Beuth Hochschule für Technik Berlin

Fachbereich VI - Informatik und Medien
Luxemburger Straße 20a
13353 Berlin
http://www.beuth-hochschule.de/vi/

Evaluierung lokaler Bild-Merkmalsextraktion mit Neuronalen Netzen

Masterarbeit zur Erlangung des akademischen Grades
Master of Engineering (M.Eng)

Jan Malte Dempewolf
Matrikelnummer: 866235

16. Januar 2019

Betreuer
Prof. Dr. Kristian Hildebrand

Gutachter
Prof. Dr. Volker Sommer

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Berlin den 16. Januar 2019

Jan Dempewolf

Abstract

The present work describes the evaluation of local image feature extraction with neural networks. Two feature extraction methods are compared: *Deep Local Features* [27] - a neural network published in 2016 and the *Scale Invariant Feature Transform*-algorithm [22], a widely used feature extraction technique in images, which was published in 1999. Thus, two fundamentally different methods are evaluated in order to evaluate the performance of this neural network relatively. *Deep Local Features* was trained with *o*utdoor data to perform a specific task and *Scale Invariant Feature Transform*, an algorithm generally formulated and not trained for specific scenes or image data. The necessary theoretical basics of image transformations, feature extraction and neural networks are introduced. Two datasets are used for the evaluation. The self-created *Outdoor* dataset - the Karlshorst dataset, taken in Berlin and a generated transformation dataset resulting from a subset of the Karlshorst dataset. Based on selected metrics for image *Matching* such as *Recall*, *Precision*, number of *Inliers* and Euclidean distances of the descriptors, the empirical behavior is assessed and visualized. As a result, it can be seen that the reviewed *Deep Local Features* architecture scores either equally well or better in both evaluated datasets. Especially regarding the behavior of *Recall* and *Precision*, the *Deep Local Feature* method is far superior in detecting *Outdoor* scenes. The hypothesis that *Recall* and *Precision* for the Karlshorst dataset will be better for *Deep Local Features* than for *Scale Invariant Feature Transform* could be confirmed. *Deep Local Features* remains robust under image transformations like smoothing, gamma correction, and zoom. As a continuation of the work an analysis of the *training* of the neural network could be done. In addition, an attempt could be made to improve the poorer performance under image-rotation-transformation by taking more images with stronger rotation in *training*.

Zusammenfassung

Die vorliegende Arbeit beschreibt die Evaluierung lokaler Bild-Merkmalsextraktion mit Neuronalen Netzen. Es werden zwei Merkmalsextraktions-Verfahren verglichen: *Deep Local Features* [27] - ein Neuronales Netz, welches erstmals 2016 veröffentlicht wurde und der *Scale Invariant Feature Transform*-Algorithmus [22], ein weit verbreitetes Verfahren zur Extraktion von Merkmalen in Bildern, welches 1999 publiziert wurde. Damit werden zwei grundsätzlich verschieden arbeitende Verfahren evaluiert, um die Leistungsfähigkeit dieses Neuronalen Netzes relativ bewerten zu können. *Deep Local Features* wurde mit *Outdoor-Trainings*-Daten trainiert um eine spezifische Aufgabe zu erfüllen und *Scale Invariant Feature Transform*, ein Algorithmus der allgemein beziehungsweise nicht für bestimmte Szenen oder Bilddaten formuliert wurde. Einleitend werden die nötigen theoretischen Grundlagen der Bild-Transformationen, Merkmalsextraktion und Neuronalen Netzen erläutert. Für die Evaluation werden zwei Datensätze genutzt. Der selbst erstellte *Outdoor*-Datensatz - Karlshorst-Datensatz, aufgenommen in Berlin und ein generierter Transformations-Datensatz resultierend aus einer Teilmenge des Karlshorst-Datensatzes. Anhand von ausgewählten Metriken für das Bild-*Matching* wie *Recall*, *Precision*, Anzahl an *Inliern* sowie euklidischen Distanzen der *Deskriptoren* wird das empirische Verhalten bewertet sowie visualisiert. Im Ergebnis wird deutlich, dass die getestete *Deep Local Features*-Architektur in beiden evaluierten Datensätzen entweder durchschnittlich gleich gut oder besser abschneidet. Besonders hinsichtlich des Verhaltens von *Recall* und *Precision* ist das *Deep Local Feature*-Verfahren beim Erkennen von *Outdoor*-Szenen deutlich überlegen. Die Hypothese der Arbeit, dass *Recall* und *Precision* für den Karlshorst-Datensatz besser für *Deep Local Features* ausfallen wird als für *Scale Invariant Feature Transform*, konnte damit bestätigt werden. Unter Bildtransformationen zeigt sich, dass *Deep Local Features* robust unter Glättung, Gamma-Korrektur und Zoom bleibt. Als Weiterführung der Arbeit könnte eine Analyse des *Trainings* des Neuronalen Netzes erfolgen. Zudem könnte versucht werden das schlechtere Abschneiden bei Bild-Rotation (Transformation) zu verbessern, indem beim *Training* mehr Bilder mit stärkerer Rotation eingehen.

Danksagung

Zunächst möchte ich mich bei meinen Betreuer Prof. Dr. Kristian Hildebrand bedanken, der mich fachlich hervorragend unterstützt und begleitet hat. Des Weiteren möchte ich meinen Eltern Viola und Uwe sowie meinem Bruder Nils, für ihre Unterstützung, meinen Dank aussprechen, ganz besonders auch meinem Cousin Kai Leippert und Freund Denis Baskan.

Inhaltsverzeichnis

Symbole

Analysis

$\lim_{a \to b} f$	Limes der Funktion f, wenn a sich b nähert
$\mathbf{H}$	Hesse-Matrix
$\mathcal{F}$	Fouriertransformation
$\mathcal{O}$	asymptotische obere Schranke
∂	partielle Ableitung
$f : A \to B$	Funktion f bildet ab von A nach B
f^{-1}	Umkehrfunktion von f
∇	Nabla-Operator
e	Eulersche Zahl

Lineare Algebra

$\mathbf{A}$	Matrix $\mathbf{A}$
$\mathbf{A}^T$	Transponierte der Matrix $\mathbf{A}$
$\mathbf{A}^{-1}$	Inverse der Matrix $\mathbf{A}$
$\mathbf{v}$	Vektor $\mathbf{v}$
$\|\mathbf{v}\|$	Norm des Vektors $\mathbf{v}$
$\mathbf{A} \cdot \mathbf{B}$	Matrixmultiplikation der Matrix $\mathbf{A}$ und $\mathbf{B}$
$\mathbf{E}$	Einheitsmatrix
$\|\mathbf{A}\|$	Determinante der Matrix $\mathbf{A}$

Andere Symbole

$\cap$	Schnittmenge
$\in$	Element von
$\wedge$	Konjunktion
$\Leftrightarrow$	Äquivalenz
$\neg$	Negation
$\overline{x}$	Durchschnitt von $x_1, \ldots, x_n$
$\Rightarrow$	Implikation
$\sim$	Proportionalität
$\sum$	Summe

$\|X\|$	Kardinalität einer Menge X

Wahrscheinlichkeitstheorie

σ	Standardabweichung
$\varphi(x, \mu, \sigma)$	Dichtefunktion der Gaußverteilung
$E(x)$ oder μ	Erwartungswert
$V(x)$	Varianz

Spezifisch

κ	*Keypoint*
$\mathfrak{f}$	*Feature* bzw. *Deskriptor*
$\mathcal{KD}$	Karlshorst-Datensatz
$\mathfrak{d}$	Dimension
$\mathfrak{K}$	Menge der *Keypoints*
d	Euklidischer Abstand
$D(x, y, \sigma)$	*Difference of Gaussians*
fn	*False Negative*
fp	*False Positive*
$G(x, y, \sigma)$	*Gaussian*
$I(x, y)$	Bild
$L(x, y, \sigma)$	Faltung von $G(x, y, \sigma)$ und $I(x, y)$
$n(x, y)$	Bildrauschen
T	Schwellenwert für *Inlier*
tn	*True Negative*
tp	*True Positive*
X_β	Verfahren X mit BF-*Nearest Neigbor* Suche und *Ratio Test* nach Lowe
X_ζ	Verfahren X mit *ckdTree Nearest Neigbor*-Suche und geometrische Verifikation mit RANSAC
z	Digitalzoom
α	Aktivierungsfunktion
$\mathbf{W}$	Gewichte eines CNNs
$\mathcal{KTD}$	Karlshorst-Transformations-Datensatz
$\mathcal{M}$	Stärke der Bild-Transformation
$\mathcal{T}_\mathcal{M}$	Bild-Transformations mit Stärke

$\mathcal{T}$	Bild-Transformation
$\mathfrak{e}_i$	Erregungszustand eines Neuron i
$\mathfrak{F}$	Menge der Merkmale
$\mathfrak{I}$	Menge der *Inlier*
$\mathfrak{i}$	*Inlier*
$\mathfrak{I}_d$	Menge der *Inlier* durch euklidische Distanz
$\mathfrak{I}_F$	Menge der *Inlier* durch euklidische Distanz sowie zusätzlichen Filter
$\mathfrak{i}_f$	falscher bzw. unechter *Inlier*
$\mathfrak{i}_t$	richtiger bzw. echter *Inlier*
ϕ	Nichtlinearität
θ	Rotationswinkel
F_1	F-Maß
M	Mathematisches Modell

Zahlenmengen

$\mathbb{C}$	Komplexe Zahlen
$\mathbb{R}$	Reelle Zahlen
$\mathbb{R}^*$	Reelle Zahlen ohne Null
$\mathbb{N}$	Natürliche Zahlen

Akronyme

CNN	*Convolutional Neural Network*
DELF	*Deep Local Features*
SIFT	*Scale Invariant Feature Transform*
LIFT	*Learned Invariant Feature Transform*
SfM	*Structure from Motion*
ReLU	*Rectified Linear Unit*
GPS	*Global Positioning System*
DLT	*Direct Linear Transformation*
IDE	*Integrated Development Environment*
TPU	*Tensor Processing Unit*
GPU	*Graphics Processing Unit*
OpenCV	*Open Source Computer Vision Library*
BSD	*Berkeley Software Distribution*
BF	*Brute Force*
SGD	*Stochastic Gradient Descent*
DFT	*Discrete Fourier Transform*
OpenCL	*Open Computing Language*
SRI	*Stanford Research Institute*
DoG	*Difference of Gaussians*
LoG	*Laplacian of Gaussian*
ADALINE	*Adaptive Linear Neuron*
MLP	*Multilayer Perceptron*
PCA	*Principal Component Analysis*
KI	Künstliche Intelligenz
CSV	*Comma Separated Values*
FCN	*Fully Convolutional Network*

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einführung

1.1. Motivation

In der heutigen Zeit, mit zunehmender Anzahl an Kameras und der damit auch steigenden Fülle digitaler Bilder, nimmt die Bedeutung der Bildverarbeitung sowie des Bildverstehens immer weiter zu. Die lokale Bild-Merkmalsextraktion hat sich dabei als starkes Werkzeug der Informatik beweisen können. Der Ansatz, nicht das gesamte Bild, sondern nur einzelne Bereiche des Bildes zu beschreiben, hat sich als sehr sinnvoll erwiesen. Die Merkmalsextraktion findet dabei Anwendungen in der Objekterkennung, beim Bild-*Stitchings* oder dem *Image Retrieval* und wird in vielen digitalen Bereichen des täglichen Lebens wie *Smartphones*, *Automotive*, *Shopping* et cetera bereits vielfach verwendet. Grundsätzlich lässt sich eine solche Extraktion mittels bewährter Algorithmen wie *Scale Invariant Feature Transform*, *Speeded Up Robust Features* usw. durchführen. Mit der zunehmenden Bedeutung und Erfolgen Neuronaler Netze in unterschiedlichsten Bereichen, wie Verstehen von Sprache oder eben der maschinellen Verarbeitung von Bilddaten, entwickelt sich das Interesse auch die Merkmalsextraktion durch ein Neuronales Netz berechnen zu lassen. Die enorme Stärke des Neuronalen Netzes besteht nun darin, basierend auf *Trainings*-Daten zunächst einmal zu Erlernen, welche Bereiche des Bildes wichtig beziehungsweise stabil sind. Damit ergibt sich die Möglichkeit Neuronale Netze für bestimmte Aufgaben zu spezialisieren. Sollen beispielsweise *Outdoor*-Szenen wiedererkannt werden, wird auch für das Training ein solcher Input verwendet. Damit erhofft man sich Vorteile gegenüber allgemeineren Verfahren wie *Scale Invariant Feature Transform* (*SIFT*) oder *Speeded Up Robust Features* (*SURF*) zu erlangen. Denn solche Verfahren extrahieren unter Umständen viele *Keypoints* in Bereichen des Bildes, welche im zweiten Bild, mit dem verglichen werden soll, überhaupt nicht vorhanden sind. Soll beispielsweise in einem Supermarkt im Bild eine Verpackung eines Produktes erkannt werden, dann wäre es natürlich sinnlos, würde das Verfahren im Bild gar nicht in diesem Bereich *Keypoints* extrahieren sondern beispielsweise im Bild-Hintergrund oder im Verkaufsregal. Um eine solche Logik oder auch Aufmerksamkeit (*Attention* beziehungsweise *Attentive*) zu erzielen gibt es einen neuen Ansatz: In 2017 wurde der Artikel *Large-Scale Image Retrieval with Attentive Deep Local Features* [28] veröffentlicht. Er beschreibt einen Ansatz der lokalen Merkmalsextraktion innerhalb einer ganzen *Pipeline* für *Image Retrieval*. Der Ansatz wurde bereits durch Hyeonwoo Noh et al. [28] evaluiert und es zeigte sich, dass *Deep Local Features* (*DELF*) die globalen und lokalen Deskriptoren nach dem neuesten Stand der Technik im großen Maßstab um beträchtliche Margen übertrifft. Abbildung 1 zeigt die gemessene *Precision-Recall*-Kurve für den getesteten *Google-Landmarks*-Datensatz aus dem Artikel von Hyeonwoo Noh et al. Eine detaillierte vergleichende Analyse sowie das Verhalten unter Bild-Transformation liegt bis zum heutigem Tag jedoch noch nicht vor. Daher soll in dieser Arbeit eine solche ausführliche, vergleichende Evaluation erfolgen und in Relation zum so erfolgreichen *Scale Invariant Feature Transform*-Algorithmus beschrieben werden.

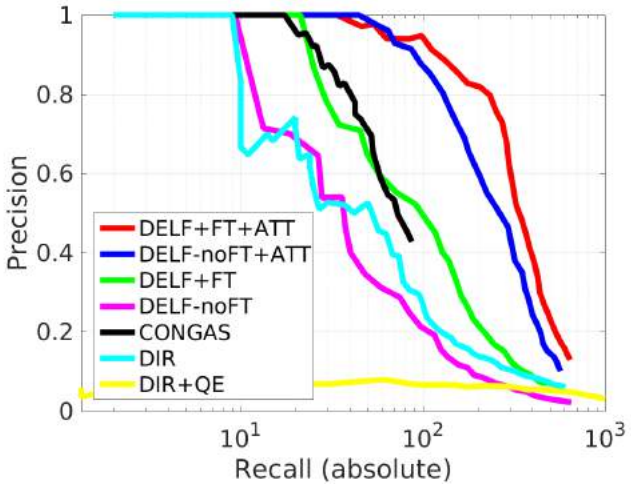


Abbildung 1: Verhalten von *Recall* gegen *Precision* für *Deep Local Features* (*DELF*), Quelle: [28]

Als Hypothese der Arbeit lässt sich formulieren, dass *Recall* und *Precision* für den Karlshorst-Datensatz besser für *Deep Local Features* ausfallen wird als für das *Scale Invariant Feature Transform*-Verfahren.

1.2. Aufbau der Arbeit

Zunächst werden in Abschnitt 2 die nötigen theoretischen Grundlagen der Bildverarbeitung eingeführt, welche nötig sind um den erstellten Datensatz zu verstehen und das Verhalten der Verfahren unter Bild-Transformation bewerten zu können. Im Anschluss wird die klassische Bild-Merkmalsextraktion mit *Scale Invariant Feature Transform* beschrieben sowie Grundlagen der Neuronen Netze und des *Machine Learnings* eingeführt. Das Neuronale Netz bzw. die verwendete Architektur wird in Abschnitt 3 erläutert. Der lokale *Feature*-Deskriptor *Deep Local Features*, welcher sich für das Abrufen von Bildern in großem Maßstab eignet, wird vorgestellt und mathematisch beschrieben. Im Abschnitt 4 - Implementierung wird die *Software*-Umgebung sowie der *Quellcode* in *TensorFlow* und *OpenCV* näher erläutert. Die Evaluierung der lokalen Bild-Merkmalsextraktion mit Neuronalen Netzen wird in Abschnitt 5 vergleichend zu *Scale Invariant Feature Transform* durchgeführt, um die *Performance* einordnen zu können. Dafür werden zunächst die nötigen Bewertungsmethoden beschrieben und das allgemeine Vorgehen des Bild-*Matchings* in *Pseudocode* erfolgt. Des Weiteren werden die zwei Datensätze - Karlshorst-Datensatz und Karlshorst-Transformations-Datensatz eingeführt. Zuletzt erfolgt in Abschnitt 6 eine Zusammenfassung und Bewertung der empirischen Analyse.

2. Theoretische Grundlagen

In folgenden Abschnitt der theoretischen Grundlagen werden Konzepte der allgemeinen Bildverarbeitung erläutert. Diese werden zum einen bei generierten Karlshorst-Transformations-Datensatz benötigt und zum anderen werden fundamentale Algorithmen beschrieben, welche für das spätere Bild-*Matching* genutzt werden. Darauf folgt die $SIFT$-Merkmalsextraktion. Da das neuronale Netz - $DELF$ mit diesem verglichen und evaluiert werden soll, muss zunächst eine theoretische Basis dafür geschaffen werden. Der letzte Abschnitt in den theoretischen Grundlagen sind die Neuronalen Netze selbst.

2.1. Bildverarbeitung

Die digitale Bildverarbeitung ist ein wichtiges Gebiet der Informatik und Elektrotechnik. Im englischsprachigem Raum bezeichnet man dieses Feld meist als *Digital Image Processing* oder auch *Computer Vision*. Die digitale Bildverarbeitung beschäftigt sich mit der Verarbeitung von digitalen Bildern mit dem finalem Zweck der Interpretation und Auswertung dieser visuellen Informationen. [12]

2.1.1. Lineare Abbildung und Matrizen

Für die Evaluation der Merkmalsextraktion ist es nötig Transformationen auf Bilder anzuwenden, um das Verhalten des *Matchings* dabei zu messen. Transformationen auf Bildern lassen sich über lineare Abbildungen beschreiben.

Theorem 1 *Sei V und W ein Vektorraum über $\mathbb{K}$. Dann sei eine lineare Abbildung $f : V \to W$ gegeben, wenn $f(x + y) = f(x) + f(y) =$ und $f(\lambda x) = \lambda f(x)$ für alle $x, y \in V$ gilt. [14]*

Sie ermöglichen es mit den Methoden der Matrix-Rechnung zu arbeiten. Das Rechnen mit Matrizen unterliegt folgenden Rechenregeln bezügliche Multiplikation mit einem Skalar, Addition und Multiplikation zweier Matrizen:

$$\lambda \mathbf{A} = \lambda(a_{ij}) = (\lambda a_{ij}) \tag{1}$$

$$\mathbf{A} + \mathbf{B} = (a_{ij}) + (b_{ij}) = (a_{ij} + b_{ij}) \tag{2}$$

$$\mathbf{A} \cdot \mathbf{B} = (a_{ij}) \cdot (b_{jk}) = \sum_{j=1}^{n} a_{ij} \cdot b_{jk}.[6] \tag{3}$$

2.1.2. Homogene Koordinaten

Um affine Abbildungen in Bildern mit Hilfe der Matrixmultiplikation abbilden zu können, bedarf es einer theoretischen Erweiterung der kartesischen Koordinaten, beziehungsweise der euklidischen Geometrie. So kann beispielsweise eine Translation, eine einfache Vektor-Addition, zunächst nicht als Matrix-Operation beschrieben werden. Eine mathematische Lösung dafür sind Homogene Koordinaten. Der Ansatz wurde 1827 von August Ferdinand Möbius, ein deutscher Mathematiker und Astronom, in seiner Arbeit *Der barycentrische Calcül* vorgestellt. Einem Vektor $x \in \mathbb{R}^n$ wird eine Dimension hinzugefügt. Diese enthält eine Komponente $h \in \mathbb{R}^*$. Für den zweidimensionalen Fall $x \in \mathbb{R}^2$ folgt somit, $x_h \in \mathbb{R}^3$. [1]

$$\mathbf{x}_h = \begin{bmatrix} hx \\ hy \\ h \end{bmatrix} \tag{4}$$

Der Parameter h kann grundsätzlich beliebig gewählt werden, muss aber ungleich Null sein. Meist wird h auf 1 gesetzt. Jedes gewöhnliche kartesische Koordinatenpaar kann also äquivalent durch

einen dreidimensionalen, homogenen Koordinatenvektor dargestellt werden. Eine 2D- Translation lässt sich nun als Matrixmultiplikation schreiben:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{5}$$

Diese Erweiterung ermöglicht das einfachere Rechnen mit Matrizen, siehe Abschnitt 2.1.1. Zudem können damit in einer Matrix mehrere Transformation gespeichert werden, was den Rechenaufwand verringert sowie eine formale Beschreibung zulässt. Dies folgt unter anderem aus Theorem 1 und Rechenregeln für Matrizen. [1]

2.1.3. Affine Transformation

Eine affine Transformation (oder einfach eine Affinität) ist eine nicht singuläre lineare Transformation, auf die eine Translation $\mathbf{t}$ folgt. In Matrix-Schreibweise lässt sich die Affinität wie folgt formulieren:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{6}$$

In Block Form lässt sie sich als

$$\mathbf{x}' = \mathbf{H_A x} = \begin{bmatrix} \mathbf{A} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \mathbf{x} \tag{7}$$

schreiben, wobei $\mathbf{A}$ eine nicht singuläre $2x2$ Matrix und $\mathbf{t}$ ein $2x1$ Spaltenvektor ist.

Theorem 2 *Sei A eine quadratische Matrix, existiert* $\mathbf{A}^{-1}$*, heißt* $\mathbf{A}$ *invertierbar oder regulär, sonst singulär. Des weiteren gilt,* $\mathbf{A} \cdot \mathbf{A}^{-1} = \mathbf{E}$ *mit* $\mathbf{E}$ *als Einheits-Matrix. Folgende Aussagen sind äquivalent:* $\exists \mathbf{A}^{-1} \Leftrightarrow det\mathbf{A} \neq 0$*. [6]*

Eine planare affine Transformation hat sechs Freiheitsgrade, entsprechend der sechs Matrixelementen. Die Transformation kann daher aus einer drei Punkt-Korrespondenzen berechnet werden. Die Bild-Rotation, eine Affine Transformation, um den Winkel θ (gegen den Uhrzeigersinn) lässt sich in Matrixform

$$\mathbf{A}_{Rotation} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \tag{8}$$

als $\mathbf{A}_{Rotation}$ beschreiben.

Abbildung 2: Bild Pferde-Statue

Abbildung 3: Bild Pferde-Statue, $\theta = \frac{\pi}{8}$

Das Bild in Abbildung 2 wird um 22.5° gedreht, das Resultat wurde in Abbildung 3 visualisiert. Eine hilfreiche Methode zum Verständnis der geometrischen Effekte der linearen Operation **A** einer affinen Transformation ist die Zusammensetzung zweier grundlegender Transformationen, nämlich Rotationen und nicht-isotroper (*Isotropie*) Skalierungen. Dabei sei die Skalierung als Matrix

$$\mathbf{A}_{Skalierung} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \tag{9}$$

mit s_x als Skalierung in x-Richtung, s_x Skalierung in y-Richtung, gegeben. [12] Die affine Matrix **A** kann wie folgt

$$\mathbf{A} = \mathbf{R}(\theta)\mathbf{R}(-\phi)\mathbf{D}\mathbf{R}(\phi) \tag{10}$$

zerlegt werden. Dabei sei $\mathbf{R}(\theta)$ und $\mathbf{R}(\phi)$ Rotationen um den Winkel θ bzw. ϕ, mit **D** als Diagonal-Matrix mit folgenden Einträgen:

$$\mathbf{D} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \tag{11}$$

Diese Dekomposition folgt aus der Singulärwertzerlegung.

Theorem 3 *Eine Matrix $Q \in R^{n\times n}$ heißt orthogonal, falls $Q^TQ = E$. [2]*

Mit Theorem 3 lässt sich die folgende Gleichung umformen.

$$\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^T \overset{\mathbf{V}^T\mathbf{V}=E}{=} \mathbf{U}\mathbf{V}^T\mathbf{V}\mathbf{D}\mathbf{V}^T \tag{12}$$

Die affine Matrix **A** ist daher die Verkettung einer Drehung um ϕ, gefolgt von einer Skalierung um λ_1 sowie λ_2 (in der gedrehten x- und y-Richtung), eine Drehung zurück um $-\phi$ und einer letzten Drehung um θ. Die vorherigen Schritte werden im folgendem visualisiert.

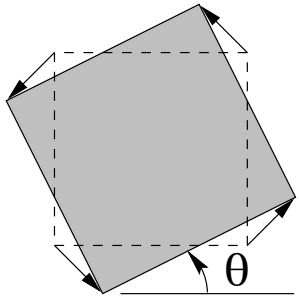


Abbildung 4: $\mathbf{R}(\theta)$, Quelle: [10]

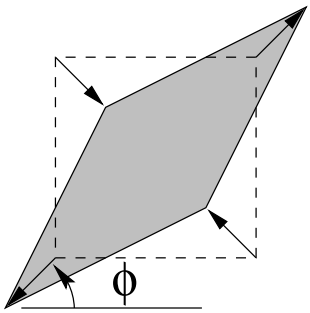


Abbildung 5: $\mathbf{R}(-\phi)\mathbf{D}\mathbf{R}(\phi)$, Quelle: [10]

In der Abbildung 5 kann man erkennen, dass die Skalierungsrichtungen der Verformung orthogonal sind.

Invarianz Da eine affine Transformation eine nicht-isotrope Skalierung umfasst, werden die Ähnlichkeitsinvarianten von Längenverhältnissen und Winkeln zwischen Linien unter einer Affinität nicht beibehalten.

(i) Parallele Geraden: Betrachtet man zwei parallele Geraden, schneiden sich diese in einem Punkt$(x1, x2, 0)^T$ im Unendlichen. Bei einer affinen Transformation wird dieser Punkt auf einen anderen Punkt im Unendlichen abgebildet. Folglich werden die parallelen Geraden auf Geraden abgebildet, die sich weiterhin im Unendlichen schneiden und sind daher nach der Transformation parallel.

(ii) Längenverhältnis von parallelen Geraden-Segmenten: Die Längen-Skalierung eines Geraden-Segmentes hängt nur vom Winkel zwischen der Geraden-Richtungen und den Skalierungs-Richtungen ab. Angenommen die Gerade hat den Winkel α zur x-Achse der orthogonalen Skalierungs-Richtung, dann entspricht die Stärke (*Magnitude*) der Skalierung

$$M_\lambda = \sqrt{(\lambda_1 \cos\alpha)^2 + (\lambda_2 \sin\alpha)^2} \tag{13}$$

dem euklidischen Abstand. Diese Skalierung ist bei allen Geraden mit derselben Richtung gleich und bricht daher im Verhältnis von parallelen Segment-Längen ab.

(iii) Verhältnis von Flächen: Diese Invarianz kann direkt aus der Zerlegung abgeleitet. Rotationen und Übersetzungen wirken sich nicht auf die Fläche aus, daher sind hier nur die Skalierungen durch λ_1 und λ_2 von Bedeutung. Der Effekt ist, dass die Fläche um $\lambda_1\lambda_2$ skaliert wird, was gleich $\det \mathbf{A}$ ist. Somit wird die Fläche einer beliebigen Form durch $\det \mathbf{A}$ skaliert, sodass die Skalierung für ein Verhältnis von Flächen aufgehoben wird. Es ist ersichtlich, dass dies nicht für eine projektive Transformation gilt.

Eine Affinität ist orientierungserhaltend oder -umkehrend, je nachdem, ob $\det \mathbf{A}$ positiv oder negativ ist. Durch $\det \mathbf{A} = \lambda_1\lambda_2$ hängt die Eigenschaft nur vom Vorzeichen der Skalierungen ab. [10]

2.1.4. Projektive Transformation

Die Projektive Transformation ist eine allgemeine, nicht singuläre lineare Transformation von homogenen Koordinaten. Dies verallgemeinert die affine Transformation aus Abschnitt 2.1.3, welche die Zusammensetzung einer allgemeinen nicht singulären linearen Transformation inhomogener Koordinaten und einer Translation ist. Die Transformation kann in Blockform beschrieben werden, wobei der Vektor $\mathbf{v}^T = (v1, v2)$ ist.

$$\mathbf{x}' = \mathbf{H}_P\mathbf{x} = \begin{bmatrix} \mathbf{A} & \mathbf{t} \\ \mathbf{v}^T & v \end{bmatrix} \mathbf{x} = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ v_1 & v_2 & v \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{14}$$

$$\Longleftrightarrow$$

$$x' = a_{11}x + a_{12}y + t_x \cdot 1 \tag{15a}$$
$$y' = a_{21}x + a_{22}y + t_y \cdot 1 \tag{15b}$$
$$1 = v_1x + v_2y + v \cdot 1 \tag{15c}$$

Die Matrix hat neun Elemente, wobei nur ihr Verhältnis von Bedeutung ist, sodass die Transformation durch acht Parameter angegeben wird. Man beachte, dass es nicht immer möglich ist, die Matrix so zu skalieren, dass v Eins ist, da v auch Null sein kann. Eine projektive Transformation zwischen zwei Ebenen kann aus vier Punkt-Korrespondenzen berechnet werden, wobei auf keiner Ebene drei kollinear (*Kollinearitaet*) sind. Im Gegensatz zu Affinitäten (Affine Transformation) kann in P^2 nicht zwischen Orientierungserhaltungs- und Orientierungsumkehrungs-Projektionen unterschieden werden.

Invarianz Die grundlegendste projektive Invariante ist das Querverhältnis von vier kollinearen Punkten: Ein Längenverhältnis einer Linie ist unter Affinitäten invariant, jedoch nicht unter Projektivität.

Theorem 4 *Das Doppelverhältnis ist die grundlegende projektive Invariante von $P1$. Bei 4 Punkten $\mathbf{x}_i$ ist das Doppelverhältnis definiert als*

$$Cross(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4) = \frac{|\mathbf{x}_1\mathbf{x}_2||\mathbf{x}_3\mathbf{x}_4|}{|\mathbf{x}_1\mathbf{x}_3||\mathbf{x}_2\mathbf{x}_4)|} \tag{16}$$

mit

$$|\mathbf{x}_i\mathbf{x}_j| = \det \begin{bmatrix} x_{i1} & x_{j1} \\ x_{i2} & x_{i2} \end{bmatrix}. \tag{17}$$

Ein Doppelverhältnis von Längen auf einer Geraden ist somit eine projektive Invariante. [10]

2.1.5. Lineare Filter

Im Gegensatz zu Punktoperatoren nutzen Filter auch die Umgebung eines Bildpunktes. Filter erhalten ein Eingangsbild und geben ein gefiltertes Ausgangsbild. Die Änderung dürfen nur im Ausgangsbild vorgenommen werden, da sonst eine Abhängigkeit von der Filter-Richtung gegeben wäre. Für lineare Filter gilt:

$$h = \alpha_1 \cdot h_1 + \alpha_2 \cdot h_2 \tag{18}$$

mit h_1, h_2 und h als Filteroperatoren und Konstanten $\alpha_1, \alpha_2 \in \mathbb{R}$. Jede Linearkombination aus linearen Filtern h_1 und h_2 führt wiederum zu einem linearen Filter h. Mathematisch betrachtet handelt es sich um eine Faltung. Allgemein können Filter für unterschiedliche Aufgaben eingesetzt werden: Bildverbesserung, Kantendetektion usw. [3]

Tiefpassfilter Diese Art der Filter glätten ein Bild, sie entfernen Grauwertkanten sowie Rauschen. Dies sind Anteile, welche in einer 2D-Fouriertransformation $\mathcal{F}(u, v)$ des Bildes $I(x, y)$ in den hohen Ortfrequenzen zu finden sind. Es folgt die kontinuierliche Integraltransformation

$$\mathcal{F}(u, v) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} I(x, y) \cdot e^{-i2\pi(ux+vy)}\, dx\, dy, \quad \mathcal{F}(u, v) \in \mathbb{C} \tag{19}$$

mit $f(x, y)$ als beliebiger Funktion. Die diskrete Berechnung für Bilder der Größe $M \times N$ wird mit der Diskrete Fourier-Transformation

$$\mathcal{F}(u, v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} I(x, y) \cdot e^{-i2\pi(\frac{ux}{M} + \frac{vy}{N})} \tag{20}$$

berechnet. [7]

(a) `original` (b) `blur1` (c) $\mathcal{F}(u,v)$ von `original` (d) $\mathcal{F}(u,v)$ von `blur1`

Abbildung 6: Frequenzraum für Bild `antenne0` sowie gefiltertes (Gauß-Filter) Bild

Nach dem Filtern mit dem Tiefpass bleiben niederfrequente Anteile übrig: Bildflächen, in denen die Grauwerte wenig variieren. [3] Die Abbildung 6d zeigt an den Rändern dunklere Bereiche auf als in Abbildung 6c, was einer Filterung der hohen Frequenzen entspricht (größere Entfernung zum Koordinatenursprung) und ein Erhalten der niederen Frequenzen nahe dem Ursprung.

Mittelwert-Filter Die trivialste Form der Glättung der Grauwerte ist sicherlich, die Bildung des Mittelwerts der benachbarten Pixel eines Bildpunktes. Dafür werden meist quadratische Filter verwendet, um die Symmetrie zu erhalten. Linearen Filteroperationen werden mit Hilfe von Filterkernen durchgeführt. Dabei werden für jedes Pixel des Ausgangsbildes alle unter dem Kern befindlichen Pixel des Eingangsbildes mit dem jeweiligen Filterkoeffizienten multipliziert und anschließend addiert. Letztlich handelt es sich somit um Kreuzkorrelation eines Bildes $I(x,y)$ mit einem Filter $h(x,y)$. [3]

$$g(x,y) * h(x,y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} I(x+u, y+v)\, du\, dv \tag{21}$$

Gauß-Filter Die Filtermatrix dieses Glättungsfilters entspricht einer diskreten, zweidimensionalen Gaußfunktion

$$\mathbf{H}(x,y) = e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{22}$$

wobei die Standardabweichung σ den Radius der glockenförmigen Funktion definiert. Das mittlere Bildelement erhält das maximale Gewicht, die Werte der übrigen Koeffizienten nehmen mit steigender Entfernung zur Mitte ab. Die Gaußfunktion ist isotrop, vorausgesetzt die Filtermatrix ist groß genug für eine ausreichende Näherung. [1] Die Werte des diskreten Filterkerns eines $m \times m$ des Gauß-Filters werden über die Binomialkoeffizienten eines Binoms der Ordnung $m-1$ bestimmt. Es folgt die Gleichung für ein Binom 2. Ordnung, um einen 3-dimensioanlen Filterkern zu definieren:

$$(a+b)^2 = 1a^2 + 2ab + 1b^2. \tag{23}$$

Der Mittlere Wert (3×3) bzw. mittlere Zeile oder Spalte ($m > 3$) des Filters sind wieder die Binomialkoeffizienten aber multipliziert mit $m-1$. Beispielhaft wird nun der Filterkern

(Approximation) für einen 5×5 Gauß-Filter visualisiert. [3]

$$h(x,y) = \frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix} \tag{24}$$

2.1.6. Gamma-Korrektur

Der Begriff der Helligkeit oder auch Intensität ist essentiell für das Verständnis von Bildern. Der Zusammenhang der Intensität wird offenbar über den numerischen Wert des einzelnen Pixel im Bild reflektiert. Die Menge des einfallenden Lichtes hat einen Einfluss auf die Helligkeit des Bildes. Nachdem ein Bild unter gewissen Licht-Verhältnissen aufgenommen wurde, soll diese Größe (Intensität) manipuliert werden. Die Gamma-Korrektur sei eine Punktoperation, die dazu dient, die unterschiedlichen Charakteristiken von Aufnahme- und Ausgabegeräten zu kompensieren und Bilder entsprechend anzupassen.

Mathematische Beschreibung Für die Beschreibung der Gamma-Korrektur dient eine Potenzfunktion.

$$y = f_\gamma(a) = a^\gamma \tag{25}$$

mit $a \in [0,1]$ und $\gamma > 0$.

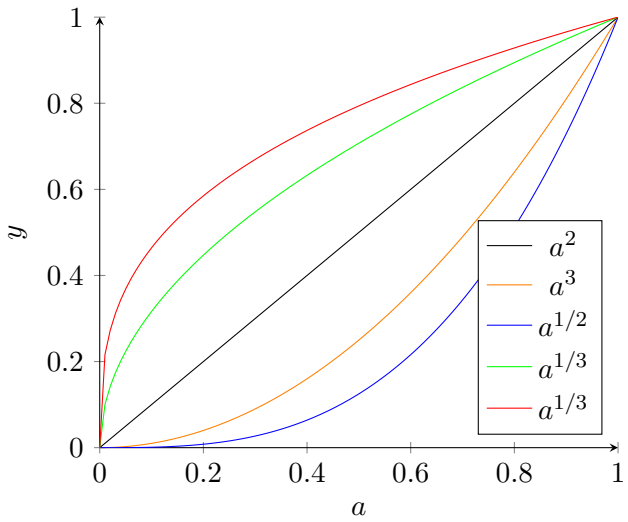


Abbildung 7: Gamma-Korrektur

Die Funktionen in Abbildung 7 sind stetig und streng monoton wachsend. Eine weitere positive Eigenschaft der Potenzfunktion ist ihre einfache Invertierbarkeit. [1]

$$a = f_\gamma^{-1} = y^{\frac{1}{\gamma}} \tag{26}$$

2.1.7. Bildrauschen

Rauschen kann aus unterschiedlichen Perspektiven betrachtet werden. Wenn Rauschen adaptiv vorliegt, lässt sich folgende Beschreibung formulieren.

$$I(x,y) = f(x,y) + n(x,y) \tag{27}$$

Dabei ist I das reale Bild, f das ideale Bild und n das Rauschen. Damit ist der Zusammenhang eine einfache Summation und linear. [3]

***Salt and Pepper* Rauschen** Das *Salt and Pepper* Rauschen wird im Allgemeinen durch einen Defekt des Kamera-Sensors, durch Software-Fehler oder durch Hardware-Fehler bei der Bilderfassung oder -übertragung verursacht. Es ist nur ein Teil aller Bild-Pixel beschädigt, während andere Pixel nicht betroffen sind. Der Rauschwert kann entweder minimal (0) oder maximal (255) der Graustufe des Bildes sein. Bei einem Bild (8-Bit pro Pixel) liegt der typische Intensitätswert für Pfeffer-Rauschen nahe bei 0 und für Salz-Rauschen nahe 255. [15]

$$n(x,y) = \begin{cases} 0, & \textit{Pepper} \text{ Rauschen} \\ 255, & \textit{Salt} \text{ Rauschen} \end{cases} \tag{28}$$

2.1.8. Vergrößern und Schrumpfen Digitaler Bilder

Zunächst sei das einfachste Verfahren des Vergrößerns sicherlich die Pixelreplikation, also das Verdoppeln der Pixel.[36] Dieses Feld der Bildverarbeitung steht im Zusammenhang zur Bild-Abtastung und Quantisierung. Zoomen kann dabei als eine Art der Überabtastung und Verkleinern des Bildes als Unterabtastung verstanden werden. Das Zoomen benötigt zwei Schritte: Erzeugung neuer Pixel Positionen sowie die Berechnung der Grauwert bezüglich dieser Koordinaten. Ein einfaches Beispiel soll nun folgen: Angenommen man hätte ein Bild der Größe 500×500, es soll im den Faktor 1,5 vergrößert werden, was eine neue Bild-Größe von 750×750 ergibt. Konzeptuell kann man ein Raster 750×750 über das Originalbild legen. Natürlich würde dies zu Lücken führen, da ein größeres Raster über ein kleineres Bild gelegt wird. Um das gesamte Bild auszufüllen, wird nun für die 250 Pixel, welche noch keinen Grauwert haben, nach dem nächsten Nachbarn (*Nearest Neighbor Interpolation*) im Originalbild gesucht und deren Grauwert zugewiesen. Danach wird die Original-Größe des Bildes berücksichtigt und das Bild an diese Größe angepasst. [7] *Nearest Neighbor Interpolation* ist sehr schnell, hat jedoch den Nachteil, dass es bei starker Vergrößerung zum Schachbrett-Effekt führt. Der Ansatz über die bilinearen Interpolation

$$v(x,y) = ax + by + cx \cdot y + d \tag{29}$$

zeigt diesen Effekt nicht. Elegantere Verfahren arbeiten mit Interpolationsverfahren, welche die Größe des Bildes erhöhen, aber die eigentliche Auflösung des Bildes nicht ändern. Der Digitale Zoom nutzt diese Interpolationsverfahren. [36] Im Folgendem wird das Zoomen visualisiert. Dafür wurde die Software Umgebung Scipy [52] verwendet. Die Methode `scipy.ndimage.zoom` arbeitet darin mit *Spline*-Interpolation.

Abbildung 8: Bild Pferde-Statue ohne Zoom

Abbildung 9: Bild Statue 2,1-fachem Zoom

Für die Abbildung 9 wurden Splines dritter Ordnung (kubische)

$$s_i(x) = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i \tag{30}$$

genutzt.

2.1.9. Fundamentale Algorithmen und Datenstrukturen

Im Folgendem wird zunächst der RANSAC-Algorithmus beschrieben. Dieser wird für das *Matching* der Datensätze eine entscheidende Rolle spielen, um nach dem Finden von potentiellen *Inliern* zu einem gewissen Maße besser zu validieren, ob es sich um einen echten *Inlier* $\mathfrak{i}_t$ handelt. Des Weiteren werden die nötigen Methoden zur Berechnung der *Homography* beschreiben, da genau diese beim *Matching* bei *DELF* geschätzt werden soll. Zuletzt wird die wichtige Datenstruktur des k-d-Baums erläutert, welche ebenfalls für das *Matching* benötigt wird, im Speziellem zum schnelleren approximativem Finden von nächsten Nachbarn (*Nearest Neighbor*).

RANSAC-Algorithmus Der *Random Sample Consensus*-Algorithmus ist ein allgemeiner und sehr erfolgreicher robuster Schätzer. Er wurde erstmals 1981 von Fischler und Bolles [5] bei SRI International veröffentlicht. Der Algorithmus macht es möglich Parameter für ein mathematisches Modell zu schätzen, im Besonderem unter der Voraussetzung, dass die zugrunde liegenden Daten eine hohe Anzahl an Ausreißern (*Outliers*) enthalten. Es handelt sich bei dem Verfahren um einen nicht deterministischen, iterativen Algorithmus in dem Sinne als, dass er nur mit einer bestimmten Wahrscheinlichkeit ein gutes Ergebnis liefert, wobei diese Wahrscheinlichkeit mit zunehmender Anzahl von Iterationen zunimmt. [10] An einem einfachen Beispiel, das sich leicht visualisieren lässt, soll der RANSAC-Algorithmus beschrieben werden. Das Schätzen der Parameter einer Geradengleichung soll aus einer Menge von zweidimensionalen Punkten erfolgen: Man kann sich vorstellen, dass eine eindimensionale affine Transformation

$$x' = ax + b \tag{31}$$

zwischen entsprechenden Punkten auf zwei Linien geschätzt werden soll. Das in Abbildung 10 dargestellte Problem ist folgendes: Finde in einer Menge aus 2D-Datenpunkten die Linie, die die Summe der rechtwinkligen Abstände im Quadrat (orthogonale Regression) minimiert, vorausgesetzt, dass keiner der gültigen Punkte abweicht von dieser Linie um mehr als t Einheiten.

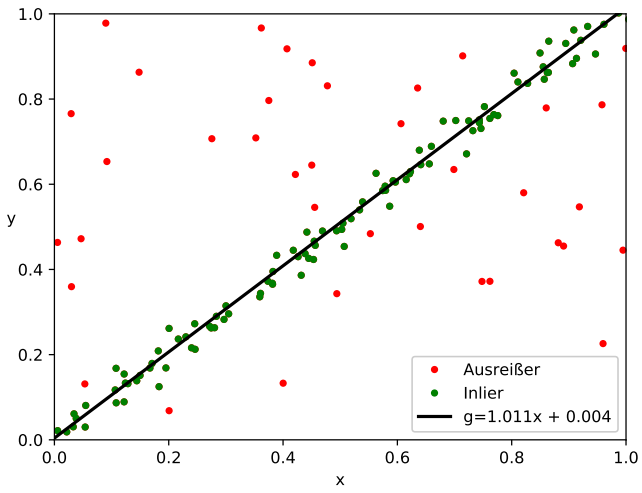


Abbildung 10: Ausgleichsgerade g mit 17 Iterationen für 145 Datenpunkte, $t = 0,05$

Dies sind eigentlich zwei Probleme: eine Linie, die zu den Daten passt und eine Klassifizierung der Daten in *Inlier* (gültige Punkte) und Ausreißer (*Outliers*). Die Schwelle t wird gemäß dem Messrauschen eingestellt (zum Beispiel $t = 3\sigma$). Es gibt viele Arten robuster Algorithmen für dieses Problem, deren Verwendung in gewissem Maße vom Anteil der Ausreißer abhängt. Wenn zum Beispiel bekannt ist, dass es nur einen Ausreißer gibt, kann jeder Punkt der Reihe nach gelöscht und die Linie vom Rest geschätzt werden. Der Algorithmus arbeitet wie folgt: Zwei Punkte werden zufällig ausgewählt. Das Punkte-Paar definiert eine Gerade. Die Unterstützung für diese Linie wird anhand der Anzahl der Punkte gemessen, die innerhalb eines Distanz-Schwellenwert t liegen. Diese zufällige Auswahl (Punkte-Paar) wird mehrmals wiederholt und die Gerade mit der meisten Unterstützung wird als robuste Schätzung betrachtet. Die Punkte innerhalb des Schwellenabstands t sind die *Inlier* und bilden den gleichnamigen Konsenssatz. Wird ein Punkte-Paar ausgewählt, bei dem zumindest einer der beiden Punkte ein Outlier ist, wird diese Gerade folglich wenig Unterstützung erhalten. [10]
Nun soll eine allgemeine Formulierung des *Random Sample Consensus*-Algorithmus erfolgen. Seien mindestens s Datenpunkte gegeben, um die freien Parameter des Modells zu instanziieren. Die vier Parameter des Algorithmus sind s, t, T und N werden übergeben, S sind die Daten.

Algorithmus 1 RANSAC(s,S,t,T,N)

1: **for** `i = 0; i < N; i++` **do**
2: wähle zufällig eine Stichprobe von s Datenpunkten aus der Menge S aus
3: schätze das Modell $M(s)$ aus dieser Teilmenge
4: bestimme die *Inlier* S_t, die innerhalb des Distanz-Schwellenwert t liegen
5: **if** $|S_t| > T$ **then**
6: **return** M
7: die größte Konsensmenge S_t wird ausgewählt und M wird unter Verwendung aller Punkte in der Teilmenge S_t neu geschätzt
8: **return** M

Algorithmus 1 beschreibt das allgemeine RANSAC-Verfahren zum Schätzen der Parameter eines mathematischen Modells M.

Berechnung der *Homography* Zu bestimmen sei, bei gegebenen $n > 4$ Bildpunkt-Korrespondenzen (x_i, x_i'), die *Maximum Likelihood* Schätzung $\hat{\mathbf{H}}$ der *Homography*-Zuordnung zwischen zwei Bildern.

Berechnung des Fehlers im Bild Dabei wird ein Punkt x_i im Bild I_1 mit $\mathbf{H}$ transformiert und der Abstand zum Punk x_i' bestimmt. $\mathbf{H}$ wird variiert um folgenden Ausdruck zu minimieren:

$$E = \sum_{i=1}^{n} d(\mathbf{x_i'}, \mathbf{H}\mathbf{x_i})^2 \tag{32}$$

Fehler bei der symmetrischen Übertragung Im Falle der symmetrischen Kosten-Funktion:

$$E_S = \sum_{i=1}^{n} d(\mathbf{x_i}, \mathbf{H}^{-1}\mathbf{x_i}')^2 + d(\mathbf{x_i}', \mathbf{H}\mathbf{x_i})^2 \tag{33}$$

Sei $\hat{\mathbf{x}}_\mathbf{i}' = \hat{\mathbf{H}}\hat{\mathbf{x}}_\mathbf{i}$, dann lässt sich der Algorithmus wie folgt formulieren. [10]

Algorithmus 2 Estimation_of_Homography(x_i, x_i')

1: Initialisierung: Berechnung einer erster Schätzung von $\hat{\mathbf{H}}$, um einen Ausgangspunkt für die geometrische Minimierung zu erhalten. Verwendung des linear normalisierten DLT-Algorithmus oder RANSAC (Algorithmus 1), um $\hat{\mathbf{H}}$ aus vier Punkt-Korrespondenzen zu berechnen.
2: Geometrische Minimierung von: $\sum_{i=1}^{n} d(\mathbf{x_i}, \hat{\mathbf{x}}_\mathbf{i})^2 + d(\mathbf{x_i}', \hat{\mathbf{x}}_\mathbf{i}')^2$

Multidimensionale Daten Es gibt viele unterschiedliche Repräsentationen von hochdimensionalen Daten. Die Art der Repräsentationen ist abhängig von der Dimension $\mathfrak{d}$ sowie der Domäne. Der einfachste Ansatz die Daten zu speichern, ist eine sequenzielle Liste. Für diesen Fall besteht keinerlei Sortierung der Daten. Allgemein gilt, bei N hochdimensionalen Datenpunkten benötigt ein exaktes *Match* für eine Anfrage (*Query*) eines beliebigen Datenpunktes

$$\mathcal{O}(N \cdot \mathfrak{d}), \tag{34}$$

da im *Worst Case* alle N Daten mit deren $\mathfrak{d}$ Dimensionen durchsucht werden(*Brute-Force*). [34]

Der *k-d-Baum* Eine relativ effiziente Datenstruktur für das Suchen von nächsten Nachbarn (*Nearest Neighbor*) in geometrischen Daten ist der *k-d-Baum*. Er stellt eine natürliche Verallgemeinerung des eindimensionalen Suchbaums dar. Sei U eine Menge aus n Punkten im Raum $\mathbb{R}^k$, welche gespeichert werden sollen. Zur Vereinfachung soll zunächst angenommen werden, dass sich die Punkte in U in alle Koordinaten $u_1 \dots u_k$ unterscheiden. Im Folgendem wird der Fall $k = 2$ betrachtet, somit gilt $\mathbf{u} = \begin{bmatrix} x, y \end{bmatrix}^\top$. Es wird eine der Koordinaten als *Splitkoordinate* gewählt (beliebig). Weiterhin wurde die x-Koordinate gewählt. Zudem wird der sogenannte *Splitwert* s gewählt, welcher ungleich der Einträge der x-Koordinaten aller Punkte in U ist. Dann wird die Punktmenge U durch die *Splitgerade* $g_s = s$ in zwei Teilmengen

$$\begin{aligned} U_{<s} &= \{(x, y) \in U, x < s\} = D \cap \{g_s < s\} \\ U_{>s} &= \{(x, y) \in U, x > s\} = D \cap \{g_s > s\} \end{aligned} \tag{35}$$

zerlegt. Auf die resultierenden Teilmengen $U_{<s}$ und $U_{>s}$ wird wieder die Teilung aus Gleichung 35 angewendet, nur wird diesmal die y-Koordinate als *Splitkoordinate* genutzt. Danach wird wieder die x-Koordinaten verwendet. Der Vorgang wird so lange wiederholt bis einelementige Punktmengen übrig bleiben. Diese werden nicht mehr weiter zerteilt. Auf diese Weise entsteht dann der *2-d Baum* für die Punktmenge U. Damit entspricht jeder innere Knoten des Baumes einer *Splitgeraden*. [18]

2.2. Merkmalsextraktion mit *SIFT*

Der *SIFT*-Algorithmus, stehend für *Scale Invariant Feature Transform* (*Skaleninvarianz*), ist ein Verfahren in der Computer Vision, um lokale Merkmale in Bildern zu erkennen und zu beschreiben. Der Algorithmus wurde 1999 von David Lowe [22] veröffentlicht und von der *University of British Columbia* patentiert. Das Ziel des Verfahrens ist die Lokalisierung von interessanten bzw. markanten Bildpunkten und zwar robust gegenüber typischen Bildtransformationen, auch über mehrere Bilder oder auch ganze Bildfolgen hinweg. *SIFT* verwendet dabei das Konzept des Skalenraums (*Scale Space*) um Bildereignisse über mehrere Skalenebenen bzw. bei unterschiedlicher Bildauflösung zu lokalisieren. Dadurch ergibt sich unter anderem eine gewisse Resistenz gegenüber Größenänderung. Damit ist es dann beispielsweise möglich ein beliebiges Objekt zu verfolgen, wenn sich dieses auf die Kamera zubewegt und sich dessen Bild-Größe kontinuierlich ändert. Des Weiteren lässt sich damit in einer *Stiching*-Anwendung Bilder mit unterschiedlichen Zoom-Einstellungen in Übereinstimmung bringen. Die rechte Abbildung 11 illustriert beispielhaft die *Keypoints* für das bekannte Bild von Lenna. Neben der ursprünglichen Implementierung existieren Weiterentwicklungen in den Softwareumgebungen wie OpenCV oder AutoPano. In den letzten Jahren, konnte eine Beschleunigung des Originalverfahrens durch algorithmische Vereinfachungen und den Einsatz von GPUs erreicht werden. [1] Die Berechnung von *SIFT-Features* besteht aus vier Schritten:

Abbildung 11: Visualisierung *SIFT-Keypoints* mit *OpenCV* für Lenna mit Betrag des *Keypoints* und Orientierung, Quelle des Originalbilds: [60]

(i) Skalenraum Extrema-Erkennung: Die erste Berechnungsstufe durchsucht alle Skalen und Bildpositionen. Es wird effizient implementiert, indem eine *Difference of Gaussians*-Funktion verwendet wird, um potentielle Interessenspunkte zu identifizieren, die in Bezug auf Skalierung und Orientierung unverändert (invariant) sind.

(ii) Keypoint-Lokalisierung: An jedem Kandidatenstandort bzw. Bildposition ist ein detailliertes Modell zur Bestimmung von Position und Skalierung nötig. *Keypoints* werden auf der Grundlage von Stabilitätsmessungen ausgewählt.

(iii) Orientierungszuordnung: Jeder *Keypoint*-Position wird eine oder mehrere Orientierungen zugewiesen, die auf den lokalen Bildgradientenrichtungen basieren. Alle zukünftigen Operationen werden für Bilddaten ausgeführt, die für jedes Merkmal relativ zur zugewiesenen Ausrichtung, Skalierung und Position transformiert wurden, wodurch eine Invarianz für diese Transformationen bereitgestellt wird.

(iv) *Keypoint-Deskriptor*: Die lokalen Bildgradienten werden auf der ausgewählten Skala in der Region um jeden *Keypoint* gemessen. Diese werden in eine Darstellung umgewandelt, die ein erhebliches Maß an lokalen Formverzerrungen und Beleuchtungsänderungen zulässt. [21]

2.2.1. Detektion von Extrema im *Scale-Space*

Es sollen Bereiche im *Skalenraum* gefunden werden, welche invariant(*Invarianz*) im Bezug auf Translation, Skalierung und Rotation sind. Zudem sollten der Bereich nur minimal durch Rauschen und leichte Verzerrung beeinträchtigt werden. [22] Für die weitere Betrachtung wird die Dichtefunktion der Gauß-Verteilung mit der Standardabweichung σ und dem Erwartungswert μ

$$\varphi(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \quad x \in]-\infty, \infty[\tag{36}$$

eine entscheidende Rolle spielen. [6] Denn für die Bild-Glättung hat sich der Gauß-Kernel, als besonders bewährt erwiesen (nach Witkin 1983 [37]).

$$G(x, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}} \tag{37}$$

Die Abbildungsvorschrift für die zweidimensionale DoG (*Difference of Gaussians* $G(x, y, \sigma)$) lautet wie folgt: Aus dem Produkt in x und y Richtung, erhält man die zweidimensionale Dichtefunktion der Gauß-Verteilung, welche in der nachfolgenden Abbildung 12 visualisiert ist.

$$G(x, y, \sigma) = G(x, \sigma) \cdot G(y, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}} \cdot \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{y^2}{2\sigma^2}} = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{38}$$

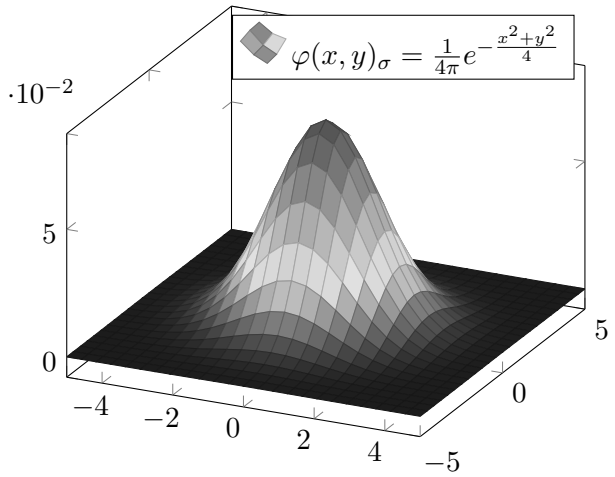


Abbildung 12: Zweidimensionale Dichtefunktion der Gauß-Verteilung mit $\sigma = \sqrt{2}$

Der *Scale-Space* ist definiert als Funktion $L(x, y, \sigma)$, mit $I(x, y)$ als Input-Bild.

$$L(x, y, \sigma) = G(x, y, \sigma) \cdot I(x, y) \tag{39}$$

Die *Difference of Gaussians*-Funktion $D(x, y, \sigma)$ resultiert aus der Differenz zweier benachbarter Skalen, die durch einen konstanten multiplikativen Faktor k getrennt sind. [21]

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) \cdot I(x, y) = L(x, y, k\sigma) - L(x, y, \sigma) \tag{40}$$

Im Folgendem sei das Bild $I(x, y)$ Lenna (siehe [60]), auf dieses wird die *Difference of Gaussians*-Funktion angewendet und illustriert. In Abbildung 13a wurde mit einem (3×3) Kernel geglättet, in Abbildung 13b wurde mit einem (5×5) Kernel gearbeitet ($\sigma = \sqrt{2}$). Abbildung 13a hat folglich eine geringere Glättung als Abbildung 13b. In der Differenz $L(x, y, k\sigma) - L(x, y, \sigma)$ werden nun besonders Kanten als Minima im Bild sichtbar. Flächige Bereiche wie der Hintergrund werden in der DoG auf eher hohe Grauwerte abgebildet. Die weißen Anteile (geringer Grauwert) in der

DoG sind die Bereiche des Bildes, welche sich unter der Glättung wenig bis gar nicht verändert haben. Genau diese Bereiche sind somit gut geeignet für die lokale Merkmalsextraktion.

(a) $L(x, y, \sigma)$ (b) $L(x, y, k\sigma)$ (c) $D(x, y, \sigma)$

Abbildung 13: Lenna mit zwei Glättungen und DoG

Wie in Abbildung 14 dargestellt, wird mit unterschiedlichen Varianzen (hier wurden fünf Stück angedeutet) pro Oktave gearbeitet. Eine Oktave beschreibt die Ebene der Skalierung des Verfahrens, in der nächsten wird das Bild um den Faktor 2 verkleinert.

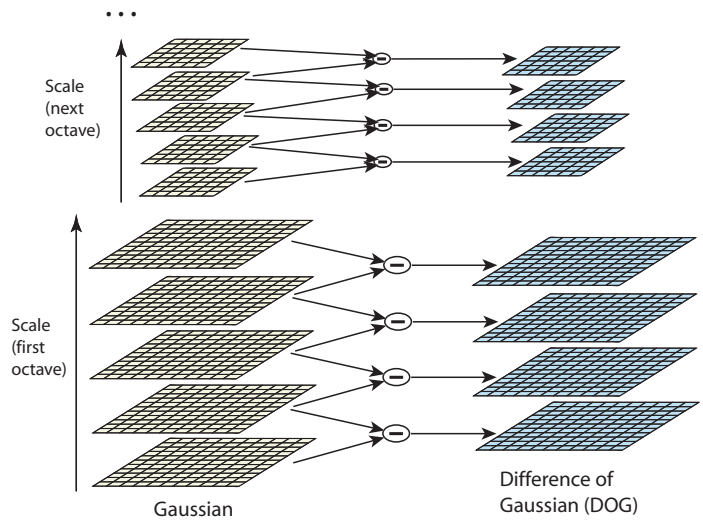


Abbildung 14: Oktaven, Quelle: [21]

Die DoG-Funktion liefert eine gute Approximation für die normalisierte *Laplacian of Gaussian* (LoG). Sei $\nabla^2 G = \Delta G = \sum_{i=1}^{n} \frac{\partial^2 G}{\partial x_i^2}$, dann sei

$$\frac{\partial G}{\partial \sigma} = \sigma \nabla^2 G \approx \frac{G(x, y, k\sigma) - G(x, y, \sigma)}{k\sigma - \sigma} \tag{41}$$

eine hinreichend gute Approximation. Der Fehler der Approximation geht für die Verringerung des Parameters k gegen null, $(\lim_{k \to 1} \sigma(k-1) = 0)$. In der Praxis hat sich jedoch gezeigt, dass die Güte der Approximation fast keinen Einfluss auf die Stabilität der Detektion der Extrema hat. Durch Multiplikation der Gleichung 41 mit $\sigma(k-1)$ folgt.

$$(k-1)\sigma^2 \nabla^2 G \approx G(x, y, k\sigma) - G(x, y, \sigma) \tag{42}$$

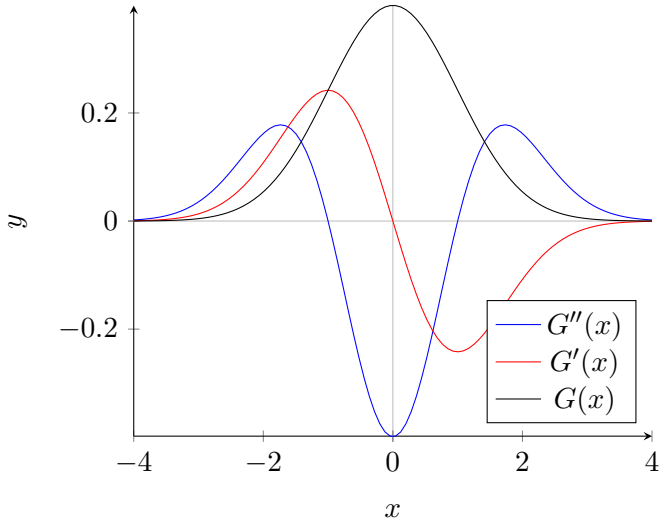


Abbildung 15: Eindimensionale Gaußfunktion mit $\sigma = 1$, $\mu = 0$ mit erster und zweiter Ableitung

Die Abbildung 15 illustriert den Verlauf der eindimensionalen Gaußfunktion und ihrer ersten und zweiten Ableitung. [1]

2.2.2. Detektion des lokalen Extrema

Um ein lokales Extremum in $D(x, y, \sigma)$ zu finden, wird jeder Pixel im aktuellen Bild, mit seinen acht Nachbarn, sowie mit den neun Nachbarn in der höheren und niedrigen Skalierung verglichen. Es handelt sich um ein lokales Maximum, wenn alle Nachbarn kleiner sind als $D(x, y, \sigma)$ (analog für ein lokales Minimum). Es stellte sich heraus, dass drei Skalierungen pro Oktave die besten Ergebnisse liefern, da die Anzahl an wiederholt detektierten *Keypoints* bei unterschiedlichen Bildtransformationen, bei dieser Anzahl an Skalierung, mit über 80% ein Maximum annahm. [21]

2.2.3. Akkurate *Keypoint* Lokalisierung

Wenn ein Kandidat für einen *Keypoint* gefunden wurde, wird die Umgebung dieses Pixels analysiert. Es werden die charakteristischen Eigenschaften Position (*Location*), Skalierung sowie das Verhältnis der Hauptkrümmungen festgestellt. Dieser Ansatz verhindert Punkte auszuwählen, welche geringen Kontrast haben oder entlang einer Kante lokalisiert wurden. Die erste Implementierung von 1999 setzte den Keypoint auf das Zentrum des Punktes, hinsichtlich Skalierung und Position. Der neue Ansatz besteht darin, eine dreidimensionale, quadratische Funktion zu nutzen, um die interpolierte Position des Maximums zu bestimmen. Sei $\mathbf{x} = \begin{bmatrix} x, y, \sigma \end{bmatrix}^\top$. Es wird die Taylorreihen-Entwicklung (siehe [6]) mit Entwicklungsstelle $\mathbf{a}$

$$T_n(\mathbf{x}) = \sum_{n=0}^{\infty} \frac{D^{(n)}(\mathbf{a})}{n!}(\mathbf{x} - \mathbf{a})^n = D(\mathbf{a}) + D'(\mathbf{a})(\mathbf{x} - \mathbf{a}) + \frac{f''(\mathbf{a})}{2}(\mathbf{x} - \mathbf{a})^2 + \dots \tag{43}$$

der Skalenraumfunktion $D(x, y, \sigma)$ bis zum quadratischen Term genutzt, mit $\frac{\partial D}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial D}{\partial x} \\ \frac{\partial D}{\partial y} \\ \frac{\partial D}{\partial \sigma} \end{bmatrix}$

$$D(\mathbf{x}) = D + \frac{\partial D^\top}{\partial \mathbf{x}}\mathbf{x} + \frac{1}{2}\mathbf{x}^\top \frac{\partial^2 D}{\partial \mathbf{x}^2}\mathbf{x} \tag{44}$$

so verschoben, dass der Ursprung am Abtastpunkt liegt. Der Vektor $\mathbf{x} = \begin{bmatrix} x, y, \sigma \end{bmatrix}^\top$entspricht demnach dem Offset bzw. der Verschiebung zum Abtastpunkt. Die Lokalität des Extremum $\hat{x}$

wird durch Ableiten der Funktion nach x

$$\hat{x} = \frac{\partial^2 D}{\partial \mathbf{x}^2}^{-1} \frac{\partial D}{\partial \mathbf{x}} \tag{45}$$

und Null-setzen bestimmt. Die Ableitung sowie die Hesse-Matrix von D werden durch Nachbarn des Abtastpunktes approximiert. Wenn der Abstand $\hat{x}$ größer als in einer der drei Dimensionen $0,5$ ist, dann bedeutet dies, dass das Extremum näher an einem anderen Abtastpunkt liegt. Das finale $\hat{x}$ wird mit dem Abtastpunkt gelinkt und eine Schätzung der Position des Extremums zu erhalten. [29] Durch Substitution von Gleichung 45 in 44 ergibt sich diese Gleichung,

$$D(\hat{x}) = D + \frac{1}{2} \frac{\partial D^\top}{\partial \mathbf{x}} \hat{x} \tag{46}$$

welche instabile Extrema mit geringem Kontrast erkennen kann. Im Artikel von [21] wurden Extrema mit $\mid D(\hat{x}) \mid < 0,03$ ausgeschlossen.

Beseitigung der Kanten-Reaktion Die DoG wird einen starken Ausschlag an Kanten haben, selbst wenn die Position entlang der Kante schlecht bestimmt ist und daher für Rauschen instabil ist.

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix} \tag{47}$$

Ein schlecht definierter *Peak* in der DoG-Funktion hat eine große Hauptkrümmung über die Kante, aber eine kleine in der senkrechten Richtung. Die Hauptkrümmung lässt sich mit Hilfe der 2×2 Hesse-Matrix bestimmen. Die Eigenwerte von $\mathbf{H}$ sind proportional zu Hauptkrümmung von D. Für die Analyse ist lediglich das Verhältnis derer entscheidend. Sei α der größte Eigenwert, β der kleinere. Es gelte

$$Spur(\mathbf{H}) = D_{xx} + D_{yy} = \alpha + \beta \tag{48}$$

$$Det(\mathbf{H}) = D_{xx} D_{yy} - D_{xy}^2 = \alpha\beta \tag{49}$$

Im unwahrscheinlichen Fall, dass die Determinante negativ ist, haben die Krümmungen unterschiedliche Vorzeichen, sodass der Punkt als kein Extremum verworfen wird. Es sei $r = \frac{\alpha}{b}$ und es gelte $r \geq 1$.

$$\frac{Spur(\mathbf{H})^2}{Det(\mathbf{H})} = \frac{(\alpha + \beta)^2}{\alpha\beta} = \frac{(r+1)^2}{r} \tag{50}$$

Damit lässt sich überprüfen, ob das Verhältnis der Hauptkrümmungen unter einem bestimmten Schwellwert r_x liegt.

$$\frac{Spur(\mathbf{H})^2}{Det(\mathbf{H})} < \frac{(r_x + 1)^2}{r_x} \tag{51}$$

Diese Methode ist sehr effizient hinsichtlich des Rechenaufwands, da die Eigenwerte nicht direkt berechnet werden.

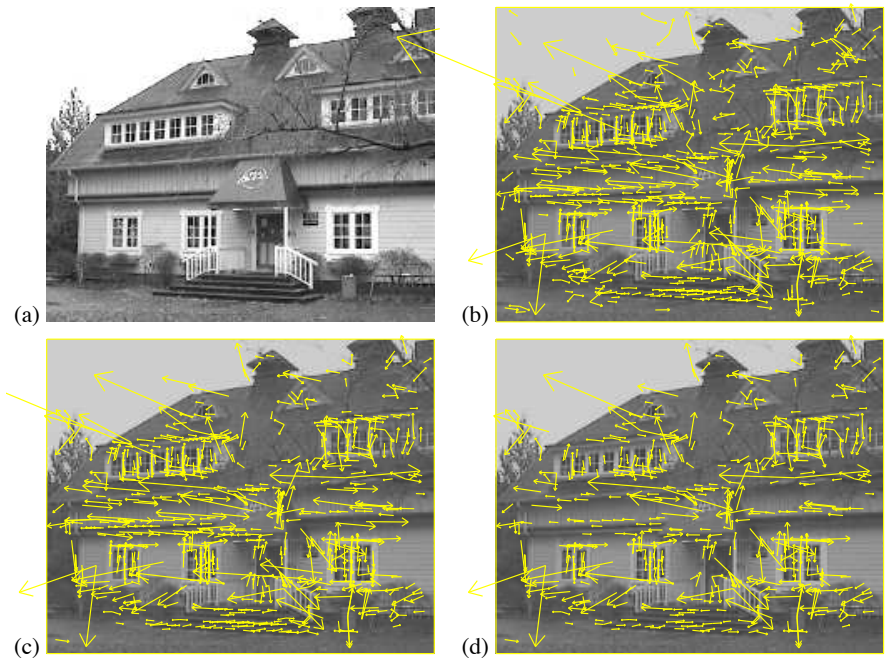


Abbildung 16: Phasen der Keypoint Auswahl: (a) Originalbild, (b) 832 Keypoints at Maxima und Minima der DoG, (c) Schwellwert für den Kontrast 729 Keypoints, (d) Schwellwert Verhältnis der Hauptkrümmungen 536 Keypoints, Quelle: [21]

In Abbildung 16 stellt der Ortsvektor der Pfeile die Position des Keypoints, die Länge den Betrag des Gradienten dar. Die Richtung des Vektors beschreibt den Winkel. [21]

2.2.4. Orientierungszuordnung

Durch Zuweisen einer konsistenten Orientierung zu jedem *Keypoint*, basierend auf lokalen Bildeigenschaften, kann der *Keypoint-Deskriptor* relativ zu dieser Orientierung dargestellt werden und somit eine Invarianz zur Bild-Drehung erzielen. Die Skalierung des Keypoints wird verwendet, um das Gauß'sche geglättete Bild L mit der nächstliegenden Skalierung auszuwählen, so dass alle Berechnungen skaleninvariant ausgeführt werden. Durch Differenzbildung der Pixel in L lässt sich die Stärke m des Gradienten und sein Winkel θ berechnen. [21]

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2} \tag{52}$$

$$\theta(i,j) = \tan^{-1}(\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)}) \tag{53}$$

Ein Orientierungs-Histogramm wird aus den Gradienten-Orientierungen von Abtastpunkten innerhalb einer Region um den *Keypoint* gebildet. *Peaks* im Histogramm deuten auf eine dominanten Winkel des lokalen Gradienten hin. [21]

2.2.5. Lokaler *Deskriptor*

Lokale Bild-*Deskriptoren* lassen sich als Vektoren beschreiben. Sie können in Gleitkommazahl- oder binärer Darstellung repräsentiert werden. Das Ziel der Vektor-Darstellung, des lokalen Bildbereichs, ist die Charakteristik beziehungsweise die Besonderheit dessen zu erfassen und gleichzeitig eine große Robustheit gegenüber Bild-Transformationen, Wechsel des Standpunkts, Skalierungsänderungen, Bild-Unschärfe, Bild-Rauschen etc. zu erreichen. Durch die Extraktion dieser

Charakteristiken kann dann eine Korrespondenz zu Bildern der gleichen Szene oder ähnlichen Bildern gefunden werden. Allerdings ist es nicht einfach, einen guten lokalen *Deskriptor* zu entwerfen, der exzellente beschreibende Eigenschaften sowie Robustheit vereint. Denn genau diese beiden Fähigkeiten des*Deskriptors* stehen sich kontradiktorisch gegenüber. [4] Jedem *Keypoint* kann nun also eine Position, Stärke sowie Orientierung zugewiesen werden. Der nächste Schritt besteht in der eigentlichen Berechnung des *Deskriptors*. Die Anforderungen sind die gute Unterscheidbarkeit sowie die Invarianz gegenüber Variationen wie z. B. Änderung der Beleuchtung oder des 3D-Blickpunkts. [21]

2.2.6. Darstellung des *Deskriptors*

Abbildung 17 visualisiert die Berechnung des *Keypoint-Deskriptors*. Ein *Deskriptor* wird gebaut, indem zuerst der Betrag des Gradienten und seine Orientierung an jedem Bildabtastpunkt in einer Region um den *Keypoint* berechnet wird. Diese werden durch ein Gauß-Kernel gewichtet, welcher durch den überlagerten Kreis in Abbildung 17 angedeutet wird. Diese *Samples* werden dann in Orientierungs-Histogrammen akkumuliert, die die Inhalte über 4×4-Unterregionen zusammenfassen, wie rechts gezeigt, wobei die Länge jedes Pfeils der Summe der Beträge des Gradienten nahe dieser Richtung innerhalb der Region entspricht. Die Abbildung 17 zeigt ein (2×2)-Orientierungshistogramm-*Array*. Allerdings wurde die besten Ergebnisse von Lowe mit einem 4×4-Array von Histogrammen mit jeweils 8 Orientierungs-*Bins* erzielt. Daher verwenden die Experimente in seinem Artikel einen $4 \times 4 \times 8 = 128$ dimensionalen Merkmalsvektor für jeden *Keypoint*. [21]

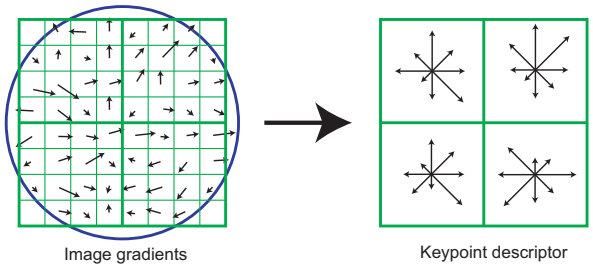


Abbildung 17: Bildgradienten und zugehöriger *Keypoint-Deskriptor*, Quelle: [21]

2.2.7. Anwendung für die Objekterkennung

Die Objekterkennung wird durchgeführt, indem zunächst jeder *Keypoint* unabhängig von der Datenbank, der aus Trainingsbildern extrahierten *Keypoints*, abgeglichen wird. Viele dieser anfänglichen Übereinstimmungen sind aufgrund von mehrdeutigen Merkmalen oder Merkmalen, die durch Background Clutter entstehen, falsch. Dann wird jeder *Cluster* durch Ausführen einer detaillierten geometrischen Anpassung an ein Modell überprüft und das Ergebnis wird verwendet, um die Interpretation zu akzeptieren oder abzulehnen. [21]

Keypoint Matching Der beste Kandidat für ein *Keypoint Match* aus einem beliebigem Bild A ist dadurch gegeben, den nächsten Nachbarn (*Nearest Neighbor*) in de Menge an Keypoints $\mathfrak{K}_B$ im anderen Bild B zu finden. Dabei sei erwähnt, dass für ein *Keypoint-Match* in Bezug zu DELF, die Bezeichnung *Inlier* $\mathfrak{i}$ genutzt wurde. Da im Artikel von Lowe diese Bezeichnung verwendet wurde, soll die Beschreibung dazu, ohne Beschränkung der Allgemeinheit gelten. Der *Nearest Neighbor* ist dadurch definiert, dass die euklidische Distanz $\mathrm{d}(\mathfrak{f}_A, \mathfrak{f}_B)$ ihrer *Deskriptoren* minimal ist. Viele *Features* aus $\mathfrak{f}_A$ werden gar kein korrektes Match in B finden, da diese aus Stördaten im Bild-Hintergrund (*Background Clutter*) resultieren oder gar nicht im Bild B lokal extrahiert wurden. Daher ist es sinnvoll, *Deskriptoren* nicht in Erwägung zu ziehen, die gar keine guten *Matches* sind. Ein globaler *Threshold* für die Distanz schneidet nicht gut ab, da bestimmte *Deskriptoren* markanter sind als andere. Daher bietet sich eine andere Methode an, wobei die Distanz d des Matches mit der geringsten Distanz mit der des zweitgeringsten bzw. zweitbesten Nachbar. Man könnte sich vorstellen, dass die zweitnächste Übereinstimmung (*Match*) eine

Schätzung der Wahrscheinlichkeitsdichte falscher Übereinstimmungen in diesem Teil des Merkmalsraums liefert und gleichzeitig bestimmte Fälle von Merkmalsmehrdeutigkeit identifiziert. In der Implementierung von Loewe, mit einem gewähltem Ratio für den Threshold von $0,8$, konnte 90 % der falschen Matches($\mathfrak{i}_f$) gefiltert werden und nur 5 % richtige Matches ($\mathfrak{i}_t$) wurden damit entfernt. Dieser Ansatz soll für die spätere Implementierung in dieser Masterarbeit verwendet werden und wird als *Ratio Test* bezeichnet. [21]

Bestimmung der affinen Parameter Eine affine Transformation berücksichtigt korrekt die 3D-Drehung einer planaren Ebene bei orthografischer Projektion, die Annäherung kann jedoch für die 3D-Drehung nichtplanarer Objekte schlecht sein. Eine allgemeinere Lösung wäre die Lösung der Fundamentalmatrix (Luong und Faugeras, 1996 sowie Hartley und Zisserman, 2000). Eine fundamentale Matrixlösung erfordert jedoch mindestens 7 Punkte, im Vergleich zu nur 3 Punkten für die affine Lösung und erfordert in der Praxis sogar noch mehr Übereinstimmungen für eine gute Stabilität. Lowe nutzte für die Erkennung nur drei Merkmalsübereinstimmungen, sodass die affine Lösung einen besseren Ausgangspunkt bietet und wir Fehler in der affinen Approximation berücksichtigen können, indem große Restfehler(*Residual Errors*) berücksichtigt werden. Dabei sei die affine Transformation eines Modell Punktes $p_m = \begin{bmatrix} x & y \end{bmatrix}^T$ zu einem Bildpunkt $p_b = \begin{bmatrix} u & v \end{bmatrix}^T$. Die affinen Transformation sei dabei gegeben durch

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}. \tag{54}$$

Für die Berechnung der sechs Transformations-Parameter werden mindestens drei Punkte $p_1 = \begin{bmatrix} u_1 & v_1 \end{bmatrix}^T$, $p_2 = \begin{bmatrix} u_2 & v_2 \end{bmatrix}^T$ und $p_3 = \begin{bmatrix} u_3 & v_3 \end{bmatrix}^T$ benötigt.

$$\begin{bmatrix} x_1 & y_1 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_1 & y_1 & 0 & 1 \\ x_2 & y_2 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_2 & y_2 & 0 & 1 \\ x_3 & y_3 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_3 & y_3 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t_x \\ t_y \end{bmatrix} = \begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \end{bmatrix} \tag{55}$$

Die Gleichung 55 lässt sich als Lineares System schreiben:

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{56}$$

Für den Fall, dass mehr als drei Punkte verwendet werden, handelt es sich nun um ein Lineares Ausgleichsproblem. Gesucht sei zu einer gegebenen Matrix $\mathbf{A} \in R^{m \times n}$ und $\mathbf{b} \in R^n$, die Lösung mit

$$\|\mathbf{A}\mathbf{x}^* - \mathbf{b}\|_2 = \min_{x \in R^n} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2. \tag{57}$$

Theorem 5 *$x^* \in R^n$ ist genau dann Lösung des Linearen Ausgleichsproblems (Gleichung 57), wenn x^* Lösung der Normalengleichungen*

$$\mathbf{A}^\top \mathbf{A}x = \mathbf{A}^\top \mathbf{b} \tag{58}$$

ist. Das System der Normalengleichungen hat stets mindestens eine Lösung. Sie ist genau dann eindeutig lösbar, wenn $Rang(\mathbf{A}) = n$ [2].

Die Lösung der entsprechenden Normalengleichungen, minimiert somit die Summe der Quadrate der Entfernungen von den projizierten Modellorten zu den entsprechenden Bildorten. [21]

$$\mathbf{x}^* = (\mathbf{A}^\top \mathbf{A})^{-1} \mathbf{A}^\top \mathbf{b} \tag{59}$$

2.3. Neuronale Netze

Die Entstehung der neuronalen Informationsverarbeitung ist kein Verdienst der klassischen Informatik oder ihrer Vorläufer. Entscheidende Beiträge leisteten vielmehr Biologen, Neurophysiologen und Kognitionswissenschaftler. Neuronale Netze bieten ein mächtiges Werkzeug zur Lösung von Problemen der Informatik, gegenüber dem Ansatz von klassischen Programmierstilen (modular, objektorientiert et cetera) bzw. dem Beschreiben jeder einzelnen Routine durch Logik (Quellcode). [19]

2.3.1. Biologische neuronale Netze

Schon der Begriff Künstliche Intelligenz nimmt Bezug auf Leistungen der biologischen Informationsverarbeitung, welche als Vorbild und Existenzbeweis für Lösungen dient. Neuronale Netze bilden ein Teilgebiet der Neuroinformatik, welche sich mit Erforschung biologischer Informationsverarbeitung mit den Methoden der Informatik und Informationstechnologie beschäftigt. Der Zweck der biologischen Informationsverarbeitung ist es, Verhalten sowie internen Regulationsprozessen in Abhängigkeit von den gegebenen Umweltsituationen, so zu organisieren, so dass das Lebewesen sich behauptet.[9] Die Abbildung 18 zeigt beispielhaft die Nervenzellen im Gehirn einer Katze.

Abbildung 18: Illustration von Nervenzellen aus der Kleinhirnrinde einer Katze, Quelle: [59]

2.3.2. Künstliche neuronale Netze

Elemente für die Modellierung neuronaler Netze sind die Aktivität (auch Erregung) mit der zugehörigen Erregungsdynamik, die Übertragungsgewichte mit der zugehörigen Gewichtsdynamik sowie die Netzwerktopologie.[9]

1. Erregung und Erregungsdynamik
   (i) Erregungszustand (auch Aktivität): Für ein gegebenes Neuron i beschreibt $\mathfrak{e}_i$ den Erregungszustand. Dies kann in einem diskreten Intervall oder auch kontinuierlich sein. Der Vektor $\mathfrak{e} = \left[\mathfrak{e}_1\ , \mathfrak{e}_2, \ldots, \mathfrak{e}_n\right]^T$ beschreibt den Erregungszustand des gesamten Netzes. Im zeit-kontinuierlichen Fall ist $\mathfrak{e}$ eine Funktion von t. Im zeit-und ortskontinuierlichen Fall gilt für die globale Erregung $\mathbf{e}(x,t)$.
   (ii) Aktivierungsfunktion (auch Übertragungsfunktion): Sei $\alpha_i(\mathbf{e})$ die Aktivierungsfunktion. Sie beschreibt den zukünftigen Erregungszustand des Neuron i lokal durch die Zustände der benachbarten Zellen $\mathfrak{e}_j$. Im kontinuierlichem Fall entsteht ein Aktivierungsfunktional.
   (iii) Globale Erregungsdynamik: Für bestimmte Analysen (Stabilität) kann die Netzwerkenergie definieren, welche die Erregungsdynamik global formuliert.
2. Gewichte und Gewichtsdynamik
   (i) Übertragungsgewichte: Die Verbindung zwischen zwei Neuronen $\mathfrak{e}_i, \mathfrak{e}_j$ wird als Gewicht w_{ij} bezeichnet und geht in die Aktivierungsfunktion mit ein. Da die Verbindungen eine Richtung haben, gilt allgemein $w_{ij} \neq wji$.

(ii) Lernregeln: Lokale Regeln für die Manipulation von Gewichten. Diese hängen nur vom gegenwärtigen Gewicht der Verbindung sowie des Erregungszustand der beteiligten Zelle ab (unbeaufsichtigtes Lernen).

(iii) Globale Gewichtsdynamik: Für bestimmte Anwendungen können globale Anforderungen an das Netzwerk formuliert werden, welche dann zur Einstellung der Gewichte genutzt werden (überwachtes Lernen).

3. Netzwerktopologie Neuronale Netze sind in der Regel nicht vollständig verbunden, sondern weisen komplexe Strukturen wie Schichten auf.

Im Folgendem wurde ein Modellneuron (siehe Abbildung 19) eines neuronalen Netzes visualisiert.

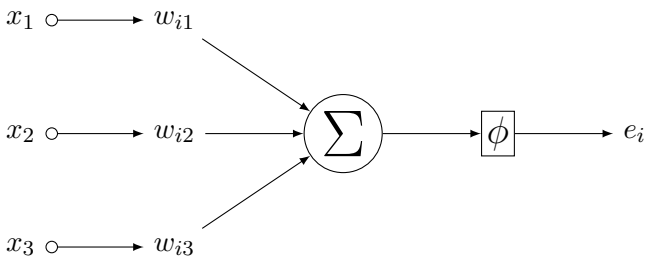


Abbildung 19: Modellneuron

Dabei seien x_j die Eingänge, w_{ij} die Übertragungsgewichte, ϕ ist eine Nichtlinearität und $\mathfrak{e}_i$ die resultierende Erregung. [9]

Aktivierungsfunktion Die Aktivierungsfunktion α ordnet einer Menge an Eingangssignalen x_j eine Erregung $\mathfrak{e}_i$ zu. Dazu gehen die Gewichte w_{ij} sowie eine nichtlineare Kennlinie ϕ ein. Selten ist es auch nützlich nichtlineare Interaktionen des *Inputs* zu formulieren. Im Allgemeinen sind x_j die Erregungen der übrigen Zellen des Netzes, im zeit-diskreten Fall zum vorherigen Zeittakt. McCulloch und Pitts führten 1943 das logistische Neuron, dabei sei u das Potential,

$$\mathfrak{e}_i = \phi\left(\sum_j w_{ij}x_j - \theta\right) \;\; mit \;\; \phi(u) = \begin{cases} 0, & \text{für } u < 0 \\ 1, & \text{für } u \geq 0 \end{cases} \tag{60}$$

ein, wobei $\mathfrak{e}_i$, w_{ij} und θ diskret sind ($\mathfrak{e}_i \in \{0,1\}$, $w_{ij} \in \{-1,1\}$ und $\theta \in \mathbb{N}$). Widrow und Hoff ließen 1960 auch kontinuierliche Gewichte zu (Adaptive Linear Neuron, ADALINE). Der konstante Schwellwert θ (auch *Bias*) wird wie ein konstanter Eingang mit Gewicht $w_{i0} = -\theta$ behandelt. Kontinuierliche Werte aller beteiligten Größen und stetig oder stetig differenzierbare Nichtlinearitäten beschreiben das sogenannte semilineare Neuron. Gängige Aktivierungsfunktionen und Kennlinien sind:

$$\mathfrak{e}_i = \phi\left(\sum_j w_{ij}x_j\right) \tag{61}$$

mit

$$\phi(u) = \frac{2}{\pi}\arctan(\lambda u); \quad \phi(u) = \frac{1}{1+e^{-\lambda u}}; \quad \phi(u) = \begin{cases} 0, & \text{für } u < 0 \\ x, & \text{für } 0 \geq x \leq 1 \\ 1, & \text{für } u > 1 \end{cases} \tag{62}$$

Grundtypen von neuronalen Netzen Nach der Dimension von Eingang und Ausgang lassen sich folgende Unterscheidungen treffen:

1. *Klassifikatoren* teilen die Menge der Eingangsvektoren in Klassen ein. Im einfachsten Fall detektiert ein einziges Ausgangs-Neuronen ein Muster einer bestimmten Klasse. $\mathbb{R}^J \rightarrow \{0,1\}$.

2. *Assoziatoren* ordnen bestimmten Inputvektoren bestimmte Outputvektoren zu. Dabei ist die Anzahl assoziierter Paare relativ klein (etwa der Dimension des Inputraums). Nicht gespeicherte Inputvektoren können mit Zwischenwerten oder mit einem gespeicherten Outputvektor beantwortet werden. Bei letzterem spricht man von einem klassifizierenden Assoziator:
$\{\mathbf{x}^1, \ldots, \mathbf{x}^P\} \subset \mathbb{R}^{\mathbb{J}} \rightarrow \{\mathfrak{e}^1, \ldots, \mathfrak{e}^P\} \subset \mathbb{R}^I$.

3. *Abbildende Netzwerke* erzeugen eine Zuordnung zwischen beliebigen Inputvektoren und zugehörigen Outputvektoren. Sie werden genutzt, um Funktionen bzw. Operatoren zu approximieren. $\mathbb{R}^J \rightarrow \mathbb{R}^I$. [9]

Klassifizierende Netzwerke: Das *Perzeptron* Semilineare Neuronen sind lineare Klassifikatoren. Ein einzelnes Modellneuron mit der Aktivierungsfunktion

$$\mathfrak{e} = \begin{cases} 1, & \text{für } \sum_{j=0}^{J} w_j x_j \geq \theta \\ 0, & \text{für } \sum_{j=0}^{J} w_j x_j < \theta \end{cases} \tag{63}$$

kann als linearer *Klassifikator* auf dem Raum der Inputvektoren $\mathbf{x} = \left[\mathbf{x}_1, \ldots, \mathbf{x}_J\right]^T$ aufgefasst werden. Die Abbildung 20 visualisiert dies für einen zweidimensionalen Input und $\theta = 2$.

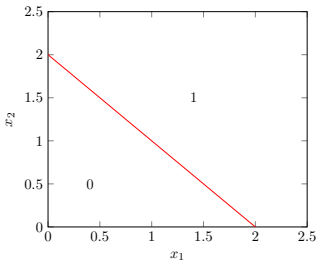


Abbildung 20: Trennende rote Gerade für zweidimensionalen Eingang, $\theta = 2$, $x_2 + x_1 = \theta = 2 \implies x_2 = -x1 + 2$

2.3.3. *Deep Feedforward*-Netzwerke

Deep Feedforward-Netzwerke, oft auch *Multilayer Perceptrons* (MLPs) genannt, sind die fundamentalen *Deep Learning*-Modelle. Das Ziel eines *Feedforward*-Netzwerks besteht darin, eine Funktion f^* anzunähern. Es definiert eine Abbildung $y = f(\mathbf{x}, \theta)$ und lernt den Wert der Parameter θ, der zu der besten Funktions-Approximation führt. Diese Modelle werden *Feedforward* genannt, weil die Information durch die Funktion fließt, die von x ausgewertet wird. Dabei werden Zwischen-Berechnungen erzeugt, die verwendet werden, um f zu definieren, und schließlich die Ausgabe y. *Feedforward Neural Networks* werden Netzwerke genannt, da sie typischerweise durch Zusammensetzen vieler verschiedener Funktionen dargestellt werden. Dem Modell ist ein gerichteter, azyklischer Graph zugeordnet, der beschreibt, wie die Funktionen zusammengesetzt sind. Diese Verkettung lässt sich mathematisch wie folgt beschreiben:

$$f(\mathbf{x}) = f^{(n)}(f^{(n-1)}(\cdots(f^{(1)}(\mathbf{x})))). \tag{64}$$

Diese Ketten-Struktur repräsentiert auch die *Layer* des Netzwerks. So ist die Funktion $f^{(n)}$ der erste *Layer* und $f^{(1)}$ der letzte. Die Gesamtlänge der Kette gibt die Tiefe (*Depth*) des Modells an. Während des Trainigs wird versucht, $f(\mathbf{x})$ so gut wie möglich an $f^*(\mathbf{x})$ anzupassen. Die Trainingsdaten liefern uns verrauschte, ungefähre Beispiele für $f^*(\mathbf{x})$, die an verschiedenen Trainingspunkten ausgewertet werden. Jedem Beispiel $\mathbf{x}$ wird ein *Label* zugewiesen, $y \approx f^*(\mathbf{x})$.

Das Verhalten der anderen Schichten (*Layer*) wird von den Trainingsdaten nicht direkt beschrieben. Der Lern-Algorithmus muss entscheiden, wie diese Schichten verwendet werden, um die gewünschte Ausgabe zu erzeugen. Die Trainingsdaten allein geben jedoch noch keinen Aufschluss darauf, was jeder einzelne *Layer* tun soll. Da die Trainingsdaten nicht die gewünschte Ausgabe für jede dieser Schichten zeigen, werden diese Schichten als *Hidden Layers* bezeichnet. Eine Möglichkeit, *Feedforward*-Netzwerke zu verstehen, besteht darin, mit linearen Modellen zu beginnen. Lineare Modelle, wie die logistische Regression und die lineare Regression, sind attraktiv, weil sie effizient und zuverlässig funktionieren. Sie haben jedoch einen offensichtlichen Nachteil, sie sind beschränkt auf lineare Funktionen und können damit nicht jedes Verhalten approximieren. Um das lineare Modell zu erweitern, wird das lineare Modell auf einen transformierten Input $\phi(\mathbf{x})$ angewendet, wobei ϕ eben eine nichtlineare Funktion ist. [8] Die Strategie des *Deep Learning* besteht darin ϕ zu lernen. Dabei ist das allgemeine Modell durch

$$y = f(\mathbf{x}, \theta, \mathbf{w}) = \phi(\mathbf{x}, \theta)^\top \cdot \mathbf{w} \tag{65}$$

gegeben. Dabei sei θ eine Menge an Parametern, die wir verwenden, um ϕ aus einer breiten Klasse von Funktionen zu lernen. $\mathbf{w}$ sind Parameter, die von $\phi(x)$ auf die gewünschte Ausgabe abbilden. Bei diesem Ansatz parametrisieren wir die Repräsentation als $\phi(\mathbf{x}, \theta)$ und verwenden den Optimierungs-Algorithmus, um das θ zu finden, welches einer guten Repräsentation entspricht. [8]

ReLU Die gleichgerichtete lineare Aktivierungsfunktion (*Rectified Linear Activation Function*), ist die Standard-Aktivierungsfunktion, die für die meisten neuronalen *Feedforward*-Netzwerke genutzt wird. Die Anwendung dieser Funktion auf die Ausgabe einer linearen Transformation führt zu einer nichtlinearen Transformation.

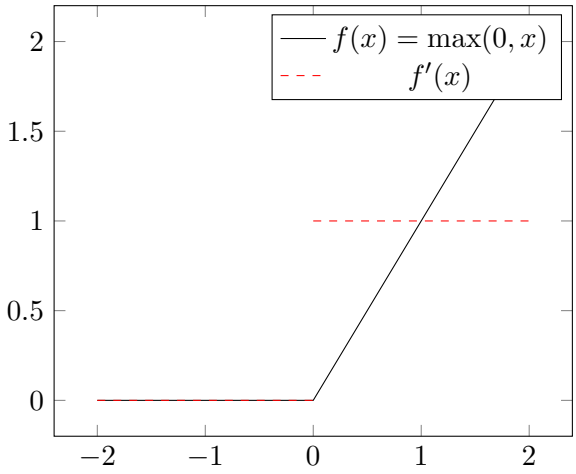


Abbildung 21: ReLU Funktion

Die Funktion bleibt jedoch nahe an einem linearem Verhalten, denn sie ist stückweise linear. Daher behält die ReLU (siehe Abbildung 21) viele der Eigenschaften bei, die lineare Modelle mit gradientenbasierten Methoden leicht optimierbar machen. [8]

2.3.4. Gradientenbasiertes Lernen

Der größte Unterschied zwischen den linearen Modellen und neuronalen Netzwerken ist, dass die Nichtlinearität eines neuronalen Netzwerks bewirkt, dass die meisten interessanten *Cost*-Funktionen nicht konvex sind. Dies bedeutet, dass neuronale Netze üblicherweise mit iterativen, gradientenbasierten Optimierern trainiert werden, die lediglich die Kostenfunktion auf einen sehr niedrigen Wert bringen, anstatt die linearen Gleichungslöser zum Trainieren von linearen Regressionsmodellen zu verwenden. [8]

Gradient Descent Die Methode erlaubt die Funktion f zu minimieren. Dabei geht das Verfahren in jedem Schritt in Richtung des negativen Gradienten, mit ϵ als Lernrate.

$$\mathbf{x}' = \mathbf{x} - \epsilon \cdot \nabla f(\mathbf{x}) \tag{66}$$

Dadurch gibt es allgemein für Neuronale Netze auch keine globale Konvergenz-Garantie. Stochastischer *Gradient Descent*, der auf nicht konvexe *Cost*-Funktionen angewendet wird, ist empfindlich gegenüber der Anfangsbedingung. Für neuronale *Feedforward*-Netzwerke ist es wichtig, alle Gewichte auf kleine zufällige Werte zu initialisieren. Der *Bias* kann auf Null oder ebenfalls auf kleine positive Werte gesetzt werden. [8]

2.3.5. *Loss* Funktion

Die *Loss*-Funktion auch *Cost*-Funktion, bezeichnet mit L, die minimiert oder maximiert werden soll, wird als objektive Funktion oder Kriterium bezeichnet. Sie wird auch als Kostenfunktion, Verlustfunktion oder Fehlerfunktion bezeichnet. [8]

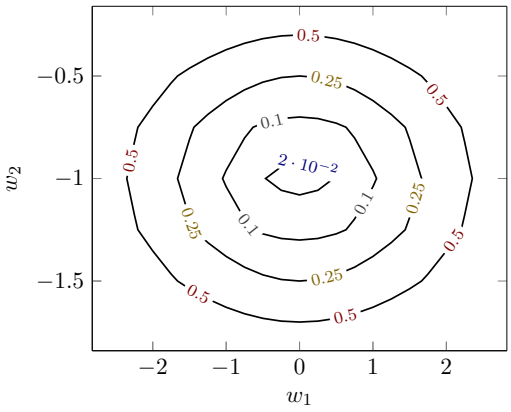


Abbildung 22: Höhenlinien einer beliebigen Loss-Funktion mit zwei Gewichten w_1 und w_2

2.3.6. *Softmax* Funktion

Die *Softmax*-Funktion erhält einen N-dimensionalen Vektor aus reellen Zahlen als *Input* und wandelt diesen in einen Vektor aus reellen Zahlen im Bereich $[0, 1]$, welche in Summe 1 ergeben.

$$softmax(\mathbf{a}) = p_i = \frac{e^{a_i}}{\sum_{k=1}^{N} e_k^a} \tag{67}$$

Wie der Name vermuten lässt, handelt es sich bei der *Softmax*-Funktion um eine weiche Version der *Max*-Funktion. Anstatt einen Maximalwert auszuwählen, wird ein anderer Ansatz verwendet wie in Gleichung 67 beschrieben, wobei maximales Element den größten Teil der Verteilung erhält, andere kleinere Elemente jedoch auch etwas davon. Die Eigenschaft der Softmax-Funktion, eine Wahrscheinlichkeitsverteilung auszugeben, macht sie zur probabilistischen Interpretation von *Klassifizierungs*-Aufgaben geeignet. [54]

Ableitung Aufgrund der wünschenswerten Eigenschaft der *Softmax*-Funktion, eine Wahrscheinlichkeitsverteilung zu modellieren, wird sie oft als letzte Schicht in neuronalen Netzwerken verwendet. Dazu muss die Ableitung bzw. der *Gradient* berechnet und während der *Backpropagation* an die vorherige Ebene übergeben werden. [54]

$$\frac{\partial p_i}{\partial a_j} = \frac{\partial \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}}}{\partial a_j} \tag{68}$$

Die Quotienten-Regel der Analysis wird eingesetzt: Sei $f(x) = \frac{g(x)}{h(x)}$, dann folgt die Ableitung durch $f'(x) = \frac{g'(x)h(x) - h'(x)g(x)}{h(x)^2}$. Für die *Softmax*-Funktion ist $g(x) = e^{a_i}$ und $h(x) = \sum_{k=1}^{N} e^{a_k}$. Dabei muss nun eine Fallunterscheidung gemacht werden:

$$\frac{\partial p_i}{\partial a_j} = \begin{cases} p_i(1 - p_j), & \text{wenn } i = j \\ -p_j p_i, & \text{wenn } i \neq j \end{cases}. \tag{69}$$

Cross Entropy Loss Die Kreuzentropie (*Cross Entropy*) gibt an, wie weit das Modell von der Ausgabeverteilung entfernt ist und wie die ursprüngliche Verteilung wirklich ist. Sie ist definiert als $H(y,p) = -\sum_i y_i \log(p_i)$ und ist eine Alternative zum *Squared Error*. Sie wird verwendet, wenn unter Knoten-Aktivierungen die Wahrscheinlichkeit verstanden werden kann, dass jede Hypothese wahr sein könnte, daher wird der *Cross Entropy Loss* in neuronalen Netzwerken verwendet, die *Softmax*-Aktivierungen im *Output-Layer* haben. [54]

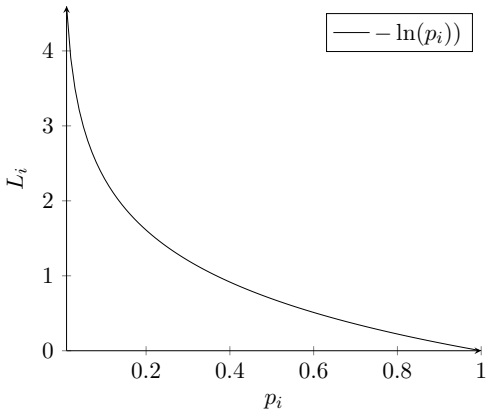


Abbildung 23: *Cross Entropy Loss*

Die Abbildung 23 zeigt, dass bei einer hohen Wahrscheinlichkeit für die Klasse i, der zugehörige *Loss* gering ist. Für Wahrscheinlichkeiten die gegen Null gehen, geht der *Loss* gegen unendlich.

2.3.7. *Backpropagation*

Die *Backpropagation* ist eine Methode, die in künstlichen neuronalen Netzen verwendet wird, um einen Gradienten zu berechnen, der für die Veränderung der Gewichte w benötigt wird. Dazu ist es nötig Ableitungen der Funktionen im neuronalen Netz zu bilden. Die Kettenregel der Analysis ist für die Berechnung des Gradienten essentiell. Sei $y = g(x)$ eine Funktion und $z = f(g(x)) = f(y)$ eine Verkettung. Es gelte folgende Beziehungen:

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx} \tag{70}$$

Von der Ableitungsregel kennt man $\frac{dz}{dy}$ als äußere Ableitung und $\frac{dy}{dx}$ als innere. Diese Methode lässt sich auf vektorwertige Funktionen übertragen. Sei $x \in R^m$, $y \in \mathbb{R}^n$ und $g : R^m \to \mathbb{R}^n$. Die Funktion f bildet in ein Skalar ab, mit $f : \mathbb{R}^n \to R$. Sei $y = g(x)$ und $z = f(y)$. Damit lässt sich

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i} \tag{71}$$

festhalten. [8]

'Convolutional Networks are an attempt to solve the dilemma between small networks that cannot learn the training set, and large networks that seem overparameterized...'

Yann LeCun, Corinna Cortes, Christopher J.C. Burges

2.3.8. *Convolutional Neural Network*

Convolutional Networks [20], auch bekannt als *Convolutional Neural Networks* oder CNNs, sind eine spezialisierte Art neuronaler Netze zur Verarbeitung von Daten, die eine bekannte gitterartige Topologie aufweisen. *Convolutional Neural Networks* sind in der praktischen Anwendungen enorm erfolgreich. Der Begriff *Convolutional* deutet an, dass das Netzwerk eine mathematische Operation verwendet, die als Faltung bezeichnet wird. Die Faltung ist eine spezielle Art der linearen Operation. CNNs sind neuronale Netze, die Faltung anstelle von allgemeiner Matrixmultiplikation in mindestens einer ihrer Schichten verwenden. [8]

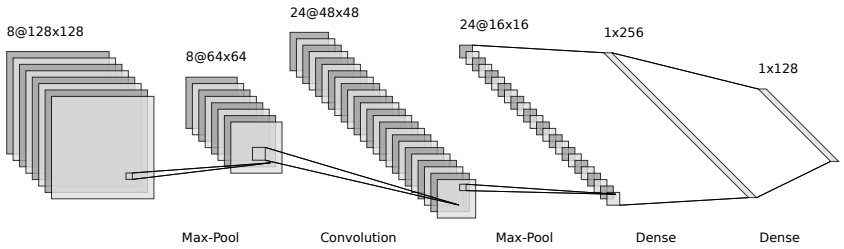


Abbildung 24: beispielhafte LeNet Architektur, Quelle: [50]

Die obige Abbildung 24 illustriert beispielhaft eine typisches CNN mit LeNet Architektur.

Faltungs-Operation In seiner allgemeinsten Form ist Faltung eine Operation auf zwei Funktionen eines reellwertigen Arguments. Sei w eine Gewichts-Funktion, a beschreibt die Verschiebung und t ist eine Variable der Zeit.

$$s(t) = \int x(a)\, w(t-a)\, da \tag{72}$$

Typischerweise wird diese Operation mit einem Stern (*) beschrieben.

$$s(t) = (x * w)(t) \tag{73}$$

In der Terminologie der CNNs wird das erste Argument (in diesem Beispiel die Funktion x) für die Faltung oft als die Eingabe und die zweite als *Kernel* bezeichnet. Die Ausgabe wird als *Feature Map* beschrieben. Wenn wir nun annehmen, dass x und w nur für die ganzzahlige t definiert sind, können wir die diskrete Faltung definieren:

$$s(t) = s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)\, w(t-a). \tag{74}$$

In in der Praxis wird die unendliche Summation natürlich als eine Summation über eine endliche Anzahl implementiert. Im maschinellen Lernen ist die Eingabe normalerweise ein mehrdimensionales Datenfeld und der *Kernel* ist ein mehrdimensionales *Array* an Parametern, die durch das

Lernen angepasst werden. Im Bereich der *Computer Vision*, wird ein zweidimensionales Bild I als Eingabe verwendet. Daraus folgt dann ein zweidimensionalen *Kernel* K.

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i-m, j-n)\, K(m,n) \tag{75}$$

Viele *Machine Learning*-Bibliotheken nutzen eine verwandte Funktion, die als Kreuzkorrelation bezeichnet wird, die das gleiche tut wie die Faltung, aber ohne den Kernel zu spiegeln. [8]

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)\, K(m,n) \tag{76}$$

Vorteile der Faltung Es gibt Vorteile von CNNs gegenüber klassischen Neuronalen Netzen. Der geringe Aufwand einer Faltung ist sicherlich einer der am schnellsten Ersichtlichen. Denn während ein linearer Klassifizierer in einem Bild jeden Pixel als *Input* für den nächsten *Layer* verwendet, wird in der Faltung nur eine gewisse Teilmenge der Pixel verwendet. Zudem werden die Gewichte der Faltungs-Matrix für mehrere Bereiche des Bildes verwendet, der Begriff ist im englischen als *Paramter Sharing* bekannt. Dadurch ist auch die Beschreibung bzw. Repräsentation des Netzes einfacher. Außerdem bietet die Faltung eine Möglichkeit, mit *Input* variabler Größe zu arbeiten.

Pooling Eine typische Schicht eines CNNs besteht aus drei Stufen. In der ersten Stufe führt die Schicht mehrere Faltungen parallel durch, um eine Reihe linearer Aktivierungen zu erzeugen. In der zweiten Stufe wird jede lineare Aktivierung durch eine nichtlineare Aktivierungsfunktion wie die gleichgerichtete lineare Aktivierungsfunktion ausgeführt. Diese Stufe wird manchmal als *Detektor*-Stufe bezeichnet. In der dritten Stufe wird eine *Pooling*-Funktion verwendet, um die Ausgabe der Ebene weiter zu modifizieren. Eine *Pooling*-Funktion ersetzt die Ausgabe des Netzes an einer bestimmten Stelle durch eine statistische Zusammenfassung des benachbarten *Outputs*. Eine vielfach verwendete spezifische Methode wird im folgenden erläutert. [8]

Max Pooling Diese Operation teilt das Eingabe-Bild in sich nicht überlappender Rechtecke (siehe Abbildung 24) auf und gibt für jeden dieser Teilbereiche das Maximum aus.

Sei beispielhaft $\mathbf{I_{4,4}} = \begin{bmatrix} I_{11} & I_{12} & I_{13} & I_{14} \\ I_{21} & I_{22} & I_{23} & I_{24} \\ I_{31} & I_{32} & I_{33} & I_{34} \\ I_{41} & I_{42} & I_{43} & I_{44} \end{bmatrix}$ und $mp\colon R^{4\times4} \to R^{2\times2}$ die Abbildungsvorschrift.

$$mp(\mathbf{I}) = \begin{bmatrix} \max(I_{11}, I_{12}, I_{21}, I_{22}) & \max(I_{13}, I_{14}, I_{23}, I_{24}) \\ \max(I_{31}, I_{32}, I_{41}, I_{42}) & \max(I_{33}, I_{34}, I_{43}, I_{44}) \end{bmatrix} = \begin{bmatrix} MP_{11} & MP_{12} \\ MP_{21} & MP_{22} \end{bmatrix} = \mathbf{MP_{2,2}} \tag{77}$$

Die Intuition ist, dass die genaue Position eines *Features* weniger wichtig ist, als seine grobe Position im Vergleich zu anderen *Features*. Die *Pooling*-Schicht dient dazu, die räumliche Größe der Darstellung zu verringern, die Anzahl der Parameter und den Umfang der Berechnung im Netzwerk zu reduzieren und somit auch die Überanpassung (*Overfitting*) zu steuern. [42] [44][8]

***Layer* und ihre Dimension** Sei W_1 die Breite, H_1 die Höhe und D_1 die Tiefe des *Input*-Bildes. Des weiteren sie K die Anzahl an Filtern, F die räumliche Größe, S gibt die Schrittweite an, b ist der *Bias* und P ist das *Padding* am Rand des Bildes. Durch die Faltungs-Operation ergeben sich folgenden Dimensionen.

$$W_2 = \frac{W_1 - F + 2P}{S+1} \tag{78}$$

$$H_2 = \frac{H_1 - F + 2P}{S+1} \tag{79}$$

$$D_2 = K \tag{80}$$

Dadurch ergibt sich eine Anzahl an Parametern $N = F \cdot F \cdot D1$ pro Filter und eine insgesamt Anzahl an Parametern für das CNN mit $N \cdot K + K \cdot b$. [44]

2.3.9. Residual Neural Network

Tiefe *Convolutional Neural Networks* haben einen entscheidenden Anteil beim Fortschritt des Maschinellen Lernens. Tiefe Netze enthalten dabei unterschiedliche Grade an *Features* und *Klassifizierern*. Nun kann der Grad der Feature durch eine höhere Anzahl an *Layern* angereichert werden und damit nimmt die Tiefe des Netzes zu. Die Netzwerk-Tiefe hat entscheidenden Einfluss auf die *Performance* des Netzes. Besonders leistungsstark zeigen sich sehr tiefe Netze [35]. Als tief gilt in diesem Zusammenhang eine Tiefe von 16 bis 30. Nun könnte der Gedanke entstehen, dass die Erhöhung der Tiefe (*Depth*) immer bessere Verfahren produzieren würde. Es gibt jedoch ein Verhalten von Netzen dieser Tiefe, welches dem negativ entgegen steht. Mit der Zunahme der Tiefe wird das numerische Verhalten des Gradienten schlecht. Es kann zu einer Explosion des Gradienten oder einem numerisches Verschwinden führen und damit die Konvergenz negativ beeinflussen. Diese Verhalten ist ein großes Problem beim *Training* des Netzes. Diese Problematik wurde jedoch weitgehend durch eine normalisierte Initialisierung und Normalisierungs-Schichten gelöst, was es ermöglicht Netze mit dutzenden Schichten zu trainieren. Dabei wird die Konvergenz mit dem Verfahren des stochastischen Gradienten (SGD) für die *Backpropagtion* ermöglicht.

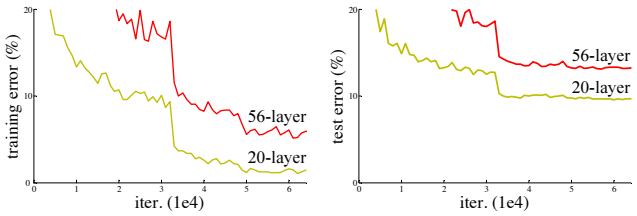


Abbildung 25: Fehler beim *Training* links, Fehler Test rechts, 20-schichtige und 56-schichtige Netzwerke, das tiefere Netzwerk hat einen höheren *Trainings*-Fehler und damit auch einen *Test*-Fehler, Quelle: [11]

Nachdem die Konvergenz gegeben ist, taucht nun ein *Degradations*-Problem auf. Mit zunehmender Netzwerk-Tiefe wird die Genauigkeit gesättigt und nimmt dann stark ab. Allerdings wird eine solche Verschlechterung nicht durch Überanpassung verursacht und das Hinzufügen weiterer Schichten zu einem entsprechend tiefen Modell führt zu höheren *Trainings*-Fehlern, wie in Abbildung 25 visualisiert ist. Die Verschlechterung (der *Trainings*-Genauigkeit) zeigt , dass nicht alle Systeme ähnlich leicht zu optimieren sind. Betrachten wir eine flachere Architektur und ihre tiefere Version. Für die Konstruktion des tieferen Modells gibt es nun doch eine Lösung: Die hinzugefügten Ebenen sind Identitäts-Abbildungen und die anderen Ebenen werden aus dem erlernten flacheren Modell kopiert. Das Vorhandensein dieser konstruierten Lösung zeigt an, dass ein tieferes Modell keinen höheren *Trainings*-Fehler erzeugen sollte als sein flacheres Gegenstück. Allerdings zeigen Experimente, dass die heutigen *Solver* keine guten Lösungen finden, zumindest nicht in realistisch begrenzter Zeit. Die Verschlechterung der *Trainigs*-Genauigkeit wird nun durch tiefe *Residul Networks* gelöst. Anstatt zu hoffen, dass nur wenige gestapelte *Layer* direkt die gewünschtes darunter liegendes Abbildung $H(\mathbf{x})$ approximiert, wird dieser *Layer* explizit ein Rest-Abbildung vornehmen. Aus formalen Gründen, werden die gestapelten nichtlinearen Schichten in eine andere Abbildung als

$$F(\mathbf{x}) = H(\mathbf{x}) - \mathbf{x} \tag{81}$$

zusammengefasst. Dabei sei $\mathbf{x}$ die Identität des Eingangs $\mathbf{x}$. Die ursprüngliche Abbildung wird zu $F(\mathbf{x})+\mathbf{x}$ umgeformt. Es zeigt sich, dass es einfacher ist die Residual-Zuordnung zu optimieren als die ursprüngliche, nicht referenzierte Abbildung. Im Extremfall wäre es bei einer optimalen Identitäts-Zuordnung einfacher, den *Residuum* auf Null zu setzen, als eine Identitäts-Zuordnung durch einen Stapel nichtlinearer Schichten anzupassen.

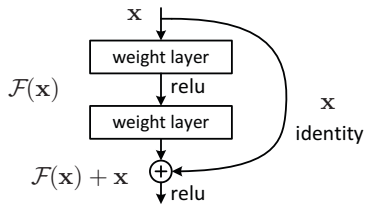


Abbildung 26: Baustein des *Residual Learnings*, Quelle: [11]

Die Formulierung von $F(\mathbf{x}) + \mathbf{x}$ kann durch *Feedforward*-Netzwerke mit sogenannten *Shortcut*-Verbindungen realisiert werden, siehe Abbildung 26. *Shortcut*-Verbindungen sind solche die, mindestens einen *Layer* überspringen. Für den Baustein des *Residuals* führen die Verknüpfungs-Verbindungen einfach eine Identitäts-Abbildung durch, und ihre Ausgaben werden zu den Ausgaben der gestapelten Schichten hinzugefügt. Positiver Weise erhöhen die Identitäts-Verknüpfungen weder Anzahl zusätzliche Parameter noch die numerische Komplexität. Zudem kann das gesamte Netzwerk durch SGD mit *Backpropagation* trainiert werden, ohne die *Solver* zu modifizieren. [11] Für die *DELF*-Architektur ist das *50-Layer ResNet* (siehe [11] `ResNet50`) besonders relevant, da diese Architektur in *Deep Local Features* implementiert wurde.

2.3.10. Informations-Theorie

Die Informationstheorie ist ein Zweig der angewandten Mathematik, bei dem es darum geht, zu bestimmen, wie viele Informationen in einem Signal vorhanden sind. Es wurde ursprünglich entwickelt, um das Senden von Nachrichten von einzelnen Alphabeten über einen verrauschten Kanal zu untersuchen, beispielsweise die Kommunikation per Funk. [8] Sei die Selbst-Information eines Ereignisses $x = x$, geschrieben als

$$I(x) = -\log\, P(x). \tag{82}$$

Selbst-Information behandelt nur ein einzelnes Ergebnis. Wir können die Unsicherheit in einer gesamten Wahrscheinlichkeitsverteilung mithilfe der Shannon-Entropie quantifizieren:

$$H(x) = \mathbb{E}_{x\sim P}[I(x)] = -\mathbb{E}_{x\sim P}[\log\, P(x)]. \tag{83}$$

Wenn wir zwei getrennte Wahrscheinlichkeitsverteilungen $P(x)$ und $Q(x)$ über dieselbe Zufallsvariable x haben, können wir anhand der Kullback-Leibler Divergenz messen, wie unterschiedlich diese beiden Verteilungen sind:

$$D_{KL}(P||Q) = \mathbb{E}_{x\sim P}[\log\, P(x) - \log\, Q(x)]. \tag{84}$$

Eine mit der KL-Divergenz eng verwandte Größe ist die Kreuzentropie $H(P,Q) = H(P) + D_{KL}(P||Q)$, die der KL-Divergenz ähnelt, jedoch fehlt der linke Term: [8]

$$H(P,Q) = \mathbb{E}_{x\sim P}[\log\, Q(x)]. \tag{85}$$

2.3.11. Fluch der Dimensionalität

Konfigurationen Viele maschinelle Lern-Probleme werden äußerst schwierig, wenn die Anzahl der Dimensionen in den Daten hoch ist. Dieses Phänomen ist als Fluch der Dimensionalität

bekannt. Die Problematik besteht darin, dass die Anzahl an möglichen Konfigurationen einer Menge an Variablen v, exponentiell mit der Anzahl der Variablen wächst. Um $\mathfrak{d}$-Dimensionen und v-Werte zu unterscheiden, werden

$$\mathcal{O}(v^{\mathfrak{d}}) \tag{86}$$

benötigt. [8]

Distanzen Ein weiteres Problem besteht bei der Bestimmung von Distanzen in hochdimensionalen Räumen. Im Allgemeinen sind Ähnlichkeits-Maße, die auf Entfernungen basieren, empfindlich gegenüber Schwankungen innerhalb einer Daten-Verteilung oder der Dimensionalität eines Daten-Raums. Diese Abweichungen können die Qualität der Lösung, die Effizienz der Suche oder beides einschränken. Für Normen in hohen Dimensionen wurden von mehreren Forschern die Frage aufgeworfen, ob der Ansatz des nächsten Nachbarn sinnvoll ist. Die euklidische Distanz der Punkte A und B in Abbildung 27 im Dreidimensionalem ergibt sich als $d(A,B) = \sqrt{(A_{x1} - B_{x1}) + (A_{x2} - B_{x2}) + (A_{x3} - B_{x3})}$. Das Verhältnis der Varianz der Länge eines beliebigen Punktvektors $\|\mathbf{X}_d\|$ zur Länge des Mittelpunktvektors (mit $E(\|\mathbf{X}_d\|)$ bezeichnet) mit zunehmender Daten-Dimensionalität zu Null konvergiert, dann verschwindet die proportionale Differenz zwischen dem Abstand am weitesten entfernten Punkt D_{max} und dem Abstand D_{min} am nächsten Punkt.

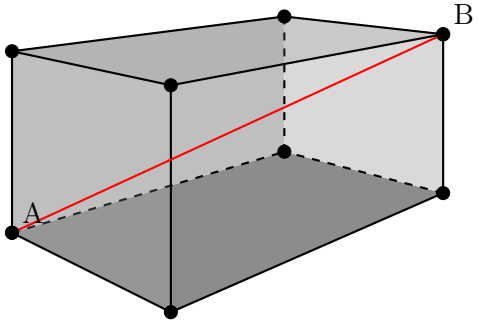


Abbildung 27: Distanzen in drei Dimensionen

$$Wenn \quad \lim_{d\to\infty} V(\frac{\|\mathbf{X}_d\|}{E(\|\mathbf{X}_d\|)}) = 0, \quad dann \quad \frac{D_{max} - D_{min}}{D_{min}} \to 0 \tag{87}$$

Bei einer breiten Palette von Daten-Verteilungen und Metriken nimmt der relative Kontrast mit zunehmender Dimensionalität ab. Dieser Konzentrations-Effekt des Entfernungs-Maßes verringert den Nutzen des Maßes zur Diskriminierung(*Discrimination*). Dieses Phänomen, der Fluches der Dimensionalität, ist recht allgemein und tritt für ein breites Spektrum von Datenverteilungen und Abstandsmaßen auf. [13]

3. *Deep Local Features*

Deep Local Features (*DELF*) ist eine Architektur, die für eine groß angelegte Bild-Suche geeignet ist. Das neue *Feature* basiert auf *Convolutional Neural Networks* (CNNs), die nur mit Bild-*Level-Annotationen* auf einem Landmarken-Bild-Datensatz trainiert wurden. Um semantisch nützliche lokale Funktionen für die Bilder-Suche zu identifizieren, wird ein Aufmerksamkeits-Mechanismus (*Attention*) für die Auswahl des *Keypoints* eingeführt, der die meisten Netzwerk-Ebenen mit dem *Deskriptor* teilt. Dieses *Framework* kann für den Abruf von Bildern als ein *Drop In*-Ersatz für andere *Keypoints* und *Deskriptoren* verwendet werden, wodurch eine genauere Merkmalsanpassung und geometrische Verifizierung ermöglicht wird. Um den vorgeschlagenen *Deskriptor* zu evaluieren, wurde ein neuer großer Datensatz eingesetzt, der als *Google-Landmarks*-Datensatz bezeichnet wird und Herausforderungen wie Hintergrund-*Clutter*, partielle Verdeckung, mehrere Landmarken, Objekte in variablen Maßstäben usw. beinhaltet. *DELF* kann globalen und lokalen *Deskriptoren* auf dem neuesten Stand der Technik im großen Rahmen um signifikante Margen übertreffen. [28] Beispielhaft wurden *DELF-Keypoints* in Abbildung 28 für die Weltzeituhr visualisiert.

Abbildung 28: extrahierte *DELF-Keypoints* für die Weltzeituhr am Berliner Alexanderplatz, Originalbild-Quelle: [57]

3.0.1. Einführung

Image Retrieval ist eine der fundamentalen Aufgaben im Bereich der Bildverarbeitung und hat viele direkte praktische Anwendungen wie die Produktsuche oder die Bestimmung des Ortes (Lokalisierung) mit Hilfe von Bilddaten. In den vergangenen Jahrzehnten hat eine enorme Entwicklung dieser Wiedererkennungssysteme stattgefunden. Dabei hat in den letzten Jahren die Bedeutung von Methoden, basierend auf *Convolutional Neural Networks*, für das Erlernen globaler *Deskriptoren*, besonders zugenommen. Der Ansatz über globale Verfahren hat jedoch den Nachteil, dass er nicht robust gegenüber Stördaten, Verdeckung, Wechsel des Standpunkts sowie der Variation der Lichtstärke ist. Zudem fehlt die Fähigkeit ähnliche Teilbereiche des Bildes (*Patches*) zu finden (matchen). Daher wird ein neue Ansatz verfolgt - *CNN* basierte lokale *Deskriptoren*, welche *Matching* von *Patches* ermöglichen. Allerdings waren viele dieser Verfahren nicht optimiert um semantisch wichtige *Features* zu detektieren. Das Verfahren von *DELF*, welches mit schwacher *Supervision* trainiert (nur Zugehörigkeit des Bildes zu einer Klasse wird benötigt) und daher keine *Label* für Objekte oder *Patches* benötigt. *DELF* vereinigt damit die Extraktion von lokalen Merkmalen sowie Bild-Wiedererkennung (*Retrieval*). Im *DELF* Verfahren wird das *Attention*-Modell sehr stark mit dem *DELF Deskriptor* gekoppelt. Es wird die selbe CNN-Architektur verwendet und generiert *Feature*-Bewertungen (*Attention*) mit sehr wenig zusätzlichen Berechnungsaufwand. [28]

3.0.2. *Google-Landmarks-Dataset*

Im Vergleich zu existierenden Datensätzen wie dem *Oxford-Buildings-Dataset* [30] und dem *Paris-Dataset* [31] ist der *Google-Landmarks*-Datensatz (siehe Abbildung 29) entscheidend größer. Er enthält diverse Landmarken, zudem ist der Datensatz schwieriger in Anbetracht der verwendeten Szenen. Insgesamt enthält *Google-Landmarks* 1.060.709 Bilder von 12.894 Landmarken. Damit sind pro Lokalität circa 82 Bilder gegeben. Die Anzahl an *Query*-Bildern beträgt 111.036. Die Bilder wurden an unterschiedlichen Lokalitäten auf der ganzen Welt aufgenommen. Zu jedem Foto gibt es die GPS-Koordinaten. In den meisten Datensätzen sind Bilder zentrisch im Bezug auf die Landmarke, dies begünstigt die Nutzung von globalen *Deskriptoren*. Der *Google-Landmark-Dataset* enthält realistischere Bilder, mit Stördaten im Bildvordergrund sowie Hintergrund, teilweise Überdeckung der Landmarke et cetera. In der Evaluierung des Algorithmus wurden zudem *Query*-Bilder eingesetzt, welche gar keine Landmarken enthalten. Diese *Distractors* geben die Möglichkeit die Robustheit des Verfahrens gegenüber irrelevanten Bilder und Bildrauschen auszuwerten. Um eine Aussage über *Ground Truth* zu treffen, wurden visuelle Eigenschaften und die GPS-Informationen verwendet. [28]

Abbildung 29: Verteilung der Geo-Lokalisierung des *Google-Landmarks*-Datensatz, Bilder aus 4.872 Städten in 187 Ländern. Quelle: [28]

3.0.3. *Image Retrieval* mit $DELF$

Das $DELF$ Verfahren zum Finden von ähnlichen Bildern kann in vier Schritte aufgeteilt werden:

(i) Dichte Lokale Merkmalsextraktion mit großer Anzahl n an *Keypoints* im Bild

(ii) Auswahl der *Keypoints* mit Anzahl m ($m < n$)

(iii) Reduktion der Dimensionalität

(iv) Indizierung und *Retrieval* [28]

3.0.4. Dichte lokale Merkmalsextraktion

Es wird eine gewisse Menge an *Features* n von einem Bild extrahiert. Dazu wird ein *Fully Convolutional Network* (FCN) eingesetzt. Es besteht aus einem *Feature Extraction Layer* eines CNNs, trainiert mit *Classification Loss*. Für die Architektur wurde das *Residual Neural Network* `ResNet50` [11] verwendet. Für die Bildskalierung wurde eine *Bild-Pyramide* konstruiert, für jede Skalierung wird das FCN einzeln angewendet. Die resultierenden *Feature Maps* bilden damit ein Raster aus lokalen *Deskriptoren* innerhalb dieser Skalierung. Die benötigten Pixel-Koordinaten der *Keypoints* werden aus den Zentren des rezeptiven Feldes berechnet. Die Berechnung beruht auf der Berücksichtigung der Konfiguration der Faltungs- und *Pooling*-Schichten des FCN. Die Größe des rezeptiven Feldes bei Original-Skalierung beträgt (291×291). Durch Verwendung der oben erwähnten Bild-Pyramide ergeben sich so Merkmale, welche unterschiedlich große Bildregionen beschreiben. Das bereits trainierte `ResNet50`-Model [11] wurde als Ausgang genutzt. Danach wurde die Verfeinerung der Diskriminierung der lokalen *Deskriptoren* vorgenommen.

Für das *Training* wurden lediglich Bilder mit *Labeln* bezüglich der Position genutzt. Es wurde die Kreuzentropie-*Loss*-Funktion verwendet.

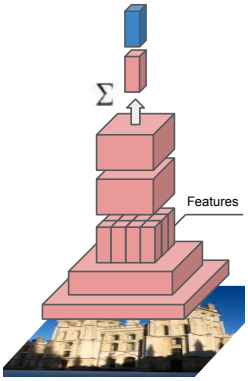


Abbildung 30: Training des *Deskriptors*, Quelle: [28]

In Abbildung 30 wird die Verfeinerung der Diskriminierung visualisiert, Abbildung 31 zeigt das *Training* der *Attention*. Die Eingabe-Bilder werden zunächst zentriert, um quadratische Bilder zu erzeugen und auf (250×250) skaliert. Zufällige (224×224) Bilder werden dann für das *Training* verwende. Durch diesen Ansatz lernt der *Deskriptor-Layer* Repräsentationen, die für die Erkennung von Landmarken entscheidend sind. Daher sind weder Objekt- noch *Patch-Level-Labels* erforderlich, um eine Verbesserung der lokalen *Deskriptoren* zu erhalten. [28]

3.0.5. *Attention*-basierte *Keypoint* Auswahl

Anstatt alle extrahierten *Features* direkt zum Abfragen bzw. *Matching* von Bildern zu verwenden, wird eine zusätzliche Technik eingeführt, um nur eine Teilmenge der *Features* auszuwählen. Da ein erheblicher Teil der dicht extrahierten Merkmale für die Erkennungs-Aufgabe irrelevant ist und Unordnung verursacht, was den Abruf-Vorgang negativ beeinflusst, ist die Auswahl der *Keypoints* sowohl für die Genauigkeit als auch für die Rechen-Effizienz von Abruf-Systemen wichtig. [28]

Lernen mit schwacher *Supervision* Es wird ein Landmarken-*Klassifizierer* trainiert, der Wichtigkeit bzw. Relevanz für lokale *Feature-Deskriptoren* explizit misst. Um die Funktion y zu trainieren, werden *Features* durch eine gewichtete Summe zusammengefasst, wobei die Gewichte vom *Attention* Netzwerk vorhergesagt werden. Die *Trainings*-Methode ähnelt der Merkmalsextraktion in Abschnitt 3.0.4, hinsichtlich der *Loss*-Funktion sowie des genutzten Datensatzes.

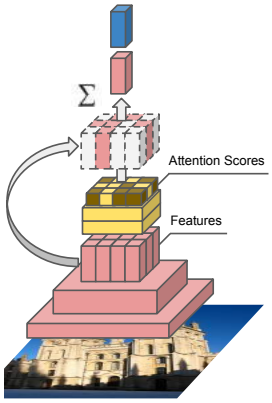


Abbildung 31: *Training* der *Attention*, Quelle: [28]

Abbildung 31 illustriert den Ablauf des *Trainings*. Dabei ist das *Attention*-Netzwerk, bestehend aus zwei Schichten, gelb hervorgehoben. Dadurch wird eine Einbettung für das gesamte Eingabe-Bild generiert, die dann zum *Trainieren* eines auf *Softmax* basierenden Landmarken-*Klassifizierer* verwendet wird. Im folgenden soll das *Training* $\mathbf{y} = \begin{bmatrix} y_1, \dots, y_m \end{bmatrix}^\top$ für m Klassen genauer beschrieben werden. Sei $\mathfrak{f}_i \in \mathbb{R}^d$ und beschreibe ein d-dimensionales Merkmal (*Feature*) $\mathfrak{f}_i = \begin{bmatrix} C_1, \dots, C_d \end{bmatrix}^\top$, welches erlernt werden soll. Die Anzahl der *Features* sei n, somit ist $\mathfrak{f} \in \mathbb{R}^n$, beschrieben durch $\mathfrak{f} = \begin{bmatrix} \mathfrak{f}_1, \dots, \mathfrak{f}_n \end{bmatrix}$ und damit n-dimensional. Sei weiterhin $\alpha(\mathfrak{f}_i, \theta)$ die *Score*-Funktion für die *Feature*, die Parameter der Funktion a werden mit θ bezeichnet. Sei $\mathbf{W} \in \mathbb{R}^{m \times d}$ und beschreibe die Gewichte des CNNs.

$$\mathbf{W}_{m,d} = \begin{bmatrix} w_{1,1} & \cdots & w_{1,d} \\ \vdots & \ddots & \vdots \\ w_{m,1} & \cdots & w_{m,d} \end{bmatrix} \tag{88}$$

Damit lässt sich das *Training* beschreiben als Multiplikation der Matrix W und dem Produkt aus *Score*-Funktion sowie des zugehörigem *Feature*. Es wird über die Anzahl an *Feature* (n) summiert.

$$\mathbf{y} = \mathbf{W} \cdot \sum_{i=1}^{n} \alpha(\mathfrak{f}_i, \theta) \cdot \mathfrak{f}_i \tag{89}$$

Eine Dimensionsanalyse zeigt, $\mathbf{y} = (\mathbf{W} \in \mathbb{R}^{M \times d}) \cdot (\mathfrak{f}_i \in \mathbb{R}^{d \times 1}) \implies \mathbf{y} \in \mathbb{R}^M$. Sei L die *Loss*-Funktion, $\mathbf{I}^\top = \begin{bmatrix} 1, \dots, 1 \end{bmatrix}$ der Eins-Vektor und der Spaltenvektor $\mathbf{y}^* = \begin{bmatrix} \mathbf{y_1^*}, \dots, \mathbf{y_m^*} \end{bmatrix}$ die tatsächliche Klasse des Bildes. Der Ansatz über die Kreuzentropie im diskreten Fall (siehe Gleichung 85) führt auf:

$$L = -\mathbf{y}^* \cdot \ln(\frac{\mathbf{e^y}}{\mathbf{I}^\top \cdot \mathbf{e^y}}) = -\begin{bmatrix} y_1^*, \dots, y_m^* \end{bmatrix} \cdot \ln(\frac{\begin{bmatrix} e^{y_1} \\ \vdots \\ e^{y_m} \end{bmatrix}}{\begin{bmatrix} 1, & \dots, 1 \end{bmatrix} \cdot \begin{bmatrix} e^{y_1} \\ \vdots \\ e^{y_m} \end{bmatrix}}) = -\mathbf{y}^* \cdot \ln(\frac{\mathbf{e^y}}{\sum_{i=1}^{m} e^{y_i}}). \tag{90}$$

Im Nenner steht die Summennorm des $\mathbf{e^y}$ Vektors. Das Argument des Logarithmus ist die *Softmax*-Funktion, siehe Gleichung 67. Damit lässt sich der *Loss* als

$$L = \mathbf{y}^* \cdot \ln(\frac{\mathbf{e^y}}{\|\mathbf{e^y}\|_1}) \overset{\text{Gleichung 67}}{=} -\mathbf{y}^* \cdot \ln(softmax(\mathbf{y})) = -\mathbf{y}^* \cdot \ln(p_i) \tag{91}$$

beschreiben. Die Parameter der *Score*-Funktion α werden durch *Backpropagation* trainiert. Der Gradient ergibt sich durch

$$\frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial y} \sum_{i=1}^{n} \frac{\partial y}{\partial \alpha_i} \frac{\partial \alpha_i}{\partial \theta} = \frac{\partial L}{\partial y} \sum_{i=1}^{n} W \cdot \mathfrak{f}_i \frac{\partial \alpha_i}{\partial \theta} \tag{92}$$

über partielle Ableitungen. [28]

Pipeline Die Kopplung aus dichter lokale Merkmalsextraktion und *Attention*-basierten *Keypoint* Auswahl ergibt dann eine Pipeline.

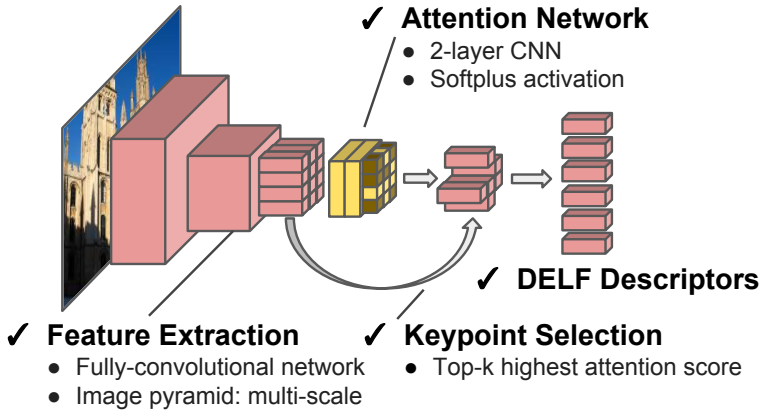


Abbildung 32: *DELF-Pipeline*, Quelle: [58]

Visualisiert wird dieser Ablauf beispielhaft in Abbildung 32.

Attention Training Im vorgeschlagenen Rahmen werden sowohl die *Deskriptoren* als auch das *Attention*-Modell implizit mit Beschriftungen (*Labeln*) auf Bild-Ebene erlernt. Leider stellt dies den Lernprozess vor einige Herausforderungen. Während die *Feature*-Repräsentation und die *Score*-Funktion gemeinsam durch *Backpropagation* trainiert werden kann, stellte sich heraus, dass diese Vorgehensweise in der Praxis schwache Modelle generiert. Daher wurde eine zweistufige *Trainings*-Strategie verwendet. Zunächst wird der *Deskriptor* gelernt, danach wird die *Score*-Funktion bei dem nun festgelegten *Deskriptor* angelernt. Eine weitere Verbesserung des Modells wird durch zufällige Bildskalierung während des Aufmerksamkeits-*Trainings* erreicht. Dies ist intuitiv, da das Aufmerksamkeits-Modell in der Lage sein sollte, effektive Bewertungen für Merkmale in verschiedenen Skalierungen zu generieren. Dazu wurden die Eingabe-Bilder anfangs in der Mitte beschnitten, um quadratische Bilder zu erzeugen, und auf (900×900) skaliert. Zufällige (720×720) Zuschnitte werden dann extrahiert und schließlich mit einem Faktor $\gamma <= 1$ zufällig skaliert. [28]

Eigenschaften Ein unkonventioneller Aspekt des Systems besteht darin, dass die *Keypoint*-Auswahl nach der *Deskriptor*-Extraktion erfolgt, was sich damit zu $SIFT$ unterscheidet, wo *Keypoints* zuerst erfasst und später beschrieben werden. Traditionelle *Keypoint*-Detektoren konzentrieren sich auf die wiederholte Erkennung von *Keypoints* unter verschiedenen Bildaufnahme-Bedingungen, die nur auf ihren niedrigen Eigenschaften basieren. Für eine Erkennungs-Aufgabe, wie zum Beispiel das Abrufen von Bildern, ist es jedoch auch wichtig *Keypoints* zu wählen, die verschiedene Objekt-Instanzen unterscheiden. Mit der *DELF-Pipeline* werden beide Ziele erreicht, indem ein Modell trainiert wird, welches die Semantik höherer Ebenen in der *Feature-Map* codiert, und das Lernen, diskriminierende *Features* für die Klassifizierungs-Aufgabe auszuwählen. Dies steht im Gegensatz zu kürzlich vorgeschlagenen Techniken zum Lernen von *Keypoint*-Detektoren, wie $LIFT$ [38], die *Trainings*-Daten basierend auf $SIFT$-Übereinstimmungen sammeln. Obwohl das $DELF$-Modell nicht darauf beschränkt ist, Invarianzen hinsichtlich Position und Standpunkt zu lernen, lernt es dies implizit - ähnlich wie bei CNN-basierten Bild-*Klassifikations*-Techniken. [28]

3.0.6. Reduktion der Dimension

Die Dimensionalität ausgewählter Merkmale wird reduziert, um eine verbesserte Wiederauffindung(*Retrieval*)-Genauigkeit zu erreichen, wie es üblich ist. Erstens werden die ausgewählten Merkmale L_2-normalisiert, und ihre Dimension wird durch die *Principal Component Analysis*

(PCA) auf 40 reduziert, was einen guten Kompromiss zwischen Kompaktheit und Diskriminierung darstellt. [16] Abschließend durchlaufen die *Features* noch einmal eine L_2-Normalisierung. [28]

4. Implementierung

In diesem Kapitel wird die Implementierung der Komponenten der Merkmalsextraktion sowie des *Matchings* im speziellen für $DELF$ und allgemein für $SIFT$ beschrieben. Im Folgendem wird die Umgebung bzw. die genutzten Frameworks für die Implementierung ausgeführt.

4.1. Umgebung

4.1.1. Betriebssystem und Programmiersprache

Folgende Software bzw. Betriebssysteme wurden für die Implementierung verwendet: *macOS High Sierra* ist das verwendete Betriebssystem. Als Programmiersprachen wurde *Python 2.7* und *Python 3.5* ([49]) bei der Implementierung für $DELF$ und $SIFT$ verwendet.

4.1.2. *Anaconda*

Die *Open Source Anaconda*-Distribution liefert einen *Package Manager*, für *Python* und *R* im Feld der *Data Science* und *Machine Learnings*. Über 200 Pakete werden automatisch mit Anaconda installiert, 2000 zusätzliche *Open-Source*-Pakete (einschließlich *R*) können mit dem Befehl *conda install* einzeln aus dem Anaconda-*Repository* installiert werden. Mit Anaconda wurden die drei Projekte aufgesetzt und die nötigen *Packages* geladen und installiert. [39]

4.1.3. *TensorFlow*

TensorFlow ist eine *Open-Source-Software*-Bibliothek für numerische Berechnungen. Die flexible Architektur ermöglicht eine einfache Ausführung von Berechnungen auf einer Vielzahl von Plattformen (CPUs, GPUs, TPUs), Desktop-Rechnern über *Server-Clustern* bis hin zu mobilen Geräten. Ursprünglich von Forschern und Ingenieuren des *Google Brain Teams* in der KI-Organisation von *Google* entwickelt, bietet es eine starke Unterstützung für das maschinelle Lernen. TensorFlow wird in vielen anderen wissenschaftlichen Bereichen eingesetzt. [55]

4.1.4. *OpenCV*

OpenCV (*Open Source Computer Vision Library*) steht unter einer BSD-Lizenz und ist daher sowohl für den akademischen als auch für den kommerziellen Gebrauch frei. Es verfügt über *C ++*- , *Python* und *Java*-Schnittstellen und unterstützt *Windows*, *Linux*, *Mac OS*, *iOS* und *Android*. *OpenCV* wurde für die Rechen-Effizienz entwickelt und konzentriert sich auf Echtzeit-Anwendungen. In optimiertem *C* und *C ++* geschrieben, kann die Bibliothek die *Multi Core*-Verarbeitung nutzen. Mit *OpenCL* aktiviert, kann die *Hardware*-Beschleunigung der zugrunde liegenden heterogenen Rechenplattform genutzt werden. [45]

4.2. Projekt Struktur

Die Implementierung wurde in drei *PyCharm* ([48]) Projekte aufgeteilt: $SIFT$, $DELF$ und eine Visualisierung der *Attention* für $DELF$.

4.3. Merkmalsextraktion

Für die Merkmalsextraktion mit $DELF$ wird der *TensorFlow-Code* von *Github* ([40]) verwendet. Die Merkmalsextraktion mit $SIFT$ wird mit der *OpenCV*-Implementierung ([53]) durchgeführt.

4.4. *Matcher*

Es wurden zwei unterschiedliche *Matcher* implementiert. Diese werden im Folgendem beschrieben:

Brute-Force* und *Ratio-Test Der *Brute-Force-Matcher* ist simpel aber aufwändig. Er nimmt den *Deskriptor* eines Merkmals $\mathfrak{f}$ aus dem ersten Satz ($\mathfrak{F}_A$) und vergleicht mit allen anderen Merkmalen im zweiten Satz ($\mathfrak{F}_B$) durch Abstandsberechnung. Der nächste, also der mit der geringsten Distanz d_{min}, wird zurückgegeben. Um den *BF-Matcher* nutzen zu können, wird zuerst das `BFMatcher`-Objekt erstellt. Es sind zwei optionale Parameter erforderlich. Der erste Parameter ist der `normType`. Er gibt die zu verwendende Abstands-Metrik an. Standardmäßig wird die L_2 (`cv2.NORM_L2`) genutzt. Der zweite Parameter ist die boolesche Variable, `crossCheck`, die standardmäßig falsch ist. Der `BFMatcher.knnMatch` wird genutzt um die besten Übereinstimmungen zu erhalten. In der Implementierung wir `k = 2` genutzt, damit wir der von D.Lowe, in seinem Artikel erklärten Verhältnistest(*Ratio Test* [21]), genutzt. [46] Der Ansatz wurde in Abschnitt 2.2.7 beschrieben.

***cKDTree* und RANSAC** Diese Klasse stellt einen Index für eine Menge von k-dimensionalen Punkten bereit, mit deren Hilfe die nächsten Nachbarn eines beliebigen Punktes schnell nachgeschlagen werden können. Der verwendete Algorithmus ist in Maneewongvatana und Mount 1999 ([23]) beschrieben. Die allgemeine Idee ist, dass der *k-d-Baum* ein binärer Baum ist, dessen Knoten jeweils ein achsenorientiertes n-dimensionales Hyperrechteck

$$\mathbf{R} = \prod_{i=1}^{n} [x_i, y_i] = [x_1, y_1] \times [x_2, y_2] \times \cdots \times [x_n, y_n] \tag{93}$$

darstellen. [33] Jeder Knoten gibt eine Achse an und teilt die Menge der Punkte auf der Grundlage davon auf, ob ihre Koordinate entlang dieser Achse größer oder kleiner als ein bestimmter Wert ist. [41] RANSAC (siehe 2.1.9)ist ein iterativer Algorithmus zur zuverlässigen Abschätzung von Parametern von einer Teilmenge von *Inliern* aus dem Datensatz. [51] Der RANSAC erhält als *Input* die *Keypoints-Inlier*, für diese werden dann die Parameter einer affinen Transformation (siehe 2.1.3) geschätzt. Auf Grundlage dessen, werden die finalen *Inlier* in einem Bildpaar erzeugt (gefiltert).

Bezeichnung der Verfahren Der Grund zwei unterschiedliche *Matching*-Verfahren in dieser Arbeit zu nutzen, besteht darin, dass $DELF$ mit *cKDTree* und RANSAC in [40] erfolgreich implementiert wurde. Das $SIFT$-Verfahren arbeitet jedoch optimal mit *Brute-Force* und *Ratio Test*. Um eine ausgewogene Bewertung zu treffen, wird in der folgenden Empirische Analyse mit beiden Verfahren evaluiert. Denn es wäre möglich, dass $DELF$ zwar mit *cKDTree* und RANSAC besser abschneidet, für *Brute-Force* und *Ratio Test* jedoch schlechter als $SIFT$ performt. Für die kürzere Schreibweise der Verfahren steht in folgendem der Index ζ für ein *Matching* mittels *cKDTree* und RANSAC und der Index β für ein *Matching* mit *Brute-Force* und dem *Ratio Test* nach Lowe.

4.5. $DELF$ *Python* Implementierung

4.5.1. $DELF$ Merkmalsextraktion

In der Datei `delf_config_example.pbtxt` sind die wichtigsten Paramter für die Merkmalsextraktion gespeichert. Zunächst wird der Modell-Pfad angegeben.
Die *Python*-Funktion `tf.saved_model.loader` bietet Lade-Funktionen für ein gespeichertes Modell. Danach folgen die Bild Skalierungen für $DELF$.
Es wurden folgende Skalierungen verwendet: $S = \{0,25\ 0,3536\ 0,5\ 0,7071\ 1,0\ 1,4143\ 2,0\}$.
Des Weiteren werden wichtige Parameter für die *Principal Component Analysis*-Dimension-Reduktion definiert. Die Dimension der *Deskriptoren* wurde wie im Artikel [28] auf $\mathfrak{d} = 40$ gesetzt.
Im Folgendem *Python-Code* wird die TensorFlow-Klasse `Graph` verwendet. Ein `Graph` enthält eine Reihe von `tf.Operation`-Objekte, die Rechen-Einheiten darstellen und `tf.Tensor`-Objekte,

die die Daten-Einheiten beschreiben, die zwischen Operationen fließen. Ein Standard-`Graph` ist immer registriert und kann durch Aufruf von `tf.get_default_graph` aufgerufen werden. Eine andere typische Verwendung umfasst den Kontextmanager `tf.Graph.as_default`, der den aktuellen *Default Graph* überschreibt. [56] Die Klasse `Session` wird zur Ausführen von TensorFlow-Operationen benötigt. Ein `Session`-Objekt kapselt die Umgebung, in der Operation-Objekte ausgeführt werden, und `Tensor`-Objekte werden ausgewertet. [55]

```
#...more code

  # Tell TensorFlow that the model will be built into the default Graph.

  # A Graph contains a set of tf.Operation objects, which represent units of
    computation
  # Reading list of images.
  with tf.Graph().as_default():
    # A queue with the output strings.
    filename_queue = tf.train.string_input_producer(image_paths, shuffle=False)
    # A QueueRunner for the Queue is added to the current Graph's QUEUE_RUNNER
    collection.
    print('filename_queue:', filename_queue)
    # A Reader that outputs the entire contents of a file as a value.
    reader = tf.WholeFileReader()
    _, value = reader.read(filename_queue)
    print('value:', value)
    # Decode a JPEG-encoded image to a uint8 tensor, output an RGB image.
    image_tf = tf.image.decode_jpeg(value, channels=3)

    # A Session object encapsulates the environment in which Operation objects are
     executed, and Tensor objects are evaluated.
    with tf.Session() as sess:
      # Initialize variables.

      # An Op that initializes global variables in the graph.
      init_op = tf.global_variables_initializer()
      sess.run(init_op)

      # Loading model that will be used.
      # Loads the model from a SavedModel as specified by tags.
      tf.saved_model.loader.load(sess, [tf.saved_model.tag_constants.SERVING],
                                 config.model_path)
      # Returns the default graph for the current thread.
      graph = tf.get_default_graph()
      input_image = graph.get_tensor_by_name('input_image:0')
      print('input_image:', input_image)
      input_score_threshold = graph.get_tensor_by_name('input_abs_thres:0')
      print('input_score_threshold:', input_score_threshold)
      input_image_scales = graph.get_tensor_by_name('input_scales:0')
      input_max_feature_num = graph.get_tensor_by_name(
          'input_max_feature_num:0')
      boxes = graph.get_tensor_by_name('boxes:0')
      raw_descriptors = graph.get_tensor_by_name('features:0')
      feature_scales = graph.get_tensor_by_name('scales:0')
      attention_with_extra_dim = graph.get_tensor_by_name('scores:0')
      attention = tf.reshape(attention_with_extra_dim,
                             [tf.shape(attention_with_extra_dim)[0]])

      # Extract DELF features from input image
      locations, descriptors = feature_extractor.DelfFeaturePostProcessing(
          boxes, raw_descriptors, config)

```

```
#...more code
```

Listing 1: Ausschnitt aus extract_features_karlshorst_dataset.py für *DELF* - Quelle des Ausgangscodes: [40]

Der obige *Code* in extract_features_karlshorst_dataset.py führt die Merkmalsextraktion für die in der `list_images_karlshorst.txt` Datei enthaltenen Bilder durch und speichert die *Deskriptoren* mit der Datei-Endung `.delf` ab.

4.5.2. *DELF Matching*

***Python-Code* für $DELF_\beta$-*Matching* für Karlshorst-Datensatz** Für $DELF_\beta$ wird nun das *Matching* durchgeführt. Dabei wurde ein $Ratio = 0,75$ verwendet.

```
def run_matching_BFMatcher_ratio(image_1_path, image_2_path, features_1_path,
    features_2_path):
    tf.logging.set_verbosity(tf.logging.INFO)
    # Read features
    locations_1, _, descriptors_1, _, _ = feature_io.ReadFromFile(
        features_1_path)
    num_features_1 = locations_1.shape[0]
    tf.logging.info("Loaded image 1's %d features" % num_features_1)
    locations_2, _, descriptors_2, _, _ = feature_io.ReadFromFile(
        features_2_path)
    num_features_2 = locations_2.shape[0]
    tf.logging.info("Loaded image 2's %d features" % num_features_2)
    # Find nearest-neighbor matches using a BFMatcher
    # BFMatcher with default params
    matcher = cv2.BFMatcher(normType=cv2.NORM_L2, crossCheck=False)
    matches = matcher.knnMatch(np.asarray(descriptors_1, np.float32), np.asarray(
    descriptors_2, np.float32), k=2)

    distances = 0
    # Apply ratio test
    good = []
    for m, n in matches:
        if m.distance < 0.75 * n.distance:
            good.append([m])
            distances = distances + m.distance

    if len(good) == 0:
        return num_features_1, num_features_2, len(good), float('Inf'), good,
    locations_1, locations_2

    else:
        return num_features_1, num_features_2, len(good), distances/len(good),
    good,  locations_1, locations_2
```

Listing 2: Ausschnitt aus `match_images.py` für *DELF* - Quelle des Ausgangscodes: [47]

Der obige Code ist ein Ausschnitt aus der `match_images.py` Datei mit der Methode `run_matching_BFMatcher_ratio()`. Es werden die zuvor extrahierten *Deskriptoren* geladen und gemacht. Zudem werden wichtige Werte wie Anzahl an genutzten *Keypoints*, Anzahl der *Inlier* und euklidischer Abstand in eine CSV-Datei geschrieben.

***Python-Code* für $DELF_\zeta$-*Matching* für Karlshorst-Datensatz** Für $DELF_\zeta$ wird nun das *Matching* durchgeführt.

```
def run_matching_cKDTree_ransac(image_1_path,  image_2_path, features_1_path,
    features_2_path):
    tf.logging.set_verbosity(tf.logging.INFO)
    # Read features.
```

```
    locations_1, _, descriptors_1, _, _ = feature_io.ReadFromFile(
        features_1_path)
    num_features_1 = locations_1.shape[0]
    tf.logging.info("Loaded image 1's %d features" % num_features_1)
    locations_2, _, descriptors_2, _, _ = feature_io.ReadFromFile(
        features_2_path)
    num_features_2 = locations_2.shape[0]
    tf.logging.info("Loaded image 2's %d features" % num_features_2)
    # Find nearest-neighbor matches using a KD tree.
    d1_tree = cKDTree(descriptors_1)
    distances, indices = d1_tree.query(
        descriptors_2) #removed: distance_upper_bound=_DISTANCE_THRESHOLD

    # Select feature locations for putative matches.
    locations_2_to_use = np.array([
        locations_2[i,]
        for i in range(num_features_2)
        if indices[i] != num_features_1
    ])
    locations_1_to_use = np.array([
        locations_1[indices[i], ]
        for i in range(num_features_2)
        if indices[i] != num_features_1
    ])
    if len(locations_2_to_use) == 0 or len(locations_1_to_use) == 0:
        print('Nothing found.')
        return num_features_1, num_features_2, 0
    else:
     # Perform geometric verification using RANSAC.
     _, inliers = ransac(
        (locations_1_to_use, locations_2_to_use),
        AffineTransform,
        min_samples=3,
        residual_threshold=20,
        max_trials=1000)

     avarage_distance_inliers = calculate_distance_inliers(distances, inliers)
     if inliers is None:
         return num_features_1, num_features_2, float('inf'), locations_1_to_use,
    locations_2_to_use
     else:
         return num_features_1, num_features_2, inliers, avarage_distance_inliers,
    locations_1_to_use, locations_2_to_use, locations_1, locations_2
```

Listing 3: Ausschnitt aus `match_images.py` für *DELF* - Quelle des Ausgangscodes: [40]

Der obige *Code* zeigt die Methode `run_matching_cKDTree_ransac()` in der `match_images.py` Datei. Es werden die zuvor extrahierten *Deskriptoren* geladen. Mittels *k-d-Baum* werden die nächstgelegenen Nachbarn gefunden. Danach wird die geometrische Verifikation mit RANSAC durchgeführt. Zudem werden wichtige Werte wie Anzahl an genutzten *Keypoints*, Anzahl der *Inlier* und euklidischer Abstand in eine CSV-Datei geschrieben.

4.5.3. Visualisierung der *DELF Attention*

Dieses Projekt ist ein *Wrapper* für *DELF TensorFlow-Code*. Unter anderem kann damit die Aufmerksamkeit (*Attention*) für Bilder visualisieren, damit sichtbar wird, wo die *DELF-Keypoints* ausgewählt werden. [43]

5. Empirische Analyse

In der empirischen Analyse soll nun das Verhalten bezüglich des *Matchings*, aber auch Informationen über *Keypoints* sowie der *Attention Map* erfolgen. Dazu werden zwei eigene *Outdoor*-Datensätze eingeführt: Der erste ist der Karlshorst-Datensatz, welcher Aufnahmen von *Outdoor*-Szenen enthält. Der zweite ist der Karlshorst-Transformations-Datensatz, welcher unterschiedliche Bildtransformationen zu den Szenen aus dem ersten Datensatz umfasst.

5.1. Bezeichnung der Elemente der Merkmalsextraktion und *Matching*

Für die empirische Analyse gelten die folgenden Bezeichnungen. Ein *Feature* oder auch Merkmal sei ein Vektor,

$$\mathfrak{f} \in \mathbb{R}^n, \tag{94}$$

welcher den *Deskriptor* beschreibt. Ein zugehöriger *Keypoint* sei als

$$\kappa \in \mathbb{R}^2 \tag{95}$$

mit einer x- und y-Koordinate bezeichnet. Für das *Matching* werden zwei Bilder I_A und I_B verglichen. Die Menge aller extrahierten m *Feature* aus Bild I_A sei

$$\mathfrak{F}_A = \{\mathfrak{f}_1, \cdots, \mathfrak{f}_m\}. \tag{96}$$

Analog gilt dies für die Menge an m *Keypoints*

$$\mathfrak{K}_A = \{\kappa_1, \cdots, \kappa_m\} \tag{97}$$

aus einem Bild I_A.

Abbildung 33: Merkmalsextraktion

Ein *Inlier* sei definiert als zwei ähnliche Merkmale (*Nearest Neighbor*) $\mathfrak{f}_A$ und $\mathfrak{f}_B$ aus Bild I_A und I_B, welche einen gewissen Abstand d aufweisen und zudem geometrische-(RANSAC) oder *Ratio*-Kriterien erfüllen und wird mit $\mathfrak{i}$ bezeichnet. Analog sei die Menge an *Inliern* $\mathfrak{I}$.

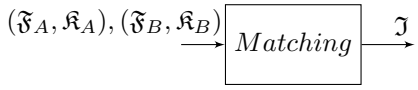


Abbildung 34: *Matching*

Allgemein gilt $|\mathfrak{F}_A| = |\mathfrak{K}_A|$, da jedem *Keypoint* genau ein Merkmal zugeordnet wird.

5.2. Bewertungskriterien und Methoden

Im Folgenden sei $\mathbf{a} = \begin{pmatrix} x & y \end{pmatrix}^T$ ein *Keypoint* mit einer x- und y-Koordinate im Bild I_x.

Theorem 6 *Die euklidische Norm (auch L_2-Norm) eines Vektors $\mathbf{v} \in \mathbb{R}^n$ sei definiert als $\|\mathbf{v}\|_2 := \sqrt{v_1^2 + \cdots + v_n^2}$. [6]*

Theorem 7 *Der euklidische Abstand d zweier Vektoren $\mathbf{v}$ und $\mathbf{w} \in \mathbb{R}^n$ sei definiert als $d(\mathbf{v}, \mathbf{w}) := \sum_{i=1}^{n} \sqrt{v_i^2 - w_i^2}$.*

5.2.1. Bildpaar

Um das *Matching* bewerten zu können, werden *Performance*-Kriterien benötigt. Diese Bewertungskriterien basieren auf der Anzahl korrekter Übereinstimmungen und falscher Übereinstimmungen, die für ein Bildpaar (I_A, I_B) festgestellt werden. Sei a eine Punkt bzw. *Keypoint* im Bild I_x und b einen Punkt im Bild I_y. a und b seien ein *Match* $M(a, b)$ bzw. *Inlier* $\mathfrak{I}_d$, wenn der euklidische Abstand (Theorem 7) zwischen ihren *Deskriptoren* $\mathfrak{f}_A, \mathfrak{f}_B \in \mathbb{R}^n$ kleiner als Schwellenwert t ist.

$$d(\mathfrak{f}_A, \mathfrak{f}_B) < t \tag{98}$$

Es zeigt sich, dass eine Bewertung der Algorithmen über den globalen(für $SIFT$ und $DELF$ der gleiche) Schwellenwert t nicht sinnvoll ist, da die Dimension des $SIFT$-*Deskriptors* 128 ($\mathfrak{f}_{SIFT} \in \mathbb{R}^{128}$) und die des $DELF$-*Deskriptors* 40 ($\mathfrak{f}_{DELF} \in \mathbb{R}^{40}$) ist. Trotzdem lässt sich grundsätzlich mit diesem Ansatz eine Ähnlichkeit zweier Deskriptoren innerhalb des Verfahrens sehr gut bestimmen und *Inlier*-Kandidaten ($\mathfrak{I}_d$) finden. [25] [24] [26] [17]

Matcher Im folgendem wird ein allgemeiner Bild-*Matcher* mit Verfahren X Input Bild A und B beschreiben:

Algorithmus 3 Matcher$(\mathfrak{F}_A, \mathfrak{F}_B, \mathfrak{K}_A, \mathfrak{K}_B, X)$

1: bestimme Menge der *Inlier* $\mathfrak{I}_d$ mit euklidischer Distanz durch $d(\mathfrak{F}_A, \mathfrak{F}_B)$ mit X
2: bestimme Menge der *Inlier* $\mathfrak{I}_F \subset \mathfrak{I}_d$ (Filter) durch zusätzliche Logik für $(\mathfrak{K}_A, \mathfrak{K}_B)$ oder $(\mathfrak{F}_A, \mathfrak{F}_B)$ mit X
3: **return** $\mathfrak{I}_F$

Der Algorithmus bzw. Verfahren 3 beschreibt damit zusammenfassend die Verfahren X_β und X_ζ (siehe Abschnitt 4.4).

5.2.2. Datei-Ebene

Für das Testen der Algorithmen werden nun Metriken eingeführt, welche auf Datei-Ebene arbeiten. Ob ein Bild einer gewissen Klasse angehört oder nicht bzw. die gleiche Szene zeigt. Präzision (*Precision*) beschreibt die Wahrscheinlichkeit, dass ein gefundenes Bild, relevant ist.

$$precision = \frac{tp}{tp + fp} \tag{99}$$

Der *Recall* sei die Wahrscheinlichkeit, dass ein relevantes Bild in der Suche gefunden wird.

$$recall = \frac{tp}{tp + fn} \tag{100}$$

Das traditionelle F-Maß oder der ausgeglichene *F-Score* ist das harmonische Mittel aus *Precision* und *Recall*. [32]

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{101}$$

5.3. Karlshorst-Datensatz

Um die Verfahren zu evaluieren werden Bilddaten benötigt. Dazu wurden im Herbst 2018 mit einem *iPhone SE* (Kamera mit 12 Megapixeln) Aufnahmen von *Outdoor*-Szenen in Berlin-Karlshorst gemacht. Es wurden 13 unterschiedliche Szenen fotografiert. Das erste Bild der Szene, wird im folgendem mit einem Anhang '0' bezeichnet. Des Weiteren sei $\mathcal{KD}$ die Menge aller 130 Bilder des Karlshorst-Datensatz.

(a) `turm0` (b) `ausguck0` (c) `grusel_haus0` (d) `bushalte0`

(e) `hochhaus0` (f) `pferd_links0` (g) `schule_bahn0` (h) `pferd_rechts0`

(i) `schule_vorn0` (j) `trabrennbahn0` (k) `tribuehne0`

(l) `antenne0` (m) `baum0`

Abbildung 35: Karlshorst-Datensatz $\mathcal{KD}$ mit 13 *Query*-Bildern

In Abbildung 35 wurden die 13 *Query*-Bilder visualisiert. Wie erkennbar ist, sind die Aufnahmen sehr unterschiedlich. Die Lichtverhältnisse variieren und der Winkel zur Szene ist stark variabel. Zudem werden Gebäude, wie in Abbildung 35i die Schule, durch einen Baum verdeckt. Zu diesen Start-Bildern bzw. *Query*-Bildern gibt es neun weiter Aufnahmen, welche die gleiche Szene zeigen aber aus einer anderen Position aufgenommen wurden und leicht verschiedene Lichtverhältnisse aufweisen. Sie besitzen den Anhang '`1-9`'.

(a) `bushalte0` (b) `bushalte1` (c) `bushalte2` (d) `bushalte3`

(e) `bushalte4` (f) `bushalte5` (g) `bushalte6` (h) `bushalte7`

(i) `bushalte8` (j) `bushalte9`

Abbildung 36: `bushalte0` - `bushalte9`, 10 Aufnahmen pro Klasse bzw. Szene aus $\mathcal{KD}$

Insgesamt gibt es damit 10 Bilder pro Szene (inklusive *Query*-Bild). Beispielhaft wird dies für die Szene `bushalte0` in Abbildung 36 dargestellt. Das Bild in Abbildung 36a ist von einer erhöhten Position im Bezug zur Bushaltestelle aufgenommen worden. Zudem hat diese Aufnahme die maximale Distanz zwischen Kamera und Bushaltestelle. Die Aufnahme 'bushalte1' wurde auf etwa gleichem Niveau (Straße) geschossen. In Abbildung 36h wurde die Kamera stark nach links bewegt um eine schräge Perspektive zu erzeugen. Das Bild 36j ist eine frontale Aufnahme der Haltestelle und innerhalb der zehn Aufnahmen die mit dem geringsten Abstand zwischen Kamera und Szene. Wie in Abbildung 35 zu sehen gibt es 13 Klassen und zehn Aufnahmen, für den Karlshorst-Datensatz ergeben sich damit 130 Bilder.

5.3.1. Zwei *Matching* Vorgehen

Wie bereits erwähnt enthält der Datensatz 130 Bilder, für 13 Szenen. Um eine kürzere Schreibweise für die folgende Analyse zu finden werden nun zwei unterschiedliche *Matching* Vorgehensweisen beschrieben. Der zunächst naheliegende Ansatz ist sicherlich innerhalb der Szene - zehn Aufnahmen von Orten in Karlshorst nach *Inliern* zu suchen. Im Folgenden seien alle Bilder in einer Szene definiert als $\mathbf{S} = \{I_0, \cdots, I_9\}$. Die gesamte Menge an *Inliern* pro Szene sei $\mathfrak{I}_\mathbf{S}$. Dabei sei I_0 das *Query*-Bild.

Algorithmus 4 query_same_scene($\mathbf{S}$)

1: **for** `i = 0; i < 10; i++` **do**
2: bestimme Menge der *Inlier* $\mathfrak{I}_i$ für Bildpaar $(I_0, S(i))$
3: füge Menge der *Inlier* $\mathfrak{I}_i$ zu $\mathfrak{I}_\mathbf{S}$ hinzu
4: **return** $\mathfrak{I}_\mathbf{S}$

Der Vorgang im Algorithmus 4 wird nun für alle Szenen im Karlshorst-Datensatz durchgeführt. Für die folgende Evaluation wird diese Form des Durchsuchen des Datensatzes als SAME_SCENE130 bezeichnet. Mit dieser Methode werden 130 Bild-Paare auf Ähnlichkeiten überprüft. Der zweite Ansatz ist ein eher globaler, aber auch realistischer. Jedes *Query*-Bild der 13 Szenen des Datensatzes wird mit jedem Bild im Datensatz gematcht. Damit ergeben sich $13 * 130 = 1690$ Bildpaare. Dabei sei $\mathcal{KD}$ die Menge aller Bilder in Karlshorst-Datensatz. Realistisch ist der *Matching*-Ansatz deswegen, da nun auch *Query*-Bilder gegen Bilder gematcht werden, die nicht zur Szene der *Query* gehören. Genau dies ist der Fall in realistischen Datensätzen und Anwendungen. Die gesamte Menge an *Inliern* pro *Query*-Bild sei im Folgendem $\mathfrak{I}_{\mathcal{KD}}$.

Algorithmus 5 query_all_scenes(I_0)

1: **for** `i = 0; i < 130; i++` **do**
2: bestimme Menge der *Inlier* $\mathfrak{I}_i$ für Bildpaar $(I_0, \mathcal{K}(i))$
3: füge Menge der *Inlier* $\mathfrak{I}_i$ zu $\mathfrak{I}_{\mathcal{K}}$ hinzu
4: **return** $\mathfrak{I}_{\mathcal{KD}}$

Der Algorithmus 5 wird für alle 13 *Query*-Bilder der Szenen im Karlshorst-Datensatz durchgeführt. Dieser Ansatz sei im folgendem als ALL_SCENES1690 bezeichnet.

5.3.2. Merkmalsextraktion für den Karlshorst-Datensatz

5.3.3. Anzahl extrahierte *Keypoints*

Dem $DELF$-Verfahren, wie auch dem $SIFT$-Verfahren wird eine maximal mögliche Anzahl an *Features* vorgegeben. Für die Evaluation wurde der Wert $m = 1000$ gewählt. Im Folgenden wird die Anzahl extrahierter *Keypoints* in zwei Tabellen beschreiben.

Verfahren	`antenne`	`ausguck`	`baum`	`bushalte`	`grusel_haus`	`hochhaus`	`pferd_links`
$DELF$	916,1	1000	1000	1000	1000	1000	866,8
$SIFT$	1000	1000	1000	1000	1000	1000	1000

Tabelle 1: Durchschnittliche Anzahl extrahierte *Keypoints* $|\overline{\mathfrak{K}}|$ für den Karlshorst-Datensatz $\mathcal{KD}$ aus 130 Paaren pro Szene Teil 1, SAME_SCENE130

Szene	`pferd_rechts`	`schule_bahn`	`schule_vorn`	`trabrennbahn`	`tribuehne`	`pferd_links`
$DELF$	998,9	1000	1000	1000	1000	993,3
$SIFT$	1000	1000	1000	1000	1000	1000

Tabelle 2: Durchschnittliche Anzahl extrahierte *Keypoints* $|\overline{\mathfrak{K}}|$ für den Karlshorst-Datensatz $\mathcal{KD}$ aus 130 Paaren pro Szene Teil 2, SAME_SCENE130

In den obigen Tabellen 1 und 2 wurde die durchschnittliche Anzahl an extrahierten *Features* für die 13 Szenen visualisiert. Es zeigt sich, dass das $DELF$-Verfahren in der Szene `antenne` sowie `pferd_links, pferd_rechts` und `turm` nicht die mögliche Anzahl von 1000 *Deskriptoren* extrahiert hat. Die Merkmalsextraktion mit $SIFT$ hingegen extrahierte für alle Szenen 1000 *Keypoints*.

5.3.4. Distanzen bei *Matching* für den Karlshorst-Datensatz

Die in Abbildung 36 visualisierte Szene `bushalte` wird nun im Detail hinsichtlich der durchschnittlichen Distanz der *Deskriptoren* analysiert. Die euklidische Distanz der Merkmalsvektoren ist ein direkter Indikator für die Ähnlichkeit zweier *Keypoints*.

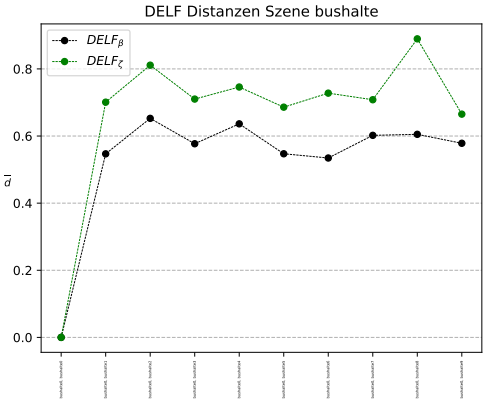


Abbildung 37: *DELF*-Abstand für Szene `bushalte` Karlshorst-Datensatz, SAME_SCENE130

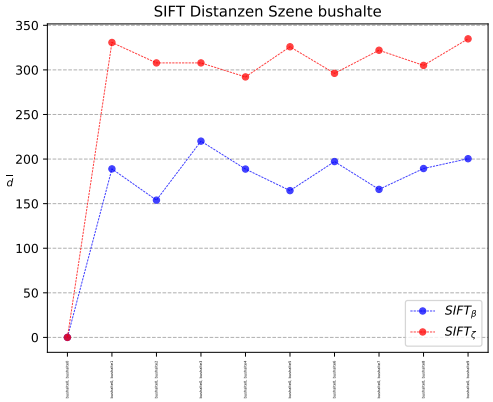


Abbildung 38: *SIFT* Abstand für Szene `bushalte` Karlshorst-Datensatz, SAME_SCENE130

Die obere linke Abbildung 37 zeigt die durchschnittliche Distanz zum Original-Bild für *DELF* `bushalte0`. In der rechten Abbildung 38 wird analog $\overline{d}$ für das *SIFT*-Verfahren aufgetragen. Für die 130 Bild-Paare aus SAME_SCENE130 wird im Folgendem die durchschnittliche Distanz innerhalb der Szene aufgetragen. Dabei sei zu beachten, dass es Bild-Paare gibt in denen keinerlei *Inlier* gefunden wurde. Für diesen Fall wird die euklidische Distanz d als unendlich (∞) definiert, zumindest numerisch. Auf Grund der unterschiedlichen Dimension der *Deskriptoren*, 40-dimensional und 128-dimensional, ist eine Visualisierung im gleichen Plot nicht sinnvoll. Der maximale Abstand wird bei $DELF_\beta$ für das Bildpaar (`bushalte0,bushalte2`) mit 0,65 festgestellt. Hinsichtlich $DELF_\zeta$ liegt der maximale Abstand bei 0,88 und trifft beim Bildpaar (bushalte0, bushalte8) auf. Die Tatsache, dass das Verfahren mit *Brute Force* geringere Distanzen erzeugt, ist ersichtlich, da tatsächlich alle *Deskriptoren* verglichen werden (bessere *Matches*) und somit auch geringe Distanzen gemessen werden. $SIFT_\beta$ findet einen maximalen Abstand für (`bushalte0, bushalte3`) mit $\overline{d}$=220,1. Das *SIFT*-Verfahren mit RANSAC hat bei (`bushalte0, bushalte9`) den durchschnittlichen euklidischen Abstand 334,8. Beide Abbildungen zeigen einen ersten Datenpunkt bei 0, welcher das *Matching* für `bushalte0,bushalte0` beschreibt. Da somit der gleiche *Deskriptor*-Vektor verglichen wird, ist $\overline{d}$=0.

Distanzen für same_scene130 Im Folgendem werden die Messungen der Distanzen für den Karlshorst-Datensatz bei SAME_SCENE130-*Matching* für alle 130 Bildpaare aufgetragen.

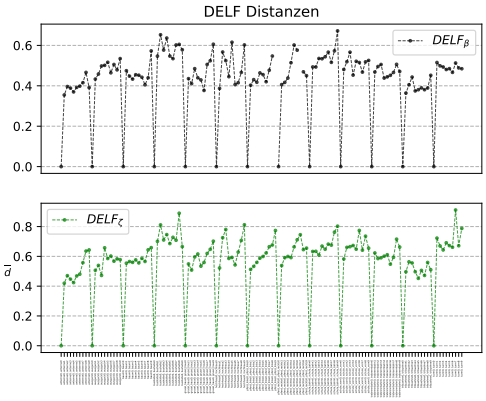


Abbildung 39: *DELF* Distanzen für Karlshorst-Datensatz $\mathcal{K}$, SAME_SCENE130

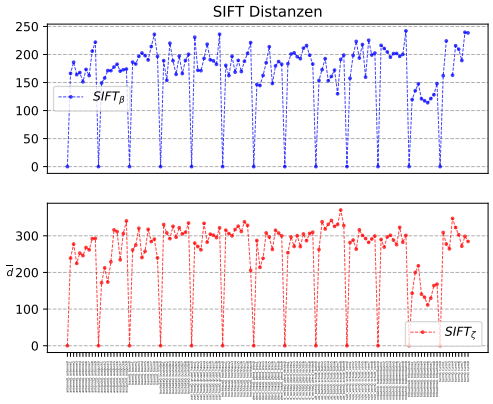


Abbildung 40: *SIFT* Distanzen für Karlshorst-Datensatz $\mathcal{K}$, SAME_SCENE130

In Abbildung 39, betreffend, p =(`pferd_links0, pferd_links9`), liegt eine solche unendliche Distanz bei $DELF_\beta$ vor. Mathematisch geschrieben: $|\mathfrak{I}(p)| = 0 \Rightarrow \overline{d} = \infty$. Im Graph erkennt man dies durch fehlende Einträge für die Distanz $\overline{d}$, da in diesem Fall dem Argument kein Wert zugeordnet werden kann. Diese Definitionslücken (Singularitäten) treten jedoch nur selten für die gemessenen Distanzen aus $\mathcal{K}$ auf.

5.3.5. Anzahl *Inlier* bei *Matching*

Für den Karlshorst-Datensatz wurde bei 130 Bild-Paaren Mittelwert und Standardabweichung der Anzahl *Inlier* für *DELF* und *SIFT* bestimmt.

Verfahren	$\lvert\overline{\mathfrak{I}}\rvert$	$\sigma(\lvert\overline{\mathfrak{I}}\rvert)$
$DELF_\zeta$	165,7	277,4
$DELF_\beta$	119,0	284,5
$SIFT_\zeta$	121,7	294,6
$SIFT_\beta$	129,1	296,2

Tabelle 3: Mittelwert und Standardabweichung der Anzahl *Inlier* für den Karlshorst-Datensatz $\mathcal{K}$ aus 130 Paaren innerhalb der Klasse bzw. Szene, SAME_SCENE130

In Tabelle 3 ist sichtbar, dass $DELF_\zeta$ die meisten *Inlier* innerhalb der 13 Szenen gefunden hat. Zudem variiert die Anzahl an *Inliern* für $DELF_\zeta$ am wenigsten mit $\sigma(|\mathfrak{I}_{DELF_\zeta}|) = 277,4$. Die geringste Anzahl an *Inliern* hat $DELF_\beta$.

Verfahren	$\lvert\overline{\mathfrak{I}}\rvert$	$\sigma(\lvert\overline{\mathfrak{I}}\rvert)$
$DELF_\zeta$	27,8	86,6
$DELF_\beta$	11,5	84,8
$SIFT_\zeta$	23,5	86,7
$SIFT_\beta$	20,0	88,5

Tabelle 4: Mittelwert und Varianz für den Karlshorst-Datensatz $\mathcal{K}$ aus 1690 Paaren innerhalb der Klasse bzw. Szene, ALL_SCENES1690

Die Messwerte in Tabelle 4 zeigen, dass $DELF_\zeta$ auch in ALL_SCENES1690 die meisten *Inlier* in allen 1690 Bildpaaren gefunden hat. Zudem variiert die Anzahl an *Inliern* nun weniger für $DELF_\zeta$ als in Tabelle 3 mit $\sigma(|\mathfrak{I}_{DELF_\zeta}|) = 86,6$. Die geringste Anzahl an *Inliern* hat wie in Tabelle 3 das $DELF_\beta$ Verfahren. In der folgenden Illustration wurde für alle 130 Bildpaare die Anzahl an zugehörigen *Inlier* (y-Achse) aufgetragen.

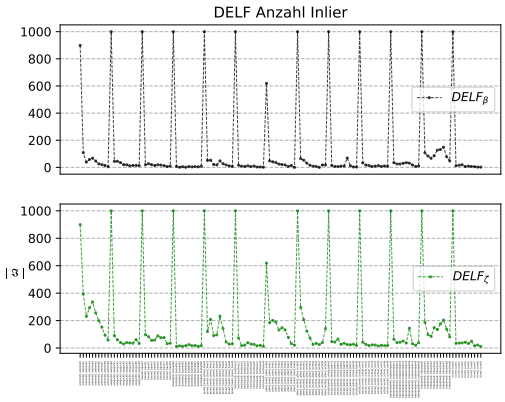


Abbildung 41: *DELF Inlier* für Karlshorst-Datensatz $\mathcal{KD}$, SAME_SCENE130

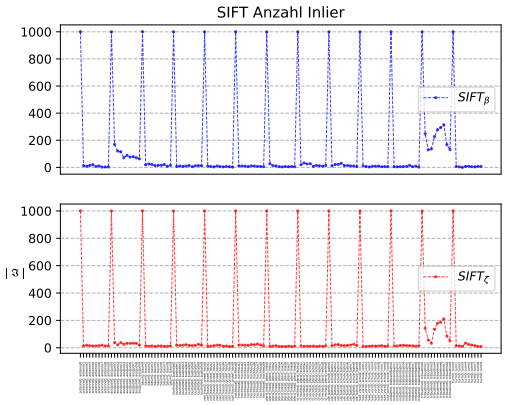


Abbildung 42: *SIFT Inlier* für Karlshorst-Datensatz $\mathcal{KD}$, SAME_SCENE130

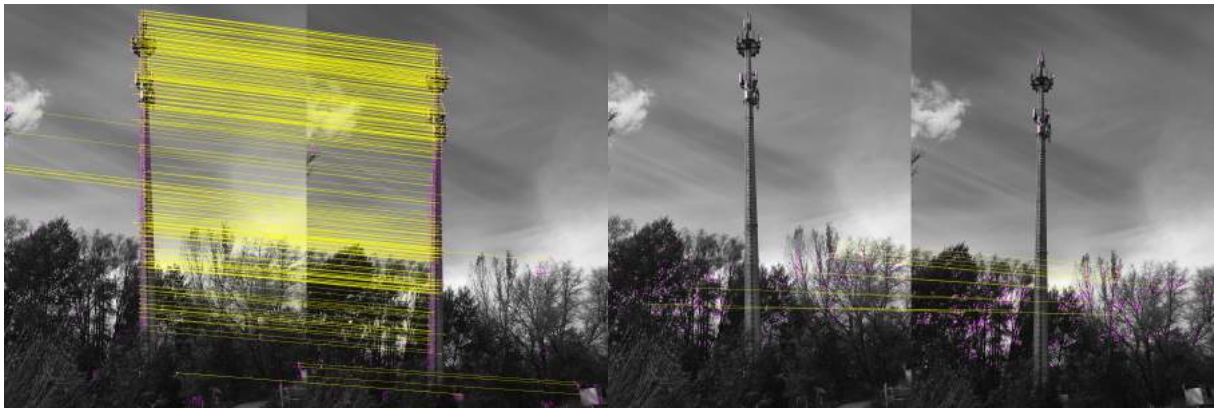
Abbildung 41 und 42 zeigt ein periodisches Verhalten. Im Abstand von 10 Bildpaaren werden circa 1000 (siehe Tabelle 1 und 2) *Inlier* gefunden. Dies liegt an der Tatsache, dass für diesen Fall ein *Matching* zwischen $Bild_0$ und $Bild_0$ also dem *Query*-Bild durchgeführt wird. Dafür werden so viele *Inlier* gefunden wie *Keypoints* im *Query*-Bild extrahiert wurden. Es gilt:

$$|\mathfrak{K}_{Bild_x}| = |\mathfrak{I}(Bild_0, Bild_x)|. \tag{102}$$

In Abbildung 41 zeigt der untere grüne Graph, das bereits in der Tabelle 3 visualisierte Ergebnis: Das $DELF_\zeta$ Verfahren findet die meisten *Inlier*. Dies ist an einem geringerem Abfallen nach dem periodischem maximalen Ausschlag für die Anzahl an *Inliern* beim *Matching* des *Query-Bildes* gegen sich selbst (Identität) erkennbar. Man kann an dieser Stelle bereits ein stabileres *Matching*-Verhalten für $DELF_\zeta$ implizieren.

5.3.6. Visualisierung

Inlier* und *Keypoints Für die Visualisierung wurden die *Keypoints* im Bild pink gezeichnet und die Verbindungs-Linien der *Inlier* werden durch gelbe Verbindungsgeraden gezeichnet. In der folgenden Abbildung wurde für das Bildpaar (`antenne0, antenne1`) aus Szene `antenne` das *Matching* visualisiert.



(a) $DELF_{\zeta}$, $|\mathfrak{I}| = 394$

(b) $SIFT_{\zeta}$, $|\mathfrak{I}| = 15$

(c) $DELF_{\beta}$, $|\mathfrak{I}| = 108$

(d) $SIFT_{\beta}$, $|\mathfrak{I}| = 13$

Abbildung 43: *Matching* Visualisierung mit den vier Verfahren zu Bild-Paar (`antenne0, antenne1`) aus $\mathcal{KD}$

Abbildung 43a mit $DELF_{\zeta}$ hat für das Bildpaar die höchste Anzahl an *Inliern*. Es wird deutlich, dass $SIFT_{\beta}$ und $SIFT_{\zeta}$ für diesen Bildpaar nur wenige *Inlier* findet.

Um das Verhalten des $DELF_\zeta$ Verfahrens besser zu verstehen, wird nun die Szene `pferd_links` $\mathcal{K}$ genauer analysiert. Die Aufnahme `pferd_links0` weißt einen maximalen Abstand für die restlichen Bilder in dieser Szene auf. Die Aufnahmen `pferd_links1 - pferd_links9` wurden aus einem geringerem Abstand zur Pferdestatue aufgenommen.

(a) `pferd_links0, pferd_links0` (b) `pferd_links0, pferd_links1`

(c) `pferd_links0, pferd_links2` (d) `pferd_links0, pferd_links3`

(e) `pferd_links0, pferd_links4` (f) `pferd_links0, pferd_links5`

(g) `pferd_links0, pferd_links6` (h) `pferd_links0, pferd_links7`

(i) `pferd_links0, pferd_links8` (j) `pferd_links0, pferd_links9`

Abbildung 44: $DELF_\zeta$ *Matching* 10 Bild-Paare zur Szene `pferd_links` aus $\mathcal{KD}$

Die Illustration 44 zeigt eine stabiles Verhalten des $DELF_\zeta$ Verfahrens. In Abbildung 44a ist gut erkennbar, dass die *Keypoints* relativ regelmäßig über das Bild `pferd_links0` verteilt sind. Auch mit Verringerung des Abstands zur Statue werden weiterhin *Inlier* gefunden. Um vergleichen zu

können, wird die gleiche Szenen für $SIFT_\zeta$ illustriert.

(a) `pferd_links0, pferd_links0` (b) `pferd_links0, pferd_links1`

(c) `pferd_links0, pferd_links2` (d) `pferd_links0, pferd_links3`

(e) `pferd_links0, pferd_links4` (f) `pferd_links0, pferd_links5`

(g) `pferd_links0, pferd_links6` (h) `pferd_links0, pferd_links7`

(i) `pferd_links0, pferd_links8` (j) `pferd_links0, pferd_links9`

Abbildung 45: $SIFT_\zeta$ *Matching* 10 Bild-Paare zur Szene pferd_links aus $\mathcal{K}$

Die Abbildung 45 zeigt ein instabiles Verhalten für $SIFT_\zeta$. Die *Inlier*, die gefunden werden, liegen meist nicht auf der Statue. In der Illustration 45a lässt sich erkennen, dass ein Hauptteil der *Keypoints* in den Bäumen liegt. In Abbildung 45j werden dann *Keypoints* in pferd_links0 in den Bäumen auf *Keypoints* in `pferd_links9` auf der Statue abgebildet. Die Ähnlichkeit der *Deskriptoren* in diesem Bereich führt damit zu sachlich falschen *Inliern*. Wie sich zeigt, lassen sich diese auch von dem genutzten RANSAC-Filter nicht vermeiden. Das menschliche Auge kann

an dieser Stelle die Schwäche des $SIFT_\zeta$ erkennen.

DELF **Attention Score** Für die Bilder `pferd_links9` und `antenne0` wurde der zugehörige *Attention Score* aufgetragen.

(a) *Attention* zu `pferd_links9` (b) `pferd_links9` mit *Keypoints*

Abbildung 46: *DELF Attention Score* für Bild `pferd_links9` aus $\mathcal{KD}$

In Abbildung 46a wurde der *Attention Score* visualisiert. Die hellen Bereiche sind die Teile des Bildes in denen besonders viele *Keypoints* extrahiert werden sollen. In Abbildung 46a hat der Himmel kaum helle Bereiche. Der Rand der Bäume (Kanten) bilden helle Linien mit hoher Aufmerksamkeit. Sonst ist die *Attention* im Bereich des Pferdes und des Reiters. Auch der Unterteil der Statue erhält eine nicht unerhebliche Wichtigkeit, jedoch nicht so hoch wie der obere Teil des Reiters und Pferdes.

(a) *Attention* zu `antenne0` (b) `antenne0` mit *Keypoints*

Abbildung 47: *DELF Attention Score* für Bild `antenne0` aus $\mathcal{KD}$

Analog wurde das Bild `antenne0` analysiert. Die sich vertikal entwickelnde Antenne hat im Bild die größte Wichtigkeit. Das Neuronale Netz hat gelernt, dass solche Strukturen wichtig sind. Der Himmel und auch der Großteil der Bäume sowie Bewuchs erhält keine große *Attention*. Abbildung 47b zeigt das *Query*-Bild der Szene antenne mit den 1000 extrahierten *Keypoints* in pink. Es lässt sich klar erkennen, dass gerade in Bereichen mit hoher Aufmerksamkeit auch viele *Keypoints* extrahiert wurden.

5.3.7. Gesamte empirische Messungen in $\mathcal{KD}$

Für die Analyse wurde der Schwellenwert für die *Inlier* von $T = 30$ festgelegt.

$$\begin{cases} tp, & (|\mathfrak{I}| \geq 30) \wedge GleicheSzene \\ fn, & (|\mathfrak{I}| < 30) \wedge GleicheSzene \\ fp, & (|\mathfrak{I}| \geq 30) \wedge \neg GleicheSzene \end{cases} \tag{103}$$

In der folgenden Tabelle werden die Messwerte für den Karlshorst-Datensatz dargestellt. Die 13 *Query*-Bilder werden innerhalb der Szene - 10 Bilder gematcht. Die Messwerte sind somit aus 130 *Matching*-Paaren erhoben worden.

Verfahren	tp	fp	fn	$precision$	$recall$	F_1	$\lvert\overline{\mathfrak{I}}\rvert$	$\overline{d}$
$DELF_\zeta$	92	0	38	1	0,708	0,414	165,7	0,560
$DELF_\beta$	45	0	85	1	0,346	0,257	119,0	0,425
$SIFT_\zeta$	29	0	101	1	0,223	0,182	121,7	252,1
$SIFT_\beta$	33	0	97	1	0,254	0,202	129,1	164,3

Tabelle 5: Empirische Messung für den Karlshorst-Datensatz $\mathcal{KD}$ aus 130 Paaren innerhalb der Klasse bzw. Szene, SAME_SCENE130

In der folgenden Tabelle werden die Messwerte für den Karlshorst-Datensatz dargestellt. Die 13 *Query*-Bilder werden gegen den gesamten Datensatz - 130 Bilder gematcht. Die Messwerte sind somit aus 1690 *Matching*-Paaren erhoben worden.

Verfahren	tp	fp	fn	$precision$	$recall$	F_1	$\lvert\overline{\mathfrak{I}}\rvert$	$\overline{d}$
$DELF_\zeta$	95	11	35	0,896	0,730	0,402	27,8	0,843
$DELF_\beta$	45	3	85	0,9375	0,346	0,253	11,5	0,485
$SIFT_\zeta$	26	48	104	0,351	0,2	0,127	23,5	297.7
$SIFT_\beta$	32	100	98	0,242	0,246	0,122	20,0	198,8

Tabelle 6: Empirische Messung für den Karlshorst-Datensatz $\mathcal{KD}$ aus 1690 Paaren, ALL_SCENES1690

Variation von T In der folgenden Abbildung wurde der Parameter T der Anzahl an *Inlier* (ab wann es als *True Positiv* gezählt wird) zwischen $T_i = \{0, 1, \dots 100\}$ variiert. Ein *True Positive* wird beispielsweise festgestellt, wenn die Anzahl an *Inliern* größer-gleich einem *Threshold* T_i ist und es sich beide Bilder des Bildpaares in der selben Szene befinden.

$$\begin{cases} tp, & (|\mathfrak{I}| \geq T_i) \wedge GleicheSzene \\ fn, & (|\mathfrak{I}| < T_i) \wedge GleicheSzene \\ fp, & (|\mathfrak{I}| \geq T_i) \wedge \neg GleicheSzene \end{cases} \tag{104}$$

Die Gleichung 104 beschreibt dabei wie die drei Werte für die Bewertung der Verfahren bestimmt werden. Dabei sei $precision = f(recall)$ die *Precision* eine Funktion des *Recalls*.

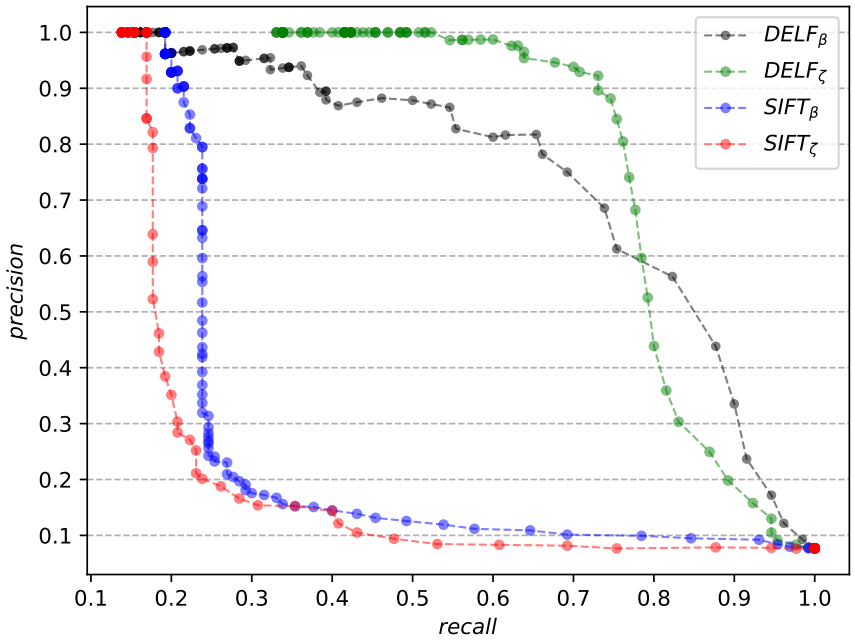


Abbildung 48: *Recall* gegen *Precision* für den Karlshorst-Datensatz $\mathcal{KD}$, ALL_SCENES1690

Im obigen Plot (Abbildung 48) beschreibt der schwarze Graph das Verhalten von $DELF_\beta$. Bei einem minimalen *Recall*-Wert von 0,138 liegt die *Precision* bei 1,0. Bei einem *Recall* von 1,0 liegt die *Precision* hingegen bei 0,077. Das grün gezeichnete $DELF_\zeta$ Verfahren hat einen minimalen Wert des *Recalls* von 0,331 und bildet dabei auf eine *Precision* von 1,0 ab. Bei einem *Recall* von 1,0 erreicht die *Precision* ebenfalls 0,077. Das $SIFT_\beta$ Verfahren wird in der x-y-Ebene als blaue Funktionsgraph visualisiert. Der minimalen *Recall*-Wert 0,192 liegt die *Precision* bei 1,0. Bei einem maximalem Recall von 1,0 wird eine Precision von 0,077 gemessen. Der vierte rote Graph - $SIFT_\zeta$ erreicht einem *Recall* 0,1384 einen Funktionswert (*Precision*) von 1,0 zu und besitzt eine minimale *Precision* von 0,077 bei entsprechendem *Recall* mit 1,0. Als Zusammenfassung lässt sich bezüglich der *Precision* folgende Reihenfolge

$$prec(DELF_\beta)_{max} = prec(DELF_\zeta)_{max} = prec(SIFT_\beta)_{max} = prec(SIFT_\zeta)_{max} = 1 \tag{105}$$
$$prec(DELF_\beta)_{min} = prec(DELF_\zeta)_{min} = prec(SIFT_\beta)_{min} = prec(SIFT_\zeta)_{min} = 0,077 \tag{106}$$

festhalten. Bezüglich des Recalls können folgende zusammenfassende Aussagen hinsichtlich lokaler Maxima und Minima getroffen werden:

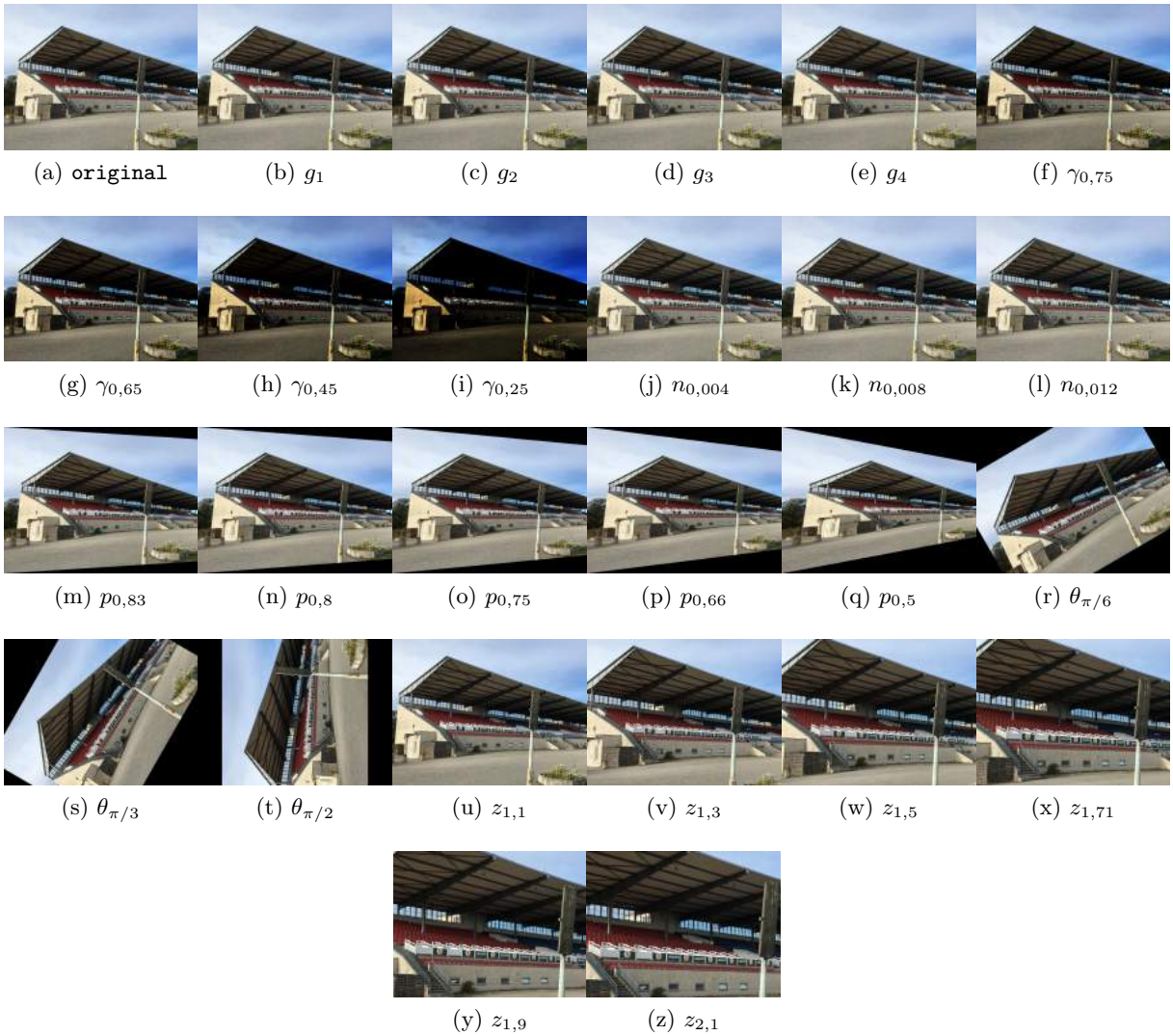
$$rec(DELF_\beta)_{max} = rec(DELF_\zeta)_{max} = rec(SIFT_\beta)_{max} = rec(SIFT_\zeta)_{max} = 1 \tag{107}$$
$$rec(DELF_\beta)_{min} = 0,138 = rec(SIFT_\zeta)_{min} < rec(SIFT_\beta)_{min} < rec(DELF_\zeta)_{min} \tag{108}$$

Die letzte Gleichung 108 verdeutlicht das visuell sichtbare Verhalten von $DELF_\zeta$. Der minimale *Recall*-Wert von 0,331 wird nicht unterschritten. Der grüne Graph in Abbildung 48 beginnt erst bei höheren x-Werten(weiter rechts). Allgemein ist ein hoher *Recall* und eine hohe *Precision* eine positive Eigenschaft eines Verfahrens. Damit lässt sich festhalten, dass $DELF_\zeta)_{min}$ bei der Variation des Parameters T am besten abschneidet. Eine Erhöhung des *Recalls* ist proportional zur Verminderung des Parameters der *Inlier*-Wertung T_i. Für $T_i = 0$ liegt der *Recall* bei 1,0, für $T_i = 100$ liegt dieser bei kleinen Werten $\leq 0,331$. Mit Hinsicht auf die *Precision* lässt sich feststellen, dass eine Erhöhung der *Inlier*-Wertung T_i auch eine Erhöhung der *Precision* folgt. Für die $SIFT$-Verfahren ist ersichtlich, dass bei der Zunahme des *Recalls* die *Precision* deutlich

schneller sinkt als bei den beiden $DELF$-Verfahren. Man könnte sagen, dass $SIFT_\beta$ und $SIFT_\zeta$ sensibler auf eine Variation von T_i reagieren.

5.4. Karlshorst-Transformations-Datensatz

Anlehnend an den *Generated-Matching*-Datensatz (erzeugter Datensatz) der Universität Freiburg [61] wurde der Karlshorst-Transformations-Datensatz erstellt. Für die Analyse war es wichtig, dass es sich um einen reinen *Outdoor*-Datensatz handelt, was für den *Generated-Matching*-Datensatz nicht gegeben war. Auf die ersten Bilder aus den 13 Szene des Karlshorst-Datensatz $\mathcal{KD}$ werden nun sechs Bild-Transformationen (mit insgesamt 26 Variationen) angewendet. In Abbildung 49 ist ein Teil des Datensatzes visualisiert. Damit ergeben sich $13 * 26 = 338$ Bilder. Der Karlshorst-Transformations-Datensatz wird im Folgenden mit $\mathcal{KTD}$ bezeichnet. Die sechs Bild-Transformationen werden nun genauer beschrieben: Für die folgende Evaluation beschreibt g eine Glättung des Bildes. Der Index beschreibt die Stärke der Glättung mittels Gauß-Filter. Das Originalbild wird vier mal geglättet $g = \{g_1, g_2, g_3, g_4\}$. Dafür wurde ein 5×5 Gauß-Filter (Abschnitt 2.1.5) genutzt. Des Weiteren wurde eine Serie aus vier Bildern mit sinkender Gamma-Korrektur erzeugt (Abschnitt 2.1.6), $\gamma = \{\gamma_{0,75}, \gamma_{0,65}, \gamma_{0,45}, \gamma_{0,25}\}$. Das bedeutet, dass die Bilder schrittweise dunkler werden. Um das Verhalten der Verfahren bei Rauschen zu analysieren, wurden drei Bilder mit *Salt* und *Pepper* Rauschen (Abschnitt 2.1.7) verrauscht.



(a) `original` (b) g_1 (c) g_2 (d) g_3 (e) g_4 (f) $\gamma_{0,75}$

(g) $\gamma_{0,65}$ (h) $\gamma_{0,45}$ (i) $\gamma_{0,25}$ (j) $n_{0,004}$ (k) $n_{0,008}$ (l) $n_{0,012}$

(m) $p_{0,83}$ (n) $p_{0,8}$ (o) $p_{0,75}$ (p) $p_{0,66}$ (q) $p_{0,5}$ (r) $\theta_{\pi/6}$

(s) $\theta_{\pi/3}$ (t) $\theta_{\pi/2}$ (u) $z_{1,1}$ (v) $z_{1,3}$ (w) $z_{1,5}$ (x) $z_{1,71}$

(y) $z_{1,9}$ (z) $z_{2,1}$

Abbildung 49: Teil des Karlshorst-Transformations-Datensatz $\mathcal{KTD}$, 26 Bildern für `tribuehne0`

Dabei wurde mit einer relativen Anzahl an defekten Pixeln gearbeitet, $n = \{n_{0,004}, n_{0,008}, n_{0,012}\}$. Dass bedeutet, dass im letzten Bild dieser Reihe $n_{0,012}$ sind $1,2\,\%$ der Pixel *Salt* oder *Pep-*

per. Der Datensatz enthält fünf perspektivische Transformation. In jedem Schritt nimmt der vertikale relative Abstand der rechten oberen und rechten unteren Ecke des Bildes ab, $p = \{p_{0,83}, p_{0,8}, p_{0,75}, p_{0,66}, p_{0,5}\}$. Es wurden drei Rotationen ausgeführt, $\theta = \{\theta_{\pi/6}, \theta_{\pi/3}, \theta_{\pi/2}\}$. Es wurden sechs Digitalzooms erzeugt, zunehmend $z = \{z_{1,1}, z_{1,3}, z_{1,5}, z_{1,71}, z_{1,9}, z_{2,1}\}$. Das bedeutet $z_{1,1}$ wurde um 10 % gezoomt, $z_{2,1}$ um 110 %.

5.4.1. *Matching* Vorgehen

Der Datensatz enthält 338 Bilder, resultierende aus 13 *Query*-Bildern (unverändert) und 25 Bildtransformationen. Das *Query*-Bild und die 25 Transformationen ergeben somit jeweils 26 Bilder (26 *13 = 338). Um eine kürzere Schreibweise für die folgende Analyse zu finden wird nun die Vorgehensweisen beschrieben. Im folgenden seien alle 25 Bildtransformationen sowie das *Query*-Bilde definiert als $\mathcal{T} = \{I_0, \cdots, I_{25}\}$. Die gesamte Menge an *Inliern* aller Transformationen sei im folgendem $\mathfrak{I}_\mathcal{T}$. Dabei sei I_0 das *Query*-Bild.

Algorithmus 6 query_Transformationen($\mathcal{T}$)

1: **for** `i = 0; i < 26; i++` **do**
2: bestimme Menge der *Inlier* $\mathfrak{I}_i$ für Bildpaar $(I_0, \mathcal{T}(i))$
3: füge Menge der *Inlier* $\mathfrak{I}_i$ zu $\mathfrak{I}_\mathcal{T}$ hinzu
4: **return** $\mathfrak{I}_\mathcal{T}$

Der Vorgang im Algorithmus 6 wird nun für alle 13 *Query*-Bilder durchgeführt. Für die folgende Evaluation wird diese Form des Durchsuchen des Datensatzes als TRANSFORMATION338 bezeichnet. Mit dieser Methode werden somit 338 Bild-Paare auf Ähnlichkeiten überprüft.

5.4.2. Distanzen unter Transformationen

Die sechs verschiedenen Transformationen wurden in unterschiedlicher Stärke bzw. Ausmaß (*Magnitude*) angewendet. Im Folgenden wird pro Transformation die durchschnittliche euklidische Distanz aller *Inlier* festgestellt. Es gibt 13 Query-Bilder, die zugehörige Stärke bezeichnet mit $\mathcal{M}$ der Transformation $\mathcal{T}$ jeweils einmal für das *Query*-Bild angewendet wurde, ergeben sich $n = 13$ Distanzen deren Mittelwert numerisch bestimmt wird.

$$\overline{d}(\mathcal{T}_\mathcal{M}) = \frac{1}{n} \sum_{i=1}^{n} d(\mathcal{T}_{\mathcal{M}i}) \tag{109}$$

Die Werte von $\overline{d}(\mathcal{T}_\mathcal{M})$ werden im Folgendem geplottet. Die Stärke der $\mathcal{M}$ ist dabei für die x-Achse aufgetragen. Über die Stärken der Transformationen $\mathcal{M}$ von ($\mathcal{T}$ lässt sich dann noch über $\mathcal{M}$ der Mittelwert für die Distanzen bilden $\overline{d}(\mathcal{T}_\mathcal{M})$.

$$\overline{d}(\mathcal{T}) = \frac{1}{|M|} \sum_{i=1}^{|M|} \overline{d}(\mathcal{T}_i) \tag{110}$$

Distanzen für Glättung Auf die *Query*-Bilder wurden vier unterschiedliche Glättungen ausgeführt. Die Glättung nimmt in positiver x-Richtung zu.

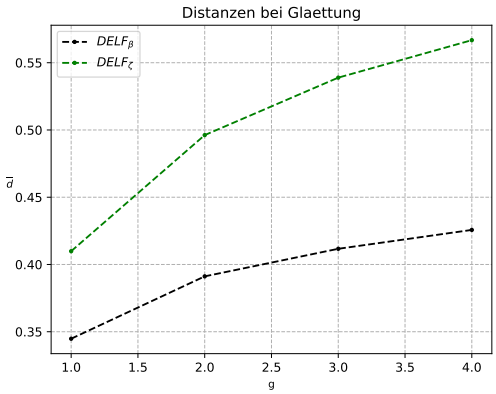


Abbildung 50: *DELF* Distanzen unter Glättung für $\mathcal{KTD}$, TRANSFORMATION338

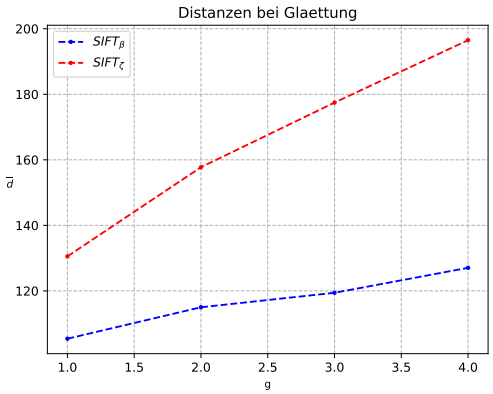


Abbildung 51: *SIFT* Distanzen unter Glättung für $\mathcal{KTD}$,TRANSFORMATION338

Die obige linke Abbildung 50 zeigt das Verhalten für das *DELF* Verfahren. Es ist erkennbar, dass $DELF_\zeta$ (grüne Graph) eine höhere durchschnittliche Distanz als $DELF_\zeta$ annimmt. Dies ist zu erwarten gewesen, da sich bereits in Abbildung 39 eine solcher Trend in $\mathcal{KD}$ abzeichnete. $DELF_\beta$ nimmt eine geringere Distanz der *Feature-Deskriptoren* an. Die obere rechte Abbildung 51 illustriert analog das Verhalten für SIFT. Es zeigt sich ein ähnliches Verhalten. Der rote Graph - $SIFT_\zeta$ liegt für alle vier Messungen oberhalb des blauen Graph-$SIFT_\beta$. Alle vier Graphen sind für die diskrete Messung monoton steigend. Allgemein lässt sich feststellen, dass eine Glättung eine Vergrößerung der durchschnittlichen euklidischen Distanz $\overline{d}(g)$ zur Folge hat.

Distanzen für Gamma-Korrektur Mit dem TRANSFORMATION338 Ansatz wurde das Verhalten unter Gamma-Korrektur untersucht. Der Gamma-Wert nimmt in positiver x-Richtung zu. Dabei sei erwähnt, dass eine Erhöhung von γ, das Bild aufhellt.

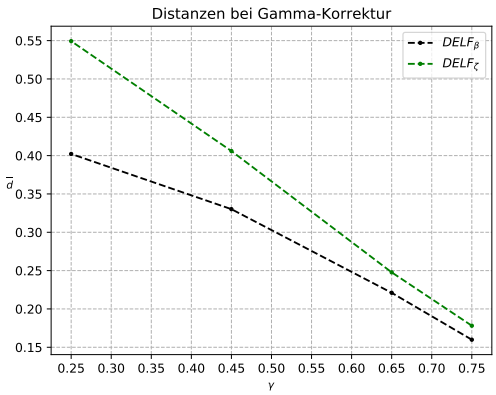


Abbildung 52: *DELF* Distanzen unter Gamma-Korrektur für $\mathcal{KTD}$, TRANSFORMATION338

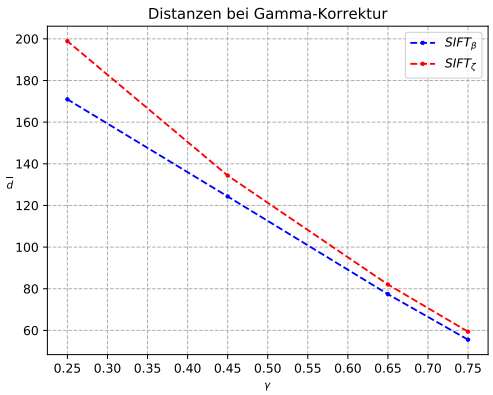


Abbildung 53: *SIFT* Distanzen unter Gamma-Korrektur für $\mathcal{KTD}$, TRANSFORMATION338

Dadurch erklärt sich auch das Abfallen der Messwerte für $DELF_\zeta$ (grüne Graph) und $DELF_\beta$ (schwarzer Graph). Für *SIFT*, siehe Abbildung 53, ergibt das gleiche monoton fallende Verhalten. Es lässt sich grundsätzlich festhalten, dass eine Verringerung von γ, was einer Verdunklung des Bildes entspricht, die euklidischen Distanz $\overline{d}(\gamma)$ zunimmt.

Distanzen für Bild-Rauschen Für das Bildrauschen wurden drei unterschiedliche Stärken des Rauschens $n(x, y)$ angewendet. Der relative Anteil defekter Pixel nimmt in positiver x-Richtung zu.

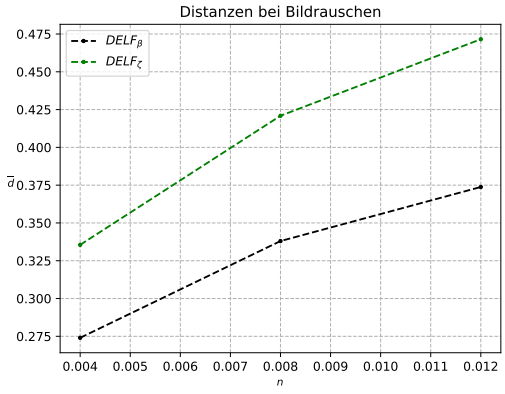


Abbildung 54: $DELF$ Distanzen für $\mathcal{KTD}$, TRANSFORMATION338

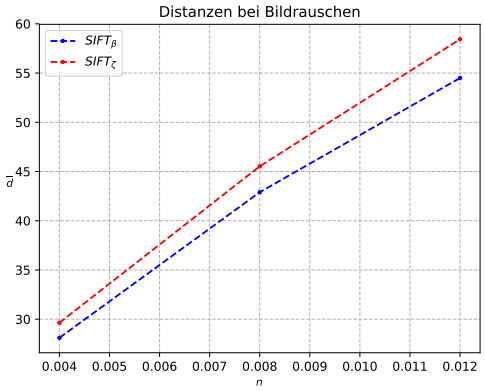


Abbildung 55: $SIFT$ Distanzen für $\mathcal{KTD}$, TRANSFORMATION338

Die obere linke Abbildung 54 zeigt ein monoton wachsendes Verhalten für $DELF_\zeta$ und $DELF_\beta$. Auch der rote Graph ($SIFT_\zeta$) und der blaue ($SIFT_\beta$) in Abbildung 55 zeigt, dass bei Erhöhung des Rauschens $\overline{d}(n)$ zunimmt.

Distanzen für Perspektivische Transformationen Der Datensatz enthält fünf perspektivische Transformation. In jedem Schritt nimmt der vertikale relative Abstand der rechten oberen und rechten unteren Ecke des Bildes ab, $p = \{p_{0,83}, p_{0,8}, p_{0,75}, p_{0,66}, p_{0,5}\}$.

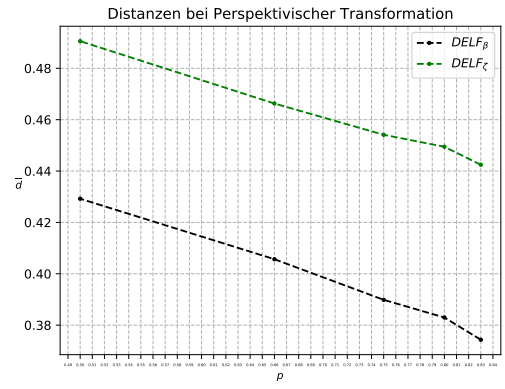


Abbildung 56: $DELF$ Distanzen für $\mathcal{KTD}$, TRANSFORMATION338

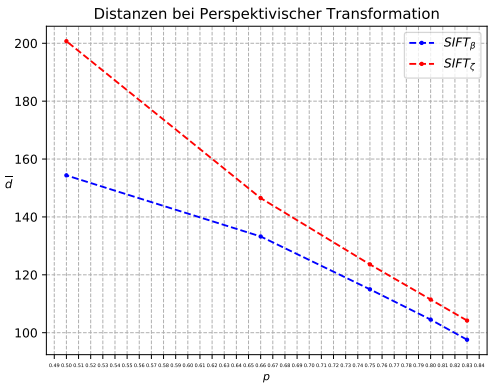


Abbildung 57: $SIFT$ Distanzen für $\mathcal{KTD}$, TRANSFORMATION338

Der vertikale relative Abstand nimmt in positiver x-Richtung zu. Eine kleinere relative Distanz entspricht dabei einer stärkeren perspektivischen Transformation als eine größere. Somit erklärt sich auch das monoton fallende Verhalten unter Perspektivische Transformationen p für $DELF$ und $SIFT$. Allgemein lässt sich festhalten, dass sich eine perspektivische Transformation negativ auf die durchschnittlichen Distanzen $\overline{d}(p)$ der *Deskriptoren* auswirkt.

Distanzen für Rotation In den folgenden Darstellungen wurde der Winkel θ auf der x-Achse aufgetragen. Messungen wurden dabei für 30°, 60° und 90° durchgeführt.

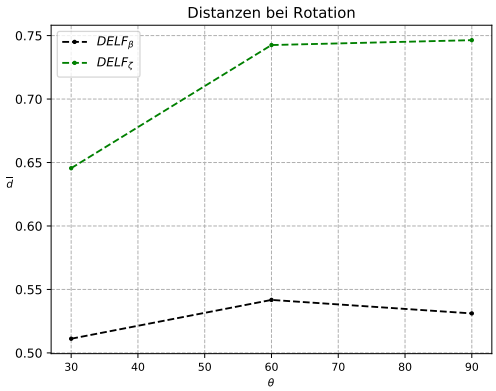


Abbildung 58: $DELF$ Distanzen für $\mathcal{KTD}$, TRANSFORMATION338

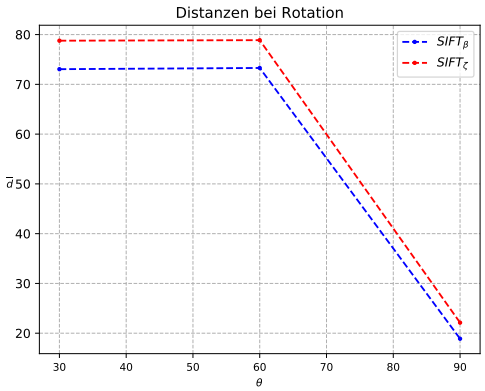


Abbildung 59: $SIFT$ Distanzen für $\mathcal{KTD}$, TRANSFORMATION338

In Abbildung 58 zeigt sich für $DELF_\beta$ kein monotones Verhalten. Den maximalen euklidischen Abstand $\overline{d}(r)$ nimmt $DELF_\beta$ bei 60 °an und fällt bei 90 °wieder ab. Für $SIFT$ zeigt sich für beide Verfahren ein fallendes Verhalten. Es lässt sich daher festhalten, dass sich für $DELF$ eine Rotation eher negative auf die Distanzen $\overline{d}(p)$ auswirkt, für $SIFT$ hingegen lässt sich das Gegenteil feststellen.

Distanzen für Zoom Es wurden sechs zunehmende Digitalzooms angewendet, diese werden auf der x-Achse aufgetragen.

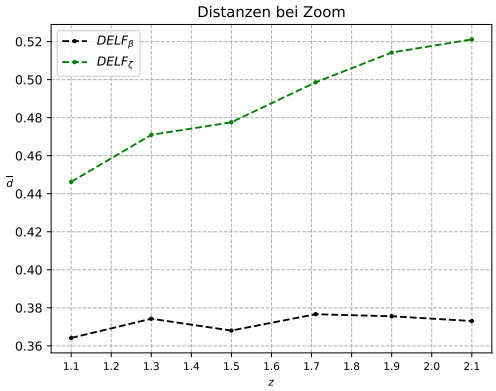


Abbildung 60: $DELF$ Distanzen für $\mathcal{KTD}$, TRANSFORMATION338

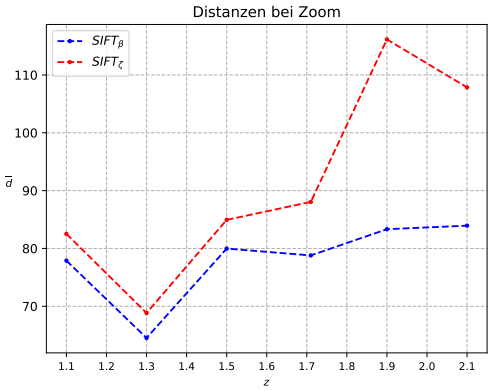


Abbildung 61: $SIFT$ Distanzen für $\mathcal{KTD}$, TRANSFORMATION338

Die linke Abbildung 60 zeigt für $DELF_\zeta$ ein monoton wachsendes Verhalten. $DELF_\beta$ hingegen bleibt fast konstant und die durchschnittliche Distanz $\overline{d}(z)$ erhöht sich nur wenig. Die obige rechte Abbildung 61 zeigt eine global zunehmende Distanz bei Zoom $\overline{d}(z)$ für $SIFT_\beta$ und $SIFT_\zeta$, wobei $SIFT_\beta$ nicht so stark wächst wie $SIFT_\zeta$.

5.4.3. *Inlier* unter Transformationen

Es wird weiter mit den 338 Bild-Paaren gearbeitet. Für die folgende Auswertung wurde der bereits beschriebene Ansatz TRANSFORMATION338 genutzt. Es gibt 13 *Query*-Bilder, die zugehörige Stärke, bezeichnet mit $\mathcal{M}$, der Transformation $\mathcal{T}$ jeweils einmal für das *Query*-Bild angewendet wurde, ergeben sich $n = 13$ Anzahl an *Inliern* deren Mittelwert numerisch bestimmt

wird.

$$|\overline{\mathfrak{I}}(\mathcal{T}_{\mathcal{M}})| = \frac{1}{n}\sum_{i=1}^{n}|\mathfrak{I}(\mathcal{T}_{\mathcal{M}i})| \tag{111}$$

Über die Stärken der Transformationen $\mathcal{M}$ von $\mathcal{T}$ lässt sich dann noch über $\mathcal{M}$ der Mittelwert für die Anzahl der *Inlier* bilden $|\overline{\mathfrak{I}}(\mathcal{T})|$.

$$|\overline{\mathfrak{I}}(\mathcal{T})| = \frac{1}{|M|}\sum_{i=1}^{|M|}|\overline{\mathfrak{I}}(\mathcal{T}_i)| \tag{112}$$

***Inlier* unter Glättung** Auf die *Query*-Bilder wurden vier unterschiedliche Glättungen ausgeführt. Nachdem das Verhalten der euklidischen Distanz der *Deskriptoren* untersucht wurde, soll dies nun für die Anzahl der *Inlier* erfolgen. Die Glättung nimmt in positiver x-Richtung zu.

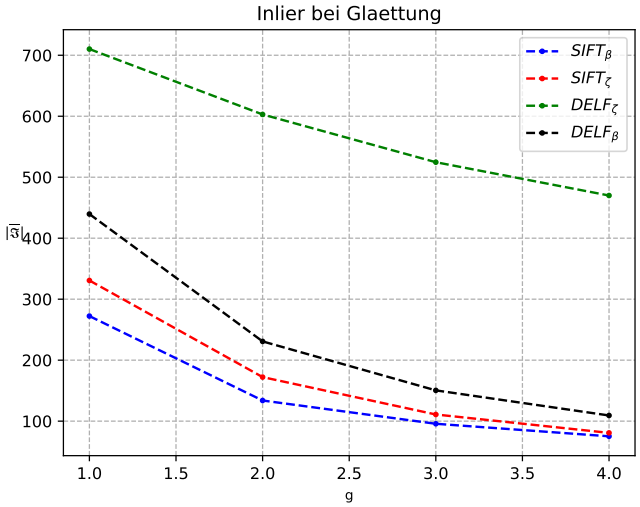


Abbildung 62: *Inlier* unter Glättung für $\mathcal{KTD}$, TRANSFORMATION338

Für alle vier Graphen in Abbildung 62 ist ein monoton fallendes Verhalten zu sehen. Dies ist konsistent mit der Analyse hinsichtlich der Distanz (siehe Abbildung 50 und 51). Ein monoton wachsendes Verhalten von $\overline{d}(g)$ steht in Beziehung zu einem monoton fallendem Verhalten von $|\overline{\mathfrak{I}}(g)|$. Zumindest für diese Transformations-Datensatz lässt sich deswegen eine Proportionalität

$$|\overline{\mathfrak{I}}(g)| \sim \overline{d}(g) \tag{113}$$

feststellen. Es lässt sich allgemein festhalten, dass $DELF_\zeta$ die meisten *Inlier* sammelt und $DELF_\beta$ die zweit-misten. Gefolgt von $SIFT_\zeta$ und danach $SIFT_\beta$, welche aber immer unterhalb der Werte von $DELF$ blieben.

***Inlier* unter Gamma-Korrektur** Pro *Query*-Bild wurde, wie bereits erwähnt, eine Serie aus vier Bildern mit sinkender Gamma-Korrektur erzeugt, $\gamma = \{\gamma_{0,75}, \gamma_{0,65}, \gamma_{0,45}, \gamma_{0,25}\}$. Das bedeutet, dass die Bilder schrittweise dunkler werden. Die Gamma-Korrektur nimmt in positiver x-Richtung zu, was bedeutet, dass die Bilder in dieser Darstellung schrittweise heller werden.

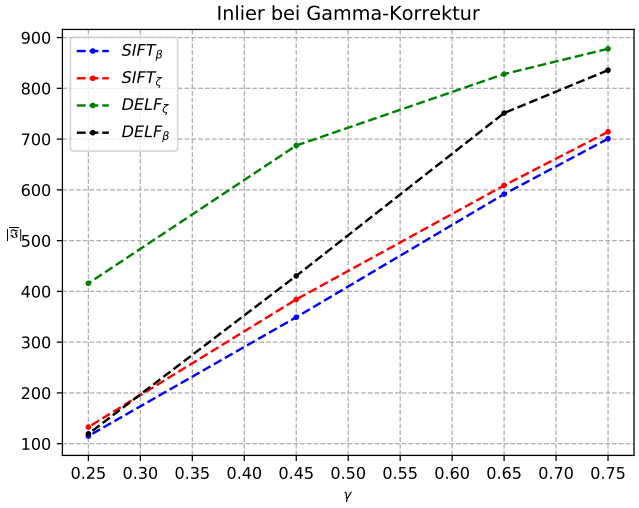


Abbildung 63: Inlier unter γ-Korrektur für $\mathcal{KTD}$, TRANSFORMATION338

Die obige Abbildung 63 zeigt ein monoton wachendes Verhalten für alle Verfahren. Interessant ist das Schneiden der Graphen $DELF_\zeta$ und $DELF_\beta$ im Bereich höherer Helligkeit. Wie schon bei den *Inliern* unter Glättung zeigt sich ein Umgekehrtes Verhalten zwischen $\overline{d}(\gamma)$ (monoton fallend) und $|\overline{\mathfrak{I}}(\gamma)|$.

***Inlier* unter Rauschen** Im folgendem wird das Verhalten der *Inlier* beim Erhöhen der defekten Pixel in Bilder untersucht.

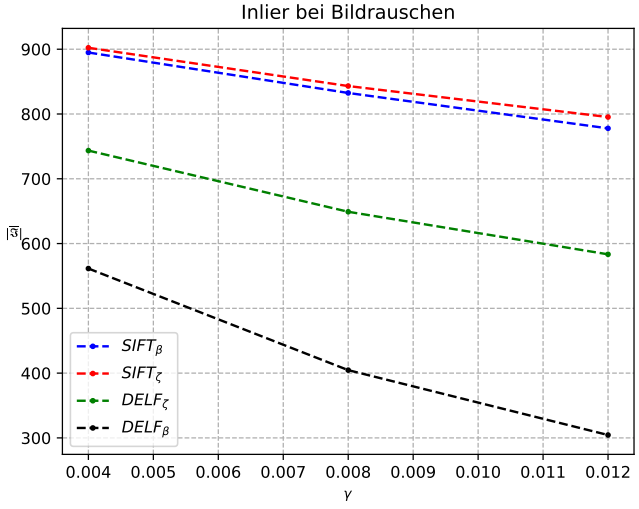


Abbildung 64: Inlier unter Rauschen für $\mathcal{KTD}$, TRANSFORMATION338

Es zeigt sich für alle Verfahren ein monoton fallendes Verhalten. Die meisten *Inlier* findet $SIFT_\zeta$ gefolgt von $SIFT_\beta$. Bei Bildrauschen finden beide $DELF$ Verfahren deutlich weniger *Inlier*.

***Inlier* unter Perspektivischer Transformation** Für die Perspektivischen Transformationen $p = \{p_{0,83}, p_{0,8}, p_{0,75}, p_{0,66}, p_{0,5}\}$ ergaben sich die folgenden Messungen. Dabei sei erwähnt, dass eine Erhöhung der Werte der x-Achse einer Verringerung der Stärke der Transformation entspricht.

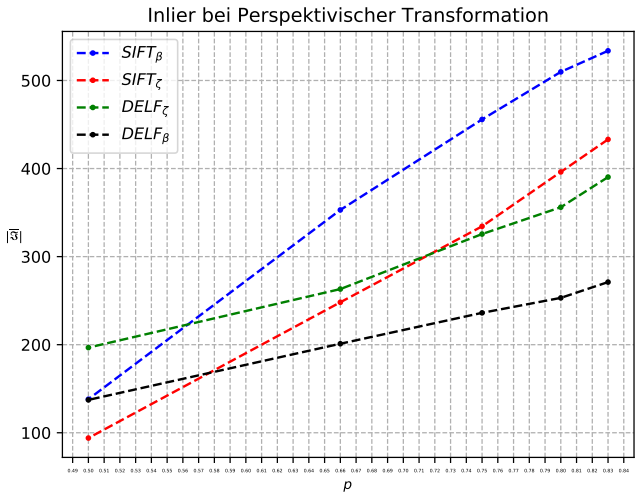


Abbildung 65: Inlier unter Persp. Transf. für $\mathcal{KTD}$, TRANSFORMATION338

Unter Perspektivischer Transformation kann $SIFT_\beta$ (blauer Graph in Abbildung 65) die meisten *Inlier* finden, danach folgt $DELF_\zeta$. $DELF_\beta$ findet am wenigsten *Inlier* unter den angewendeten Perspektivischer Transformationen $p_\mathcal{M}$. Für alle Graphen zeigt sich ein monoton wachsendes Verhalten.

***Inlier* unter Rotation** Es wurden drei Rotationen im Abstand von 30 °ausgeführt.

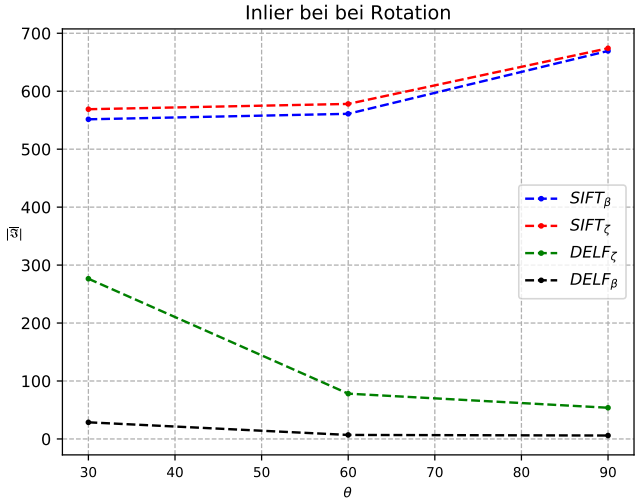


Abbildung 66: Inlier unter Rotation für $\mathcal{KTD}$, TRANSFORMATION338

Die obige Illustration 66 zeigt, dass sich einem zunehmende Rotation positiv auf $SIFT_\zeta$ $SIFT_\beta$ auswirkt. $DELF$ hingegen zeigt ein monoton fallendes Verhalten und kann bei $\theta = \pi/2$ fast keine *Inlier* mehr finden. Die Rotation ist wohl die Transformation in der das Verhalten von $DELF$ zum einen sehr schlecht ist und zu anderem sehr stark von $SIFT$ abweicht.

***Inlier* unter Zoom** Für die sechs angewendeten Zooms ergaben sich die folgenden Anzahlen der *Inlier*.

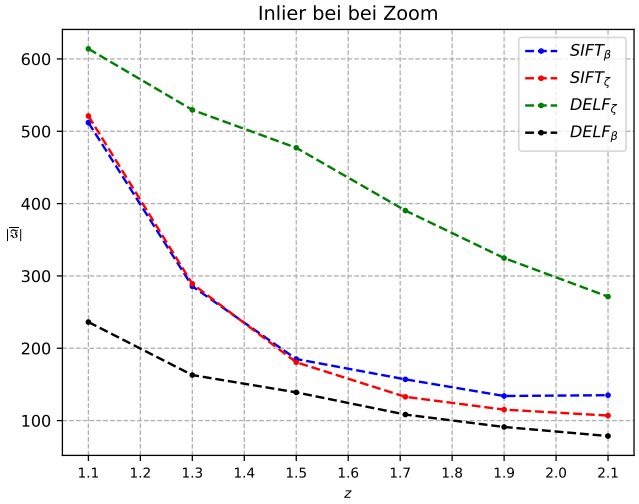


Abbildung 67: Inlier unter Zoom für $\mathcal{KTD}$, TRANSFORMATION338

Die Abbildung 67 zeigt einen monoton fallenden Verlauf für alle Verfahren. Die meisten *Inlier* kann dabei $DELF_\zeta$ und die wenigsten $DELF_\beta$ erzielen. $SIFT_\beta$ erzielt leicht mehr *Inlier* als $SIFT_\zeta$.

5.4.4. Distanzen gegen Anzahl *Inlier* nach Transformation für $DELF$

Im Folgendem wird die Anzahl der *Inlier* gegen die gemessenen Distanzen in $\mathcal{KTD}$ für $DELF$ aufgetragen. Dabei werden die sechs Transformationen gemittelt und beschrieben.

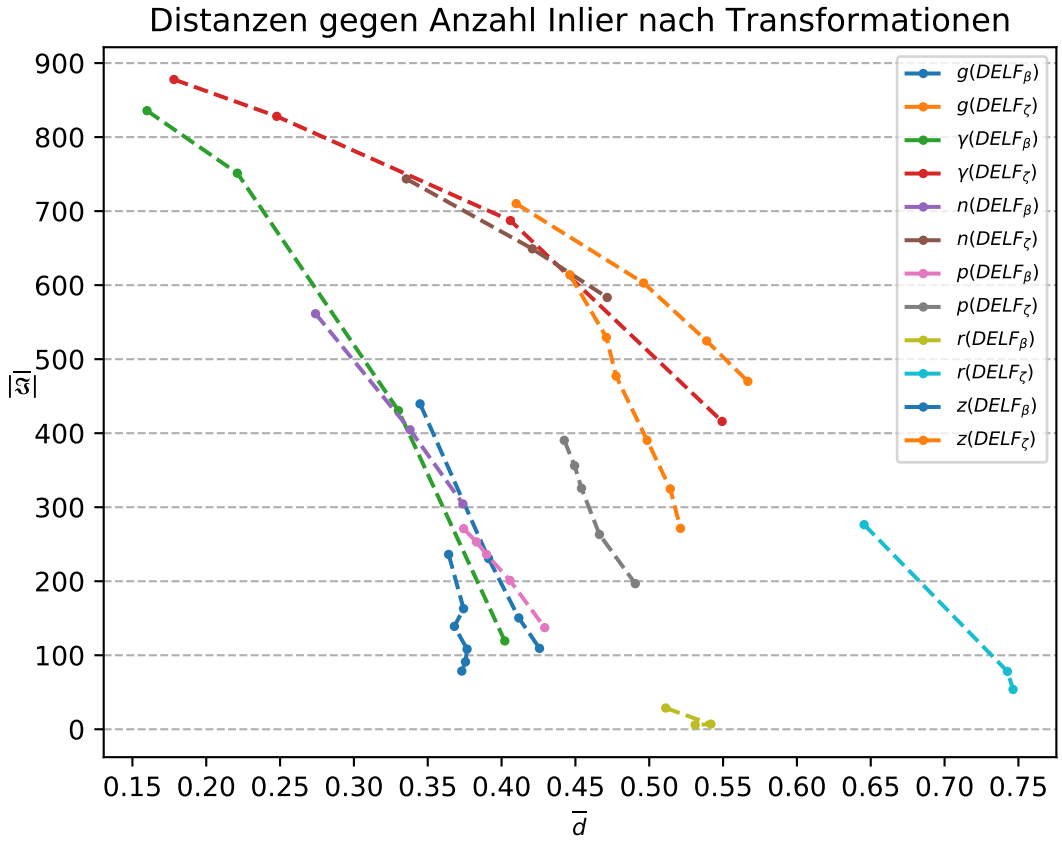


Abbildung 68: Durchschnittliche Distanzen gegen durchschnittliche Anzahl *Inlier* nach Transformationen für $DELF$ $\mathcal{KTD}$, TRANSFORMATION338

Die Abbildung 68 visualisiert die Messungen der *Inlier* in Bezug zur zugehörigen Distanz bei Veränderung der Stärke der Transformation $\mathcal{M}$. Es ist eine messbare Proportionalität

$$|\overline{\mathfrak{I}}(\mathcal{T}_\mathcal{M})| \sim \overline{d}(\mathcal{T}_\mathcal{M}) \tag{114}$$

erkennbar. Allgemein lässt sich festhalten, dass mit Zunahme der Stärke der Transformation, die durchschnittlichen Distanzen $\overline{d}(\mathcal{T}_\mathcal{M})$ steigen und sich die Anzahl der *Inlier* $|\overline{\mathfrak{I}}(\mathcal{T}_\mathcal{M})|$ dadurch auch verringert. In Abbildung 68 zeigt sich bereits deutlich, dass die Rotation für $DELF_\zeta$ und $DELF_\beta$ eine enorme Erhöhung der Distanzen erzeugt sowie auch die Anzahl an *Inliern* extrem reduziert. Die höchste durchschnittliche Distanz der *Deskriptoren* bei Rotation erreicht $DELF_\zeta$.

5.4.5. Mittelwerte pro Transformation für Anzahl *Inlier*

Um festzustellen welcher der Bild-Transformationen $\mathcal{T}$ sich wie stark auf das *Matching* ausgewirkt haben, werden die Mittelwerte für die Anzahl der *Inlier* gegeben durch $|\overline{\mathfrak{I}}(\mathcal{T})|$ bestimmt. Die sechs Transformationen werden in der folgenden Tabelle beschrieben.

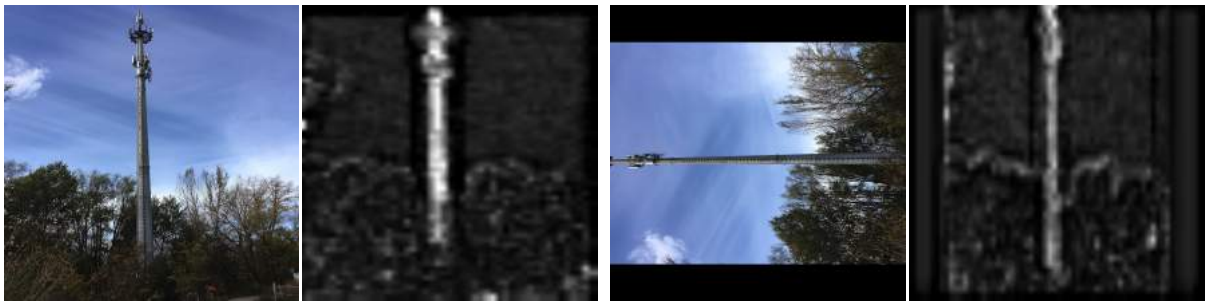
Verfahren	$\lvert\overline{\mathfrak{J}}(g)\rvert$	$\lvert\overline{\mathfrak{J}}(\gamma)\rvert$	$\lvert\overline{\mathfrak{J}}(n)\rvert$	$\lvert\overline{\mathfrak{J}}(p)\rvert$	$\lvert\overline{\mathfrak{J}}(r)\rvert$	$\lvert\overline{\mathfrak{J}}(z)\rvert$
$DELF_\zeta$	576,9	702,2	658,6	306,4	136,1	434,5
$DELF_\beta$	232,5	534,2	423,5	219,7	13,8	136,0
$SIFT_\zeta$	173,7	460,0	846,9	301,3	607,0	224,3
$SIFT_\beta$	144,3	439,1	835,1	398,1	594,0	234,8

Tabelle 7: Empirische Messung für $\lvert\overline{\mathfrak{J}}(\mathcal{T})\rvert$ aus $\mathcal{KTD}$ aus 338 Paaren innerhalb Szene, TRANSFORMATION338

Die erste Spalte in Tabelle 7 beschreibt das durchschnittliche Verhalten der *Inlier* unter Glättung. $DELF_\zeta$ hat die höchste Anzahl an *Inliern*, gefolgt von $DELF_\beta$. $SIFT_\zeta$ und $SIFT_\beta$ erzielen beiden weniger *Inlier* als $DELF$. Für die zweite Spalte, die Gamma-Korrektur, zeigt sich ein ähnliches Verhalten. Wieder erzielt $DELF_\zeta$ die maximale Anzahl. An dieser Stelle lässt sich festhalten, dass die $DELF$ besser als $SIFT$ mit Glättung und Gamma-Korrektur umgehen kann, was $\lvert\overline{\mathfrak{J}}\rvert$ betrifft. Ein umgekehrtes Bild ergibt sich für die Analyse bei Bildrauschen. Die höchste Anzahl an *Inliern* erzielt dabei $SIFT_\zeta$ gefolgt von $SIFT_\beta$. Die beiden $DELF$ Verfahren liefern deutlich weniger *Inlier*. Das *Matching*-Verhalten bei Rauschen für $DELF$ fällt eher schlecht aus. Bei Perspektivischer Transformation hat das $SIFT_\beta$-Verfahren. An zweiter Stelle ist $DELF_\zeta$. Wie bereits erwähnt, schneidet $DELF$ bezüglich der Rotation schlechter ab. $SIFT_\zeta$ ist mit 607 *Inliern* sehr robust gegenüber Rotation. In der letzten Spalte zeigt sich für Zoom $DELF_\zeta$ am stabilsten. Allerdings erreicht $DELF_\beta$ mit 136,0 am wenigsten *Inlier* innerhalb der vier getesteten Verfahren unter Zoom.

5.4.6. Visualisierung

Attention Im Folgendem soll die *Attention Map* bei Rotation für das Bild `antenne` untersucht werden.



(a) `antenne_original` (b) *Attention* zu `antenne_original` (c) `antenne` mit $\theta = \pi/2$ (d) *Attention* zu `antenne` mit $\theta = \pi/2$ (zurückgedreht)

Abbildung 69: *Attention*-Unterschiede zwischen Bild `antenne` und `antenne` mit $\theta = \pi/2$

Abbildung 69b zeigt enorme Ausschläge auf der Fläche der Antenne. In der rechten Abbildung 69d fällt dieser Ausschlag geringer aus. Dies liegt daran, dass die *Attention* in dieser Weise trainiert wurde. Es wird sich jedoch in der Analyse der Distanzen d und Anzahl *Inlier* $\lvert\mathfrak{J}\rvert$ negativ für $DELF$ unter Rotation auswirken. Es ist erkennbar, dass in der linken Abbildung 69b das sich vertikal entwickelnde Objekt hohe *Attention* (sehr hell und große Fläche), während in der rechten Abbildung 69d, welches einer Rotation (siehe $SIFT$ in Abbildung 49t) unterliegt und nur für die Illustration 69d zurück gedreht wurde, das vorher vertikal vorliegende Objekt sich nun in horizontale Richtung entwickelt.

Matching Wie das *Matching* im Detail aussieht wird im folgendem illustriert. Als Beispiel wird das *Query*-Bild `baum0` gewählt. Dies wird für $DELF_\zeta$ und um einen Vergleich zu haben auch für $SIFT_\zeta$ durchgeführt.

(a) g_4, $|\mathfrak{I}| = 533$ (b) $\gamma_{0,25}$, $|\mathfrak{I}| = 567$ (c) $n_{0,012}$, $|\mathfrak{I}| = 725$

(d) $p_{0,5}$, $|\mathfrak{I}| = 272$ (e) $\theta_{\pi/2}$, $|\mathfrak{I}| = 106$ (f) $z_{2,1}$, $|\mathfrak{I}| = 355$

Abbildung 70: $DELF_\zeta$ *Matching*-Visualisierung für sechs Transformationen zu Szene `baum` aus $\mathcal{KTD}$

Abbildung 70 visualisiert das *Matching* bei maximalen Bild-Transformation für $DELF_\zeta$. Dabei sei erwähnt, dass 1000 *Keypoints* extrahiert wurden. Es zeigt sich, wie schon in Tabelle 7, ein sehr robustes Verhalten bei Rauschen (siehe Abbildung 70c), Gamma-Korrektur (siehe Abbildung 70b) und Glättung (siehe 70a). Zudem illustriert Abbildung 70e das vergleichsweise (siehe Abbildung 71e) schwache Verhalten von $DELF$ bei Bild-Rotation.

(a) g_4, $|\mathfrak{I}| = 127$ (b) $\gamma_{0,25}$, $|\mathfrak{I}| = 175$ (c) $n_{0,012}$, $|\mathfrak{I}| = 798$

(d) $p_{0,5}$, $|\mathfrak{I}| = 96$ (e) $\theta_{\pi/2}$, $|\mathfrak{I}| = 858$ (f) $z_{2,1}$, $|\mathfrak{I}| = 156$

Abbildung 71: $SIFT_\zeta$ *Matching*-Visualisierung für sechs maximale Transformationen zu Szene `baum` aus $\mathcal{KTD}$

Abbildung 71 zeigt das *Matching* bei maximalen Bild-Transformation für $SIFT_\zeta$.

5.4.7. Gesamte empirische Messungen in $\mathcal{KTD}$

Um generelle Aussagen zu den Verfahren für $\mathcal{KTD}$ zu machen, wird nun eine Zusammenfassung der gemessenen Daten erfolgen. Dafür wurden wie auch schon beim Karlshorst-Datensatz die Werte für *True Positive, False Positive* und *False Negative*. Dabei sei erwähnt, dass der Karlshorst-Transformations-Datensatz nicht in erster Linie deswegen erzeugt wurde, sondern um das Verhalten der *Inlier* und Distanzen unter Transformationen zu evaluieren. Trotzdem sollen der Vollständigkeit halber die *Precision, Recall* und F_1 berechnet werden.

Verfahren	tp	fp	fn	*precision*	*recall*	F_1	$\overline{\lvert\mathfrak{I}\rvert}$	$\overline{d}$
$DELF_\zeta$	336	0	2	1	0.994	0,499	484,8	0,46
$DELF_\beta$	302	0	36	1	0,893	0,472	279,2	0,36
$SIFT_\zeta$	317	0	21	1	0,938	0,484	413,4	103,27
$SIFT_\beta$	335	0	3	1	0,991	0,498	423.9	86,88

Tabelle 8: Empirische Messung für den Karlshorst-Tranformations-Datensatz $\mathcal{KTD}$ aus 338 bildpaaren innerhalb der Klasse bzw. Szene, SAME_SCENE338

Die Tabelle 8 zeigt, dass $DELF_\zeta$ den höchsten Wert für tp mit 336 erzielt. Aus 338 Bildpaaren wurden lediglich zwei nicht als positiv erkannt. Der Recall beträgt 0,994 und der $F_1 = 0,499$, was beides die maximalen Werte für die untersuchten vier Verfahren sind. Den kleinsten *Recall*-Wert hat $DELF_\beta$. $SIFT_\beta$ hat die zweithöchste Anzahl an *True Positives* mit 335 und 3 *False Negatives*. $SIFT_\zeta$ hat die dritte Position bezüglich der tp analog zu zum *Recall* von 0,938. Bezüglich der *Inlier* zeigt sich, dass die Anzahl der *Inlier*, bei Verfahren mit hohem *Recall* ebenfalls hoch ist. Zudem zeigt sich, dass eine geringe durchschnittliche Distanz für $DELF_\zeta$ 0,46 und $DELF_\beta$ 0,36 keinen Effekt auf den *Recall*-Wert hat. Das gleiche Bild zeigt sich auf für $SIFT$. Der Unterschied der Distanzen von 103,27 und 86,88 ist genau konträr zu dem Verhalten

von *DELF*, denn für *DELF* erzielt das Verfahren mit ckdTree und RANSAC einen höheren Recall. Für *SIFT* ist genau das Gegenteil der Fall, der Ansatz mit *Brute Force* und *Ratio Test* ist hier erfolgreicher.

6. Zusammenfassung

In der empirischen Analyse für den Karlshorst-Datensatz zeigte sich *Deep Local Features* gegenüber *Scale Invariant Feature Transform* als überlegen. In Tabelle 5 erzielt $DELF_\zeta$ den höchsten F_1-Wert von 0,414. $DELF_\beta$ hat mit *True Positive* = 45 und *False Negative* = 95 schlechtere Werte und erreicht nur 0,257 bezüglich F_1. Beide DELF-Verfahren schneiden bezüglich des F_1-Wert besser ab als SIFT. Für ALL_SCENES1690 zeigt sich in Tabelle 6 ein F_1-Wert ein maximaler Wert von 0,402 für $DELF_\zeta$. $DELF_\beta$ liegt mit 0,253 wie auch schon in SAME_SCENE130 unter $DELF_\zeta$. Auch für die Suche in den 1690 Bildpaaren schneiden beide $SIFT$ Verfahren bezüglich *Recall* und *Precision* schlechter als $DELF$ ab (siehe Tabelle 6). Bei der Variation des Threshold T (ab wann es als *True Positiv* gezählt) zeigen sich beide $DELF$-Verfahren als sehr stabil, was bedeutet, dass bei zunehmendem *Recall* die *Precision* nur wenig abfällt. $SIFT$ hingegen zeigt ein sehr sensitives Verhalten (siehe Abbildung 48). Der Karlshorst-Transformations-Datensatz ergibt bezüglich des SAME_SCENE338, also der reduzierten Suche innerhalb der Transformation ein leicht anderes Ergebnis. $DELF_\zeta$ erreicht den höchsten *Recall* mit 0,994 wie auch schon im Karlshorst-Datensatz, allerdings hat $DELF_\beta$ den kleinsten *Recall*-Wert der vier getesteten Verfahren mit einem Wert von 0,893. Dies liegt an der Tatsache, dass 36 *False Negatives* vorliegen. Bei den Mittelwerten pro Transformation für Anzahl *Inlier* zeigt sich, dass $DELF$ bei der angewendeten Rotation, perspektivischer Transformation und Zoom weniger *Inlier* findet als bei Gamma-Korrektur, Glättung und Bildrauschen. Im Vergleich zu $SIFT$ schneidet $DELF$ hinsichtlich der durchschnittlichen Anzahl an *Inliern* bei Glättung, Gamma-Korrektur und Zoom besser ab, schlechter bei Rotation, Rauschen und leicht schlechter bei Perspektivischer Transformation. Es lässt sich eine vergleichsweise positives Verhalten von $DELF$ unter Gamma-Korrektur, Glättung und Bildrauschen feststellen, auch wenn die Anzahl der *Inlier* mit Stärke der Transformation abnimmt. Zusammenfassend lässt sich festhalten, dass $DELF$ in beiden getesteten Datensätzen mindestens gleich gut (durchschnittlich) oder besser abschneidet als $SIFT$. Besonders im Karlshorst-Datensatz zeigte sich die Überlegenheit von *Deep Local Features*.

6.1. Ergebnis

$DELF$ eignet sich sehr gut für die lokale Merkmalsextraktion und kann in fast allen Bewertungskriterien besser als $SIFT$ abschneiden. Damit zeigt sich, dass die lokale Bild-Merkmalsextraktion mit Neuronalen Netzen mit $DELF$ für die Aufgabe des *Image Retrieval* sowie anderer *Matching-Aufgaben* im *Outdoor*-Bereich sehr gut eingesetzt werden kann und Vorteile gegenüber $SIFT$ bietet. Die Hypothese der Arbeit, dass *Recall* und *Precision* für den Karlshorst-Datensatz besser für *Deep Local Features* ausfallen wird als für *Scale Invariant Feature Transform*, hat sich damit bestätigt.

6.2. Ausblick

Als Ausblick und Weiterführung der Arbeit könnte eine Analyse des *Trainings* des Neuronalen Netzes ($DELF$) für den Deskriptor sowie der *Attention* erfolgen. Zudem könnte versucht werden, das schlechte Abschneiden bei Bild-Rotation (Transformation) zu verbessern, indem beim *Training* des Netzes mehr Bilder mit stärkerer Rotationen verwendet werden.

Literatur

[1] Wilhelm Burger und Mark James Burge. *Digitale Bildverarbeitung: Eine algorithmische Einführung mit Java*. 3. Aufl. X.media.press. Berlin: Springer Vieweg, 2015. DOI: 10.1007/978-3-642-04604-9.

[2] Wolfgang Dahmen und Arnold Reusken. *Numerik für Ingenieure und Naturwissenschaftler*. Springer-lehrbuch. Berlin: Springer, 2006. URL: https://cds.cern.ch/record/937425.

[3] Angelika Erhardt. *Einführung in die digitale Bildverarbeitung : Grundlagen, Systeme und Anwendungen ; mit 35 Beispielen und 44 Aufgaben*. Wiesbaden: Vieweg + Teubner, 2008.

[4] Bin Fan, Zhenhua Wang und Fuchao Wu. *Local Image Descriptor: Modern Approaches*. 1st. Springer Publishing Company, Incorporated, 2016.

[5] Martin A. Fischler und Robert C. Bolles. „Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography“. In: *Commun. ACM* 24.6 (Juni 1981), S. 381–395. ISSN: 0001-0782. DOI: 10.1145/358669.358692. URL: http://doi.acm.org/10.1145/358669.358692.

[6] Detlef Wille Gerhard Merziger Günter Mühlbach und Thomas Wirth. *Formeln + Hilfen zur höheren Mathematik*. Hannover: Binomi Verlag, 2010.

[7] Rafael C. Gonzales und Paul Wintz. *Digital Image Processing (2Nd Ed.)* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1987.

[8] Ian Goodfellow, Yoshua Bengio und Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: http://www.deeplearningbook.org.

[9] Günther Görz. *Einführung in die künstliche Intelligenz*. Addison Wesley, 1995.

[10] Richard Hartley und Andrew Zisserman. *Multiple View Geometry in Computer Vision*. 2. Aufl. New York, NY, USA: Cambridge University Press, 2003.

[11] Kaiming He u. a. „Deep Residual Learning for Image Recognition“. In: *CVPR*. IEEE Computer Society, 2016, S. 770–778.

[12] T. Hermes. *Digitale Bildverarbeitung: eine praktische Einführung ; [auf CD: die Bildverarbeitungssoftware Orasis 3D]*. Hanser, 2005. URL: https://books.google.de/books?id=NYBy%5C_8YSm4UC.

[13] Michael E. Houle u. a. „Can Shared-Neighbor Distances Defeat the Curse of Dimensionality?“ In: *Scientific and Statistical Database Management, 22nd International Conference, SSDBM 2010, Heidelberg, Germany, June 30 - July 2, 2010. Proceedings*. 2010, S. 482–500. DOI: 10.1007/978-3-642-13818-8_34. URL: http://www.dbs.ifi.lmu.de/~zimek/publications/SSDBM2010/SNN-SSDBM2010-preprint.pdf.

[14] K. Jänich. *Lineare Algebra*. Springer-Lehrbuch. Springer, 2013. URL: https://books.google.de/books?id=Mu5DT0F0hkMC.

[15] Firas Jassim. *Kriging Interpolation Filter to Reduce High Density Salt and Pepper Noise*. World of Computer Science und Information Technology Journal, 2013. URL: https://arxiv.org/pdf/1302.1300.pdf.

[16] Hervé Jégou und Ondrej Chum. „Negative evidences and co-occurrences in image retrieval: the benefit of PCA and whitening“. In: *ECCV - European Conference on Computer Vision*. Firenze, Italy, Okt. 2012. URL: https://hal.inria.fr/hal-00722622.

[17] Yan Ke und Rahul Sukthankar. „PCA-SIFT: A more distinctive representation for local image descriptors“. In: 2004, S. 506–513.

[18] R. Klein. *Algorithmische Geometrie: Grundlagen, Methoden, Anwendungen*. eXamen.press. Springer Berlin Heidelberg, 2006.

[19] K.P. Kratzer. *Neuronale Netze: Grundlagen und Anwendungen*. Hanser, 1991. URL: https://books.google.de/books?id=TQvJAQAACAAJ.

[20] Yann Lecun u. a. „Gradient-based learning applied to document recognition“. In: *Proceedings of the IEEE*. 1998, S. 2278–2324.

[21] David G. Lowe. „Distinctive Image Features from Scale-Invariant Keypoints“. In: *Int. J. Comput. Vision* 60.2 (Nov. 2004), S. 91–110. ISSN: 0920-5691. DOI: 10.1023/B:VISI.0000029664.99615.94. URL: https://doi.org/10.1023/B:VISI.0000029664.99615.94.

[22] David G. Lowe. „Object Recognition from Local Scale-Invariant Features“. In: *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*. ICCV '99. Washington, DC, USA: IEEE Computer Society, 1999, S. 1150–. URL: http://dl.acm.org/citation.cfm?id=850924.851523.

[23] Songrit Maneewongvatana und David M. Mount. „Analysis of approximate nearest neighbor searching with clustered point sets“. In: *CoRR* cs.CG/9901013 (1999). URL: http://arxiv.org/abs/cs.CG/9901013.

[24] K. Mikolajczyk und C. Schmid. „An Affine Invariant Interest Point Detector“. In: *Proceedings of the 7th European Conference on Computer Vision-Part I*. ECCV '02. London, UK, UK: Springer-Verlag, 2002, S. 128–142. URL: http://dl.acm.org/citation.cfm?id=645315.649184.

[25] Krystian Mikolajczyk und Cordelia Schmid. „A Performance Evaluation of Local Descriptors“. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 27.10 (Okt. 2005), S. 1615–1630. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2005.188. URL: https://doi.org/10.1109/TPAMI.2005.188.

[26] Krystian Mikolajczyk und Cordelia Schmid. „Scale &Amp; Affine Invariant Interest Point Detectors“. In: *Int. J. Comput. Vision* 60.1 (Okt. 2004), S. 63–86. ISSN: 0920-5691. DOI: 10.1023/B:VISI.0000027790.02288.f2. URL: https://doi.org/10.1023/B:VISI.0000027790.02288.f2.

[27] Hyeonwoo Noh u. a. „Image Retrieval with Deep Local Features and Attention-based Keypoints“. In: *CoRR* abs/1612.06321 (2016). arXiv: 1612.06321. URL: http://arxiv.org/abs/1612.06321.

[28] Hyeonwoo Noh u. a. „Large-Scale Image Retrieval with Attentive Deep Local Features“. In: Okt. 2017, S. 3476–3485. DOI: 10.1109/ICCV.2017.374.

[29] K. B. Petersen und M. S. Pedersen. *The Matrix Cookbook*. Version 20121115. Technical University of Denmark, Nov. 2012. URL: http://www2.imm.dtu.dk/pubdb/p.php?3274.

[30] J. Philbin u. a. „Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition“. In: 2007. URL: http://www.robots.ox.ac.uk/~vgg/data/oxbuildings/.

[31] J. Philbin u. a. „Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition“. In: 2008. URL: http://www.robots.ox.ac.uk/~vgg/data/parisbuildings/.

[32] D. M. W. Powers. „Evaluation: From precision, recall and f-measure to roc., informedness, markedness & correlation“. In: *Journal of Machine Learning Technologies* 2.1 (2011). URL: https://bioinfopublication.org/files/articles/2_1_1_JMLT.pdf.

[33] B. Rovan. *Mathematical Foundations of Computer Science 1990: Banska Bystrica, Czechoslovakia, August 27-31, 1990 Proceedings*. Lecture Notes in Computer Science. Springer, 1990. URL: https://books.google.de/books?id=-PuyAAAAIAAJ.

[34] Hanan Samet. *Foundations of Multidimensional and Metric Data Structures (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005.

[35] Karen Simonyan und Andrew Zisserman. „Very Deep Convolutional Networks for Large-Scale Image Recognition“. In: *CoRR* abs/1409.1556 (2014). arXiv: 1409.1556. URL: http://arxiv.org/abs/1409.1556.

[36] H. Süße und E. Rodner. *Bildverarbeitung und Objekterkennung: Computer Vision in Industrie und Medizin*. Springer Fachmedien Wiesbaden, 2014. URL: https://books.google.de/books?id=YDBwBAAAQBAJ.

[37] Andrew P. Witkin. „Scale-space Filtering“. In: *Proceedings of the Eighth International Joint Conference on Artificial Intelligence - Volume 2*. IJCAI'83. Karlsruhe, West Germany: Morgan Kaufmann Publishers Inc., 1983, S. 1019–1022. URL: http://dl.acm.org/citation.cfm?id=1623516.1623607.

[38] Kwang Moo Yi u. a. „LIFT: Learned Invariant Feature Transform“. In: *CoRR* abs/1603.09114 (2016). arXiv: 1603.09114. URL: http://arxiv.org/abs/1603.09114.

Software- und Internetquellen

[39] *Anaconda Distribution*. 2018. URL: https://www.anaconda.com (besucht am 06. 12. 2018).

[40] André Araujo. *Github Master Research DELF*. 2018. URL: https://github.com/tensorflow/models/tree/master/research/delf (besucht am 01. 09. 2018).

[41] *cKDTree*. 2018. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.cKDTree.html (besucht am 30. 12. 2018).

[42] *Convolutional Neural Networks (LeNet)*. 2019. URL: http://deeplearning.net/tutorial/lenet.html.

[43] Insik Kim. *Github Delf Enhanced*. 2018. URL: https://github.com/insikk/delf_enhanced (besucht am 01. 10. 2018).

[44] Fei-Fei Li, Justin Johnson und Serena Yeung. *Convolutional Neural Networks for Visual Recognition*. 2017. URL: http://cs231n.stanford.edu/2017/.

[45] *OpenCV*. 2018. URL: https://opencv.org (besucht am 18. 11. 2018).

[46] *OpenCV*. 2018. URL: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_matcher/py_matcher.html (besucht am 19. 11. 2018).

[47] *OpenCV Matcher*. 2018. URL: https://docs.opencv.org/3.4/dc/dc3/tutorial_py_matcher.html (besucht am 02. 09. 2018).

[48] *Pycharm IDE*. 2018. URL: https://www.jetbrains.com/pycharm/ (besucht am 26. 12. 2018).

[49] *Python Programmiersprache*. 2018. URL: https://www.python.org (besucht am 26. 12. 2018).

[50] *Schematics LeNet*. 2018. URL: http://alexlenail.me/NN-SVG/LeNet.html (besucht am 06. 12. 2018).

[51] *Scikit Image Ransac*. 2018. URL: http://scikit-image.org/docs/dev/api/skimage.measure.html#skimage.measure.ransac (besucht am 18. 11. 2018).

[52] *Scipy*. 2018. URL: https://www.scipy.org (besucht am 29. 11. 2018).

[53] *SIFT OpenCV*. 2018. URL: https://docs.opencv.org/3.1.0/da/df5/tutorial_py_sift_intro.html (besucht am 01. 08. 2018).

[54] *Softmax and Crossentropy*. 2019. URL: https://deepnotes.io/softmax-crossentropy (besucht am 12. 01. 2019).

[55] *TensorFlow*. 2018. URL: https://www.tensorflow.org (besucht am 18. 11. 2018).

[56] *TensorFlow Graph*. 2018. URL: https://www.tensorflow.org/api_docs/python/tf/Graph (besucht am 01. 08. 2018).

Bildquellen

[57] *Alexanderplatz Weltzeituhr*. URL: https://www.freeimageslive.co.uk/free_stock_image/alexanderplatz-world-clock-jpg.

[58] Andre Araujo. *DELF Poster*. URL: https://andrefaraujo.github.io/files/posters/2017-10-22-iccv-delf-poster.pdf.

[59] *Camara Lucida Zeichnung einer Purkinje-Zelle in der Kleinhirnrinde einer Katze, von Santiago Ramón y Cajal*. URL: https://commons.wikimedia.org/wiki/File:Purkinje_cell_by_Cajal.png.

[60] Original full portrait by Dwight Hooker, scan: Alexander Sawchuk u. a. *Well-known Standard Test Image Lenna*. 1973. URL: https://en.wikipedia.org/wiki/Lenna#/media/File:Lenna_(test_image).png.

[61] P. Fischer, A. Dosovitskiy und T. Brox. *Descriptor Matching with Convolutional Neural Networks: a Comparison to SIFT*. Mai 2014. URL: http://lmb.informatik.uni-freiburg.de/Publications/2014/FDB14.

A. Anhang

Im Anhang befindet sich ein *USB-Stick* mit dieser Arbeit im PDF-Format, der *Quellcode* in *Python* und die zwei evaluierten Datensätze. Die Daten sind auch unter diesem *Hyperlink* für *Google Drive* zu finden: Masterarbeit-Daten.