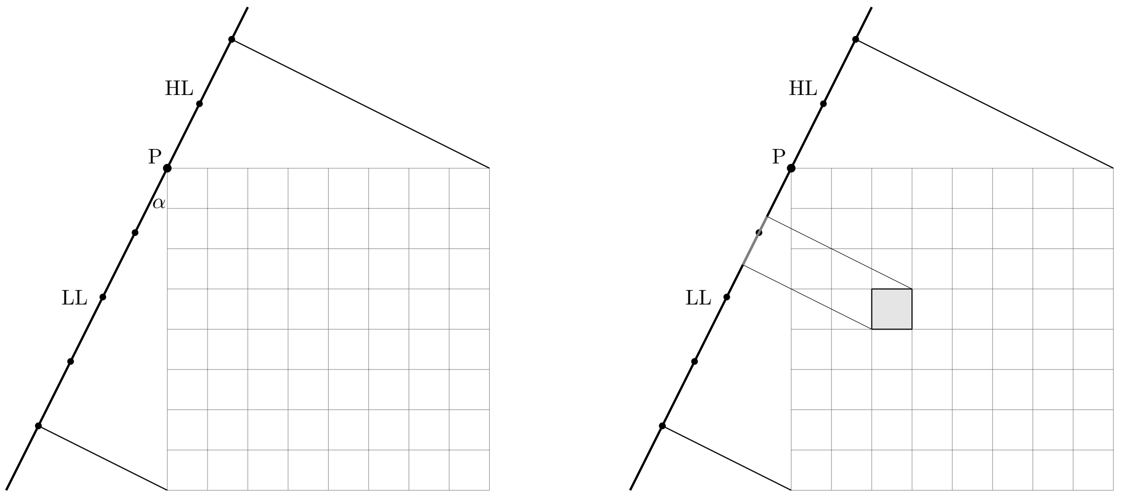
In the proposed method, the projection line is placed to the left side of the image, and it is rotated by α degrees around point P, which is in the top left corner. In this case, $\alpha \in [0, \frac{\pi}{2}]$.

While rotating the line, the lowest and highest points of the orthogonal projection divide it into two segments around point P. The length of these can be easily given



(a) (b)

Figure 2.9. A sample vehicle image in subfigure (a) and the sinogram of the Radon transform for the same image for angles $[0; 2\pi)$ in subfigure (b).



(a) (b)

Figure 2.10. (a) The proposed method with fixed bin sizes. Rotational angle α is 30 degrees and the number of bins is 10. (b) The value of one pixel can affect two (or more) projection bins, according to the resolution.

as

$$\begin{aligned} LL &= \cos(\alpha)N \\ HL &= \sin(\alpha)N. \end{aligned} \tag{2.16}$$

To create a fixed number of subsegments for all angles, segment $LL + HL$ is divided into S equal parts (demonstrated in Figure 2.10a), where S stands for the number of bins, resulting in $\frac{LL+HL}{S}$ in the resolution. Finally, the $(x; y)$ projection position of each pixel is given as

$$\begin{aligned} start &= \sqrt{(x+1)^2 + y^2} \cos(\arctan(y, x+1) + \alpha) \\ end &= \sqrt{x^2 + (y+1)^2} \cos(\arctan(y+1, x) + \alpha) \end{aligned} \tag{2.17}$$

as seen in Figure 2.10b.

Based on S, each pixel is projected into one or more subsegments of the projection line. Each pixel should increase the value of all affected bins, proportionately. Algorithm 1 defines the technique in pseudo language.

Algorithm 1 Method to calculate Multi-Directional Fixed Length Projections of an Image.

```
function MDIPFL($I, N, S, StepSize$)
    $R \leftarrow new\ array[]$
    for $\alpha := 0 \rightarrow \frac{\pi}{2}$ step $StepSize$ do
        $LL \leftarrow \cos(\alpha) \cdot N$
        $HL \leftarrow \sin(\alpha) \cdot N$
        $res \leftarrow (LL + HL)/S$
        for all $px \in I$ do
            $[start, end] \leftarrow$ POSITIONOF($px.X, px.Y, \alpha$)
            RATIONALACCUMULATION($R, res, start, end, px$)
        end for
    end for
    return $R$
end function
```

The results for each direction are calculated for angles between 0 and $\frac{\pi}{2}$ for an $N \times N$ sized squared matrix **I**, and collected into R resulting matrix. In practice, a step size is used at the iteration of α. After LL and HL is specified, according to (2.16), the resolution of the segment *res* is given by dividing the projection line into S pieces. Each pixel px in image **I** is processed by calculating the position of the projection using POSITIONOF, which is based on (2.17), and the correct values of R are increased proportionately, represented as function RATIONALACCUMULATION.

The behavior of the RATIONALACCUMULATION procedure can be described as:

- If the number of affected bins is 1, the pixel value is added to the bin entirely
- If the number of affected bins is 2, a ration based on the projection segments is added to each affected bin
- If the number of affected bins is more than 2, fully covered bins are increased by the whole value, and the value of partially affected bins are raised according to the portion of the projection.

The effects of differently chosen resolution values are visualized in Figure 2.11. By analyzing the potential S values, it is clear that three separate cases can be distinguished: $S < N$, $S = N$ and $S > N$.

In the first case, having $S < N$, the number of bins is lower than the width of the squared matrix. In this case, a single pixel could only affect the content of one bin, in some cases a maximum of two bins. By using a shorter vector, the information is compressed and memory cost is reduced.

In case the bin number is equal to the width, formally given as $S = N$, the case is the same as above: a maximum of two bins can be affected. However, it is interesting to point out that in this case, the horizontal and vertical projections are given as the sums of rows and columns.

If the chosen resolution is high, described as $S > N$, the number of bins affected by a single pixel is greater than or equal to one. As a special limit $S > \sqrt{2}N$ can be mentioned, which case a single pixel will always affect more than one bin, not influenced by the rotational angle. It is also worth mentioning that for high resolutions the projection functions will have the same curvature, as the multiple bins will have a stretching effect.

The extension of this method to $[0;\pi]$ is done by moving point P to the upper right corner, and rotating HL and LL with it respectively – or the same result could be achieved by rotating the matrix counter-clockwise, or just switch the indexing of the algorithm accordingly. The method does not need to be extended to a full circle, as the projections are equal on the $[0;\pi)$ and $[\pi;2\pi)$ sections [K2], similarly as the mirror-effect in the Radon-transform.

The resulting matrix of projected values for vehicle images is visualized in Figure 2.12. The difference compared to the Radon transform is as expected: the sinusoids of the picture edges are eliminated.

Notable, that the presented method works with squared matrices; however, a generalization to other two-dimensional rectangular inputs could be defined by small modifications to the algorithm. In the case of a $N \times M$ sized input, the lengths of the projection line is given as $\cos(\alpha)N + \sin(\alpha)M$. The function used to define the beginning and length of a given pixel projection needs to be altered likewise.

While it is possible to extend the method accordingly, based on the intended use case of the technique, $N \times N$ inputs are used in this study.

2.2.1 Memory and computational cost

Although the method provides the necessary results, the performance needs further investigation. The calculation complexity of the 4D signature defined in Section 1.2 is

$$T_1(N) = \mathcal{O}(4N^2) = \mathcal{O}(N^2), \tag{2.18}$$

because for every projection angle (horizontal, vertical, diagonal and antidiagonal) every pixel is affected exactly once. This implies that the performance is directly proportional to the image size. The runtime of the proposed method is

$$T_2(N) = \mathcal{O}(\mathit{StepNumber} \cdot N^2) = \mathcal{O}(N^2) \tag{2.19}$$

where $\mathit{StepNumber} \gg 4$, because this method is defined for multiple directions, more than 4. While the step size is greater than in the case of the 4D signature, the computational complexity of calculating a projection vector is N^2. As a result, it can be concluded that the performance of both methods depends on the image size and the number of projection directions.

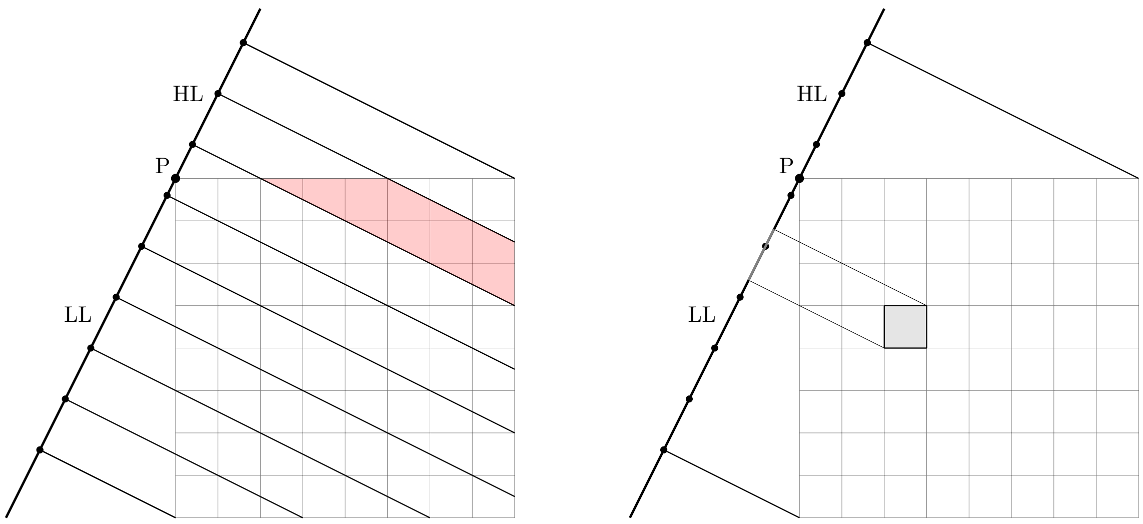
By analyzing the memory cost of the method, it can be declared that the 4D signature uses

$$2N + 2(2N - 1) = 6N - 2$$

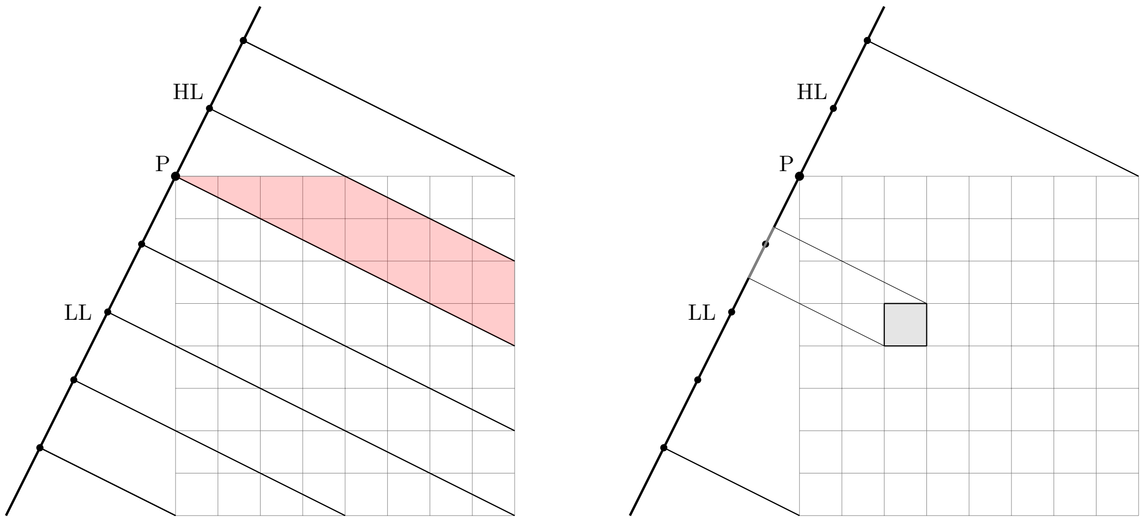
double-precision floating point numbers to store the four vectors while the presented method uses $\mathit{StepNumber} \cdot S$ doubles, where $\mathit{StepNumber} \gg 4$, $N \leq S \leq \sqrt{2}N$.

As described earlier, a chosen value for S is less than N causes the compression of the data, resulting in information loss while greater values result in redundancy.

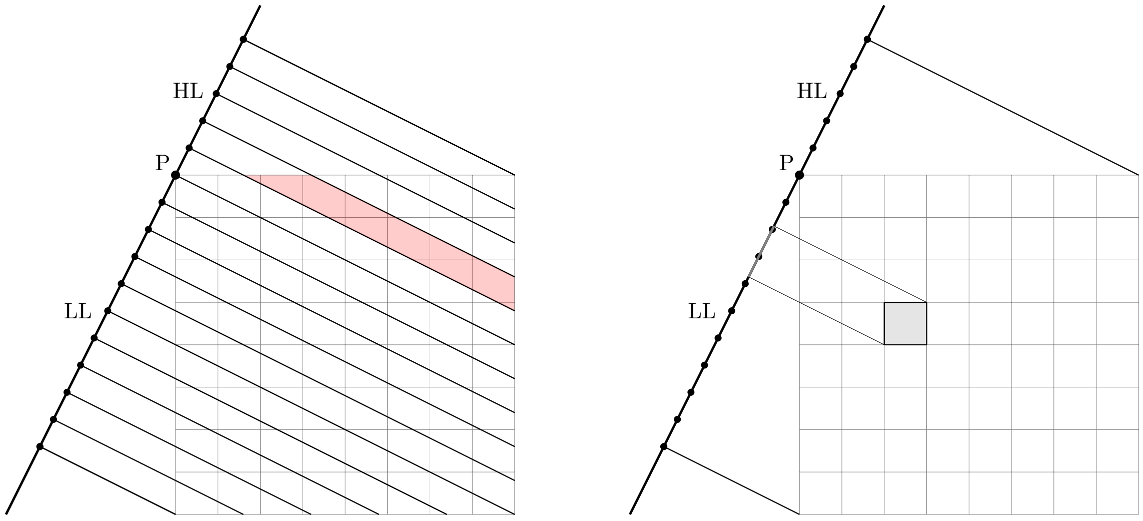
When using 4D signatures defined in [46], the longest projections are the diagonals, as seen in equation (1.4) while the horizontal and vertical sums are almost half (1.1). The difference of the vector lengths could cause difficulties storing and handling the data: instead of a matrix, an array of different length arrays should be used.



(a)

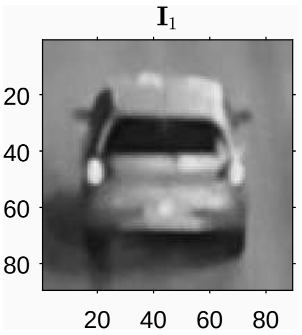


(b)

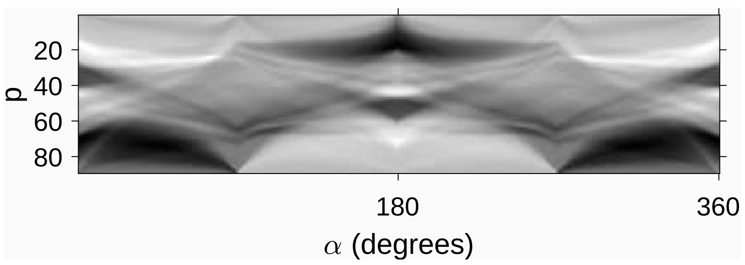


(c)

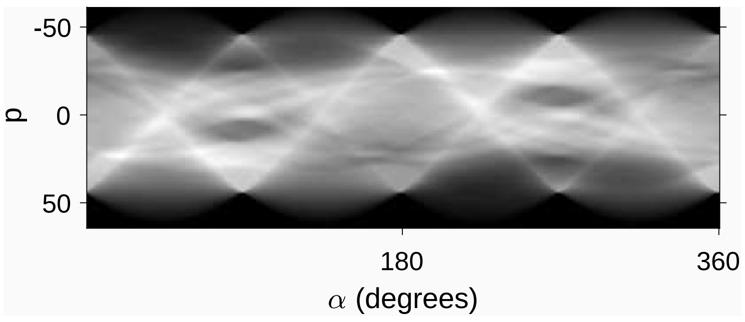
Figure 2.11. The affected pixels for different bin numbers, and the affected bins for a single pixel. (a) $S = 8$, which is equal to the width and height (N) of the matrix. (b) $S = 6$, having $S < N$. (c) $S = 15$, having $S > N$, which results in a single pixel affecting multiple bins.



(a)



(b)



(c)

Figure 2.12. A sample output of the defined method. In subfigure (a) is the sample image, and in subfigure (b) is the result of the proposed method, displayed as an intensity image, similar to the sinogram. For the sample, $S = N$ resolution was used. For reference, subfigure (c) shows the result sinogram of the Radon transform, the same as in Fig. 2.9.

Software implementations of the Radon transform and Hough transform both use a 2D matrix with the dimension $StepNumber \times \lceil \sqrt{2} \rceil$, meaning that the unused cells of the matrix are filled with empty data. The main advantage of the proposed method is that it does not store any empty values [K2], and the memory consumption is not affected by the image size or projection angle; it only depends on the predefined resolution and the sampling rate of rotation.

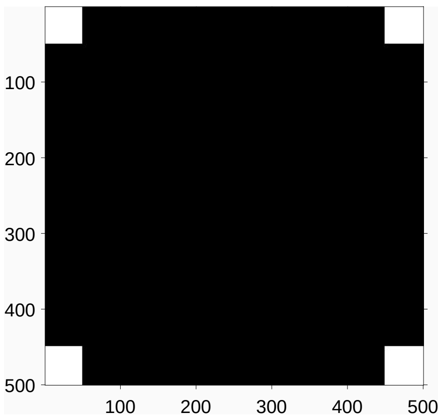
As the input matrix is not modified in the iterations, and there is no dependency between calculation steps, multi-level parallelization of the algorithm can be achieved.

2.2.2 Properties of the transformation

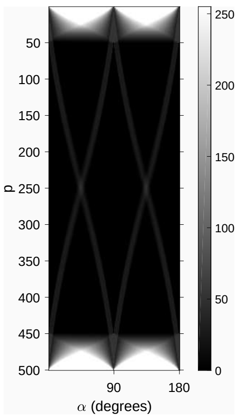
The method and the Radon-transform both define in a mapping from the two-dimensional image to its projections from multiple angles; however, some differences are present.

First of all, the so-called sinusoid effect is eliminated in case of the presented MDIPFL transformation. The cause of the sinus form of transformed points in the representation space is deducted in (2.11f). In case of the MDIPFL method, the projection lengths are not affected by the rotational angle, having a fixed function length for all elements of the projection signature.

The behavior of the corner points after transformation can be observed in Figure 2.13. It is clearly visible that the bins at the edges of the projection line are filled with pixel intensity values of corners at the edges. At the same time, the other two corners opposite to the projection line are represented as a straight line.

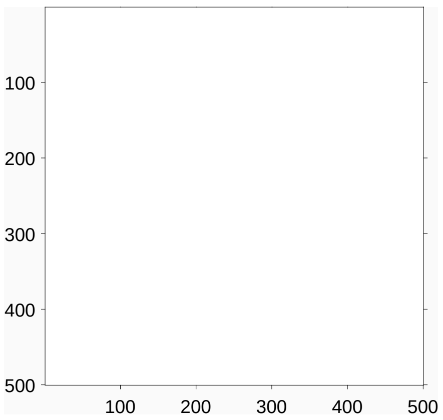


(a)

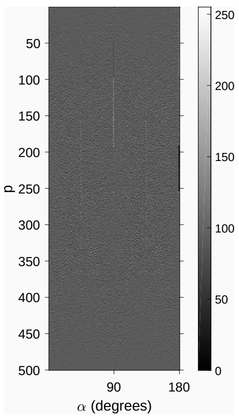


(b)

Figure 2.13. The behavior corner region points (subfigure (a)) in the transformation space of the MDIPFL method (subfigure (b)). Bin number is set to $S = N$. Note the straight diagonal lines representing the movement of the two opposite corners while the other two are projected to the start and end of the projection line.



(a)



(b)

Figure 2.14. The output of the MDIPFL method for a unit matrix type image with maximum intensity values in all positions. The image is shown in subfigure (a) and the result of the transformation is in subfigure (b).

The effects of transforming a unit matrix using the MDIPFL method results in a matrix where all values are equally 0.5, independently of the image size or the resolution.

The phenomena is visualized in Figure 2.14. It is notable, that the operations with floating point numbers cause rounding errors, which are amplified after normalization.

Horizontal or vertical translation of an object in an image also has a different effect on the result of the transformation when compared to the Radon transform. While no sinus wave is present, the horizontal movement of the object (Figure 2.15) produces two straight lines on domains $[0; \frac{\pi}{2}]$ and $[\frac{\pi}{2}; \pi]$.

The Radon transform has a well-known property of mirroring, meaning that the projections in the range of $[0; \pi)$ and $[\pi; 2\pi)$ are equal, but inverted, given in (2.7).

The MDIPFL transform has the same property: rotational angles are only relevant on the $[0; \pi)$ domain, inversion of the projection function is similarly observable.

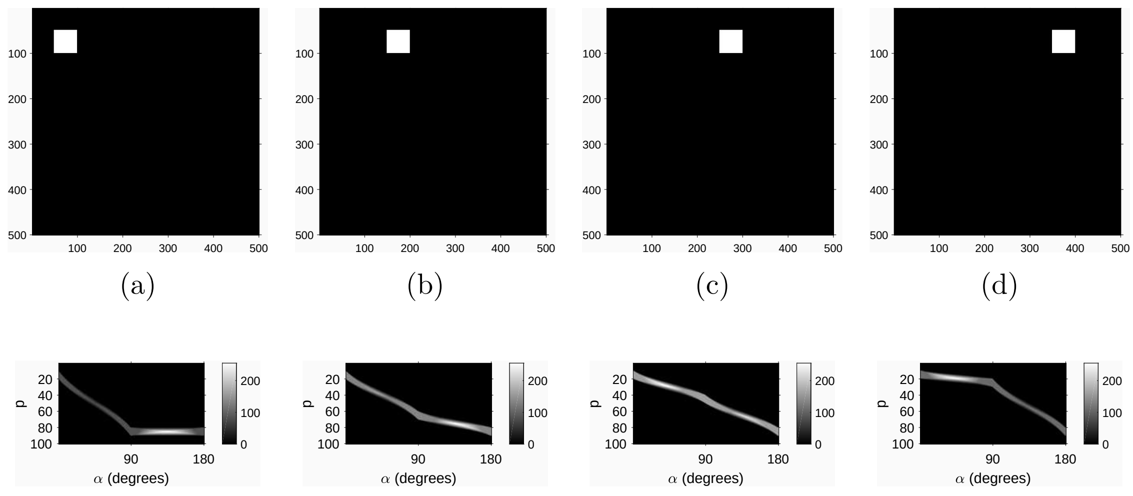


Figure 2.15. The results of horizontal translation when applying the MDIPFL transformation: subfigures (a), (b), (c) and (d) show the input image where the object is shifted to the right. Subfigures (e), (f), (g) and (h) present the result of transforms for each input image. The number of bins is set to $S = 100$.

The projection function for angle α is equal to the projection for angle $\alpha + 2n\pi$, where $n \in \mathtt{N}$, defining that the values for domain $[0; 2\pi)$ are continuously repeating.

Rotation of an object in the image resulted in a circular shifting phenomenon in the Radon space. As the presented MDIPFL method is also a mapping to projections for rotational angles, a very similar effect is visible in transformed space (Figure 2.16).

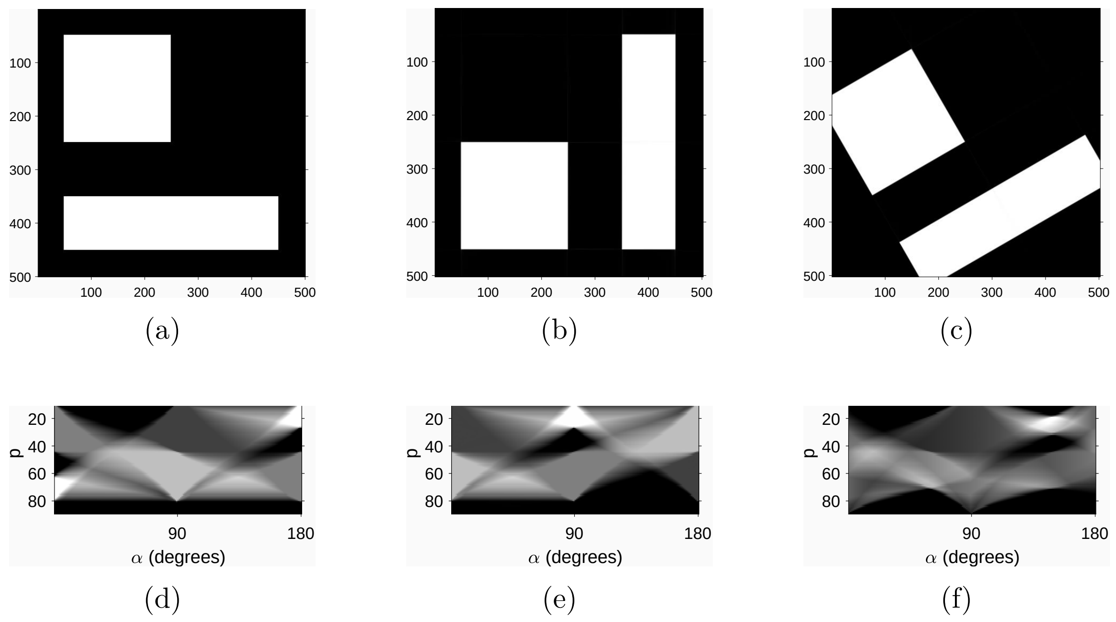


Figure 2.16. The effects of image rotation to the result of the MDIPFL transform. Subfigure (a), (b) and (c) show the input images, where (a) is the original, (b) (c) are rotated by 90 degrees and 30 degrees counter-clockwise, respectively. Subfigures (d), (e) and (f) are the visualized results of the transformation of (a), (b) and (c).

Because the resolution is fixed, the width of each bin is dependent on the projection angle. Therefore, while circular shifting of features in the transform space is visible, the actual forms are flexible.

As a conclusion, when compared to the Radon transform, the defined method has a similar computational cost with adjustable memory expense. There are similarities in the properties, and despite the differences caused by the fixed projection sizes, the transformation gives a regular description of the contents of the image.

Thesis 1.1 *I have designed and implemented a method of mapping multi-directional projection vectors using fixed bin numbers regardless of the rotation angle. The memory cost of the result is independent of the image size; it is only affected by the rotation step number and the number of bins.*

Publications pertaining to thesis: [K2], [K4], [K5].

2.3 Data-parallel implementation

The application of a graphics accelerator seems to be the best choice for parallel processing, as the calculation is based on a large number of elements, and the processing of these are more or less independent [64]. The idea of using the architecture of a massive number of processing units in graphical accelerators to solve computationally intense cases created General-Purpose computing on Graphical Processing Units (GPGPU).

The modern graphical processing units (GPU) has thousands of processing elements grouped into blocks. On a GPU device, a large amount of memory is accessible. When running a calculation on a GPU, the first step must be the transferring of the input data from the memory of the host computer to the device memory. This is the memory transfer time of initializing, which is raised with the time necessary to move the results back from the graphical processor to the memory of the hosting computer. These transfer times should be taken into account when designing the application [65]; it would be clearly wrong to try to access the memory of the host during the calculation, as it would significantly increase the runtime of the whole procedure.

Also, the understanding of memory architecture [65] of the board is crucial: the data transferred from the host is located in the global memory of the device. However, each multiprocessor also has its own on-chip memory, and the access of these blocks are performing better than addressing the global memory.

The code implemented on the GPU is referenced as a compute kernel. The design of the kernel procedures determines the performance of the solution. To achieve the best performance, optimal breakdown of the task is necessary. The aim is to use all multiprocessing units, keeping in mind that access to common variables could result in a race condition.

While there are well-known synchronization methods to deal with shared variables [66], a data-parallel approach gives the best performance in the case of a multiprocessor-based implementation, such as the GPU. To achieve such an algorithm, a thorough redesign should be done, giving special attention to the shared variables and their role in the procedure.

A naive approach to implement the calculation would be based on breaking the inner for loop into individual threads. In the case of this method (as previously given in Algorithm 1), the outer for loop moves the projection angle, and the inner for loop calculates the projections pixel-by-pixel.

To parallelize the processing of every iteration of the inner loop, $N \times N$ processing units are necessary, which is given in the case of GPUs. After the parallel computations are done on all threads, the outer loop could advance and projections could be calculated for the next angle. The drawback of this approach would be caused by the fine granularity of the problem: each thread has only a few instructions before finishing; therefore, the handling of synchronization would waste notable time.

The only possible way to increase performance is to dissolve the outer loop as well. It is notable, that creating a coarsely granulated solution by rendering multiple pixels together would still have the problem of synchronization overhead, also the efficiency could be damaged as a result of possibly unused processing elements.

The correct usage of the memory architecture [65] of the device is crucial: transfer and access times could have remarkable effects on runtime if designed badly. The main memory of the device – the global memory – could be accessed by the threads; however, the access times are better if the less accessible storages, which are assigned to the blocks (shared memory) or the even less accessible registers belonging to the threads themselves are used.

The proposed solution to achieve data-parallelism is inspired by the memory architecture of the GPU. Logical blocks are defined inside the image and, in these blocks, single threads are assigned to each pixel of the image. These threads will calculate results for every angle, individually (Alg. 2).

First, the local image parts are copied into the shared memory for each block. Memory allocations for shared variables are also done. After the initial steps, a block--level projection positioning is calculated: taking the border pixels of the block, the affected line segments are calculated for every rotational angle. The start positions of these segments are stored in the shared memory of the block.

After the dispositions are calculated, each thread of the block is assigned to each element of the matrix and the projection positions are calculated. By having the starting and closing positions of the segment on the projection line, the affected bins can be recognized. These values of the bins are then proportionally increased by the value of the actual element.

After the calculation is done, the outcomes are positioned and summarized for each angle. The results are first summarized thread-safely in the shared memory, then the results of blocks are accumulated in the global memory, from where the final results are transferred back to the host.

Algorithm 2 Kernel procedure to calculate the image projection for multiple directions.

```
procedure MDIPFL_Kernel($blk, I_G, N, S, R_G$)
    $I_S \leftarrow$ getBlock($I_G$, $blk.X$, $blk.Y$)
    $R_S \leftarrow$ new array[]
    $disp_S \leftarrow$ new array[]
    for $\alpha := 0 \to \frac{\pi}{2}$ step StepSize do
        $LL \leftarrow \cos(\alpha) \cdot N$
        $HL \leftarrow \sin(\alpha) \cdot N$
        $res \leftarrow (LL + HL)/S$
        $[start, end] \leftarrow$ positionOf($blk.X$, $blk.Y + 1$, $\alpha$)
        $disp_S[\alpha] \leftarrow \lfloor start/res \rfloor$
    end for
    for all $t \in threads$ do
        for $\alpha_L := 0 \to \frac{\pi}{2}$ step StepSize do
            $LL_L \leftarrow \cos(\alpha_L) \cdot N$
            $HL_L \leftarrow \sin(\alpha_L) \cdot N$
            $res_L \leftarrow (LL_L + HL_L)/S$
            $[start_L, end_L] \leftarrow$ gPos($blk.X$, $blk.Y$, $t.X$, $t.Y$, $\alpha_L$)
            rationalAccumulation($R_S$, $res_L$, $start_L$, $end_L$, $I_S[t.X]$)
        end for
    end for
    summarization($R_S$, $disp_S$, $R_G$)
end procedure
```

The proposed method in Alg. 2 uses all three mentioned levels from the memory architecture: indexes G, S and L indicate that the variables are stored in the global, shared or local memory, respectively. The following list contains comments and explanations for each member of the procedure:

- *blk*: image block identifier
- I_G: image in global memory
- N: image size (width & height)
- S: number of bins
- R_G: result container in global memory
- $blk.X$, $blk.Y$: coordinates of the block
- getBlock(I_G, x, y): returns the block starting at x, y from A matrix
- I_S: image in shared memory
- R_S: results in shared memory
- $disp_S$: precalculated dispositions in block memory
- α: rotation angle

- LL, HL: projection line segment lengths
- res: resolution: length of each bin
- $\textsc{positionOf}(x, y, \alpha)$: position of a block with the coordinates of x, y on the projection line for α rotation
- $start$, end: starting and ending of the projection on the projection line
- $threads$: container representing every thread on a block
- t: a single thread
- α_L: rotation of a single pixel, iterated locally
- LL_L, HL_L: projection line segment lengths used by a thread for a specific rotation
- $t.X, t.Y$: position of a pixel
- $\textsc{gPos}(bx, by, x, y, \alpha)$: returns the projection position of a pixel referred at x, y relatively to the block bx, by, for α rotation
- $start_L$, end_L: starting and ending position of the projection, handled locally
- $\textsc{rationalAccumulation}(R_S, r_L, start, end, v)$: accumulates R_S with v, having r_L resolution from $start$ to end using thread safe atomic increment
- $\textsc{summarization}(R_S, disp, R_G)$: R_S values are summarized atomically into R_G based on the dispositions $disp_S$

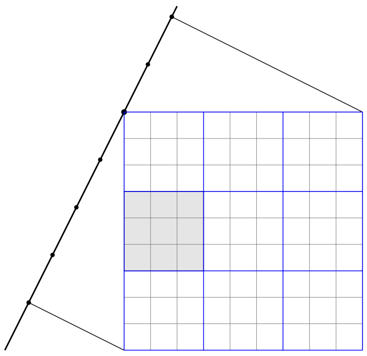
To further explain the steps of the parallel method, refer to Figure 2.17 for visual explanation.

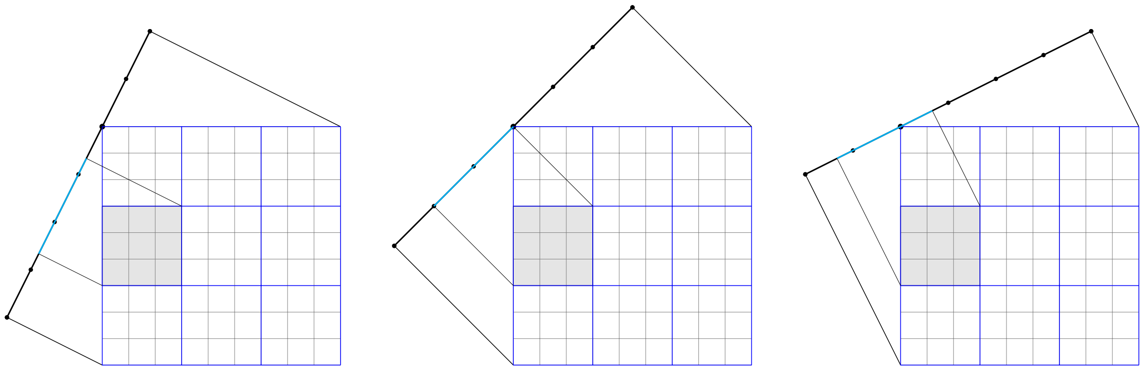
As a first step, the image is divided into logical blocks (Figure 2.17a). These blocks will be handled separately. Each block has an identifier; in the algorithm it is referred to as coordinates $blk.X$, $blk.Y$. The $\textsc{getBlock}(I_G, blk.X, blk.Y)$ returns the selected block from the image. In implementation, this method can be replaced by a function that returns the indices or boundaries of the selected region.

The selected block is copied to the block's memory, therefore reducing the on-device overhead caused by continuous addressing of the global memory. On the shared memory of the blocks, an array to store the results (R_S) and an array to store the dispositions for each rotational angle ($disp_S$) are declared.

For every rotational angle, the projection segment of the block is defined, and a disposition is stored (Figure 2.17b). It is easily understandable that projection position of a single pixel on the image can shift after continuous rotation. This is similar to the sinusoid phenomena of the Radon transform and the effect is in a relationship with distance of the pixel from the middle of the image.

The data-parallel processing is done by rendering a thread to a single pixel and calculating the projections for every angle (Figure 2.17c). These operations are independent of each other; however, R_S is a shared array for storing the results. To avoid potential faults caused by the possible race condition, an atomic operation is used for adding. It is notable, that mutual exclusion could also be used; however,


(a) Logical blocks are identified inside the image. These will be handled separately inside the GPU.


(b) For every rotational angle, the global disposition is calculated. The demonstrated angles from left to right: 30, 45 and 60 degrees.

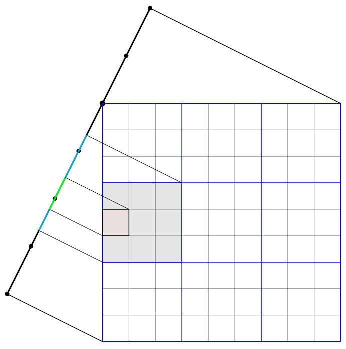

(c) Inside the blocks, the threads are assigned directly to the elements, and the calculation of their local projections is done simultaneously.

Figure 2.17. Visualized steps of the parallel method for multi-directional projections with fixed vector length.

the overhead of such a synchronization method would have a negative effect on the performance.

As a final step of the algorithm, the merging of each R_S block-level result is done into the global R_G array of results. In this step, the values in the result array are shifted by the α-dependent relevant $disp_S$ offset and added to the corresponding R_G element. As multiple blocks could access the same element simultaneously, atomic addition is applied similarly as before to avoid a race condition.

2.3.1 Results and evaluation

The sequential algorithm was implemented in the Microsoft .NET environment in C# language as a console application, and to implement the kernel of the parallel solution the NVIDIA CUDA environment was used in CUDA C [67] and C++ programming languages.

The following hardware configuration has been used during the tests:

- Processor: Intel Core i7-2600
- Architecture: Sandy Bridge
- Number of cores: 4
- Memory: 16 GB DDR2.

During the testing of the GPU-accelerated implementation, the *NVIDIA GeForce GTX Titan X* graphics accelerator was used:

- Number of CUDA cores: 3072
- Memory: 12 GB GDDR5
- Architecture: Maxwell.

The host computer of the graphics accelerator was the same workstation mentioned above.

During the tests, different image sizes from 160×160 to 1600×1600 pixels were used. In the parallel solution, the block image size was 16×16, resulting in a number of 256 parallel threads for each block. The rotation of the line was given in degrees, starting from 0 to 90, with a step size of five degrees. This resulted in a set of 19 vectors starting with the horizontal, and ending with the vertical projection.

As the results in Table 2.1 show, the difference of processing times for small images (160×160, 320×320) is not significant. The runtimes of both solutions show a linear relationship with the number of pixels, given as $\mathcal{O}(N^2)$ (Figure 2.18). However, the processors of the graphics accelerator are able to simultaneously execute multiple threads, meaning that multiple pixels are processed parallelly. The complexity of the GPU accelerated solution is still given as $\mathcal{O}(N^2)$; however, by addressing the total number of simultaneously accessible threads as t, the sequential runtime as T_1 and the possible overhead caused by memory transfers, synchronization and thread management as T_{cost}, the parallel runtime can be given as

$$T_t = \left\lceil \frac{T_1}{t} \right\rceil + T_{cost}. \tag{2.20}$$

Table 2.1. Processing of the projection calculation methods on different image sizes [K4].

Reference size (pixels)	**CPU runtime (ms)**	**GPU runtime (ms)**
160×160	93	125
320×320	363	150
480×480	821	184
640×640	1458	242
800×800	2278	303
960×960	3282	381
1120×1120	4465	485
1280×1280	5830	593
1440×1440	7381	721
1600×1600	9112	856

The visualization of T_t shows linear behavior, with discrete jumps [68] over the increase of pixel numbers. After the overhead of the preparation and collection phases is taken into account, the form becomes gradual.

The achieved speedup of t threads is generally formalized as

$$S_t = \frac{T_1}{T_t}, \tag{2.21}$$

where $t \leq N^2$.

The relatively high runtime compared to the sequential solution for small matrices is caused by the overhead of memory transfer time: while the sequential solution worked on data in-place, the parallel solution uses the device memory of the graphical processor, for which the data must be transferred from the host memory to the device memory. It is also worth mentioning that other on-device memory transfers (allocations in shared memory and copying from global device memory) also require time, which is not applicable to the sequential method.

As the computational complexity of the sequential algorithm has a squared relationship with the matrix size (as given in (2.19)), the sequential runtime show similar behavior with increasing image sizes.

The GPU-based solution is able to process multiple pixels simultaneously, having a gradually increasing effect on the runtime when increasing the image size.

An interesting evaluation of the proposed method is by benchmarking the Radon transform function of *MATLAB* and comparing the runtime with the results of the GPU implementation of the method introduced in [K4] (Table 2.2). The test inputs and parameters were the same as in the cases above.

It is assumed that the *MATLAB* function is well-defined and an optimal method is implemented to reach the highest available computational performance. It is notable that the Radon function results in more data than the method provided here; nevertheless, the runtimes are slightly different.

Thesis 1.2 *I have designed and implemented the data-parallel version of the multi-directional image projection algorithm for graphical processors, which allows acceleration proportional to the number of execution units.*

Publications pertaining to thesis: [K4], [K5].

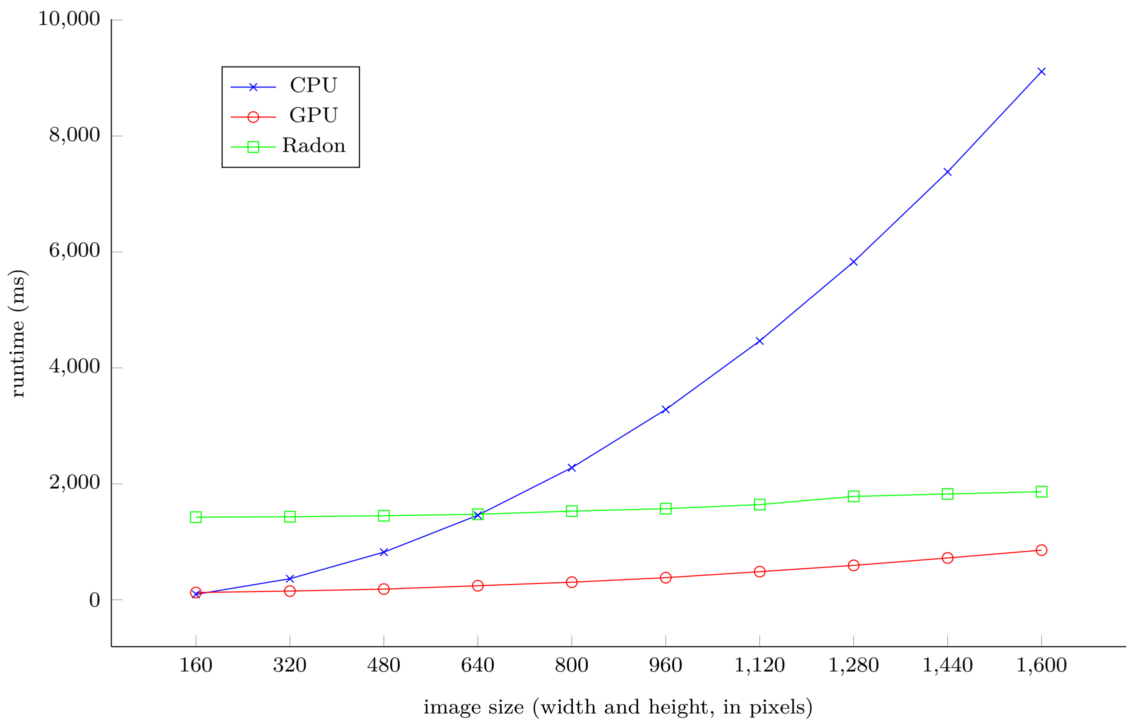


Figure 2.18. Comparison of runtimes for different image sizes [K4]. The horizontal axis displays the width of the squared images in pixels. The vertical axis shows the runtimes in milliseconds. The CPU and GPU implementations of the presented method were compared with the runtime of *MATLAB*'s GPU-accelerated Radon transform [K4].

Table 2.2. Runtime of the proposed method compared to the Radon transform function in *MATLAB* [K4].

Reference size (pixels)	**GPU (ms)**	**Radon transform (ms)**
160×160	125	1425
320×320	150	1432
480×480	184	1450
640×640	242	1475
800×800	303	1529
960×960	381	1571
1120×1120	485	1642
1280×1280	593	1783
1440×1440	721	1825
1600×1600	856	1865

2.4 Object matching using multi-directional image projections with fixed bin number

To apply the defined method for object matching, a measurement method and a similarity scoring must be defined. An adequate evaluation of the hypothesis would be to apply the previously introduced correlation-based similarity measure of each projection. The results then can be compared with the 4D (horizontal-vertical-diagonal-antidiagonal) signatures, as reference.

The dataset used for evaluating the method consists of 253 manually cropped and labeled images of 21 different vehicles. The source videos were recorded in the downtown of Budapest, at night. The artificial light sources resulted in a similar output as in the case of tunnels [4, 46]. Each image is grayscaled, and contains the object and parts of the background. Background subtraction methods and other enhancements were not applied.

The point of view of the detected vehicles are the same; the camera was directed towards the distancing vehicles. The width and height of the squared images are in average 100 pixels. Sizes vary from 48×48 to 150×150, sparsely with a few larger (200×200, 290×290) instances.

2.4.1 Two- and four-dimensional projections

For reference, the two- and four-dimensional signatures are calculated for the images, described in Section 1.2. While the method in [46] uses a two-step alignment technique, for these experiments a coarse alignment is applied using the PCC formula with a shifting technique (as given in (1.8)).

In Figure 2.19, a sample result of the 4D signature method is visualized: as the images are of different sizes, the shifting technique is applied to find the best fit over the longer function. The correlation values for the best fits are

$$\begin{aligned}
\rho_V &= \max_S \rho_l(\boldsymbol{\pi}_{V1}, \boldsymbol{\pi}_{V2}, s) = 0.909, \\
\rho_H &= \max_S \rho_l(\boldsymbol{\pi}_{H1}, \boldsymbol{\pi}_{H2}, s) = 0.956, \\
\rho_D &= \max_S \rho_l(\boldsymbol{\pi}_{D1}, \boldsymbol{\pi}_{D2}, s) = 0.697, \\
\rho_A &= \max_S \rho_l(\boldsymbol{\pi}_{A1}, \boldsymbol{\pi}_{A2}, s) = 0.954.
\end{aligned}$$

These values show a strong connection between the compared functions, even in the case of ρ_D, where the result is clearly affected by the differences. As the values are positive, the rectifier defined in (1.9) leaves them unchanged.

The similarity value μ is given as the normalized Euclidean norm of the four values as

$$\mu = \frac{\sqrt{0.909^2 + 0.956^2 + 0.697^2 + 0.954^2}}{2} = 0.885.$$

If the two-dimensional signature of row and column sums is used, the same score of similarity would be

$$\mu = \frac{\sqrt{0.909^2 + 0.956^2}}{\sqrt{2}} = 0.933.$$

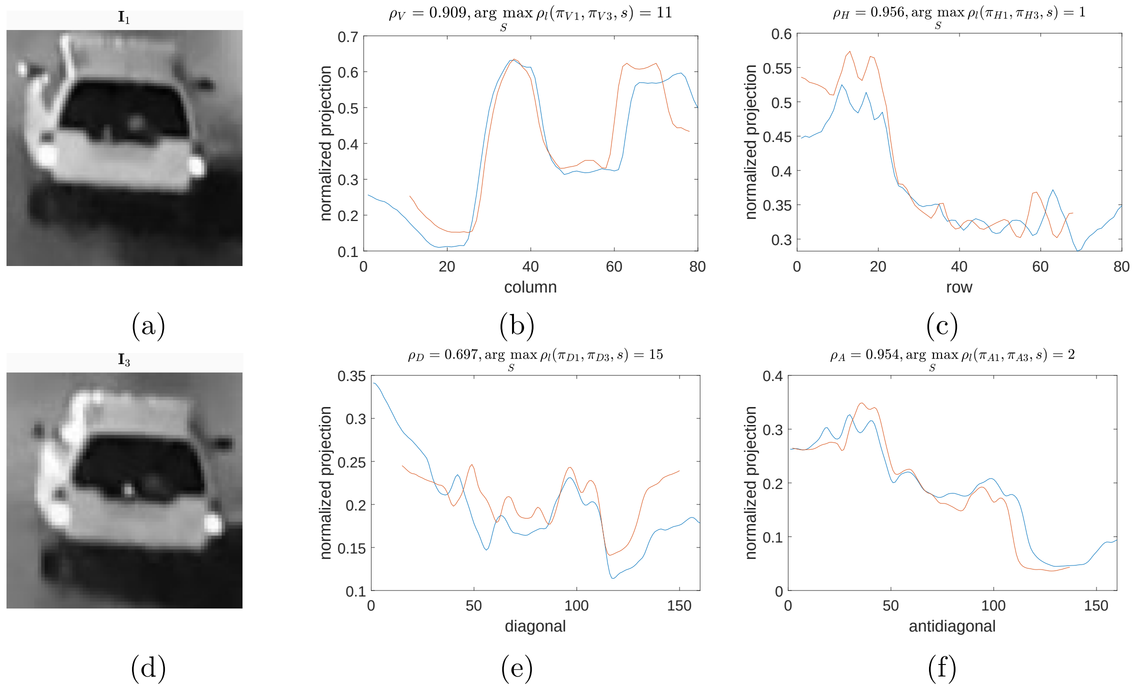


Figure 2.19. The calculated horizontal, vertical, diagonal and antidiagonal projections for different observations of the same vehicle. In subfigure (a) and (d) are images $\mathbf{I}_1$ and $\mathbf{I}_2$, in subfigures (b) and (c) the normalized vertical and horizontal projections, and in subfigures (e) and (f) the diagonal and antidiagonal projections are shown.
Note the difference caused by the blink on the left side of the vehicle on $\mathbf{I}_2$. [K5].

Note, that by using the same weights for every projection function, the effects of extremas in the similarity score are significant.

A sample output for not matching vehicles is shown in Figure 2.20. While it is clearly understandable that the two objects are not similar, the correlations are strong, having

$$\mu = \frac{\sqrt{0.687^2 + 0.908^2 + 0.796^2 + 0.944^2}}{2} = 0.840$$

for the 4D signature comparison, and

$$\mu = \frac{\sqrt{0.687^2 + 0.908^2}}{\sqrt{2}} = 0.805$$

for the 2D signature similarity.

The effect is similar to the so-called *Anscombe's quartet* datasets [69], pointing out that the Pearson correlation coefficient does not completely characterize the relationship of two functions.

The cause of high similarity scores for different vehicles can be explained by the data itself. While the vehicles are not the same, they are similar in the fact that both objects are vehicles. Therefore, the basic visual properties for vehicles are discoverable in their appearance: both have two backlights and a rear windshield and two rearview mirrors. The shadows caused by the lighting are similar as well, similarly affecting the projections.

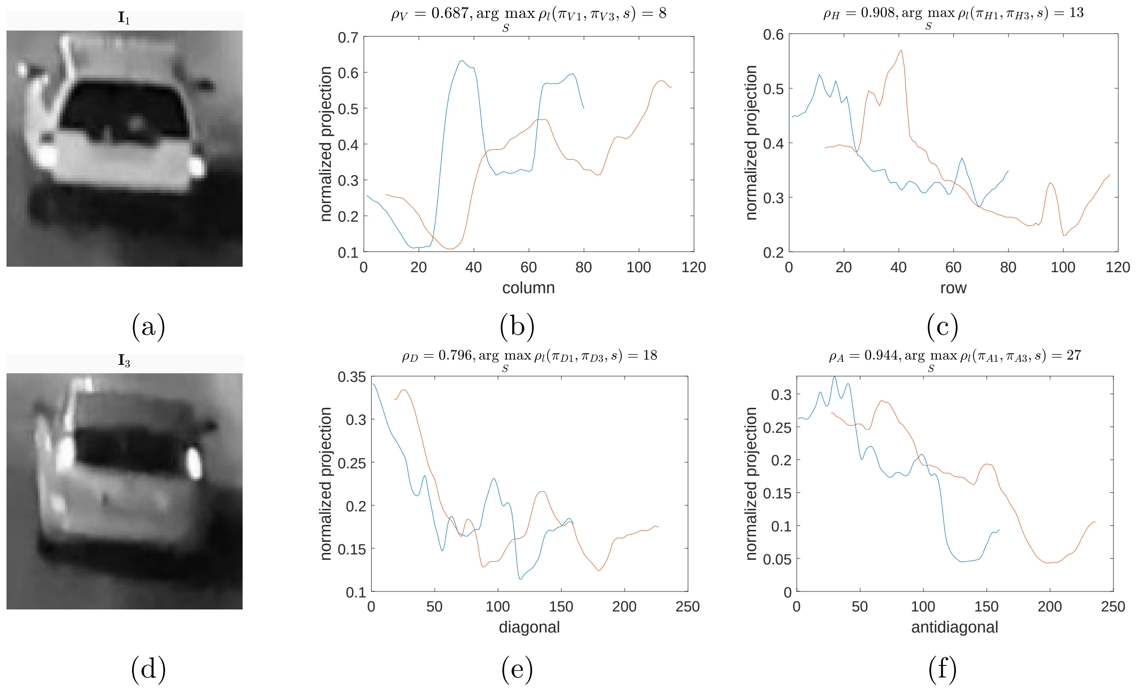


Figure 2.20. The calculated horizontal, vertical, diagonal and antidiagonal projections for observations of two different vehicles. In subfigure (a) and (d) are images $\mathbf{I}_1$ and $\mathbf{I}_3$ of the vehicles; in subfigures (b), (c), (e) and (f) the normalized vertical, horizontal, diagonal and antidiagonal projections are shown.
Note the high correlation coefficients for the clearly different functions [K5].

Other explanations can be given based on the behavior of the correlation coefficient for two very different sized functions, which could be handled by noise filtering and by rescaling the projection functions.

In Figure 2.21, the results of evaluating all available true pairs in the dataset are visualized: the horizontal, vertical, diagonal and antidiagonal projections are calculated for both observations and normalized to fit to the $[0; 1]$ interval. The correlation coefficients show a strong relationship between the functions; therefore, the calculated similarity scores are high.

It is also observable in Figure 2.21 that the lowest similarity μ for the same instances is 0.6 while the largest value is 0.98, which is quite convincing.

When comparing all different vehicles with each other, the results show great spread, represented in Figure 2.22. It is clear that although true matches have a high score, the calculated coefficients for false matches show a large spread on the complete domain of $[0; 1]$.

To further understand the distribution of similarity scores for object matching, the results for true and false pairs are visualized on a histogram in Figure 2.23.

If a simple classification is done, where it is desired that 50% of the true matches should pass, the line should be drawn to 0.82 (Table 2.3). However, using this threshold, 19.29% of the different vehicles would also pass as false positives, which is way too high.

This is caused by the high variance between the similarity values calculated for different vehicles: while the minimum value is 0.27, the highest calculated similarity is 97.69 with a 10.43% standard deviation. The application of the 2D signature

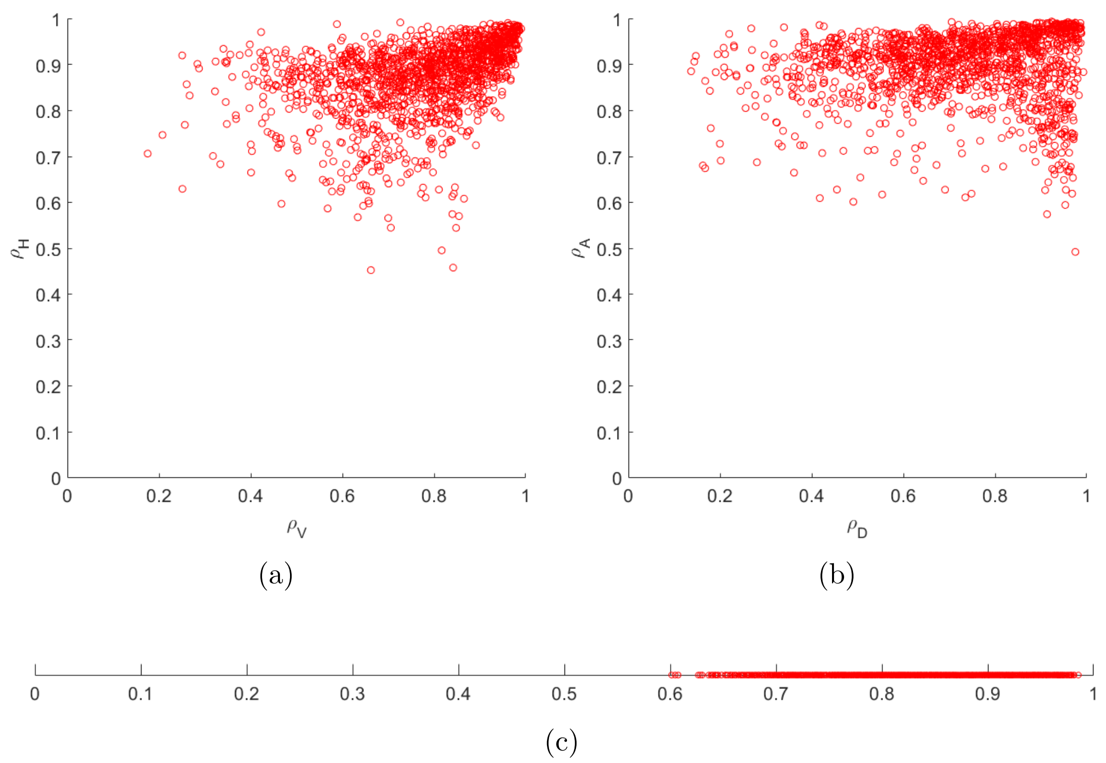


Figure 2.21. The measured correlation coefficients and calculated similarity scores in the case of true pairs using the 4D signatures. The scatter plot in subfigure (a) shows the values of ρ_H and ρ_V. In subfigure (b), the diagonal ρ_D and antidiagonal ρ_A correlations are visualized. The μ similarity is presented in subfigure (c) [K5].

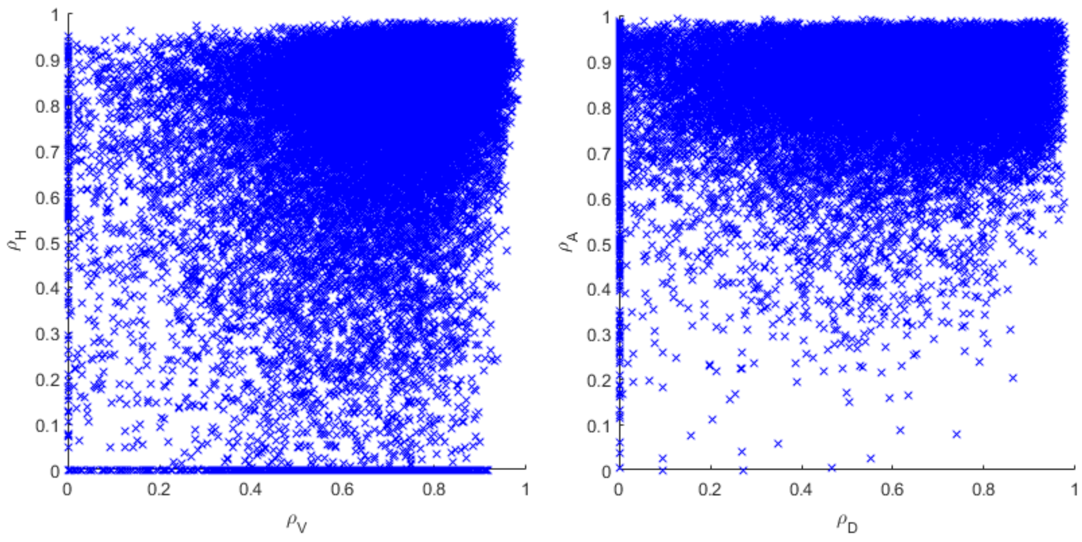
Table 2.3. Performance of the 2D and 4D projection signatures for object matching. Note that the calculated similarity score is very high for the false pairs as well [K5].

	2D	**4D**
Threshold to pass 50% of true matches	0.833	0.820
Portion of false matches above this	22.79%	19.29%
Threshold to pass 80% of true matches	0.740	0.763
Portion of false matches above this	56.75%	48.85%
Median of the similarity values of true matches	0.833	0.820
Median of the similarity values of false matches	0.760	0.761
Minimum of the similarity on true matches	0.479	0.601
Maximum of the similarity on false matches	0.978	0.976

shows the same low results: the threshold should be set to $\mu \geq 0.833$, resulting in 22.79% false positives.

It is notable that there are a number of methods to deal with high false positive rates: previously mentioned rescaling and noise removal methods could be applied to the projection functions. Having an alternative to the Euclidean norm for similarity scoring could also change the distribution of similarity scores (Figure 2.24). It is notable, that such changes would result in similar or worse distributions.

Another approach would be to take the significance of each calculated coefficient into account. A naive method to generate weights could be done by analyzing the components of the similarity score. Declare P_H^+ as a set of correlation coefficients



(a) (b)

Figure 2.22. The measured correlation coefficients in the case of false pairs using the 4D signatures. Subfigure (a) shows a scatter diagram of measured ρ_H horizontal and ρ_V vertical correlations. In subfigure (b) is a similar diagram with the diagonal (ρ_D) and antidiagonal (ρ_A) coefficients.
Note the large spread and high amount of very strong similarities. Also, it is notable that the negative correlations are removed and corresponding values are set to zero [K5].

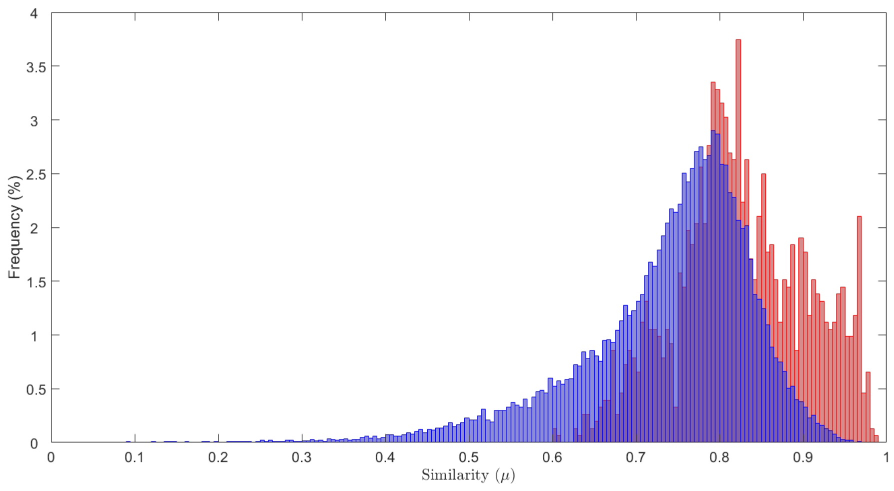
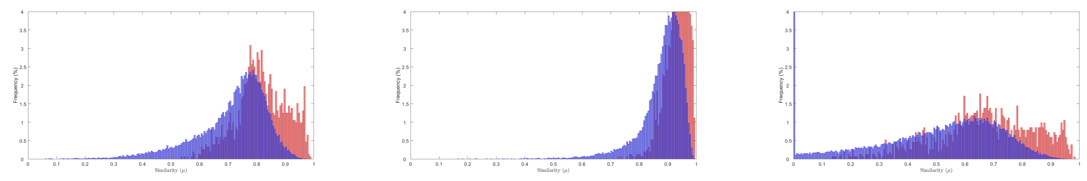


Figure 2.23. A histogram of the distribution of the similarities calculated for the same (red) and different (blue) objects [K5].

calculated from $\boldsymbol{\pi}_H$ horizontal projections for all observation pairs for the same instances. P_V^+, P_D^+ and P_A^+ are defined similarly for the correlation values calculated from the vertical, diagonal and antidiagonal projection vectors.

The averaged correlation values for true pairs are



(a) (b) (c)

Figure 2.24. Distributions of similarity scores generated by using alternatives to the Euclidean norm (Figure 2.23). In the subfigure (a) the average, in (b) the maximum and in (c) the minimum values were used from the four calculated coefficients.

$$\begin{aligned}\overline{\mathrm{P}_H^+} &= 0.8707\\ \overline{\mathrm{P}_V^+} &= 0.7675\\ \overline{\mathrm{P}_D^+} &= 0.7241\\ \overline{\mathrm{P}_A^+} &= 0.8967,\end{aligned}$$

while the similarly calculated mean of correlations for false pairs are

$$\begin{aligned}\overline{\mathrm{P}_H^-} &= 0.7510\\ \overline{\mathrm{P}_V^-} &= 0.6785\\ \overline{\mathrm{P}_D^-} &= 0.5838\\ \overline{\mathrm{P}_A^-} &= 0.8463.\end{aligned}$$

The differences of these averages are

$$\begin{aligned}\mathit{diff}_H &= \overline{\mathrm{P}_H^+} - \overline{\mathrm{P}_H^-} = 0.1197\\ \mathit{diff}_V &= \overline{\mathrm{P}_V^+} - \overline{\mathrm{P}_V^-} = 0.0890\\ \mathit{diff}_D &= \overline{\mathrm{P}_D^+} - \overline{\mathrm{P}_D^-} = 0.1403\\ \mathit{diff}_A &= \overline{\mathrm{P}_A^+} - \overline{\mathrm{P}_A^-} = 0.0504,\end{aligned}$$

from where the proposed weights could be given as

$$\begin{aligned}w_H &= \frac{\mathit{diff}_H}{\mathit{diff}_H + \mathit{diff}_V + \mathit{diff}_D + \mathit{diff}_A} = 0.2996\\ w_V &= \frac{\mathit{diff}_V}{\mathit{diff}_H + \mathit{diff}_V + \mathit{diff}_D + \mathit{diff}_A} = 0.2229\\ w_D &= \frac{\mathit{diff}_D}{\mathit{diff}_H + \mathit{diff}_V + \mathit{diff}_D + \mathit{diff}_A} = 0.3512\\ w_A &= \frac{\mathit{diff}_A}{\mathit{diff}_H + \mathit{diff}_V + \mathit{diff}_D + \mathit{diff}_A} = 0.1262\end{aligned}$$

The relatively small differences between the horizontal and vertical projection significances indicate that the overall score is affected by the error of all projections;

Table 2.4. Performance of the multi-directional projections, with bin number set as N, 2N-1, 25, 50, 100 and 300. As visible, constant bin numbers perform better [K5].

	N	**2N-1**	**25**	**50**	**100**	**300**
Threshold to pass 50% of true matches	0.819	0.819	0.881	0.875	0.873	0.872
Portion of false matches above this	19.94%	19.84%	5.06%	5.22%	5.24%	5.25%
Threshold to pass 80% of true matches	0.769	0.768	0.804	0.795	0.793	0.792
Portion of false matches above this	48.40%	48.67%	21.26%	21.64%	21.82%	21.85%
Median of the similarity values of true matches	0.819	0.819	0.882	0.875	0.873	0.872
Median of the similarity values of false matches	0.766	0.766	0.697	0.691	0.689	0.688
Minimum of the similarity on true matches	0.573	0.571	0.566	0.557	0.554	0.553
Maximum of the similarity on false matches	0.970	0.970	0.968	0.964	0.962	0.962

a single angle cannot be highlighted. As a result, the effects of weighting would be minimal. It is also important to point out that the change of data source, environmental changes, for example, the direction of lighting could prove that an approach of giving more significance to the diagonal projection over the antidiagonal is not general.

2.4.2 Multi-directional projections

When applying the defined method for multi-directional projections, several variables could be set: first, the *StepSize* between each projection angle should be set. In the experiments $StepSize = 5$ degrees ($\frac{\pi}{36}$ radian) was used.

The resolution of the projection line is also a significant tradeoff variable. Setting it to a constant number would be ideal. For example, $S = 100$ could be used, as it is the average of the image sizes, it will compress less of the data. Also some redundancy will come up on smaller images. It is also important that the runtime for the calculation of correlation coefficients shortens significantly, as no sliding window is needed, as the vector sizes are equal.

The value of S bin number could be dynamic, depending on the value of the image size. By setting it to N for all images, every projection will have a bin number aligned to the number of rows and columns. In this case, the comparison of each projection function is still necessarily done by using a sliding window.

Using a larger dynamic value for bin number reduces the effects of compression. The method with the least compression of the data is the application of $2N - 1$ resolution, which is the exact element number of the diagonal. The results for these three different settings are shown in Table 2.4.

When the proposed method is used with relative bin numbers, and results are matched with the technique described before, the pass-rate and the portion of false positives are nearly the same.

However, when using a fixed number as the projection length for all input images, results show that the number of false positives reduces significantly. For example, for bin number 25, the limit which passes through 50% of the true matches is drawn at $\mu \geq 0.881$, which is higher than the border set at the two and four-dimensional signatures. The portion of false matches above this limit is only 5.06%, which is around four times better than the false positives counted using the 2D and 4D signatures.

The distribution of similarity scores for a fixed 25 bins is visualized on a histogram in Figure 2.25. The mean average difference between these calculated similarity

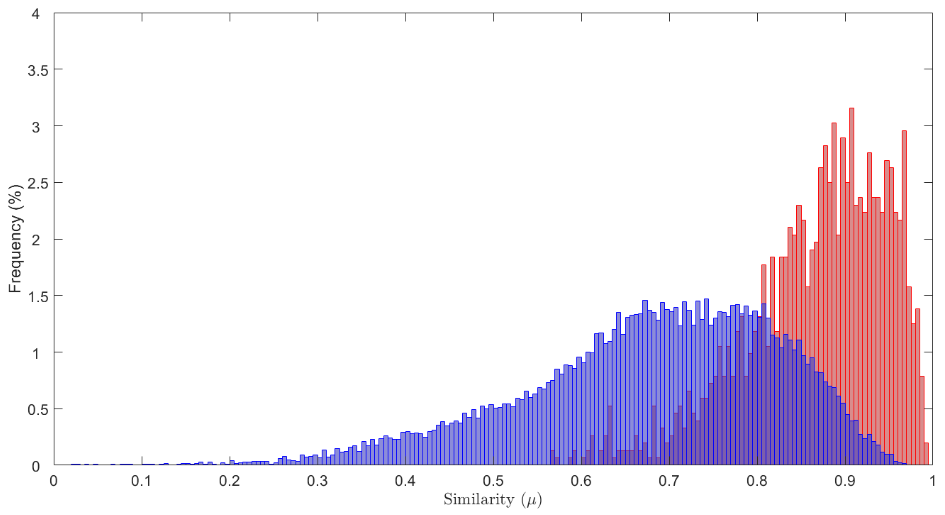


Figure 2.25. Histogram of the similarities measured using the multi-directional projection method, with a fixed bin number of 25, and *StepSize* set to five degrees. Red columns show the percentage for the comparison of the same and blue columns present the calculated values for different objects [K5].

scores and values based on the 4D signature is 0.0939.

It can be concluded, that the high false match rate is caused by the moving window: as previously discussed, the highest correlation is handled as the best fitting, which leads to high values.

It is interesting to point out that, as results show, using a higher constant bin number results in lower accuracy. This is caused by the correlation calculation, where, in the case of longer vectors, the small differences are summed and result in weaker coefficients. The phenomena can be explained by the generalization by compression effect: as the small details are removed, a tolerance to changes is developed.

As increased bin numbers result in increased memory cost, it is interesting to examine the performance for different fixed bin numbers. The quasi-optimal resolution number begins at a minimum of 10 bins for all projections (Figure 2.26). It is clear to conclude that bin numbers below this limit are affected by information shortage and, therefore, the matching algorithm is closely random.

The memory cost of each method could be given by the total number of elements of the resulting data structure. For the 2D and 4D signatures, the number of elements is $2N$ and $6N - 2$, respectively. By using a fixed bin number S, the number of elements in the case of multiple fixed projections is $StepSize \cdot S$. For the exact necessary memory the number representation must be taken into account: single-precision floating point numbers are stored in 32 bits (4 bytes), double-precision floating point numbers are stored in 64 bits (8 bytes). In these simulations, elements were represented as *double* variables.

As presented in Figure 2.27, the constant memory usage of the method based on fix bin numbers is cost-efficient in the case of larger image sizes.

For further analysis, it might be interesting to present the top false positives and false negatives of the method: in Figure 2.28a the falsely excluded vehicle pairs with the lowest similarity rate are shown, while the couples of different vehicles with the

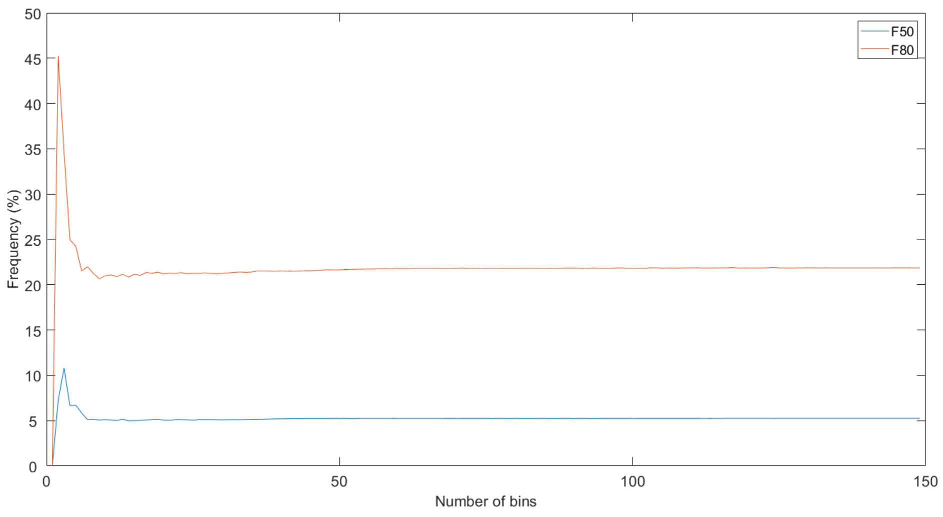


Figure 2.26. The rate of false positives if the threshold is adjusted to a limit where 50% (F50) or 80% (F80) of true matches should pass, for different number of projection bins [K5].

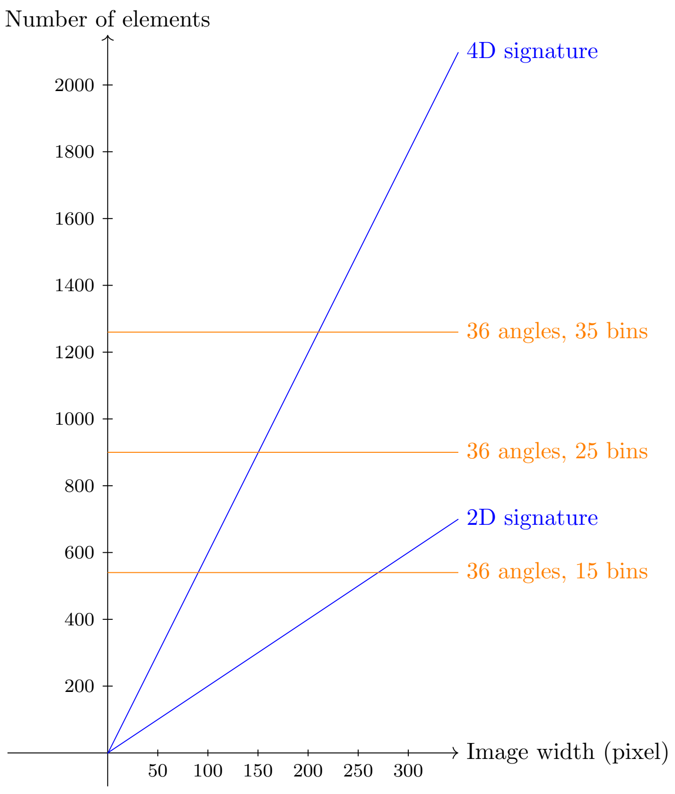
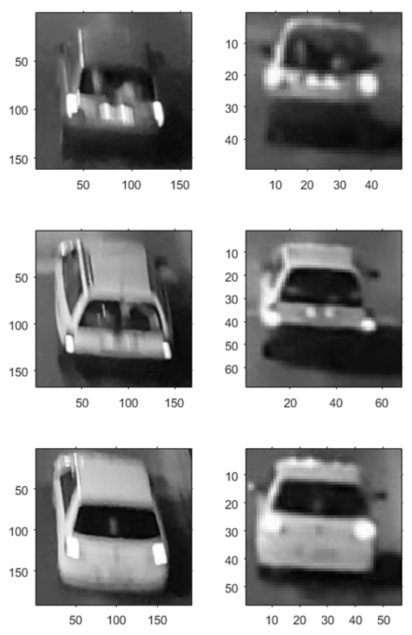


Figure 2.27. A comparison of memory cost for the 2D and 4D signatures and the multi-directional method with fixed bin numbers for different image sizes.


(a) Images of the same vehicles, detected falsely as negatives. Note the different poses and blur.

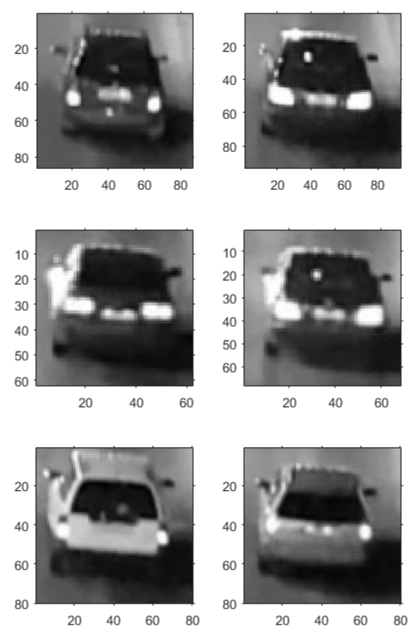

(b) Images of different vehicles, detected as positives. Note the similar blinks on the side of the cars, and that few of them are very much alike, even the same type.

Figure 2.28. A selection of various false positive and false negative pairs [K5].

highest calculated similarity are in Figure 2.28b.

As Figure 2.28a shows, the low similarity values measured for the same vehicles are mainly caused by different poses. After empirically evaluating the calculated similarities, it can be concluded that the correlation of each projection changes as the projection angle diverts from the vertical direction.

To handle the false negatives caused by the changing view, metainformation about the vehicle relative position could be used in an appearance model [70] built upon multiple views of the same instance. A multi-observation appearance model is introduced in [46] to handle the projection signature changing caused by the appearing and disappearing parts of the vehicles.

The highest false positives are caused by similar vehicles, blinks, or in few cases the same or similar type of a vehicle is falsely recognized as the same instance.

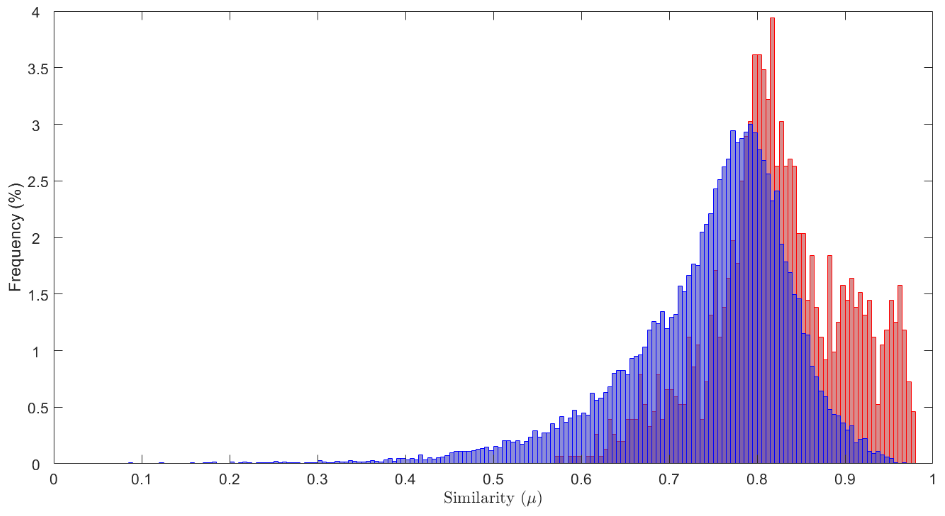
To deal with false positives, the projection matching method should be extended. Significant appearance features should be emphasized by analyzing the curvatures: there are features of the projections that are present on every rear observation of vehicles, e.g., the effects of the plate number, rear lights, etc. While these are available on every projection, the details of these specific features should decide that two signatures represent the same vehicle or not.

The Radon transform would have been a trivial choice for multi-directional projections (with varying vector lengths). For reference, the Radon transform was also applied as a feature descriptor. The method of similarity measurement is the exact same as in the case of the previously introduced 4D signature, the number of angles was extended to a total of 36, to match the parameters set for the previous experiments.

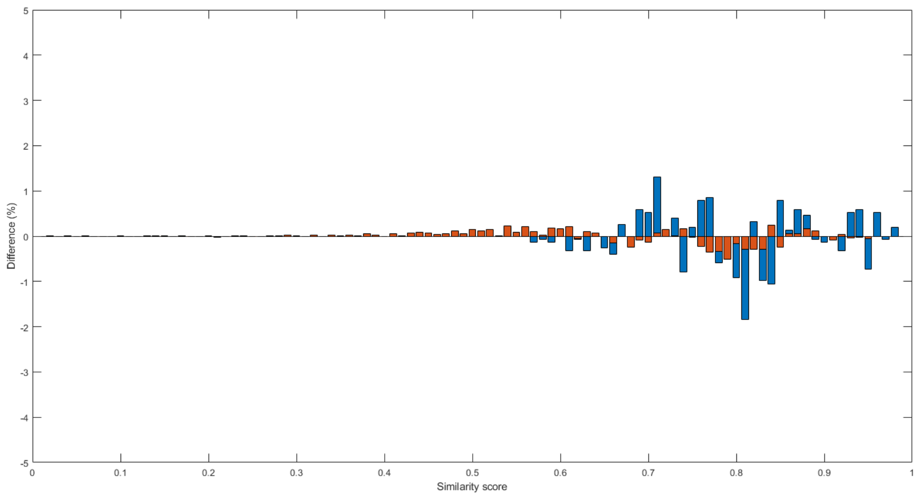
The results are summarized in Figure 2.29. As it is clearly understandable from the distribution values, the performance of the method did not improve significantly. The calculated mean average error between the 4D and the Radon based 36D signatures was 0.0261, which highlights the effect of fixed bin numbers over the application of multiple angles.

Thesis 1.3 *I evaluated the effectiveness of the fixed vector length multi-directional image projection method for object matching, comparing the results with similar projection-based, lower-dimensional image signatures, and concluded that matching accuracy increased significantly.*

Publications pertaining to thesis: [K5].



(a)



(b)

Figure 2.29. (a) Distribution of calculated similarity scores when using the Radon transform to produce multi-directional projections. The step size used for rotation was five degrees. The scores are similar to the previously introduced 4D signature-based method, visualized in Figure 2.23.
Diagram (b) shows the differences between the two histograms.

2.5 Summary

In this chapter, a novel method for multi-directional projection calculation was introduced and analyzed, which mapped the input image to its projections with a fixed number of bins. The method was inspired by the Radon transform, which was introduced in Section 2.1.

The designed method was explained in details in Section 2.2: the function was based on simple trigonometric functions. The properties of the resulting matrix showed similarities to the sinogram of the Radon transform; for example, the mirroring effect can be observed. Other properties of the representation in transform space made the method applicable as an object descriptor.

The computational complexity was in a linear relationship with the number of elements, therefore resulting in a squared relationship with image width. At the same time, the memory cost of the method only depended on the resolution setting, and the sampling rate used during rotation. In contrast, the length of each projection when applying the Radon transform was in a relationship with the image size and the projection angle.

In Section 2.3, the possibility of a parallel implementation was discussed, with a proposal of a GPU-based algorithm. The runtime of the data-parallel solution was in linear relationship with the total number of elements. The ability to use multiple processors resulted in discrete jumps in runtime, when increasing image sizes.

The possibility of small output matrices as a result of controllable parameters of the projection widths cost less memory, which had small positive effects on overhead caused by the memory transfer time.

For object matching, the application of projection functions from multiple angles provided a detailed description of the observations, improving the effectiveness of pairing. In Section 2.4 methods based and similar to the Radon transform were also tested and compared to the developed method using fix bin numbers. The effect of varying bin numbers were also analyzed, and evaluated: it was concluded that small bin numbers were adequate, resulting in a cost-efficient method.

After analyzing high similarity scores for different vehicles and low values of the same instances, it was concluded that the change in poses affected the method negatively. To deal with the changing view, the matching method should be changed to have the significant appearance features highlighted.

Chapter 3

Application of Multi-directional Projections in Siamese Convolutional Neural Networks

To deal with hyper-planes in a 14-dimensional space, visualize a 3-D space and say "fourteen" to yourself very loudly. Everyone does it.

— Geoffrey Hinton,
Introduction to Neural Networks and Machine Learning

A significant drawback of the similarity score calculation was found in the identical weighting of the correlation value of each projection. If the average of correlation coefficients is used, different angles will have the same significance, and the result is affected by rotation and translation.

An obvious approach to detect and emphasize important angles and parameters is done by applying a simple machine learning algorithm to learn the significance of each projection function, which, of course, could be developed using a multi-dimensional support vector machine.

For a more advanced solution, a deep neural network could be applied, where the input is the result of the projection transformations, represented as 2D function projections by rotation.

In this chapter, the design, implementation and evaluation of a machine learning-based approach for multi-directional projection matching is described. In Section 3.1 a brief introduction of tools and methods are given. Based on state-of-the-art techniques, a simulation is designed to analyze the applicability of projection-based descriptors. In Section 3.2, a neural architecture generation method is designed to provide the necessary structure for different signatures. As it is clear that the training and evaluation of the models on a single workstation would take weeks or months, a distributed training method is designed and explained in Section 3.3.

Finally, in section 3.4, the simulation is performed and results are collected and evaluated.

3.1 Introduction

The history of neural networks goes back to the 1940s, to the achievements of Warren S. McCulloch and Walter Pitts [71]. In the next decade, Frank Rosenblatt [72] introduced the first network, which was capable of training by examples, i.e., the first examples of supervised learning in neural networks were introduced. For a thorough summary on the history of neural networks, refer to the work of Jürgen Schmidhuber [73].

Neurons are simple connected processors: for a received input, they produce an activation based on weights. Neural networks are built by connecting these elements and structuring them in a layered architecture.

The expansion of the field and the large increase of machine learning and artificial intelligence applications are caused by deep learning. Deep learning [74, 75, 76] is basically a common name for the training of a neural network with a large number of layers, neurons, and input data. As the computational cost is increased by all of these factors, the process of training is slow. Parallel programming solutions do exist, by using widely available GPUs to execute the matrix or tensor operations [77] of backpropagation [78]. The popularity and accessibility of deep learning on GPUs both have a huge impact on what we experience today.

In deep learning, the size of the network grew with the size of the dataset. For a large number of labeled training samples, the training of the network will have a large computational and memory cost. To handle the increased hardware demands, small batches of data are used for training instead of using all samples.

The most popular approaches nowadays to classify objects in an image are based on CNNs [79]. The main idea behind the application of convolution and pooling is that the structure and neighbourhood of the input image are kept, and structural information will flow from layer to layer. In most cases, convolutional and pooling layers are used together to decrease the size of the problem, as the convolving layers extract the necessary features [80].

3.1.1 Convolutional Neural Network

It is a common term that neural networks are biologically inspired. CNNs are truly inspired by neuroscience, as the early designs of such architectures are motivated by the research of David H. Hubel and Torsten N. Wiesel.

The work of Hubel and Wiesel showed [81, 82] that the visual cortexes of cats contain neurons that are dedicated to responding to visual stimuli. These receptive fields form a visual nervous system for extensive visual recognition.

In the 1970s, Kunihiko Fukushima designed special neural networks for recognition. In his paper in 1980 [83], the Neocognitron is introduced. The multilayered structure of the network is similar to those proposed by Hubel and Wiesel. Neocognitron is capable of visual recognition of visual stimuli, independently of the position of the patterns.

The architecture of the convolutional neural network is similar to the Neocognitron; however, training of the model was different: before backpropagation, a number of methods were developed to set the weights of the network, but the robustness was missing from these approaches.

In 1989, researchers of the AT&T Bell Laboratories led by Yann LeCun proposed a method [84] for handwritten ZIP-code recognition, based on the application of backpropagation. The structure of the network is analogue to the modern understanding of CNNs [85]. These early applications [75] showed great potential, achieving practical successes in an era where neural networks were out of favour.

Since the introduction of CNNs, a number of patterns, best practices and observations of empirical research have been published [86]. These design patterns are investigated in the following.

Layer types

In a classical CNN architecture, the input layer is followed by a number of convolutional-pooling layers [87]. The convolutional layer is based on a mask in a sliding window, which is the actual element of the network containing trainable parameters and weights. The pooling layer does not learn: it is used to decrease the representation size by selecting the average or maximum values of a given region.

In some cases, the pooling layers are skipped, resulting in two or three consecutive convolutional layers followed by a single pooling layer [88].

Dense or fully-connected layers are the standard in feed-forward networks. In several cases, after a number of convolutional and pooling layer pairs the outputs are flattened, meaning that the three (or higher) dimensional structure is unfolded into a single layer of elements. Flattening is then followed by a number of dense layers, until the output layer.

Networks built strictly on convolution (and pooling), without fully connected parts are referred to as fully-convolutional nets.

Layer hyperparameters

The activation functions in hidden layers are usually rectified linear unit (ReLU) functions. Other functions, such as sigmoid or tanh, are used in the output layers to avoid the vanishing gradient problem [78].

Weight initialization methods are usually random. Zero starting values are generally not advised [89, 78].

Kernel or filter sizes of the convolutional layers are generally (but not necessarily) squared, where the width and height of the kernel is an odd number [90]. This will result in a two-dimensional activation map.

Another important parameter of these layers is the filter number: this value represents the number of activation maps to be defined. In the general structure of the convolutional network, the kernel sizes decrease layer-by-layer while the filter number increases. This is easily explained feature extraction: in the early stages, the learned features are simple edges and corners. Every following layer is able to extract more complex features, and as there are a greater number of possible features, a larger number of filters is necessary.

In the case of the pooling layers, the de facto standard is maximum pooling [90]; average or minimum pooling is not popular. The function of the pooling layer is downsampling by a given window size (this is also referenced as pool size). If pooling is done too often, it might result in losing some valuable features. So, while it is generally used, the pool sizes are smaller than the filter sizes of convolutional layers, and as previously described, they are not used as frequently as convolutional layers.

Spatial arrangement is important as well [90]. It is observable that the popular architectures use the de facto values: for pooling layers, the stride is, by default, the size of the pool size. This means that the filter is moved exactly by the size of the pool size. Another important hyperparameter is padding, where, in the case of pooling, two main techniques are present: same- and valid-padding. The general method is valid-padding, where the right-most values where the size is smaller than the pooling size are dropped. In the case of same-padding, these values are included, resulting in a larger output size. General advice is to architect a structure with valid-padding, where no cell values are dropped.

In the case of convolutional layers, the default value of stride is 1; therefore, padding is unnecessary. Interestingly, the usage of stride greater than 1 results in similar effects as the usage of a pooling layer; however, in this case, a loss of features could happen.

Number of hidden layers

Yoshua Bengio, one of the pioneers of deep learning, states in [89] that the number of hidden neurons should be "high enough"; a higher number of layers should not hurt generalization much.

In all structures, following the convolutional and pooling layers, the output is falling into fully connected (so-called dense) layers, at least one. These layers following each other have a descending neuron number; however, the pyramid-like shape is unadvised [89]; instead, gradual reduction with equal stages should be used.

3.1.2 Siamese architecture

Convolutional Neural Networks are successfully applied for visual classification problems, such as binary classification or multiclass classification. The latter is often done using a softmax function to highlight the output.

If the problem is pairing images instead of classifying them, a method based on the concept of CNN is used. The so-called Siamese Neural Network [91, 92] has two input images, whereas the output is one single value: the similarity or semantic distance of the two (Figure 3.1). The applications of face recognition systems based on this method are well known [93, 94].

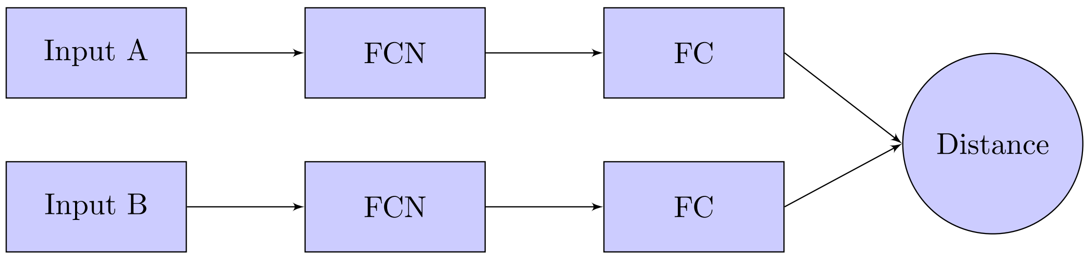


Figure 3.1. The basic structure of the "two-headed" Siamese Neural Network. The fully-convolutional (FCN) layers are followed by fully-connected (FC) layers. These heads share the same weights, and their outputs are multi-dimensional vectors. The distance of the output vectors gives the similarity of the inputs [K6].

The Siamese structure in neural networks was first introduced by Jane Bromly, Yann LeCun and associates from the AT&T Bell Labs to solve the matching of signatures [91].

The proposed architecture is based on two identical sub-networks joined at the outputs. The loss function is defined by measuring the distance between the two feature vectors. When applied, the input images of the two signatures are processed through the network, and in case the distance of features is below a predefined threshold, the signatures are accepted as pairs while those above are rejected.

In the case of object matching, a Siamese network could be used to compare input images, and give the probability of the two observations being the same instance.

An interesting fact is that the theory of one-shot learning demonstrates [95, 96] that by using a wide-enough training dataset and a wisely chosen network architecture, the network will be able to handle images never seen before. This is extremely important in real-life applications: given that the training dataset cannot and should not contain every possible observable instance, the method should be able to handle objects never seen before; in other words, the model should be able to learn differences effectively only from a small number of examples.

In summary, a short explanation of CNNs is that the network learns what an object looks like. A description on SNNs is that the network learns how to spot differences between multiple similarly looking objects.

3.1.3 Goal

As concluded at the end of Chapter 2, object matching based on multi-directional projection methods is sensitive to the similarity calculation method. Different projections having equal significance cause noise in the similarity score, which can be handled by weighting different angles of projection.

The significance of projection vectors are problem dependent; therefore, a general solution cannot be given. Instead, an approach to find the most important features in projection maps can be defined, applying a machine learning-based approach.

The goal of the research defined in this chapter is to analyze the applicability of multi-directional projection maps as object descriptors for object matching, based on Siamese architectured neural networks.

For a neural network-based comparator of structured data, a Siamese architectured CNN can be trained, and evaluated. The performance of every model greatly depends on the architecture and different hyperparameters of the training. There are methods to find an optimal set of hyperparameters; however, the case of architecture search is mostly a trial-and-error approach.

Therefore, multiple different architectures should be examined, trained and evaluated. To find multiple architectures that meet the requirements defined earlier, a generator needs to be developed.

After the architectures are generated, the training and evaluation of multiple methods should be compared. As the task of training a large number of models is very expensive, parallelizational methods need to be examined.

In summary, to prepare the experiment a Neural Architecture Generation method should be designed and developed, where the resulting Siamese models should be trained and evaluated parallelly. The interpretation and analysis of the results, that is giving the effectiveness of object matching based on projections is the final task.

3.2 Neural Architecture Generation

The efficiency of a neural network depends on the architecture and the chosen hyperparameters of the elements of the structure [76]. Data scientists and machine learning engineers struggle to find new methods to enhance the accuracy of the models [75].

For hyperparameter tuning, there are multiple methods to find the optimal setup; however, to find the best architecture, there is no exact solution.

One of the available methods is the nowadays very popular solution from the researchers of Google Brain called NASNet [97]. NASNet, the Neural Architecture Search NET, is a machine learning-based solution to find the optimal neural network architecture for a given problem. NASNet is trained on the architectures of the state-of-the-art models, and as predictions, it outputs suggested neural architectures. These architectures are then trained and evaluated, and as feedback, the validation accuracy is given back to the NASNet controller.

Because of the feedback, the NASNet is basically a Recurrent Neural Network (RNN), which generates and updates neural architectures until a defined precision is met. As this is basically an action-reward structure, it can be interpreted as a reinforcement learning solution [98].

The idea of reinforcement learning is based on the individual decision-maker, the agent, which interacts with the environment [99]. After every action, a reward (or penalty) is gained. Recent developments show [100] that a machine learning solution to generate machine learning models is actually working, which makes machine learning available to those with no knowledge of the underlying structure at all. Google Brain's AutoML [101] was introduced in January 2018 and since then, other developments have been made, introducing transfer learning [74] as well.

If the goal is to find a neural network structure and to tune the hyperparameters to find an optimal solution without any knowledge of the architecture, it is a great solution, as the result is a trained model, validated by the accuracy.

In case the task of the data scientist is to analyze the possibility of a given data preprocessing method, multiple networks and properties should be tested to be able to conclude the efficiency of the preprocessing method [K7]. For example, principal component analysis [102] could be used to reduce the input feature vector: this might result in a model that learns faster and generalizes more effectively. Also, the result could be underfitting, in the case the parameter number is lower than necessary.

The goal introduced in this section is to introduce a novel solution to recommend convolutional neural network architectures based on an algorithm, instead of using a heuristical approach. While NAS methods find the best architecture and hyperparameters by validating the model, this method suggests multiple possible setups based on the best practices of the widening field of supervised machine learning.

3.2.1 Methodology

Based on the practical recommendations in the previous section, the defined method [K8] is based on three functions:

1. defining the maximum filter and pool sizes based on the input width and height and convolutional layer number;

2. generation of layers, based on previously defined parameters while the maximum memory consumption is limited;
3. collecting a given number of model architectures (if necessary the kernel and pooling sizes are increased continuously) and finally ordering these architectures based on feature numbers and learning rate.

Window sizes

The output size of the sliding window method [90] is calculated using

$$O_W = \frac{I_W - F_W + 2P_W}{S_W} + 1, \tag{3.1}$$

where O_W represents the output width, and similarly, I_W is the input width, F_W is the filter width, P_W stands for the padding, finally S_W represents the stride. Of course, the same applies to the height of the output.

This equation stands for convolution and pooling as well. First of all, in the case of convolution, as defined earlier, the padding is zero while the stride is one. Based on this, the size of the output after the convolutional filter is

$$\mathit{ActivationMapSize} = O_W \times O_H \times F_{num}, \tag{3.2}$$

where

$$O_W = I_W - C_W + 1, \tag{3.3}$$

$$O_H = I_H - C_H + 1, \tag{3.4}$$

where C_W and C_H are the width and height of the convolutional kernel and F_{num} represents the number of filters.

If a pooling layer is applied following the convolutional layer, then the output sizes are defined as

$$O_W = \left\lfloor \frac{I_W - C_W + 1}{P_W} \right\rfloor, \tag{3.5}$$

and

$$O_H = \left\lfloor \frac{I_H - C_H + 1}{P_H} \right\rfloor, \tag{3.6}$$

where P_W and P_H represent the pooling size. Please note that the output size is still given as in (3.2) while the case of no pooling layers following the convolutional layer can be handled by setting both P_W and P_H to one.

Based on these equations and the constraints that

- convolutional kernel size should always exceed the size of the following pooling layer;
- the input sizes should always be greater than or equal to the kernel size;
- the output neuron number of the last convolutional-pooling layer pair should be larger than the element number of the following fully connected layer, but the decrease should not be significant,

the maximum sizes for the convolutional filter and pooling layers can be given. To achieve this, first, the output neuron number for a given number of convolutional and pooling layer pairs is defined.

Estimate window sizes

The idea is that if layers with the maximum filter and pooling sizes are linked after each other, the resulting output width and height gives a good approximation regarding the solvability of the problem. It is important to point out that there are two outcomes: if the output element number is too high, or one of the output dimensions length is below one. The implementation of size estimation explained in Algorithm 3 is based on exception-handling; however, other low-level control methods (e.g., negative return value) could be used the achieve the same.

Algorithm 3 Function to calculate the output size for a given input image with the size given as I_W and I_H. L_{num} is the total number of convolutional-pooling layer pairs. C_W, C_H, P_W and P_H gives the maximum sizes of the convolutional kernels and pooling windows.

```
function GETSIZE($L_{num}, I_W, I_H, C_W, C_H, P_W, P_H$)
    $O_W \leftarrow I_W$
    $O_H \leftarrow I_H$
    for $i \leftarrow 1 \dots L_{num}$ do
        $O_W \leftarrow$ FLOOR($(O_W - C_W + 1)/P_W$)
        $O_H \leftarrow$ FLOOR($(O_H - C_H + 1)/P_H$)
        if $O_W < 1$ or $O_H < 1$ then
            return 0
        end if
    end for
    return $O_W \cdot O_H$
end function
```

To get the output matrix element number after applying convolutional kernels with kernel size $C_W \times C_H$ paired with pooling filter with window size $P_W \times P_H$ on an input image sized $I_W \times I_H$, output size $O_W \times O_H$ is estimated for every layer-pair.

The values for O_W and O_H are initially set to match the size of the input and are updated in a loop iterating through the total number of convolutional-pooling layer pairs L_{num}. The assignment of O_W and O_H are based on previously given (3.5) and (3.6).

As a special outcome, the values of output window sizes could become non-positive, which indicates that the selected convolutional and pooling windows are oversized.

The goal is to get the first fitting window sizes based on the input parameters: the method to achieve this is defined in Algorithm 4.

Algorithm 4 Method to calculate the maximum window sizes for a given input image: L_{num} is the total number of convolutional-pooling layer pairs. I_W and I_H denotes the input size and r_{CP} gives the minimum ratio between convolutional and pooling window sizes. The GetSize function is defined in Algorithm 3.

```
function MaxWindowSizes($L_{num}, I_W, I_H, F_{num}, r_{CP}, FC_{size}$)
    $C_W, C_H \leftarrow 3$
    $P_W, P_H \leftarrow 1$
    $s \leftarrow \infty$
    while $s > FC_{size}$ do
        if not $(P_W > P_H)$ and $(C_W < P_W \cdot r_{CP})$ or $(C_H < P_H \cdot r_{CP})$ then
            if $C_W > C_H$ then
                $C_H \leftarrow C_H + 2$
            else
                $C_W \leftarrow C_W + 2$
            end if
        else
            if $P_W > P_H$ then
                $P_H \leftarrow P_H + 1$
            else
                $P_W \leftarrow P_W + 1$
            end if
        end if
        $s \leftarrow$ getSize$(L_{num}, I_W, I_H, C_W, C_H, P_W, P_H) \cdot F_{num}$
    end while
    return $C_W, C_H, P_W, P_H$
end function
```

The idea is based on approximation from above: for a given input image size $I_W \times I_H$, the increase of convolutional kernel $(C_W \times C_H)$ and pooling window $(P_W \times P_H)$ sizes results in reduced output element number s. The goal is to select the highest quartet of window dimensions C_W, C_H, P_W, P_H where the output element number is still valid, i.e., non-zero.

The initial values of convolutional kernel window size are set to 3, while pooling window sizes are set to 1. Note, that application of a pooling window with size 1×1 would result in an output with the exact same size as the input.

As the algorithm is looking for optima of window dimensions where the value of s is minimal, the initial value of s is set to positive infinity.

The main loop is based on the condition $s > FC_{size}$, where FC_{size} is the minimal element number of the fully-connected dense layer following the convolutional parts. If the condition is met, a possible increase in the parameters could be done.

Based on the relation of convolutional and pooling window sizes referred to as r_{CP}, eighter C_W or C_H, eighter P_W or P_H is increased. If changed, the convolutional window width and height are always expected to be an odd value, thus it is increased by two.

After changing the parameters, the previously described GetSize function (Algorithm 3) is called to calculate the number of output elements in the proposed setup. This element number is multiplied with the minimal number of filters F_{num}, giving the new value of s. Note, that the result of the function is zero, if the application of defined window sizes is invalid.

The steps are repeated until s is minimal or invalid, and the previously selected parameters are passed as the final results. These are the four values for the maximum kernel and pool sizes, widths and heights, respectively. It is important to point out, that this method enables the window sizes to be non-squared, which could be useful in the case of non-squared inputs.

Finding possible architectures

To generate the actual model architectures, first all possible convolutional and pooling layer pairs are generated into a collection. Basically, this means that for every possible convolutional kernel size every possible pool size is selected and stored.

Based on the possible layer pairs and L_{num} (the number of total layer pairs in the target architecture), all possible architectures can be generated. It is notable, that the scale of possible architectures has an exponential relation to the number of possible layer pairs, so the examination of every single architecture is not recommended.

In this solution, a special backtrack algorithm [103] is applied to find all possible architectures. In this method, described in Algorithm 5, the classical backtracking is extended with the collection of every output, which satisfies all the necessary conditions.

Algorithm 5 Searching for available solutions using a recursive backtracking search with multiple results.

```
procedure BackTrack($level, R, All$)
    $tmp \leftarrow$ AllPossibleConvPoolPairs()
    for $i \leftarrow 1 \ldots$ SizeOf($tmp$) do
        if $level = 1$ or NoCollisions($tmp[i], R[level-1]$) then
            $R[level] \leftarrow tmp[i]$
            if $level =$ SizeOf($R$) then
                if FinalCheck($R$) then
                    Add($All, R$)
                end if
            else
                BackTrack($level + 1, R, All$)
            end if
        end if
    end for
end procedure
```

Backtracking search is a recursive algorithm that collects possible sub-solutions to an array R. In this case, possible sub-solutions are convolutional-pooling layer parameters, which are in advance collected to a temporary set *tmp*.

For a network architecture of L_{num} consecutive convolutional-pooling layer pairs, exactly L_{num} elements should be tested against each other. These elements are referred to as *level* from 1 to L_{num}, which is also the size of the array R.

At the very start, the method tries to fit a possible convolutional-pooling pair to the first layer.

The benefit of using a backtracking algorithm is that after applying the first convolutional-pooling layer pair, every other layer is evaluated after adding, and if a so-called collision appears, the sub-solution on that given level is rejected, and will not be examined again in the same state. It is important to point out that a greedy approach would check every possible solution, wasting steps on invalid architectures.

For collision, the following (previously announced) properties are defined:

- the width or height of the convolutional filter following the last one cannot be greater than the previous convolutional filter;
- the number of convolutional filters cannot decrease;
- the size of the pooling window cannot increase.

If all conditions are satisfied, then no collisions are given.

Using the backtracking algorithm, the layer pairs are added after each other, and if the last empty position is fitted, the actual architecture could be stored as a possible solution; however, before that, a final check is done. This final check basically verifies the output model size, similarly as defined in Alg. 3, but instead of having the maximal values, the actual values are used.

Other properties to validate the generated architecture are also applied:

- a general rule is that whenever the output of a layer (convolutional or pooling) is squared, every following layer's window must be squared;
- in the case of pooling layers, every architecture is invalidated, where after applying the pooling layer, the size of the input is not the exact product of the pooling window dimensions and the output dimensions. As stated before, valid-padding, where cell values are dropped, is unadvised.

The last step of validation is based on the memory consumption of the model during training.

Model size estimation

While the memory consumption and training time of a model is not important in production, to compare input data and to evaluate preprocessing methods, the actual size of the parameter space and the training time are both important descriptors of the method.

To provide equal conditions an upper limit is given for the model sizes, and this upper limit is closed on with the manipulation of the training batch size.

As a rough estimation of the memory consumption of the model, the total memory cost is defined as

$$Cost = 4 \cdot TotalParameterNumber \cdot BatchSize, \tag{3.7}$$

where the *TotalParameterNumber* is the total number of weights and biases in the model. The explanation of the multiplier 4 is that these values are stored as floating point numbers on 32 bits, which means 4 bytes for every parameter.

The weight and bias variable number of the convolutional layers are trivially given by

$$ParamNum_{CONV}^{(i)} = F_{num}^{(i)} \cdot W \cdot H \cdot F_{num}^{(i-1)} + F_{num}^{(i)}, \tag{3.8}$$

giving the number of parameters for layer i from the number of filters in the actual layer multiplied with the size of the kernel, and multiplied with the filter number of the previous layer. The number of biases is equal to the number of filters, which is added to the sum. It is notable, that the size of the input image does not affect the number of trainable parameters.

There are no trainable parameters in pooling layers, so these are skipped during the estimation. The last elements of the network are in fully connected layers, where the parameter number is given as

$$ParamNum_{FC}^{(i)} = N_{num}^{(i)} \cdot N_{num}^{(i-1)} + N_{num}^{(i)}, \tag{3.9}$$

where $N_{num}^{(i)}$ stands for the number of neurons in the actual layer, whereas $N_{num}^{(i-1)}$ gives the same in the previous layer. The total number of weights is given by the product of the two numbers while the number of bias values are equivalent to the number of neurons.

To get the parameter number for the first dense layer, the neuron number of the last convolutional layer has to be defined. As the output matrix with dimensions $O_W \times O_H \times F_{num}$ is flattened, the result is a vector of processing elements with this length.

In the case of a fully connected layer and large neuron numbers, this can result in large memory consumption. For example, if a layer of 4096 neurons is followed by a layer with 2048 neurons, the total memory consumption calculated only for the connections between these two layers will result in 32 MB.

To find the optimal batch size, the following algorithm was designed (Algorithm 6): as the last step of validation, the model size is estimated, based on (3.7). If the size exceeds the limit, the current architecture is invalidated.

Algorithm 6 The estimation of the ideal batch size for training. The minimum number of training batches is defined as 10; however, this could depend on the number of training samples in a single epoch. Function RoughEstimation results in the number of bytes defined in (3.7).

```
function BatchSizeEstimate($max\_mem, min\_batch$)
    $b \leftarrow min\_batch$
    if RoughEstimation($b$) $< max\_mem$ then
        while RoughEstimation($b + 2$) $< max\_mem$ do
            $b \leftarrow b + 2$
        end while
        return $b$
    else
        return 0
    end if
end function
```

The input parameters for function BatchSizeEstimate are the upper boundary for memory consumption *max_mem*, and the minimal number of training samples in a single batch given as *min_batch*. Supplementary function RoughEsti-

MATION gives the memory cost of the model with a given total parameter number in bytes, according to (3.7).

Variable b is initially set to match the minimal number. In case the estimation with the minimal batch size results in a memory cost above the limitation, the architecture is rejected. Note, that in Algorithm 6, the signaling of rejection is done by the return value 0; however, other implementations based on exception handling or events are also feasible.

If the size is acceptable, the batch size is increased step-by-step to approach the limit, until it is exceeded. When this happens, the last valid value for b size is used as output.

Sorting results

As the last step, the eligible architectures should be sorted by pooling sizes increasingly, and by having batch sizes ordered decreasingly. This idea is based on the fact that pooling may unnecessarily decrease the feature number while, of course, a large batch number means that the training time is decreased significantly.

It is important to point out, that in order to have an acceptable number of architectures, the implemented solution increases the previously calculated maximum window sizes if the necessary architecture number is not met at the end of an iteration. Of course, a similar effect could be reached by increasing the initial values of Algorithm 4.

3.2.2 Results

As a proof of concept, the results of the method for a few input sizes are hereby demonstrated and analyzed.

First, the resulting architectures for an RGB input image sized 100×100 for a total number of convolutional-pooling layer pairs are described in List 3.1. As the input is an RGB image, the pixel information can be represented as a 3-dimensional matrix, where the third dimension gives the three planes of red, green and blue intensity values.

Note the varying convolutional kernel sizes: in the first four architectures, the first layer starts with a 9×9 kernel with different settings for the rest of the architectures. The last six architectures start with a larger 13×13 kernel followed by similarly large windows. This results in a smaller number of output neurons after flattening; therefore, the parameter number between the first dense layer is smaller, resulting in less memory cost, allowing a larger batch size.

Results show, that the algorithm rewards those structures, where convolutional and pooling kernel sizes are as low as possible, and only increases these window sizes, if there are no other possible solutions. Given so, if the kernel size is increased, the ordering is given by the batch sizes, which results in less time for epoch processing.

In the case of a non-squared input, the algorithm allows the usage of non-squared windows as well. For an input with a size of $100 \times 50 \times 1$, a few resulting architectures are described in List 3.2.

Note the different approaches of the algorithm to reduce the size of the representation: in some cases, a kernel window with a slight difference in window sizes is used to slightly reduce the difference between the width and height of the output

Input(100×100×3) → Conv(9×9@64) → Pool(2×2) → Conv(7×7@64) → Pool(2×2) → Conv(7×7@64) → Pool(2×2) → Flatten() → FC(2048), batch size: 18

Input(100×100×3) → Conv(9×9@64) → Pool(2×2) → Conv(7×7@64) → Pool(2×2) → Conv(5×5@64) → Pool(2×2) → Flatten() → FC(2048), batch size: 14

Input(100×100×3) → Conv(9×9@64) → Pool(2×2) → Conv(7×7@64) → Pool(2×2) → Conv(7×7@64) → Pool(2×2) → Flatten() → FC(2048) → FC(1024), batch size: 14

Input(100×100×3) → Conv(9×9@64) → Pool(2×2) → Conv(7×7@64) → Pool(2×2) → Conv(5×5@64) → Pool(2×2) → Flatten() → FC(2048) → FC(1024), batch size: 12

Input(100×100×3) → Conv(13×13@64) → Pool(2×2) → Conv(13×13@64) → Pool(2×2) → Conv(13×13@64) → Flatten() → FC(1024), batch size: 54

Input(100×100×3) → Conv(13×13@64) → Pool(2×2) → Conv(13×13@64) → Pool(2×2) → Conv(13×13@128) → Flatten() → FC(1024), batch size: 30

Input(100×100×3) → Conv(13×13@64) → Pool(2×2) → Conv(13×13@64) → Pool(2×2) → Conv(11×11@64) → Flatten() → FC(2048), batch size: 22

Input(100×100×3) → Conv(13×13@64) → Pool(2×2) → Conv(13×13@64) → Pool(2×2) → Conv(13×13@128) → Flatten() → FC(2048), batch size: 20

Input(100×100×3) → Conv(13×13@64) → Pool(2×2) → Conv(13×13@128) → Pool(2×2) → Conv(13×13@128) → Flatten() → FC(1024), batch size: 20

Input(100×100×3) → Conv(13×13@64) → Pool(2×2) → Conv(11×11@64) → Pool(2×2) → Conv(11×11@64) → Flatten() → FC(2048), batch size: 18

List 3.1. The top 10 result architectures for input sizes $100 \times 100 \times 3$, for a maximum number of three convolutional and pooling layer pairs. The arguments given for Conv and Pool represent the window sizes for the given layer. In the case of Conv layers, the third number represents the filter number [K8].

activation map. In other cases, non-squared pooling is used to significantly reduce the contrast between the sides of the output.

The algorithm will always privilege to increase the size of the convolutional kernel while the pooling windows are kept as small as possible. It is interesting to point out that the method aims to decrease the difference between the dimensions in the output of the layers.

It is notable, that in the top five resulting architectures, the filter number keeps constant through layers. The cause of this effect is that by increasing the number of filters, the parameter number and the memory usage increases, which is penalized through the ordering by batch sizes. Therefore, it is suggested that a higher number of these generated architectures should be applied for the original cause, to compare preprocessing methods.

Thesis 2.1 *I have developed a method based on backtracking search that provides all of the suitable convolutional neural network architectures at a given input, layer number, and memory cost.*

Publications pertaining to thesis: [K8].

Input(100×50×1) → Conv(37×35@64) → Pool(4×4) → Flatten() → FC(2048), batch size: 14

Input(100×50×1) → Conv(37×35@64) → Pool(4×4) → Flatten() → FC(2048) → FC(1024), batch size: 12

Input(100×50×1) → Conv(37×39@64) → Pool(4×3) → Flatten() → FC(2048), batch size: 14

Input(100×50×1) → Conv(37×39@64) → Pool(4×3) → Flatten() → FC(2048) → FC(1024), batch size: 12

Input(100×50×1) → Conv(37×39@64) → Pool(4×4) → Flatten() → FC(2048), batch size: 20

Input(100×50×1) → Conv(13×15@64) → Pool(2×2) → Conv(13×15@64) → Pool(2×1) → Flatten() → FC(2048), batch size: 14

Input(100×50×1) → Conv(13×15@64) → Pool(2×2) → Conv(13×15@64) → Pool(1×2) → Flatten() → FC(2048), batch size: 14

Input(100×50×1) → Conv(13×15@64) → Pool(2×2) → Conv(13×15@64) → Pool(2×1) → Flatten() → FC(2048) → FC(1024), batch size: 10

Input(100×50×1) → Conv(13×15@64) → Pool(2×2) → Conv(13×15@64) → Pool(1×2) → Flatten() → FC(2048) → FC(1024), batch size: 10

Input(100×50×1) → Conv(13×15@64) → Pool(2×2) → Conv(13×15@64) → Pool(2×2) → Flatten() → FC(1024), batch size: 46

Input(100×50×1) → Conv(9×11@64) → Pool(2×2) → Conv(9×11@64) → Conv(9×9@64) → Flatten() → FC(2048), batch size: 14

Input(100×50×1) → Conv(9×11@64) → Pool(2×2) → Conv(9×11@64) → Conv(7×9@64) → Flatten() → FC(2048), batch size: 14

Input(100×50×1) → Conv(9×11@64) → Pool(2×2) → Conv(9×11@64) → Conv(9×9@64) → Flatten() → FC(2048) → FC(1024), batch size: 12

Input(100×50×1) → Conv(9×11@64) → Pool(2×2) → Conv(9×11@64) → Conv(7×9@64) → Flatten() → FC(2048) → FC(1024), batch size: 12

Input(100×50×1) → Conv(9×11@64) → Pool(2×2) → Conv(9×11@64) → Pool(2×1) → Conv(9×9@64) → Flatten() → FC(1024), batch size: 60

Input(100×50×1) → Conv(11×9@64) → Pool(1×2) → Conv(11×9@64) → Conv(11×9@64) → Conv(7×5@64) → Flatten() → FC(2048), batch size: 14

Input(100×50×1) → Conv(11×9@64) → Pool(1×2) → Conv(11×9@64) → Conv(9×7@64) → Conv(9×7@64) → Flatten() → FC(2048), batch size: 14

Input(100×50×1) → Conv(11×9@64) → Pool(1×2) → Conv(11×9@64) → Conv(11×9@64) → Conv(7×5@64) → Flatten() → FC(2048) → FC(1024), batch size: 10

Input(100×50×1) → Conv(11×9@64) → Pool(1×2) → Conv(11×9@64) → Conv(9×7@64) → Conv(9×7@64) → Flatten() → FC(2048) → FC(1024), batch size: 10

Input(100×50×1) → Conv(11×9@64) → Pool(2×2) → Conv(9×9@64) → Conv(9×7@64) → Conv(9×7@64) → Flatten() → FC(1024), batch size: 60

Input(100×50×1) → Conv(9×11@64) → Pool(2×1) → Conv(9×11@64) → Conv(9×11@64) → Conv(9×11@64) → Conv(9×9@64) → Flatten() → FC(1024), batch size: 38

Input(100×50×1) → Conv(9×11@64) → Pool(2×1) → Conv(9×11@64) → Conv(9×11@64) → Conv(9×11@64) → Conv(7×9@64) → Flatten() → FC(1024), batch size: 36

Input(100×50×1) → Conv(9×11@64) → Pool(2×1) → Conv(9×11@64) → Conv(9×11@64) → Conv(9×11@64) → Conv(7×9@64) → Flatten() → FC(2048), batch size: 22

Input(100×50×1) → Conv(11×9@64) → Pool(2×1) → Conv(11×9@64) → Conv(11×9@64) → Conv(11×9@64) → Conv(11×9@64) → Flatten() → FC(2048), batch size: 16

Input(100×50×1) → Conv(9×11@64) → Pool(2×1) → Conv(9×11@64) → Conv(9×11@64) → Conv(9×11@64) → Conv(7×9@64) → Flatten() → FC(2048) → FC(1024), batch size: 16

List 3.2. The top five result architectures for input sizes $100 \times 50 \times 1$, for a maximum number of 1, 2, 3, 4 and 5 convolutional and pooling layer pairs, respectively [K8].

3.3 Distributed training

To be able to decide whether a preprocessing method is effective or not, neural networks should be trained and evaluated on the same ruleset for each type of input data, including the raw input data for an end-to-end approach.

This would require a large number of model training and evaluation. As it is well known, the training of a single model could require large computational power, which is nowadays handled by GPUs.

The simulation, of course, could be done on a single workstation, by processing models one after the other. However, the runtime of the simulation could be remarkably reduced by using a distributed environment with a cluster of GPU enabled workstations.

3.3.1 Master/Worker pattern

The Master/Worker pattern [66] is a good choice when the parallelizable tasks themselves are executed in a message-passing environment. In this case, each process of the model training and evaluation require a single workstation with an eligible graphical accelerator, no shared memory parallelization is available.

One of the key points of the Master/Worker pattern is that the Master can generate the tasks while the Workers are already processing them from the bag of tasks. However, in this case, the task generation can be separated from the functions of the Master.

The main advantages of using a Master/Worker pattern is that the load balancing of tasks is automatic: when a Worker finishes, it sends the results to the Master and asks for a new task, if possible.

The scheduling of the tasks is important to achieve an efficient load-balance: in the case of the Master/Worker pattern, the Worker instances ask for the next job, when the processing of the previous task is finished. If the jobs are served in decreasing order by processing times, the last jobs will take less time resulting in a finely granulated distribution. This is called the Longest Processing Time [104] ordering method.

3.3.2 Methodology

To efficiently train and analyze the generated architectures, the models should be trained concurrently, and the performance evaluation should be done likewise. As the task is to analyze the applicability of projection-based transforms as image preprocessing, the actual models are not kept, only information about the validation- and test accuracy, and processing time is relevant and collected.

It is clear that a Master/Worker parallel design is applicable, since the tasks are independent of each other, no synchronization is necessary, and the resulting logs of training and performance analysis can be stored individually.

The main structure of the Master is based on an infinite loop (Algorithm 7). The Master acts basically as a network server, listening on a given port and address, waiting for the Worker clients to connect. In case of an incoming connection, the processing of the request is handled on different threads.

In this implementation, a communication based on the TCP[1] protocol is used. The connection and communication between two sides are reliable, which is based on acknowledging received packets.

Note, that higher-level communication frameworks, for example, implementation of the MPI[2] [105] standard could be applied; however, the benefit over the cost for this exact problem makes it unnecessary.

Algorithm 7 The algorithm of the Master. For representation purposes, the inner loop is an infinite loop, which accepts incoming TCP connections on the defined IP and port; however, in an actual software cancellation can be implemented as well for to shutdown of the listener.

```
procedure Master(Ip, Port)
    ActualizeNumbers()
    while AnyActiveWorkerExists() do
        client ← Listen(Ip, Port)
        new Thread(Process(client))
    end while
end procedure
```

Initially, the task data are loaded, and the number of ongoing and finished training is checked for each task, referred to as procedure ActualizeNumbers(). In this system, models are allowed to be trained and evaluated multiple times, possibly on different Worker instances.

The behavior of the network service is described as a loop, listening on the predefined port. The function Listen() is, therefore, implemented as blocking call, where execution waits until an actual client connection is received. In case of a connection, a new thread is forked and the processing of the client request is started.

The property defined as AnyActiveWorkerExists returns a logical value based on the number of active Worker instances. If all Workers have terminated, the Master shuts down as well. In implementation this termination-detection logic could be replaced by event-based handling.

Note that the mass-creation of threads during runtime could result in a large overhead caused by thread management costs. The problem can be solved by indirectly mapping tasks to threads, using a task queue, for example a threadpool.

On an accepted incoming TCP connection on the predefined IP and port, the procedure Process(client) is called to handle the request. The behavior is explained in more detail in Algorithm 8.

Processing of the client requests is based on the first message sent by the clients. If the message is "ready", the master selects the next task, and sends the corresponding file. In the case of no available tasks, a so-called poison pill is sent to shut down the Worker. In other cases, the client is trying to send the results of finished processing.

To ensure correctness, the task selection and response part is guarded by mutual exclusion, granting a fixed, non-overlapping execution order of the inner instructions. It is notable, that this behavior results in some inevitable overhead.

[1]Transmission Control Protocol

[2]Message Passing Interface

Algorithm 8 Processing of the client requests is based on the first message of the client: if the client states it is "ready," then a new task is sent to it. In other cases, the worker is trying to send the results of a process.

```
procedure PROCESS(client)
    msg ← CLIENT.RECEIVEMSG()
    if msg = "READY" then
        MUTEX.LOCK()
        id, task ← NEXTTASK()
        if task = null then
            CLIENT.SENDMSG("POISONPILL")
        else
            CLIENT.SENDMSG(id)
            CLIENT.SENDFILE(task)
        end if
        MUTEX.RELEASE()
    else
        id ← msg
        content ← CLIENT.RECEIVEFILE()
        STOREFILE(id, content)
    end if
    CLIENT.CLOSE()
end procedure
```

The poison pill is a special task, that is used to shut down Workers in a distributed environment [66], where the Workers cannot access the task queue to check for termination. In this case, the Master is responsible for termination detection, and the Worker is notified when it tries to access the next task.

The algorithm of the Worker is described in details in Algorithm 9.

When the Workers are starting up, connection to the Master server is suspended for a random waiting time, to evade the flood of requests on the simultaneous launch of Worker instances. After the necessary sleep, the working loop starts with connecting and sending a message to the Master, stating that the station is ready to process a task. The answer from the Master can be a task or a poison pill; the latter is handled by shutting down the Worker instance.

If a task is received, the file describing the architecture is saved, and the training is started. During training, both standard and error outputs are redirected to a file stream. The training procedure itself is a loop of training iterations, followed by evaluations on the validation data; therefore, the log contains information about the changes of the training loss and the validation loss and accuracy as well.

After training is finished, the Worker reconnects to the Master, and sends the name which identifies the task instance. After the Master acknowledges, the file is collected. When the Master confirms the transfer, the Worker cleans the temporary data about the model and task, and after some time in cooldown, the working loop starts over.

The cooldown is a necessary idle to allow the operating system of the workstation to clear caches and free up allocated space in the RAM and in the GPU memory. It is referred to as cooling down because, during the idle, the temperature of the

Algorithm 9 The pseudo language representation of the Worker process. The workers repeatedly ask for the next neural architecture, and after training and evaluation, the results are sent back to the Master. Worker termination is implemented with the "poison pill" approach.

```
procedure Worker(Ip, Port, Cooldowntime)
    Sleep(Random())
    while true do
        server ← Connect(Ip, Port)
        server.SendMsg("READY")
        resp ← client.ReceiveMsg()
        if resp = "POISONPILL" then
            return
        end if
        id ← resp
        content ← server.ReceiveFile()
        file ← StoreFile(id, content)
        server.Close()
        log ← DoTrainingAndEval(file)
        server ← Connect(Ip, Port)
        server.SendMsg(id)
        server.SendFile(log)
        server.Close()
        CleanTemporaryFiles()
        Sleep(Cooldowntime)
    end while
end procedure
```

graphics accelerator does decrease.

Please note that in Algorithm 8 and 9, the defined Send and Receive functions are necessarily synchronous, blocking calls, in other cases there is a possibility for deadlock. The communication between the Master and Worker actors are visualized on a sequence diagram in Figure 3.2.

Scheduling

To minimize the total processing time, the optimal scheduling of tasks should be done. To do that, the Longest Processing Time [104] heuristics were used, where the tasks are sorted in descending order by the estimated processing times. It is not trivial to determine the processing time of a task; however, approximations can be done.

In the case of the training and evaluation of the neural networks, it is empirically concluded that the processing time is in a linear relationship with the memory usage defined as

$$\text{complexity} = \left\lceil \frac{\text{total number of training pairs}}{\text{batch size}} \right\rceil \cdot \text{total number of elements.} \quad (3.10)$$

The total number of training pairs divided by the size of batches gives the number

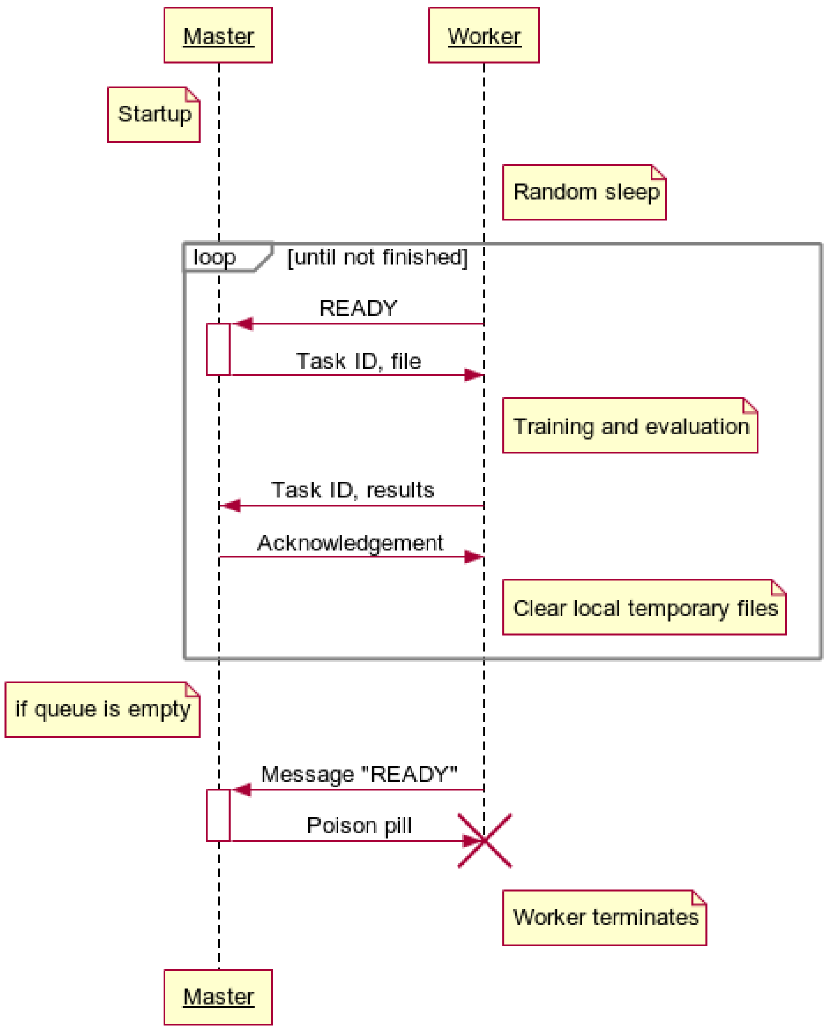


Figure 3.2. The sequence diagram of the interactions between the Master and the Worker instances. After the Master starts, the Workers take new tasks from the waiting queue, process them, and send the results back. After there is no job left, the Worker gets notified by a so-called poison pill, and then terminates [K9].

of train iterations. The estimated complexity is obtained by multiplying this number of iterations with the total number of neurons.

The results and effectiveness of the LPT ordering based on the given runtime approximation is presented in detail in the next section.

3.3.3 Results and evaluation

The actual problem for object matching was to correctly label a vehicle on different views. Multiple methods of projection transformations were analyzed, including the end-to-end method with raw images. Altogether a total of eight methods were compared. For each method, a total number of 250 neural architectures were generated based on the procedure [K8] described earlier in Section 3.2. Between a total number of one convolutional and pooling layer pair to five, 50–50 architectures were generated for each case.

This resulted in a total number of 2000 architectures, which is the input of the experiment.

The Master and the Worker clients are implemented in C# language, using the networking libraries of the .NET framework, while the training and evaluation of the neural networks are implemented with TensorFlow [106] and Keras [107] libraries of the Python programming language.

Hardware environment

The distributed training was implemented on a cluster of 25 workstations, with GeForce GTX 1050 graphics accelerators, with 2 GB onboard memory. The configuration of the host computers was the same: Intel i5-6400 CPUs with 4 cores at a 2.7 GHz clock-rate, and 8 GB RAM with maximum clock speed of 2133 MHz. The network connection between the computers was gigabit ethernet.

The *Cooldowntime* defined in the WORKER procedure in Algorithm 9 was set to 60 seconds.

Processing times

For the total analysis, all 2000 models were trained two times with the same architectures. While the total training time in a non-distributed environment would have taken more than 43 days, the distributed system finished with the tasks in less than two days. Detailed results are in Table 3.1.

In the last stages of the process, when the Master starts to run out of tasks, Workers are being shut down, one-by-one. The time difference between the first and last shutdown of Workers indicate the load balance of the processing. In the table this time is referred to as the longest idle of a given worker.

It is interesting to point out, that the speedup is extremely high, which indicates the effectiveness of the LPT method. To test this theory, based only on the model processing times of both measurements, the performance of random scheduling was calculated. Simulation of 1000 distributions was done. The results are shown in Table 3.2.

The random scheduling also produced a total runtime below two days; however, the effectiveness dropped significantly, which is well represented by the longest idle produced by the worker where the queue first runs out of tasks. While the speedup is still very high in every case, the main difference is between the load balances. The longest idle in average was 84 minutes, while in the case of the proposed scheduling, it was seven minutes (Table 3.1).

The main reason for the success of the scheduling is based on the prediction of the runtimes for the training of each model. To validate the theory, the correlation of the estimated complexity and the actual runtimes of the 4000 training process times were measured, the results are visualized in Figure 3.3.

The correlation of the estimated complexity and the real process times is measured using the Pearson correlation coefficient. A positive value over 0.5 represents a strong linear connection between the variables, which, in this case, is 0.749.

As a verification of the assumption behind the complexity measurement, a correlation heatmap is generated from the input parameters and the measured processing times (Figure 3.4).

Table 3.1. The distributed processing of the models was done in two separate runs. While there is a minimal difference between the results, the speedup and the efficiency in both cases are very high. It is also important to point out that the load balance of this scheduling is very good, the granularity of the last tasks is fine, causing a low idle time for the Worker which terminates first. Time values in this table are represented in a *HH:MM:SS* format [K9].

	Measurement #1	**Measurement #2**
Sum of times	43 days, 12:51:28	43 days, 10:40:08
Average time per worker	1 day, 17:47:39.52	1 day, 17:42:24.32
Total runtime of training	1 day, 17:50:35	1 day, 17:45:41
Longest idle	00:07:09	00:07:20
Speedup	24.97	24.97
Efficiency	99.88 %	99.87 %
Process times		
Minimum	00:06:14	00:06:11
Maximum	03:34:12	03:37:31
Mean	00:31:20.74	00:31:16.80
Median	00:26:38	00:26:33
Standard deviation	00:20:41.11	00:20:39.38

Table 3.2. The results of 1000 simulations of random scheduling. The generated runtimes are ordered increasingly, and the minimum, maximum and median values are described in three columns of this table. Time values in this table are represented in a *HH:MM:SS* format [K9].

	Minimum	**Median**	**Maximum**
Total runtime of training	1 day, 18:02:10	1 day. 18:50:10	1 day, 20:51:15
Longest idle	00:32:47	01:24:13	03:23:23
Speedup	24.85	24.39	23.29
Efficiency	99.42 %	97.57 %	93.17 %

The heatmap shows that there is a strong connection between the runtime of a task and the estimated memory cost, which confirms the base assumption behind the score calculation in (3.10).

The connection between the batch size and the runtime is also significant: the weak negative correlation shows the inverse connection, meaning that the increase of the number of samples in batches decrease the runtime. It is worth mentioning, that the increase of batch size on a large scale will have a negative effect on the performance of the model [108]; therefore, in further research, an upper boundary of batch sizes should be defined. The problem caused by this effect is a well-researched field [109]. The approach to overcoming this drawback is based on parallel and distributed training using multiple GPUs and workstations.

It is interesting, that the number of layers does not have a large impact on the

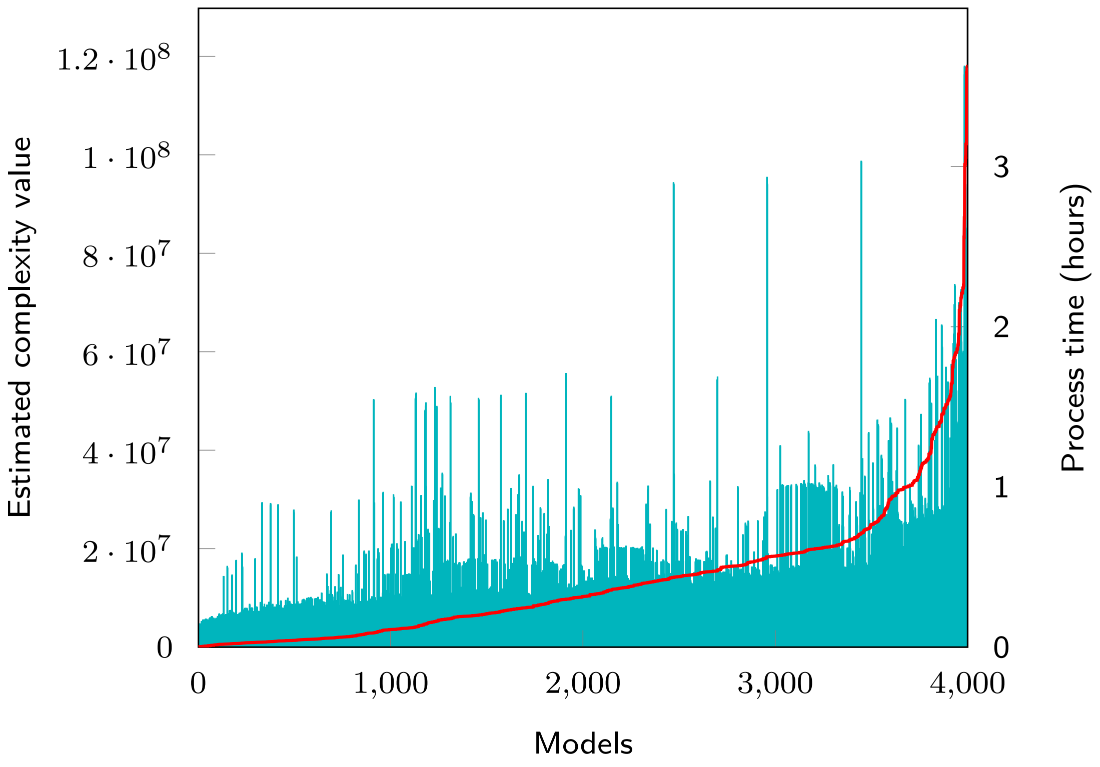


Figure 3.3. The estimated complexity values and the processing times for each model, ordered by the estimated value based on (3.10). The thick red line represents the complexity value, while the thin columns show the processing times for each model. Although there are some visible diversions between the two, the calculated correlation is strong, 0.749 [K9].

measured times, which is explained by the small number of convolutional layers used for this experiment. The large layer number of a deep network clearly has a significant computational cost factor compared to shallow networks. However, in our experiments, this effect is not relevant, which is explained by the generally small number of layers. It is noted, however, that for estimation of the complexity of deep architectures the layer number should be considered as a significant influence.

By checking the total runtime of the process, and comparing it to the average runtime, we see that the difference is around three minutes. As pointed out in [110], the lower boundary to approximate the optimal runtime can be done by simply dividing the number of processors with the sum of runtimes: it is clear that the optimal processing time cannot be shorter. There are other relaxations of the lower bound based on continuous relaxation of this defined limit, as well as measures based on other heuristics. It is worth mentioning, that based on the knowledge of the actual processing times, the optimal scheduling can be created; however, the computational complexity of such algorithms are very high [111].

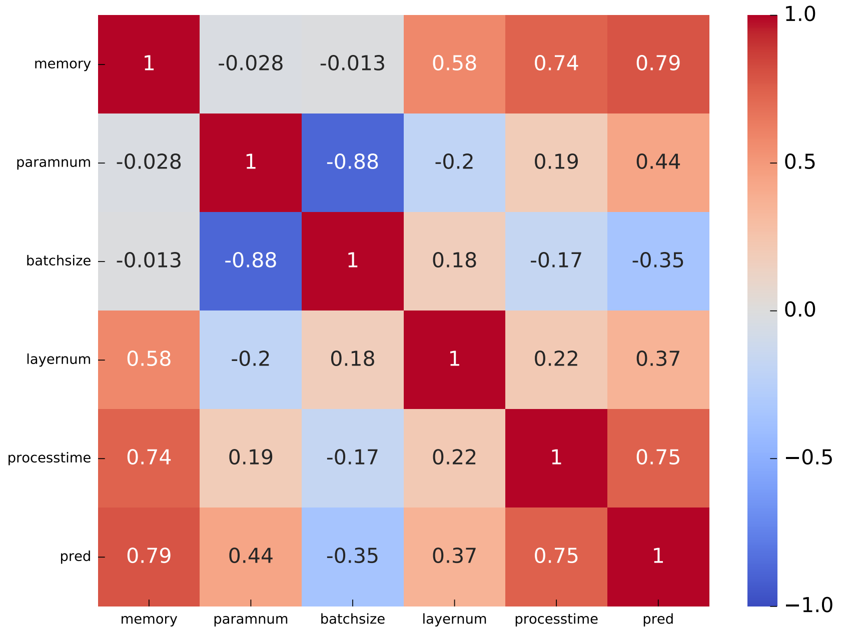


Figure 3.4. Correlation heatmap generated using the input parameters of model training and the measured processing times from the total of 4000 trainings. *pred* represents the predicted complexity based on (3.10).

Thesis 2.2 *I designed and implemented a Master/Worker model for the analysis of Siamese Convolutional neural network architectures in a distributed environment, with scheduling based on the longest processing times. In practical measurements, the parallel efficiency of the processing of the generated neural network architectures was 99.87%.*

Publications pertaining to thesis: [K9].

3.4 Results and evaluation

The goal of the research is to determine the usability of multi-directional projections as objects descriptors for object image matching. For analysis a simulation based on modern approaches for object matching is proposed.

Multiple methods and several setups are selected and defined for comparison. The outputs of these methods are two-dimensional matrices representing projection functions for different angles. From the given input sizes, the method defined in Section 3.2 generates multiple operable Convolutional architectures for a Siamese structured neural network.

Based on this method, CNN architectures are generated by the initial information of the input size and the total number of convolutional and pooling layer pairs. Multiple values are examined for the number of inner convolutional and pooling layers.

The method is also able to handle the 3D representation of RGB images, having the first two dimensions representing the image width and height, and the three layers in the third dimension represent the red, green and blue color information. Thus, neural comparator architectures for the original images are also generated.

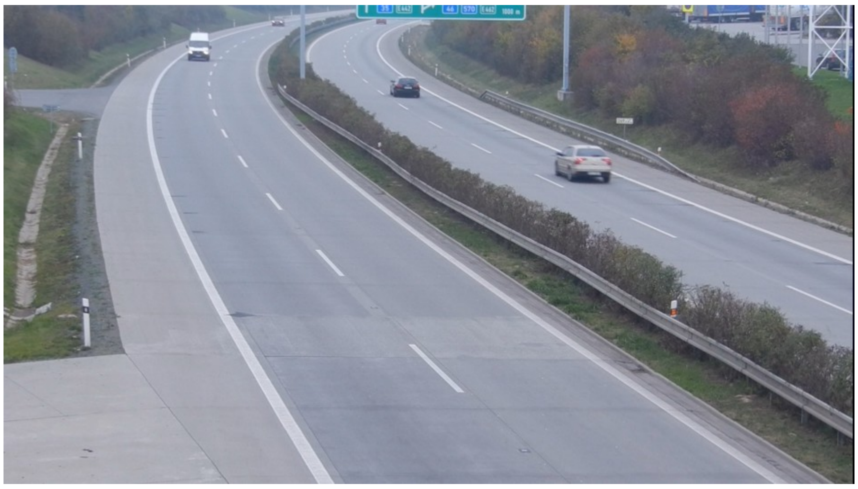


Figure 3.5. A sample frame from the Vehicle ReIdentification dataset provided for the International Workshop on Automatic Traffic Surveillance, on the 29th IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016) [112].

The algorithm rewards those structures where convolutional and pooling window sizes are as low as possible, if feasible, the pooling layer is skipped. Furthermore, the algorithm estimates the memory consumption of the generated architectures and optimizes the value of the batch size to the maximum, where a memory limit is applicable.

As an object re-identification problem, the dataset [112] used for the research was published in the International Workshop on Automatic Traffic Surveillance, on the 29th IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016). The image sequences of the dataset are annotated, so the extraction of the regions of interest was done automatically.

The regions of objects of interest were extracted from the frames of the video, and the labels were attached according to the annotations provided in the source files. A sample of the video frame is shown in Figure 3.5.

The original sizes of extracted images were kept and a rescaling of the inputs was only done if necessary as an initial step of the analyzed transformation.

To increase the efficiency, a cluster of GPU enabled computers were used for training and evaluation, based on the parallel implementation described in section 3.3.

3.4.1 Data

In those cases, where the number of extracted observations for each viewpoint are less then 15, the objects are removed from the set. After the filtering of objects with only a few observations, the final dataset contains a total number of 698 vehicle instances. The set is split into a training set and a test set using a 9 : 1 ratio, resulting in a total number of 629 instances for training and 69 for testing. For every object, a total of 30 observations from two different viewpoints are used.

For the analysis, the results of Radon transform, Trace transform and the hereby presented transformation with fixed bin numbers were generated. For further references, the abbreviation MDIPFL will be used for the method of Multi-Directional

Table 3.3. The different input types, by matrix sizes. As MDIPFL works with a fixed bin size, multiple outputs were generated and tested.

Name	Size
Radon-transform	85 x 36 x 1
Radon-transform	136 x 36 x 1
Trace-transform	137 x 72 x 1
MDIPFL transformation	25 x 36 x 1
MDIPFL transformation	50 x 36 x 1
MDIPFL transformation	85 x 36 x 1
MDIPFL transformation	136 x 36 x 1
RGB Image	96 x 96 x 3

Image Projections with Fixed Length, presented in Chapter 2. Table 3.3 contains the details of each transformation, including the description of the raw unmodified original images.

Note the two Radon transforms with different matrix dimensions: in the first case of 85×36, the images of observations are scaled to 60×60, which after the transformation for projection angles 0 to 175 with step size of 5 degree results in a 85×36 matrix. In the other case, 96×96 sized images were used as inputs for the Radon transform, resulting in 136×36 matrices under the same conditions.

The Trace transform by default is not entirely symmetric, the angles of 360 degrees are necessarily analyzed.

For the presented method, multiple resolutions were analyzed: 25 and 50 bins, and to match the matrix sizes, a 85 and 136 bin numbered transformation was generated for training.

The raw images are RGB images, resized to 96×96. It is notable, that all presented methods used resized images before the transformation, except the developed method of fix bin sizes, where the original images could be used, as the output matrix size is not affected by the input image size.

A complete list of input types, along with the abbreviations used hereafter:

- RGB Image
  The original unprocessed colored image resized to 96×96;
- Radon85
  The result of the Radon transform for a 60×60 input image for $[0; 180)$ degrees, with a step size of 5 degrees;
- Radon136
  The result of the Radon transform for a 96×96 input image for $[0; 180)$ degrees, with a step size of 5 degrees;
- Trace
  The result of the Trace transform for a 96×96 input image for $[0; 360)$ degrees, with a step size of 5 degrees;
- MDIPFL25
  The result of the MDIPFL transformation with bin number of 25;

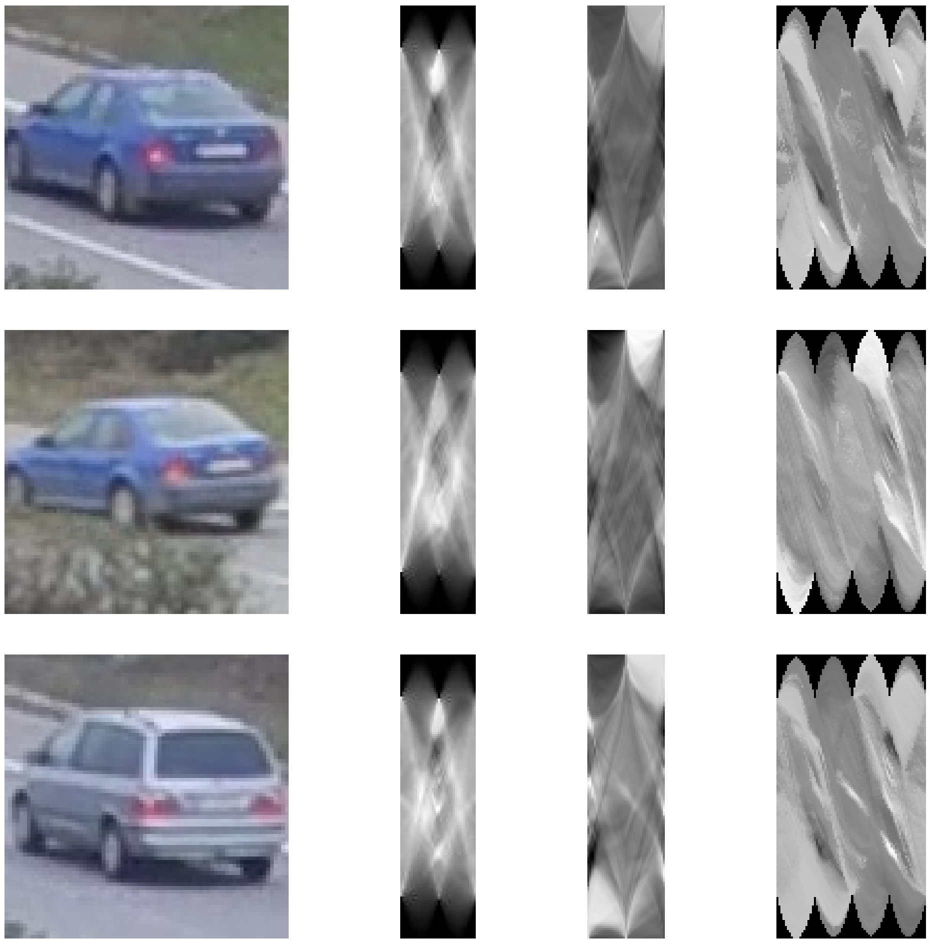


Figure 3.6. Sample images from the dataset, from left to right: the original image, the result of the Radon transform, the MDIPFL transform and the Trace transform, respectively.

- MDIPFL50
  The result of the MDIPFL transformation with bin number of 50;
- MDIPFL85
  The result of the MDIPFL transformation with bin number of 85, to match the output size of Radon85;
- MDIPFL136
  The result of the MDIPFL transformation with bin number of 136, to match the output size of Radon136.

For every transformation (each of them visualized in Figure 3.6), a total of 250 neural architectures were generated, 50 for every conv-pool layer pair number between 1 and 5.

This resulted in a sum of 2000 architectures, which is the base of the models trained in a distributed environment. The parallel training and environment are further explained in Section 3.3.

To fit the GPU memory, the upper limit of the architecture generation was set to 512 MB. During training, machine learning frameworks handle memory differently [K8]; however, to maximize efficiency in most cases an amount of more than three times the models cost (which is given by the parameter numbers times the size of batches) is allocated to store input data, feature maps and other temporary variables of backpropagation along with the network parameters.

A total of 100,000 training pairs were given to the models during the training of each. This number is more than five times higher than the number of individual training samples.

During training, a vehicle instance is selected from the dataset, and positive and negative pairs are generated to fill the batch. After forward and back-propagation, the trainable parameters are aligned, the next batch is generated.

After training finished, the test set was used to measure the prediction accuracy of the model, repeatedly. The results were written to the log file, which is collected by the Master.

The training and analysis of the 2000 models were repeated to confirm the runtimes and performance. As previously stated, the total runtime of the distributed training was over 41 hours, where the average processing time of a model is around 30 minutes. The effectiveness of the distributed processing is very high, given that the sequential processing of the same models would have taken more than 40 days. A detailed description of the runtimes is given earlier on Table 3.1.

3.4.2 Performance

After training the models, the prediction accuracy is measured using multi-way one-shot classification. The verification procedure is described in [113] and [96] as a method that demonstrates the discriminative abilities of the model.

Instead of measuring based on an absolute scale, for example distance-based thresholding, one-shot classification gives the relative performance of the separational capability. In production, when re-identifying an object, it is possible that based on some meta-information, a narrow set can be formed of possible instances. Therefore, it is more realistic to examine accuracy by a method that does not measure similarity on an absolute scale.

To formalize the method, define $x \in X$ test images, and $c \in C$ categories, having a surjective, non-injective mapping for each input x to each c as function

$$\begin{gathered} f : X \to C, \\ \forall x \in X, \exists c \in C : f(x) = c. \end{gathered} \tag{3.11}$$

This represents that every image observation belongs to one of the object instances.

During a test, a random sample image is selected from the test dataset, noted as $\hat{x}$. $\hat{x}$ will be matched against a set of images.

Define a set of N images as $\{x_i\}_{i=1}^{N}$, where

$$\begin{gathered} \forall i \nexists j : f(x_i) = f(x_j), i \neq j \\ \exists_{=1} i : f(x_i) = f(\hat{x}), x_i \neq \hat{x}. \end{gathered} \tag{3.12}$$

This states that none of the images are from the same category and exactly one image is from the same category as the selected reference image (while the images are not equal).

The model is tested with the reference image and multiple observations: one from the same instance, and one or more observations of different objects. For example, a

Table 3.4. The measured prediction accuracy for each transformation for 2,4,6,8 and 10-way classification. The models were tested with 10,000 tests and the best performances were selected for this table.

Type	**Top accuracy for N-way classification**				
	2	**4**	**6**	**8**	**10**
RGB Image	**94.34**	**85.47**	**78.86**	72.93	67.31
Radon85	93.22	83.94	75.73	70.74	66.28
Radon136	93.56	84.54	77.20	**73.32**	67.61
Trace	92.78	80.44	71.36	65.60	58.04
MDIPFL25	94.12	84.76	78.36	72.76	**68.92**
MDIPFL50	93.77	84.27	75.96	70.08	64.80
MDIPFL85	93.76	83.98	76.91	71.89	67.00
MDIPFL136	93.36	82.81	74.97	69.19	64.13

two-way comparison is done when the reference image is compared to one true and one false sample, just like a 5-way comparison is where the base image is measured against one true and four false examples.

The measurement is done by selecting the similarity between the reference and the elements of the set, defined as

$$Y = \{y_i = S(\hat{x}, x_i)\}_{i=1}^{N}, \tag{3.13}$$

where $S(a, b)$ represents predicted similarity of inputs a and b, where 1 represents a match and 0 means difference.

After calculating the similarity between the reference and all elements in the N sized set, the category corresponding to the maximum similarity is selected as

$$c^* = \arg\max_c Y, \tag{3.14}$$

which is then compared to the category of reference $\hat{x}$. If the highest similarity is measured with the true pair, as $f(\hat{x}) = c^*$, the classification is correct. If the predicted category is different, then the classificator failed.

The accuracy of the model can be measured by counting the correct classifications during a number of M tests as

$$\text{Accuracy} = \frac{\text{Number of correct classifications}}{M} \cdot 100. \tag{3.15}$$

The top results of N-way classification are detailed in Table 3.4. It is shown, that projection-based methods are suitable for neural comparison. Furthermore, in some cases, these methods outperform classical end-to-end approaches under the same conditions [K6].

For further analysis, the accuracy of different number of convolutional and pooling layer pairs are compared. In Figure 3.7, the best performances are plotted for one-shot classification tasks for increasing number of classes.

It is clear that by increasing the number of layers (and, therefore, the total number of trainable parameters), the models perform better. In the case of a single convolutional-pooling layer pair, the image-based method performs $15-20\%$ below

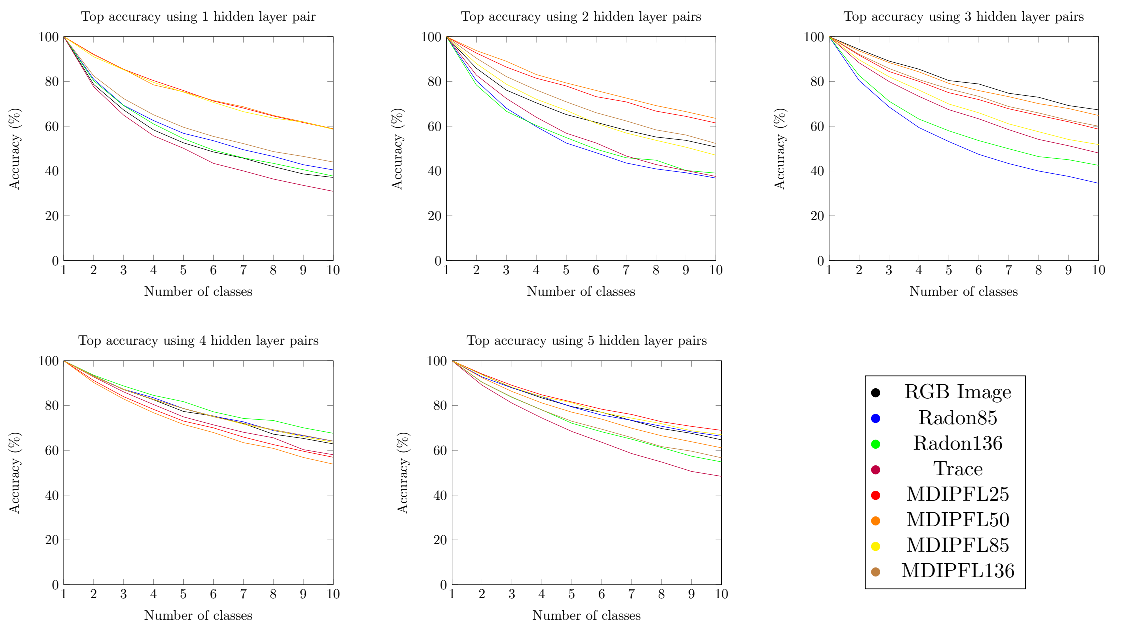


Figure 3.7. One-shot classification accuracy for classes $N = 1 \ldots 10$, grouped by different numbers of hidden convolutional-pooling layer pairs, from 1 to 5, left to right, respectively.

the top projection based models regardless of the class number. This is mainly caused by the extremely large convolutional window sizes.

A more detailed summary of classification accuracy for different methods and layer numbers can be found in appendix A.1.

The processing times of each input type regarding the number of hidden layers is visualized in Figure 3.8.

The significant increase in processing time is clear for raw image inputs: this is caused by the relatively large matrix size, causing memory cost, ultimately resulting in lower batch sizes. As a given number of training pairs should be used during training, low batch sizes increase the necessary iterations, therefore the runtime as well.

The large deviation in runtimes is caused by the difference in window sizes. Because of the memory limit (and minimal batch size), some of the generated architectures simply would not fit in the device memory; therefore, training is unadvised, or not possible. To deal with this, the generator method tries to increase the window sizes until the necessary number of architectures are found.

As previously explained, the memory cost of the model is an important measure in the implementation of machine learning solutions. To analyze the usability of projection-based preprocessing methods, both prediction accuracy and memory storage need should be optimized.

As an exact ratio of importance between the two cannot be given, multi-objective optimization [114] is used to give the best results based on the objective functions.

The objective functions are defined as

$$f_{\texttt{err}}(m) = 100 - \texttt{Accuracy}(m) \tag{3.16}$$

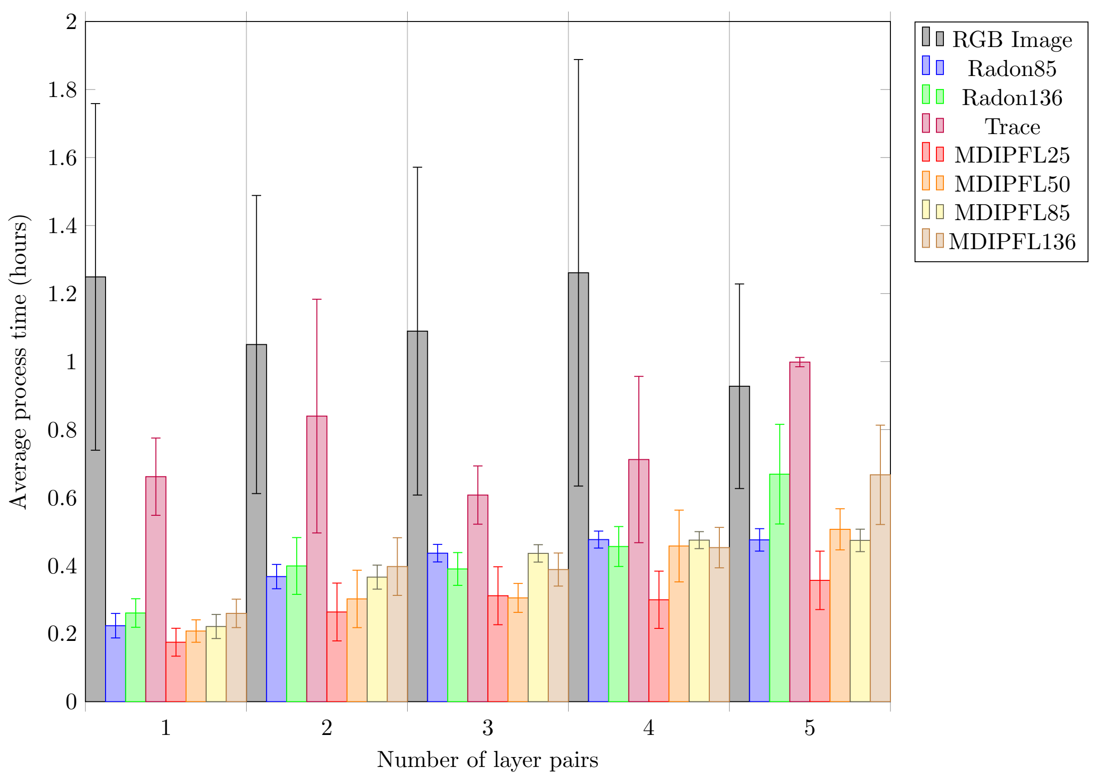


Figure 3.8. Average processing times of models grouped by input types and the number of hidden layers. Each bar represents the average processing time of the training 50 models twice. Error bars illustrate the standard deviation.

and

$$f_{\texttt{memory}}(m) = \texttt{ParameterNumber}(m), \tag{3.17}$$

where m stands for the model.

By defining $f_{\texttt{err}}$ as the error rate, and $f_{\texttt{memory}}$ with the parameter number of the model, optimization is done by pursuing to minimize the results by each objective.

The Pareto frontier [114] represents a set of multiple parameterizations, where each of the elements are Pareto efficient. An element is described as Pareto efficient – and therefore part of the Pareto frontier – if it is not Pareto dominated by any other points.

To define dominancy using the functions above, notations m_1 and m_2 will be used for two different, $m_1 \neq m_2$ models. Dominancy of m_1 over m_2 is given as

$$\begin{gathered} V = \{\texttt{err}, \texttt{memory}\} \\ m_1 \succ m_2 : \forall i \in V : f_i(m_1) \leq f_i(m_2), \exists j \in V : f_j(m_1) < f_j(m_2). \end{gathered} \tag{3.18}$$

Based on dominancy, the elements of the Pareto frontier are given as

$$\forall m \in M \nexists m' \in M : m \neq m', m' \succ m, \tag{3.19}$$

that is, the elements in the Pareto frontier are all m models, which are not dominated by (i.e., no dominator element exists) any other m' elements of set of all models M.

These elements are also referred to as Pareto optimal results.

Note that, based on this definition, elements with equal values are not dominating each other. For example,

$$\begin{gathered} m_1 \in M, m_2 \in M \\ f_{\texttt{err}}(m_1) = f_{\texttt{err}}(m_2) \\ f_{\texttt{memory}}(m_1) = f_{\texttt{memory}}(m_2) \end{gathered}$$

therefore, based on the strict definition in (3.18),

$$\begin{gathered} m_1 \not\succ m_2, \\ m_2 \not\succ m_1. \end{gathered}$$

If no other elements are dominating m_1 and m_2, both are parts of the Pareto frontier.

The results of the multi-objective optimization are visualized in Figure 3.9. The figures show that the optimal results for both $f_{\texttt{err}}$ and $f_{\texttt{memory}}$ are, in most cases, based on MDIPFL transformations. These Pareto optimal solutions also include some models based on the original images, and few are built on the Radon transform.

It is notable, the figures also point out the fact that some of the neural architectures generated were not able to learn from the given examples, as the error rates for 2, 4, 6, 8, 10-way classifications were at 50%, 75%, 83.3%, 87.5%, 90%, respectively.

CNN architectures with the highest 10-way one-shot classification accuracy are listed in Appendix A.2. For every input type, the top five architectures are illustrated.

Thesis 2.3 *I analyzed each of the multidirectional image projection methods, using them as a preprocessor for input data, to determine the effect on the performance of Siamese convolutional networks. Based on the results, I concluded that the method based on a fixed number of bins is Pareto optimal in terms of efficiency and memory requirement compared to the raw image methods considered as a reference.*

Publications pertaining to thesis: [K6].

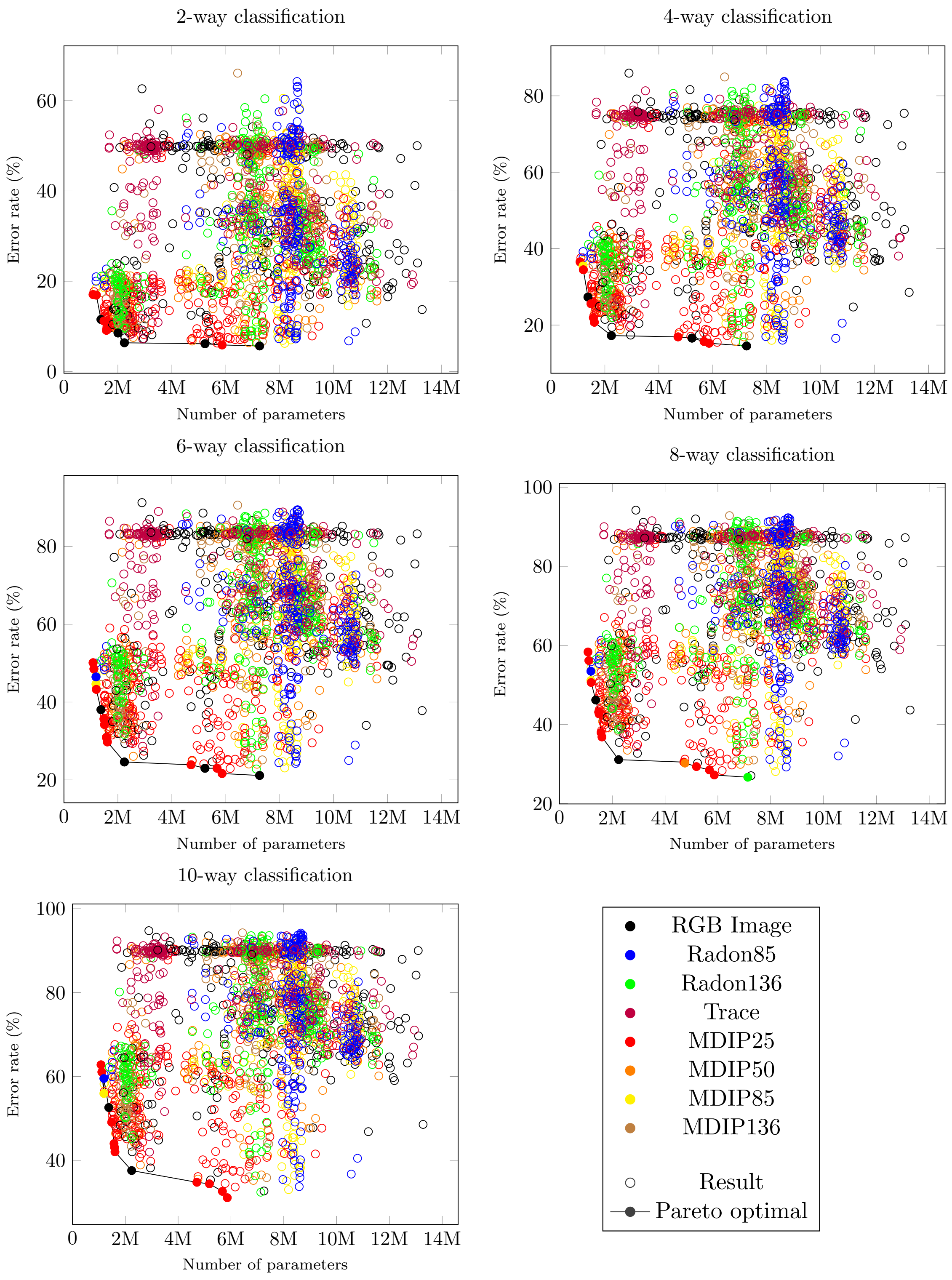


Figure 3.9. The results visualized by the number of parameters for the model and the validation error rate. Each model was tested with 10,000 validation examples. The Pareto optimal results – values that are not dominated by any other result – are visualized in the bottom left corners as the Pareto frontier [K6].

3.5 Summary

To analyze the usability of multi-directional projection-based methods for object image matching, a complex and computationally intense experiment was designed, implemented, and evaluated.

The key idea behind the trials was to apply machine learning for matching, which – in the case of images and other structured inputs – is done by Siamese architectured Convolutional Neural Networks.

For a comprehensive measurement, state-of-the-art CNN design patterns were analyzed, and based on the observations a technique to generate fully-convolutional heads was given in Section 3.2. The described method uses a backtracking search algorithm to find multiply possible solutions that satisfy the requirements. During the search for possible architectures a memory consumption limit is kept, which, in result, creates the practical benefit of possible targeting of different hardware. As a result, beside commercially available personal computers with graphical accelerators, large multi-GPU systems, or even small IoT devices with additional neural accelerators can be used as target architectures.

In this experiment, the generated architectures were trained in a distributed environment, on a cluster of GPU enabled computers. The training was managed in a Master/Worker setup, with a special LPT-based scheduling using a computational cost approximation described in section 3.3. The implementation is very effective, the achieved efficiency was nearly the same as the number of workstations.

The speedup is thoroughly analyzed in Section 3.3: in short, the achieved efficiency is extremely high, the total parallel processing time requires 4% of time when compared to the sequential method.

For training, the elements of the dataset were transformed using multiple projection transformations, including the MDIPFL transformation presented in Chapter 2. For testing purposes, the original images were also included in the simulations.

To measure the accuracy, N-way one-shot classification tasks were used with a multiple size of test image sets, to show the discriminative abilities of the trained models.

After evaluating the results in terms of accuracy, processing time and memory consumption (described in Section 3.4), it is concluded that the method based on a fixed number of bins is Pareto optimal in terms of efficiency and memory requirement compared to the raw image methods.

Chapter 4

Conclusion

Young man, in mathematics you don't understand things. You just get used to them.

— John von Neumann

This dissertation presented a method for object image matching using multi-directional image projection transformation with a fixed number of bins. This method is analyzed and compared to other similar techniques, and finally transposed into a modern machine learning-based framework.

The achieved results are summarized in a total of 6 theses, grouped into two coherent thesis groups.

Thesis group I: Achievements in Multi-directional Image Projections

Thesis 1.1

I have designed and implemented a method of mapping multi-directional projection vectors using fixed bin numbers regardless of the rotation angle. The memory cost of the result is independent of the image size; it is only affected by the rotation step number and the number of bins.

The fixed number of bins result in a fixed vector length independent of the projection angle. Using a fixed resolution for different sized images result in projection maps with equal size, which results in a constant memory cost.

The properties of the mapping show similarities to the Radon transform, which served as an inspiration.

Publications pertaining to thesis: [K2], [K4], [K5].

Thesis 1.2

I have designed and implemented the data-parallel version of the multi-directional image projection algorithm for graphical processors, which allows acceleration proportional to the number of execution units.

The data-parallel solution is designed for GPU implementation. The method uses multiple levels of the GPU device memory architecture, resulting in an efficient solution where runtime is in a linear relationship with the number of elements. However, the ability of simultaneous processing of elements results in a speedup proportional to the number of execution units.

Publications pertaining to thesis: [K4], [K5].

Thesis 1.3

I evaluated the effectiveness of the fixed vector length multi-directional image projection method for object matching, comparing the results with similar projection-based, lower-dimensional image signatures, and concluded that matching accuracy increased significantly.

The defined method is compared with two- and four-dimensional projection signature-based matching methods, as well as the Radon transform. With a fixed resolution, the memory cost does not depend on the size of the image. Also, it is identified, that performance is not harmed by using small bin numbers; therefore, a compression is achieved.

Publications pertaining to thesis: [K5].

Thesis group II: Application of Image Projections as Preprocessing in Siamese Convolutional Neural Networks

Thesis 2.1

I have developed a method based on backtracking search that provides all of the suitable convolutional neural network architectures at a given input, layer number, and memory cost.

The method generates CNN architectures based on the analyzed convolutional design patterns, while keeping a low memory cost. The algorithm calculates window sizes for decent convolutional and pooling layer pairs. Also, the memory cost of the model is estimated and the training batch size is optimized to the available maximum.

As a result, even small IoT devices with additional neural accelerators with less memory can be used as target architectures for generation.

Publications pertaining to thesis: [K8].

Thesis 2.2

I designed and implemented a Master/Worker model for the analysis of Siamese convolutional neural network architectures in a distributed environment, with scheduling based on the longest processing times. In practical measurements, the parallel efficiency of the processing of the generated neural network architectures was 99.87%.

The distributed training was done in a cluster of workstations with graphical accelerators. Measurements show, that the complexity approximation-based scheduling is very effective, resulting in a speedup near the number of workstations.

Publications pertaining to thesis: [K9].

Thesis 2.3

I analyzed each of the multidirectional image projection methods, using them as a preprocessor for input data, to determine the effect on the performance of Siamese convolutional networks. Based on the results, I concluded that the method based on a fixed number of bins is Pareto optimal in terms of efficiency and memory requirement compared to the raw image methods considered as a reference.

Finally, the results of the designed simulation were evaluated. The architectures were generated for multiple transformations based on the defined method. After training in the distributed environment, the results were analyzed in terms of one-shot classification accuracy, processing time and memory cost.

It is concluded that the MDIPFL method is Pareto optimal in terms of efficiency and memory cost; therefore, the method is well suited for low memory hardware.

Publications pertaining to thesis: [K6].

Further research

Based on the original ideas described in [46], multiple observations of the same instance could be used to increase performance. This could be done as simply as storing multiple projection signatures from the same instances in case of one-shot tasks, or an appearance model could be built by combining the collected observations.

To further improve the comparison of projection signatures, analyzing the camera position and properties should be done. Assuming that the camera is fixed, image rectification would improve the performance. Necessary camera properties could be calculated based on calibration methods.

Further plans include the analysis of the defined method for object recognition and matching for other types of data, for example, facial recognition and matching. Person identification based access control systems and surveillance are widely used for security.

As the defined method is memory efficient, the use of multi-core IoT devices should be analyzed. Industrial cameras are able to detect regions of interest, segment or even preprocess the recorded images before transfer. Therefore, a system based on IoT Smart Cameras [1] could be applied.

When designing a system using multiple smart cameras, instead of the classic centralized server-client model, a distributed environment of peer-to-peer connected smart devices should be examined [115]. Such a structure would result in greater territorial coverage with less constructional costs; in addition, the communication bottleneck to the main computer would be removed.

The machine learning-based approach is feasible. With further optimization of the training process, and hyperparameter tuning, the performance could be further enhanced. It is also concluded, that the method can be applied for one-shot learning tasks.

The efficiency of training can be increased by implementing a triplet loss technique [116]. In this case, the model would be trained with three representations: the reference, the least similar true pair, and the most similar false pair. As a result, the significant features of discrimination would be learned. As the selection of samples is based on a pre-training similarity measurement, the challenge of this method is to keep the runtime in an acceptable range [117].

The process can be further optimized by examining the possibilities in distributed training of models. The current state-of-the art techniques are based on batch division and gradient averaging. Initially, the models with equal parameters are distributed along multiple workstations. During training, the batches are divided and scattered between the nodes where the segments are used for a training loop. After the processing of batches is finished, the gradients are collected by a main node [118], averages are computed, and weights are updated on all nodes.

The memory transfer cost has a serious drawback on the efficiency of the method even on a single host multi-GPU environment with dedicated connection between the graphics accelerators. In distributed environments with multiple hosts, the negative effect of transfer times are even more significant.

State-of-the-art approaches [119, 120] of distributed deep learning use the Ring-AllReduce algorithm, removing the centralized averaging of gradients instead a ring topology-based reduction is used to pass gradients in a circle, updating all nodes to have the same model parameters.

As the projection descriptors show invariant properties, the application of transfer learning should be examined. Transfer learning is a method where pre-trained models are reused for different tasks. A very popular example is the VGG-16 model [121], which is trained on the ImageNet dataset, and is used for base architectures for different tasks. The most common approach is to remove the last few layers of the VGG-16 architecture and replace them with output layers fitting the actual problem description. The parameters of the VGG model remain unchanged during training; therefore, they act as a feature mapping for input images.

In case of projection maps, the effects of transferring knowledge when using pre-trained networks should be considered. Possible gains are higher performance and efficiency during training.

Bibliography

[1] Bernhard Rinner and Wayne Wolf. “An Introduction to Distributed Smart Cameras”. In: *Proceedings of the IEEE* 96.10 (2008), pp. 1565–1575.

[2] Chih-Chang Yu, Hsu-Yung Cheng, and Yi-Fan Jian. “Raindrop-Tampered Scene Detection and Traffic Flow Estimation for Nighttime Traffic Surveillance”. In: *IEEE Transactions on Intelligent Transportation Systems* 16 (3 2015), pp. 1518–1527.

[3] Angel Sanchez et al. “Video-Based Distance Traffic Analysis: Application to Vehicle Tracking and Counting”. In: *Computing in Science and Engg.* 13.3 (2011), pp. 38–45. ISSN: 1521-9615. DOI: `10.1109/MCSE.2010.143`.

[4] Reyes Rios-Cabrera, Tinne Tuytelaars, and Luc Van Gool. “Efficient Multi-camera Vehicle Detection, Tracking, and Identification in a Tunnel Surveillance Application”. In: *Comput. Vis. Image Underst.* 116.6 (2012), pp. 742–753. ISSN: 1077-3142. DOI: `10.1016/j.cviu.2012.02.006`.

[5] A. E. Abdel-Hakim and A. A. Farag. “Color segmentation using an Eigen color representation”. In: *2005 8th International Conference on Information Fusion* 2 (2005). DOI: `10.1109/ICIF.2005.1592043`.

[6] Vedran Jelača et al. “Real-time vehicle matching for multi-camera tunnel surveillance”. In: *Proceedings of SPIE, Real-Time Image and Video Processing*. Vol. 7871. Jan. 2011, p. 8.

[7] Johann Radon. “Über die Bestimmung von Funktionen durch ihre Integralwerte längs gewisser Mannigfaltigkeiten”. In: *Berichte über die Verhandlungen der Königlich-Sächsischen Akademie der Wissenschaften zu Leipzig, Mathematisch-Physische Klasse* (1917), pp. 262–277.

[8] Johann Radon. “On the Determination of Functions from Their Integral Values along Certain Manifolds”. In: *Medical Imaging, IEEE Transactions on* 5 (1986), pp. 170–176. ISSN: 0278-0062. DOI: `10.1109/TMI.1986.4307775`.

[9] Richard Szeliski. *Computer Vision: Algorithms and Applications*. 1st. New York, NY, USA: Springer-Verlag New York, Inc., 2010. ISBN: 1848829345, 9781848829343.

[10] Joseph L Mundy. “Object recognition in the geometric era: A retrospective”. In: *Toward category-level object recognition*. Springer, 2006, pp. 3–28.

[11] Szabolcs Sergyán. “Content based image retrieval in database of segmented images”. In: *Proceedings of 4th Slovakian-Hungarian Joint Symposium on Applied Machine Intelligence*. 2006, pp. 380–388.

[12] MM El-Gayar, H Soliman, et al. “A comparative study of image low level feature extraction algorithms”. In: *Egyptian Informatics Journal* 14.2 (2013), pp. 175–181.

[13] Zoltan Kato and Ting-Chuen Pong. “A Markov random field image segmentation model for color textured images”. In: *Image and Vision Computing* 24.10 (2006), pp. 1103–1114.

[14] Rafael C. Gonzalez and Richard E. Woods. *Digital image processing.* Upper Saddle River, N.J.: Prentice Hall, 2008. ISBN: 9780131687288 013168728X 9780135052679 013505267X.

[15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. “Fully convolutional networks for semantic segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2015, pp. 3431–3440.

[16] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-net: Convolutional networks for biomedical image segmentation”. In: *International Conference on Medical image computing and computer-assisted intervention.* Springer. 2015, pp. 234–241.

[17] Chee Sun Won. “Feature extraction and evaluation using edge histogram descriptor in MPEG-7”. In: *Pacific-Rim Conference on Multimedia.* Springer. 2004, pp. 583–590.

[18] Chris Fields. “How Humans Recognize Objects: Segmentation, Categorization and Individual Identification”. In: *Frontiers in psychology* 7 (2016), p. 400.

[19] Hans P Moravec. “Toward Automatic Visual Obstacle Avoidance”. In: *International Conference on Artificial Intelligence (5th: 1977: Massachusetts Institute of Technology).* 1977.

[20] Christopher G Harris, Mike Stephens, et al. “A combined corner and edge detector”. In: *Alvey vision conference.* Vol. 15. 50. 1988, pp. 147–151.

[21] Jianbo Shi and Carlo Tomasi. *Good features to track.* Tech. rep. Cornell University, 1993.

[22] Tony Lindeberg. “Feature detection with automatic scale selection”. In: *International journal of computer vision* 30.2 (1998), pp. 79–116.

[23] Cordelia Schmid, Roger Mohr, and Christian Bauckhage. “Evaluation of interest point detectors”. In: *International Journal of computer vision* 37.2 (2000), pp. 151–172.

[24] David G Lowe et al. “Object recognition from local scale-invariant features”. In: *Proceedings of the International Conference on Computer Vision.* 2. 1999, pp. 1150–1157.

[25] Herbert Bay et al. “Speeded-up robust features (SURF)”. In: *Computer vision and image understanding* 110.3 (2008), pp. 346–359.

[26] PM Panchal, SR Panchal, and SK Shah. “A comparison of SIFT and SURF”. In: *International Journal of Innovative Research in Computer and Communication Engineering* 1.2 (2013), pp. 323–327.

[27] Yanlin Guo et al. “Robust Object Matching for Persistent Tracking with Heterogeneous Features”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29.5 (2007), pp. 824–839. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2007.1052.

[28] Yanlin Guo et al. “Matching vehicles under large pose transformations using approximate 3D models and piecewise MRF model”. In: *IEEE Conference on Computer Vision and Pattern Recognition.* IEEE. 2008, p. 8.

[29] Tingbo Hou, Sen Wang, and Hong Qin. “Vehicle Matching and Recognition under Large Variations of Pose and Illumination”. In: *Computer Vision and Pattern Recognition Workshops, 2009. CVPR Workshops 2009. IEEE Computer Society Conference on.* IEEE, 2009, pp. 24–29. ISBN: 978-1-4244-3994-2. DOI: 10.1109/CVPRW.2009.5204071.

[30] Tae Eun Choe, Mun Wai Lee, and Niels Haering. “Traffic Analysis with Low Frame Rate Camera Networks”. In: *Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on.* IEEE, 2010, pp. 9–16. ISBN: 978-1-4244-7030-3. DOI: 10.1109/CVPRW.2010.5543801.

[31] R. Brunelli. *Template Matching Techniques in Computer Vision: Theory and Practice.* Wiley, 2009.

[32] Michael Oren et al. “Pedestrian detection using wavelet templates”. In: 1997, pp. 193–199.

[33] Constantine P Papageorgiou, Michael Oren, and Tomaso Poggio. “A general framework for object detection”. In: *Sixth International Conference on Computer Vision (IEEE Cat. No. 98CH36271).* IEEE. 1998, pp. 555–562.

[34] Paul Viola and Michael Jones. “Robust real-time object detection”. In: *International journal of computer vision* 4.34-47 (2001), p. 4.

[35] Paul Viola and Michael Jones. “Rapid Object Detection using a Boosted Cascade of Simple Features”. In: *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on.* Vol. 1. IEEE. 2001, pp. 511–518.

[36] Quan Yuan and Stan Sclaroff. “Is a detector only good for detection?” In: *2009 IEEE 12th International Conference on Computer Vision.* IEEE. 2009, pp. 1066–1073.

[37] Herbert J Ryser. “Combinatorial properties of matrices of zeros and ones”. In: *Classic Papers in Combinatorics.* Springer, 2009, pp. 269–275.

[38] Gábor T. Herman and Attila Kuba. *Discrete Tomography: Foundations, Algorithms, and Applications.* Springer Science & Business Media, 1999. ISBN: 978-1-4612-7196-3.

[39] Gábor T. Herman and Attila Kuba. *Advances in Discrete Tomography and Its Applications.* Advances in Discrete Tomography and Its Applications Applied and Numerical Harmonic Analysis. Birkhauser, 2007. ISBN: 0817636145.

[40] Stanley R. Deans. *The Radon Transform and Some of Its Applications.* New York: John Wiley and Sons, 1983.

[41] Margrit Betke, Esin Haritaoglu, and Larry S Davis. “Real-time multiple vehicle detection and tracking from a moving vehicle”. In: *Machine vision and applications* 12.2 (2000), pp. 69–83.

[42] Yanxi Liu, Robert Collins, and Yanghai Tsin. “Gait sequence analysis using frieze patterns”. In: *European Conference on Computer Vision*. Springer. 2002, pp. 657–671.

[43] Judith N Cederberg. *A course in modern geometries*. Springer Science & Business Media, 2013.

[44] Seungkyu Lee, Yanxi Liu, and Robert Collins. “Shape variation-based frieze pattern for robust gait recognition”. English (US). In: *2007 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR’07*. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. 2007. ISBN: 1424411807. DOI: 10.1109/CVPR.2007.383138.

[45] Ying Shan, Harpreet S. Sawhney, and Rakesh Kumar. “Vehicle Identification between Non-Overlapping Cameras without Direct Feature Matching”. In: *10th IEEE International Conference on Computer Vision (ICCV’05)* 1 (2005), pp. 378–385.

[46] Vedran Jelača et al. “Vehicle matching in smart camera networks using image projection profiles at multiple instances”. In: *Image and Vision Computing* 31 (2013), pp. 673–685.

[47] Richard A. Newcombe et al. “KinectFusion: Real-time Dense Surface Mapping and Tracking”. In: *Proceedings of the 2011 10th IEEE International Symposium on Mixed and Augmented Reality*. ISMAR ’11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 127–136. ISBN: 978-1-4577-2183-0. DOI: 10.1109/ISMAR.2011.6092378.

[48] Dmitry Chetverikov et al. “The trimmed iterative closest point algorithm”. In: *Object recognition supported by user interaction for service robots*. Vol. 3. IEEE. 2002, pp. 545–548.

[49] Patrick J. Drew et al. “Rapid Determination of Particle Velocity from Space-time Images Using the Radon Transform”. In: *Journal of computational neuroscience* 29.1-2 (2010), pp. 5–11. DOI: 10.1007/s10827-009-0159-1.

[50] David V Stark et al. “SDSS-IV MaNGA: characterizing non-axisymmetric motions in galaxy velocity fields using the Radon transform”. In: *Monthly Notices of the Royal Astronomical Society* 480.2 (2018), pp. 2217–2235.

[51] Yu Jeffrey Gu and Mauricio Sacchi. “Radon transform methods and their applications in mapping mantle reflectivity structure”. In: *Surveys in geophysics* 30.4-5 (2009), pp. 327–354.

[52] Fritz John. “Bestimmung einer Funktion aus ihren Integralen über gewisse Mannigfaltigkeiten”. In: *Mathematische Annalen* 109.1 (1934), pp. 488–520.

[53] Philomena Mader. “Über die Darstellung von Punktfunktionen imn-dimensionalen euklidischen Raum durch Ebenenintegrale”. In: *Mathematische Zeitschrift* 26.1 (1927), pp. 646–652.

[54] L. A. Shepp and B. F. Logan. “The Fourier reconstruction of a head section”. In: *IEEE Transactions on Nuclear Science* 21 (1974), pp. 21–43.

[55] Gene R. Gindi and Arthur F. Gmitro. “Optical feature extraction via the Radon transform”. In: *Optical Engineering* 23 (1984), pp. 499–506.

[56] Il Yong Chun, Ben Adcock, and Thomas M. Talavage. “Non-convex compressed sensing CT reconstruction based on tensor discrete Fourier slice theorem”. In: *Engineering in Medicine and Biology Society (EMBC), 2014 36th Annual International Conference of the IEEE* (2014), pp. 5141–5144.

[57] R. Clackdoyle, F. Noo, and M. S. Ould Mohamed. “Filtered-Backprojection Reconstruction Formula for 2D Tomography with Bilateral Truncation”. In: *Nuclear Science Symposium Conference Record, 2006. IEEE* 5 (2006), pp. 2895–2899.

[58] Xueling Zhu et al. “A Wavelet Multiscale De-Noising Algorithm Based on Radon Transform”. In: *Wavelet Transforms and Their Recent Applications in Biology and Geoscience* (2012), pp. 189–206.

[59] Alexander Kadyrov and Maria Petrou. “The trace transform and its applications”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 23.8 (2001), pp. 811–828.

[60] Paul VC Hough. *Method and means for recognizing complex patterns.* US Patent 3,069,654. Dec. 1962.

[61] M. van Ginkel, C.L. Luengo Hendriks, and L.J. van Vliet. *A short introduction to the Radon and Hough transforms and how they relate to each other.* Tech. rep. QI-01-2004, 2004.

[62] Richard O. Duda and Peter E. Hart. “Use of the Hough Transformation to Detect Lines and Curves in Pictures”. In: *Commun. ACM* 15.1 (Jan. 1972), pp. 11–15. ISSN: 0001-0782. DOI: `10.1145/361237.361242`.

[63] Ronny Ramlau and Otmar Scherzer. “The first 100 years of the Radon transform”. In: *Inverse Problems* 34.9 (July 2018). DOI: `10.1088/1361-6420/aacf27`.

[64] André R. Brodtkorb, Trond R. Hagen, and Martin L. SæTra. “Graphics Processing Unit (GPU) Programming Strategies and Trends in GPU Computing”. In: *J. Parallel Distrib. Comput.* 73.1 (Jan. 2013), pp. 4–13. ISSN: 0743-7315. DOI: `10.1016/j.jpdc.2012.04.003`.

[65] David B. Kirk and Wenmei W. Hwu. *Programming Massively Parallel Processors: A Hands-on approach.* Morgan Kaufmann, 2010.

[66] Timothy G Mattson, Beverly Sanders, and Berna Massingill. *Patterns for parallel programming.* Pearson Education, 2004.

[67] NVIDIA. *CUDA C Programming Guide.* NVIDIA Corporation, 2014.

[68] Sándor Szénási and Imre Felde. “Using multiple graphics accelerators to solve the two-dimensional inverse heat conduction problem”. In: *Computer Methods in Applied Mechanics and Engineering* 336 (2018), pp. 286–303. ISSN: 0045-7825. DOI: `doi.org/10.1016/j.cma.2018.03.024`.

[69] F. J. Anscombe. “Graphs in Statistical Analysis”. In: *The American Statistician* 27.1 (1973), pp. 17–21. DOI: `10.1080/00031305.1973.10478966`.

[70] Omar Javed, Khurram Shafique, and Mubarak Shah. “Appearance Modeling for Tracking in Multiple Non-Overlapping Cameras”. In: *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*. Vol. 2. CVPR ’05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 26–33. ISBN: 0-7695-2372-2. DOI: 10.1109/CVPR.2005.71.

[71] Warren S McCulloch and Walter Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[72] Frank Rosenblatt. “The perceptron: a probabilistic model for information storage and organization in the brain.” In: *Psychological review* 65.6 (1958), p. 386.

[73] Jürgen Schmidhuber. “Deep learning in neural networks: An overview”. In: *Neural networks* 61 (2015), pp. 85–117.

[74] Yoshua Bengio. “Deep learning of representations for unsupervised and transfer learning”. In: *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*. 2012, pp. 17–36.

[75] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *Nature* 521.7553 (2015), p. 436.

[76] Ian Goodfellow et al. *Deep learning*. Vol. 1. MIT press Cambridge, 2016.

[77] Rajat Raina, Anand Madhavan, and Andrew Y. Ng. “Large-scale Deep Unsupervised Learning Using Graphics Processors”. In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ICML ’09. Montreal, Quebec, Canada: ACM, 2009, pp. 873–880. ISBN: 978-1-60558-516-1. DOI: 10.1145/1553374.1553486.

[78] Yann A LeCun et al. “Efficient BackProp”. In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48.

[79] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Proceedings of the 25th International Conference on Neural Information Processing Systems*. NIPS’12. Lake Tahoe, Nevada: Curran Associates Inc., 2012, pp. 1097–1105.

[80] Yann LeCun, Koray Kavukcuoglu, and Clément Farabet. “Convolutional networks and applications in vision”. In: *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*. IEEE. 2010, pp. 253–256.

[81] David H Hubel and Torsten N Wiesel. “Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex”. In: *The Journal of physiology* 160.1 (1962), pp. 106–154.

[82] David H Hubel and Torsten N Wiesel. “Receptive fields and functional architecture in two nonstriate visual areas (18 and 19) of the cat”. In: *Journal of neurophysiology* 28.2 (1965), pp. 229–289.

[83] Kunihiko Fukushima. “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position”. In: *Biological cybernetics* 36.4 (1980), pp. 193–202.

[84] Yann LeCun et al. “Backpropagation applied to handwritten zip code recognition”. In: *Neural computation* 1.4 (1989), pp. 541–551.

[85] Yann LeCun, Yoshua Bengio, et al. “Convolutional networks for images, speech, and time series”. In: *The handbook of brain theory and neural networks* 3361.10 (1995).

[86] Yangqing Jia et al. “Caffe: Convolutional Architecture for Fast Feature Embedding”. In: *Proceedings of the 22nd ACM international conference on Multimedia.* ACM. 2014, pp. 675–678.

[87] Dmytro Mishkin, Nikolay Sergievskiy, and Jiri Matas. “Systematic evaluation of Convolution Neural Network advances on the ImageNet”. In: *Computer Vision and Image Understanding* 161 (2017), pp. 11–19.

[88] Christian Szegedy et al. “Going Deeper with Convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2015, pp. 1–9.

[89] Yoshua Bengio. “Practical Recommendations for Gradient-Based Training of Deep Architectures”. In: *Neural networks: Tricks of the trade.* Springer, 2012, pp. 437–478.

[90] Andrej Karpathy, FF Li, and J Johnson. “CS231n: Convolutional Neural Networks for Visual Recognition, 2016”. In: *URL http://cs231n. github. io* (2017).

[91] Jane Bromley et al. “Signature verification using a" siamese" time delay neural network”. In: *Advances in neural information processing systems.* 1994, pp. 737–744.

[92] Sumit Chopra, Raia Hadsell, and Yann LeCun. “Learning a similarity metric discriminatively, with application to face verification”. In: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on.* Vol. 1. IEEE. 2005, pp. 539–546.

[93] Taigman, Yaniv and Yang, Ming and Ranzato, Marc’Aurelio and Wolf, Lior. “DeepFace: Closing the Gap to Human-Level Performance in Face Verification”. In: *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition.* CVPR ’14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 1701–1708. ISBN: 978-1-4799-5118-5. DOI: 10.1109/CVPR.2014.220.

[94] Florian Schroff, Dmitry Kalenichenko, and James Philbin. “FaceNet: A Unified Embedding for Face Recognition and Clustering”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition 2015* abs/1503.03832 (2015).

[95] Oriol Vinyals and Charles Blundell and Timothy P. Lillicrap and Koray Kavukcuoglu and Daan Wierstra. “Matching Networks for One Shot Learning”. In: (2016). Ed. by D. D. Lee et al., pp. 3630–3638.

[96] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. “Siamese Neural Networks for One-shot Image Recognition”. In: *ICML 2015 Deep Learning Workshop.* 2015.

[97] Barret Zoph et al. “Learning transferable architectures for scalable image recognition”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 8697–8710.

[98] Barret Zoph and Quoc V Le. “Neural architecture search with reinforcement learning”. In: *International Conference on Learning Representations*. 2017.

[99] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. 1st. Cambridge, MA, USA: MIT Press, 1998. ISBN: 0262193981.

[100] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. “Neural Architecture Search: A Survey”. In: *Journal of Machine Learning Research* 20.55 (2019), pp. 1–21.

[101] Jia Li and F Li. “Cloud AutoML: Making AI accessible to every business”. In: (2018). URL: `https://www.blog.google/products/google-cloud/cloud-automl-making-ai-accessible-every-business/`.

[102] Ian Jolliffe. “Principal component analysis”. In: *International encyclopedia of statistical science*. Springer, 2011, pp. 1094–1096.

[103] Thomas H Cormen et al. *Introduction to algorithms*. MIT press, 2009.

[104] Michael L Pinedo. *Scheduling: theory, algorithms, and systems*. Springer, 2016.

[105] William Gropp et al. *Using MPI: portable parallel programming with the message-passing interface*. Vol. 1. MIT press, 1999.

[106] Martín Abadi et al. “Tensorflow: A system for large-scale machine learning”. In: *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 2016, pp. 265–283.

[107] Francois Chollet et al. *Keras: The Python Deep Learning library*. Astrophysics Source Code Library. 2018.

[108] Yann LeCun. *@ylecun on Twitter*. `https://twitter.com/ylecun/status/989610208497360896`. 26 Apr 2018.

[109] Dominic Masters and Carlo Luschi. “Revisiting small batch training for deep neural networks”. In: *arXiv preprint arXiv:1804.07612* (2018).

[110] Mauro Dell’Amico and Silvano Martello. “Optimal scheduling of tasks on identical parallel processors”. In: *ORSA Journal on Computing* 7.2 (1995), pp. 191–200.

[111] Alessandro Agnetis et al. “Multiagent scheduling”. In: *Berlin Heidelberg: Springer Berlin Heidelberg. doi* 10.1007 (2014), pp. 978–3.

[112] Dominik Zapletal and Adam Herout. “Vehicle Re-Identification for Automatic Video Traffic Surveillance”. In: *International Workshop on Automatic Traffic Surveillance (CVPR 2016)*. Las Vegas, US: IEEE Computer Society, 2016, pp. 1–7. ISBN: 978-0-7695-4989-7.

[113] Brenden M Lake, Ruslan R Salakhutdinov, and Josh Tenenbaum. “One-shot learning by inverting a compositional causal process”. In: *Advances in Neural Information Processing Systems 26*. Ed. by C. J. C. Burges et al. Curran Associates, Inc., 2013, pp. 2526–2534.

[114] Kaisa Miettinen. *Nonlinear multiobjective optimization.* Vol. 12. Springer Science & Business Media, 2012.

[115] Giuseppe Amato et al. “Deep learning for decentralized parking lot occupancy detection”. In: *Expert Systems with Applications* 72 (2017), pp. 327–334.

[116] De Cheng et al. “Person Re-Identification by Multi-Channel Parts-Based CNN With Improved Triplet Loss Function”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* June 2016.

[117] Alexander Hermans, Lucas Beyer, and Bastian Leibe. “In defense of the triplet loss for person re-identification”. In: *arXiv preprint arXiv:1703.07737* (2017).

[118] He Ma, Fei Mao, and Graham W Taylor. “Theano-mpi: a theano-based distributed training framework”. In: *European Conference on Parallel Processing.* Springer. 2016, pp. 800–813.

[119] Xianyan Jia et al. “Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes”. In: *arXiv preprint arXiv:1807.11205* (2018).

[120] Alexander Sergeev and Mike Del Balso. “Horovod: fast and easy distributed deep learning in TensorFlow”. In: *arXiv preprint arXiv:1802.05799* (2018).

[121] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).

Publications related to the dissertation

[K1] Gábor Kertész, Sándor Szénási, and Zoltán Vámossy. "Performance Measurement of a General Multi-Scale Template Matching Method". In: *Proceedings of INES 2015. 19th IEEE International Conference on Intelligent Engineering Systems* (Bratislava, Slovakia, Sept. 3–5, 2015). IEEE, 2015, pp. 153–158.

[K2] Gábor Kertész, Sándor Szénási, and Zoltán Vámossy. "Application and properties of the Radon transform for object image matching". In: *Proceedings of SAMI 2017. IEEE 14th International Symposium on Applied Machine Intelligence and Informatics* (Herlany, Slovakia, Jan. 26–28, 2017). IEEE, 2017, pp. 353–358.

[K3] Gábor Kertész, Sándor Szénási, and Zoltán Vámossy. "Parallelization Methods of the Template Matching Method on Graphics Accelerators". In: *Proceedings of CINTI 2015. 16th IEEE International Symposium on Computational Intelligence and Informatics* (Budapest, Hungary, Nov. 19–21, 2015). IEEE, 2015, pp. 161–164.

[K4] Gábor Kertész, Sándor Szénási, and Zoltán Vámossy. "A Novel Method for Robust Multi-Directional Image Projection Computation". In: *Proceedings of INES 2016. 20th IEEE International Conference on Intelligent Engineering Systems* (Budapest, Hungary, June 30–July 2, 2016). IEEE, 2016, pp. 239–243.

[K5] Gábor Kertész, Sándor Szénási, and Zoltán Vámossy. "Multi-Directional Image Projections with Fixed Resolution for Object Matching". In: *Acta Polytechnica Hungarica* 15.2 (2018), pp. 211–229.

[K6] Gábor Kertész, Sándor Szénási, and Zoltán Vámossy. "Multi-Directional Projection Transformations for Machine Learning based Object Matching". In: *SACI 2019 : IEEE 13th International Symposium on Applied Computational Intelligence and Informatics.* 2019, pp. 269–274.

[K7] Gábor Kertész, Sándor Szénási, and Zoltán Vámossy. "Vehicle Image Matching Using Siamese Neural Networks with Multi-Directional Image Projections". In: *12th IEEE International Symposium on Applied Computational Intelligence and Informatics, SACI 2018, Timisoara, Romania, May 17-19, 2018.* 2018, pp. 491–496.

[K8] Gábor Kertész, Sándor Szénási, and Zoltán Vámossy. “A novel method for Convolutional Neural Architecture Generation with memory limitation”. In: *Proceedings of SAMI2019. IEEE 17th World Symposium on Applied Machine Intelligence and Informatics* (Herlany, Slovakia, Jan. 24–26, 2019). IEEE, 2019, pp. 229–234.

[K9] Gábor Kertész, Sándor Szénási, and Zoltán Vámossy. “Distributed training and evaluation of projection-based descriptors in Siamese Neural Networks”. In: *Proceedings of the Sixth International Conference on Parallel, Distributed, GPU and Cloud Computing for Engineering*. 2019, Paper 25, 1–12. DOI: 10.4203/ccp.112.25.

Other, non-related publications

[KX1] Gabor Kertesz and Eva Hajnal. “Irisz Project: A Web Application for the Introduction of University Students to the Labor Market”. In: *International Symposium on Applied Informatics and Related Areas : AIS 2013 Szekesfehervar, Magyarorszag : Óbudai Egyetem, (2013).* 2013, pp. 125–129.

[KX2] Gabor Kertesz and Eva Hajnal. “Special Issues in the Development of a Large User Based Web Application”. In: *Proceedings of the IEEE 12th International Symposium on Applied Machine Intelligence and Informatics (SAMI 2014) Budapest, Magyarorszag : IEEE Hungary Section, (2014).* 2014, pp. 141–145.

[KX3] Gábor Kertész and Zoltán Vámossy. “Current challenges in multi-view computer vision”. In: *10th IEEE Jubilee International Symposium on Applied Computational Intelligence and Informatics, SACI 2015, Timisoara, Romania, May 21-23, 2015.* 2015, pp. 237–241.

[KX4] Gábor Kertész and Zoltán Vámossy. “A Brief Review of Recent Advances in Multi-View Computer Vision”. In: *Scientific Bulletin of Politechnica University of Timisoara - Transactions on Automatic Control and Computer ScienceE* 61(75) (2016), pp. 73–78. ISSN: 1224-600X.

[KX5] Gabor Kertesz et al. “Multiprocessing of an individual-cell based model for parameter testing”. In: *11th IEEE International Symposium on Applied Computational Intelligence and Informatics, SACI 2016, Timisoara, Romania, May 12-14, 2016.* 2016, pp. 491–496.

[KX6] Sandor Szenasi et al. “Comparison of Road Accident Black Spot Searching Methods”. In: *IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI 2018) Budapest, Magyarorszag : IEEE Hungary Section, (2018).* 2018, pp. 247–250.

[KX7] Sandor Szenasi et al. “Road Accident Black Spot Localisation using Morphological Image Processing Methods on Heatmap”. In: *IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI 2018) Budapest, Magyarorszag : IEEE Hungary Section, (2018).* 2018, pp. 251–256.

[KX8] Bence Danko and Gabor Kertesz. “Recognition of the Hungarian Fingerspelling Alphabet using Convolutional Neural Network based on Depth Data”. In: *IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI 2018) Budapest, Magyarorszag : IEEE Hungary Section, (2018).* 2018, pp. 41–46.

[KX9] Bence Danko and Gabor Kertesz. “Recognition of the Hungarian fingerspelling alphabet using Recurrent Neural Network”. In: *SAMI 2019 : IEEE 17th World Symposium on Applied Machine Intelligence and Informatics.* 2018, pp. 251–256.

Appendix A

Other resources

A.1 One-shot classification accuracy

Table A.1. Top measured one-shot classification accuracy for different inputs during N-way classification for one hidden convolutional-pooling layer pair.

	2	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
RGB Image	78,68	67,26	58,34	52,58	48,51	45,73	41,96	38,68	37,13
Radon85	80,99	69,32	62,52	56,82	53,53	49,51	46,51	42,81	40,49
Radon136	80,29	69,19	61,01	54,43	49,31	45,88	43,42	40,58	37,74
Trace	77,64	64,96	55,72	50,14	43,41	40,03	36,41	33,58	30,93
MDIPFL25	92,12	85,42	80,37	75,92	71,23	68,11	64,57	61,62	58,80
MDIPFL50	91,90	85,41	78,46	75,38	71,48	68,69	64,83	61,94	58,87
MDIPFL85	91,02	85,21	79,56	75,23	70,63	66,50	63,62	61,84	58,61
MDIPFL136	82,48	72,34	65,09	59,53	55,44	52,24	48,68	46,52	44,01

Table A.2. Top measured one-shot classification accuracy for different inputs during N-way classification for 2 hidden convolutional-pooling layer pairs.

	2	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
RGB Image	85,87	76,13	70,44	65,21	61,74	58,21	55,18	53,69	50,72
Radon85	80,32	68,04	59,84	52,50	48,14	43,59	40,93	39,21	36,84
Radon136	78,42	66,67	60,36	54,92	49,74	45,91	44,84	40,23	39,06
Trace	82,83	72,41	64,01	56,88	52,51	46,72	42,88	40,20	37,57
MDIPFL25	92,68	86,42	81,43	77,98	73,20	70,87	66,74	64,39	61,44
MDIPFL50	93,77	89,01	83,08	79,33	75,96	72,59	69,20	66,50	63,49
MDIPFL85	87,98	78,77	72,07	67,10	61,30	56,97	53,70	50,63	47,06
MDIPFL136	90,39	82,18	76,20	70,90	66,03	62,38	58,36	56,07	52,30

Table A.3. Top measured one-shot classification accuracy for different inputs during N-way classification for 3 hidden convolutional-pooling layer pairs.

	2	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
RGB Image	94,34	89,10	85,47	80,38	78,86	74,70	72,93	69,21	67,31
Radon85	80,43	68,63	59,39	53,13	47,42	43,31	39,98	37,58	34,52
Radon136	82,70	71,18	63,19	57,94	53,57	49,92	46,41	45,03	42,53
Trace	88,33	79,97	73,33	67,37	63,30	58,44	54,10	51,26	48,04
MDIPFL25	91,73	84,44	80,04	74,96	71,89	67,79	64,84	61,97	58,68
MDIPFL50	93,62	88,43	84,27	79,11	75,90	73,19	70,08	67,89	64,80
MDIPFL85	89,67	82,11	76,14	69,91	65,99	61,04	57,52	54,06	51,81
MDIPFL136	92,11	85,79	80,80	76,67	73,32	68,78	65,92	62,66	60,01

Table A.4. Top measured one-shot classification accuracy for different inputs during N-way classification for 4 hidden convolutional-pooling layer pairs.

	2	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
RGB Image	93,01	87,20	82,58	77,36	75,34	71,96	67,21	65,32	62,88
Radon85	93,22	87,27	83,47	78,68	75,22	72,73	68,81	66,52	64,11
Radon136	93,56	88,74	84,54	81,74	77,20	74,24	73,32	70,07	67,61
Trace	92,78	86,10	80,44	74,94	71,36	68,08	65,60	60,31	58,04
MDIPFL25	91,17	83,96	78,28	73,07	70,03	65,92	62,56	59,61	56,91
MDIPFL50	90,29	82,93	76,78	71,56	67,94	63,43	60,87	56,82	53,84
MDIPFL85	92,99	87,28	82,66	78,67	75,20	71,66	69,03	66,07	63,60
MDIPFL136	93,36	87,07	82,81	78,66	74,97	72,13	69,19	66,72	64,13

Table A.5. Top measured one-shot classification accuracy for different inputs during N-way classification for 5 hidden convolutional-pooling layer pairs.

	2	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
RGB Image	93,86	87,99	83,41	79,53	77,00	73,31	69,71	67,58	64,68
Radon85	92,78	87,89	83,94	79,40	75,73	73,39	70,74	68,22	66,28
Radon136	90,42	83,66	78,06	72,11	68,27	64,97	61,22	57,37	54,83
Trace	89,11	81,09	74,44	68,43	63,60	58,56	54,77	50,57	48,38
MDIPFL25	94,12	89,03	84,76	81,59	78,36	76,01	72,76	70,70	68,92
MDIPFL50	92,44	86,29	81,16	77,04	73,93	69,87	66,50	63,82	61,12
MDIPFL85	93,76	88,19	83,98	81,29	76,91	74,39	71,89	68,73	67,00
MDIPFL136	90,43	83,79	78,01	72,99	69,43	65,68	61,66	59,62	56,66

A.2 Generated neural architectures

Input(96×96×3) → Conv(13×13@64) → Pool(2×2) → Conv(7×7@64) → Pool(2×2) → Conv(7×7@64) → Pool(2×2) → Flatten() → FC(2048) → FC(1024) →, batch size:: 18 Input(96×96×3) → Conv(9×9@64) → Pool(2×2) → Conv(9×9@64) → Pool(2×2) → Conv(9×9@64) → Conv(5×5@128) → Conv(3×3@128) → Flatten() → FC(2048) →, batch size:: 24 Input(96×96×3) → Conv(9×9@64) → Pool(2×2) → Conv(9×9@64) → Pool(2×2) → Conv(9×9@64) → Conv(5×5@64) → Flatten() → FC(2048) →, batch size:: 24 Input(96×96×3) → Conv(9×9@64) → Pool(2×2) → Conv(9×9@64) → Pool(2×2) → Conv(9×9@64) → Conv(5×5@64) → Conv(5×5@256) → Flatten() → FC(1024) →, batch size:: 56 Input(96×96×3) → Conv(9×9@64) → Pool(2×2) → Conv(9×9@64) → Pool(2×2) → Conv(9×9@64) → Conv(5×5@64) → Conv(3×3@128) → Flatten() → FC(1024) →, batch size:: 42

List A.1. The top 5 architectures in terms of 10-way one-shot classification accuracy for RGB Image input sizes of $96 \times 96 \times 3$. The arguments given for Conv and Pool represent the window sizes for the given layer. In case of Conv layers, the third number represents the filter number.

Input(85×36×1) → Conv(9×11@64) → Pool(1×2) → Conv(7×5@64) → Conv(5×5@64) → Conv(3×3@64) → Conv(3×3@64) → Flatten() → FC(2048) →, batch size: 14 Input(85×36×1) → Conv(7×7@64) → Pool(1×2) → Conv(7×5@64) → Conv(7×5@64) → Conv(7×5@64) → Conv(5×3@64) → Flatten() → FC(2048) →, batch size: 16 Input(85×36×1) → Conv(9×11@64) → Pool(1×2) → Conv(7×5@64) → Conv(7×5@64) → Conv(3×5@64) → Flatten() → FC(2048) →, batch size: 14 Input(85×36×1) → Conv(9×11@64) → Pool(1×2) → Conv(9×9@64) → Conv(5×3@64) → Conv(5×3@64) → Flatten() → FC(2048) → FC(1024) →, batch size: 12 Input(85×36×1) → Conv(9×11@64) → Pool(1×2) → Conv(7×7@64) → Conv(5×3@64) → Conv(3×3@64) → Conv(3×3@64) → Flatten() → FC(2048) →, batch size: 14

List A.2. The top 5 architectures in terms of 10-way one-shot classification accuracy for Radon85 input sizes of $85 \times 36 \times 1$. The arguments given for Conv and Pool represent the window sizes for the given layer. In case of Conv layers, the third number represents the filter number.

Input(136×36×1) → Conv(9×11@64) → Pool(2×2) → Conv(5×7@64) → Conv(5×5@64) → Conv(5×3@64) → Flatten() → FC(2048) →, batch size: 18 Input(136×36×1) → Conv(9×11@64) → Pool(2×2) → Conv(7×7@64) → Conv(5×5@64) → Conv(5×3@64) → Flatten() → FC(2048) →, batch size: 18 Input(136×36×1) → Conv(9×11@64) → Pool(2×2) → Conv(9×7@64) → Conv(5×5@64) → Conv(5×3@64) → Flatten() → FC(2048) →, batch size: 18 Input(136×36×1) → Conv(9×11@64) → Pool(2×2) → Conv(7×5@64) → Conv(7×5@64) → Conv(5×5@64) → Flatten() → FC(2048) →, batch size: 20 Input(136×36×1) → Conv(9×11@64) → Pool(2×2) → Conv(7×7@64) → Conv(5×5@64) → Conv(3×3@64) → Flatten() → FC(2048) →, batch size: 18

List A.3. The top 5 architectures in terms of 10-way one-shot classification accuracy for Radon136 input sizes of $136 \times 36 \times 1$. The arguments given for Conv and Pool represent the window sizes for the given layer. In case of Conv layers, the third number represents the filter number.

Input(137×72×1) → Conv(15×13@64) → Pool(3×2) → Conv(15×13@64) → Conv(9×9@64) → Conv(9×9@64) → Flatten() → FC(1024) →, batch size: 42 Input(137×72×1) → Conv(9×11@64) → Pool(3×2) → Conv(9×11@64) → Conv(9×11@64) → Conv(9×11@64) → Flatten() → FC(1024) →, batch size: 52 Input(137×72×1) → Conv(15×13@64) → Pool(3×2) → Conv(13×13@64) → Conv(11×11@64) → Conv(9×7@64) → Flatten() → FC(1024) →, batch size: 44 Input(137×72×1) → Conv(15×13@64) → Pool(3×1) → Conv(13×13@64) → Conv(13×13@64) → Conv(9×7@64) → Conv(9×7@64) → Flatten() → FC(1024) →, batch size: 36 Input(137×72×1) → Conv(15×13@64) → Pool(3×2) → Conv(13×13@64) → Pool(1×2) → Conv(7×9@64) → Flatten() → FC(1024) →, batch size: 52

List A.4. The top 5 architectures in terms of 10-way one-shot classification accuracy for Trace input sizes of 137 × 72 × 1. The arguments given for Conv and Pool represent the window sizes for the given layer. In case of Conv layers, the third number represents the filter number.

Input(25×36×1) → Conv(7×5@64) → Conv(7×5@64) → Conv(5×5@64) → Conv(5×3@128) → Conv(5×3@128) → Flatten() → FC(2048) →, batch size: 22 Input(25×36×1) → Conv(7×5@64) → Conv(7×5@64) → Conv(5×5@64) → Conv(5×3@64) → Conv(5×3@128) → Flatten() → FC(2048) →, batch size: 22 Input(25×36×1) → Conv(7×5@64) → Conv(7×5@64) → Conv(5×3@64) → Conv(5×3@64) → Conv(5×3@128) → Flatten() → FC(2048) →, batch size: 20 Input(25×36×1) → Conv(7×5@64) → Conv(7×5@64) → Conv(7×5@64) → Conv(5×5@64) → Conv(3×3@128) → Flatten() → FC(2048) →, batch size: 24 Input(25×36×1) → Conv(5×7@64) → Conv(5×7@64) → Conv(5×7@64) → Conv(5×7@64) → Conv(5×5@64) → Flatten() → FC(2048) →, batch size: 22

List A.5. The top 5 architectures in terms of 10-way one-shot classification accuracy for MDIPFL25 input sizes of $25 \times 36 \times 1$. The arguments given for Conv and Pool represent the window sizes for the given layer. In case of Conv layers, the third number represents the filter number.

Input(50×36×1) → Conv(9×11@64) → Pool(1×2) → Conv(5×7@64) → Conv(5×7@64) → Flatten() → FC(2048) → FC(1024) →, batch size: 18 Input(50×36×1) → Conv(9×11@64) → Pool(1×2) → Conv(5×7@64) → Conv(5×7@64) → Flatten() → FC(2048) →, batch size: 28 Input(50×36×1) → Conv(11×9@64) → Pool(2×2) → Conv(9×9@64) → Pool(2×1) → Flatten() → FC(2048) → FC(1024) →, batch size: 18 Input(50×36×1) → Conv(11×9@64) → Conv(9×9@64) → Conv(9×7@64) → Conv(9×7@64) → Conv(5×7@64) → Flatten() → FC(1024) →, batch size: 52 Input(50×36×1) → Conv(27×27@64) → Pool(3×2) → Flatten() → FC(2048) → FC(1024) →, batch size: 18

List A.6. The top 5 architectures in terms of 10-way one-shot classification accuracy for MDIPFL50 input sizes of $50 \times 36 \times 1$. The arguments given for Conv and Pool represent the window sizes for the given layer. In case of Conv layers, the third number represents the filter number.

Input(85×36×1) → Conv(9×7@64) → Pool(1×2) → Conv(7×7@64) → Conv(7×5@64) → Conv(5×3@64) → Conv(3×3@64) → Flatten() → FC(2048) →, batch size: 16 Input(85×36×1) → Conv(9×11@64) → Pool(1×2) → Conv(7×5@64) → Conv(5×5@64) → Conv(5×3@64) → Conv(5×3@64) → Flatten() → FC(2048) →, batch size: 16 Input(85×36×1) → Conv(9×11@64) → Pool(1×2) → Conv(7×7@64) → Conv(7×5@64) → Conv(5×3@64) → Flatten() → FC(2048) →, batch size: 14 Input(85×36×1) → Conv(9×11@64) → Pool(1×2) → Conv(7×5@64) → Conv(5×5@64) → Conv(5×3@64) → Conv(5×3@64) → Flatten() → FC(2048) →, batch size: 16 Input(85×36×1) → Conv(7×7@64) → Pool(1×2) → Conv(5×5@64) → Conv(5×5@64) → Conv(5×5@64) → Conv(5×3@64) → Flatten() → FC(2048) →, batch size: 14

List A.7. The top 5 architectures in terms of 10-way one-shot classification accuracy for MDIPFL85 input sizes of $85 \times 36 \times 1$. The arguments given for Conv and Pool represent the window sizes for the given layer. In case of Conv layers, the third number represents the filter number.

Input(136×36×1) → Conv(9×11@64) → Pool(2×2) → Conv(5×5@64) → Conv(5×5@64) → Conv(3×5@64) → Flatten() → FC(2048) →, batch size: 18 Input(136×36×1) → Conv(9×11@64) → Pool(2×2) → Conv(7×5@64) → Conv(7×5@64) → Conv(5×5@64) → Flatten() → FC(2048) →, batch size: 20 Input(136×36×1) → Conv(9×11@64) → Pool(2×2) → Conv(9×7@64) → Conv(7×5@64) → Conv(3×3@64) → Flatten() → FC(2048) →, batch size: 18 Input(136×36×1) → Conv(9×11@64) → Pool(2×2) → Conv(9×7@64) → Conv(9×7@64) → Flatten() → FC(2048) →, batch size: 18 Input(136×36×1) → Conv(9×11@64) → Pool(2×1) → Conv(9×11@64) → Pool(2×1) → Conv(9×7@64) → Conv(7×5@64) → Conv(5×5@64) → Flatten() → FC(1024) →, batch size: 58

List A.8. The top 5 architectures in terms of 10-way one-shot classification accuracy for MDIPFL136 input sizes of $136 \times 36 \times 1$. The arguments given for Conv and Pool represent the window sizes for the given layer. In case of Conv layers, the third number represents the filter number.