
Evaluierung von Gesichtserkennungsmodellen für historische Gemälde am Beispiel ausgewählter Werke Lucas Cranachs zur optimierten Positionierung von Overlays

Bachelorarbeit zur Erlangung des akademischen Grades
Bachelor of Science
im Studiengang Medieninformatik
an der Fakultät für Informatik und Ingenieurwissenschaften
der Technischen Hochschule Köln

vorgelegt von: Niklas Mehlem
Matrikel-Nr.: 111 405 18
Adresse: Brucknerstraße 78
53721 Siegburg
niklas.mehlem@smail.th-koeln.de

eingereicht bei: Prof. Christian Noss
Zweitgutachter*in: Prof. Dr. Daniel Gaida

Siegburg, 13.06.2025

Inhaltsverzeichnis

Tabellenverzeichnis	III
Abbildungsverzeichnis	IV
Glossar	V
Abkürzungsverzeichnis	VII
Symbolverzeichnis	VIII
1 Einleitung	1
2 Recherche	3
3 Tests	7
3.1 Stichprobentests	7
3.2 Größentests	11
3.3 Confidencetests	11
3.4 Vergleich	13
4 Implementierung	15
5 Fazit	18
6 Formaler Aufbau	20
6.1 Reihenfolge	20
6.2 Deckblatt	20
6.3 Inhaltsverzeichnis	21
6.4 Abbildungsverzeichnis und Tabellenverzeichnis	21
6.5 Abkürzungsverzeichnis	21
6.6 Literaturverzeichnis	21
6.7 Rechtschreibung, Grammatik	22
6.8 Umfang der Arbeit	22
7 Gestaltung: Textsatz mit \LaTeX	23
7.1 Unter der Haube	24

7.2	Überschriften	24
7.3	Absätze	25
7.4	Silbentrennung	25
7.5	Aufzählungen	25
7.6	Abbildungen	26
7.6.1	Bilder	26
7.6.2	Vektorgrafiken	27
7.7	Tabellen	28
7.8	Abbildungs- und Tabellenverzeichnis	28
7.9	Formeln	28
7.10	Quellcode, Pseudocode	29
7.11	Weitere Verzeichnisse	30
7.11.1	Glossar erstellen	31
7.11.2	Abkürzungsverzeichnis erstellen	31
7.11.3	Symbolverzeichnis erstellen	31
7.12	Verweise	32
7.13	Besondere Abstände und Zeichen	32
7.14	Wahl der Grundschriftart	33
7.15	Metadaten für den pdf-Betrachter	33
7.16	Wechsel zwischen ein- und doppelseitigem Layout	34
7.17	Kompilieren	34
7.18	Diese Vorlage	35
8	Zitate und Literaturangaben	37
8.1	Zitieren	37
8.1.1	Quellenverweise	37
8.1.2	Derselbe und Ebenda	38
8.1.3	Zitate aus zweiter Hand	38
8.1.4	Abbildungen und Tabellen zitieren	38
8.2	Literaturverzeichnis	39
8.2.1	Inhalt und Anordnung der Literaturangaben	39
8.2.2	Bibliographische Angaben im Literaturverzeichnis	40
	Literatur	41
	Anhang	44

Tabellenverzeichnis

7.1	Eine einfache Tabelle	28
-----	---------------------------------	----

Abbildungsverzeichnis

7.1	Vielleicht handelt es sich hierbei um Kunst?	26
7.2	Eine schöne Grafik, die im Quellcode erzeugt wird!	27
7.3	Beispiel für ein listing	30

Glossar

Anker (engl. *anchors* oder *anchor boxes*) sind in der Objekterkennung vordefinierte Referenzrechtecke unterschiedlicher Größen und Seitenverhältnisse, die verwendet werden, um mögliche Positionen und Größen von Objekten in einem Bild vorab zu definieren. Detektoren vergleichen diese Anker mit tatsächlichen Objekten, um deren Lage und Größe genauer zu bestimmen. 6

Backbone Grundlegendes neuronales Netzwerk-Modul in einem tiefen Lernmodell, das für die Extraktion von Merkmalen aus Eingabedaten zuständig ist. Im Kontext der Bildverarbeitung bezeichnet es oft ein vortrainiertes Convolutional Neural Network, das als Basis für weitere Aufgaben wie Klassifikation oder Objekterkennung dient.. 3, 4, 6, 10

Bildpyramide Mehrstufige Darstellung eines Bildes in verschiedenen Auflösungen. Höhere Ebenen der Pyramide enthalten kleinere, herunterskalierte Versionen des Originalbildes. Wird verwendet, um Objekterkennung effizient auf unterschiedlichen Skalen durchzuführen. 4, 5

Boosting Ein Verfahren des überwachten Lernens, bei dem mehrere schwache Lernalgorithmen sequenziell trainiert und zu einem starken Klassifikator kombiniert werden. Jeder neue Klassifikator konzentriert sich auf die Fehler der vorherigen, um die Gesamtgenauigkeit zu erhöhen. 3, 5

Bounding-Box Rechteckiger Bereich in einem Bild, der ein Objekt umschließt, um dessen Lage und Größe zu bestimmen. 5, 6

Confidence Alternativ bezeichnet als *Score*. Numerischer Wert, der die Wahrscheinlichkeit angibt, mit der ein Modell eine bestimmte Vorhersage für korrekt hält. Beschreibt in der Gesichtserkennung typischerweise die Sicherheit, mit der ein erkannter Bereich als Gesicht klassifiziert wird. 7–16, 18

Detektor In der Bildverarbeitung ein System oder Algorithmus, der bestimmte Objekte in einem Bild lokalisiert. 3, 5, 6

Feature-Vektor Ein numerischer Vektor, der die Merkmale (engl. *features*) eines Objekts oder eines Bildausschnitts in strukturierter Form repräsentiert. Er dient als Eingabe für maschinelle Lernverfahren und Klassifikatoren. 4

Klassifikator Ein Algorithmus oder Modell, das Eingabedaten bestimmten Klassen zuordnet, um beispielsweise in der Bildverarbeitung Muster, Objekte oder Merkmale auf deren Vorhandensein zu überprüfen. 3, 4

Abkürzungsverzeichnis

Caffe Convolutional Architecture for Fast Feature Embedding. 3, 8, 12

CDA Cranach Digital Archive. 1, 7, 11

CNN Convolutional Neural Network. 5, 6, 9, 10, 12–14, 16, 18

HOG Histogram of Oriented Gradients. 4, 5, 9, 11

MMOD Max-Margin Object Detection. 5

MTCNN Multi-task Cascaded Convolutional Networks. 5, 10, 12–14, 16, 18

NMS Non-Maximum Suppression. 5

O-Net Output Network. 6

P-Net Proposal Network. 5

R-Net Refine Network. 5

SSD Single Shot MultiBox Detector. 3, 4, 6

SVM Support Vector Machine. 4, 30

Symbolverzeichnis

\vec{F} Kraft, vektorielle Größe. 31

1 Einleitung

Im Rahmen einer Masterarbeit wurden für das Cranach Digital Archive (CDA) neue Wasserzeichen erstellt. Infolgedessen wurde der Ansatz entwickelt, diese automatisiert auf den verschiedenen Bildern der Werke zu platzieren. Unter Zuhilfenahme von Gesichtserkennungsmodellen sollten die Gesichter aus ästhetischen Gründen ausgespart werden. Das Problem hierbei ist, dass Gesichtserkennungsmodelle hauptsächlich mit Trainingsdaten bestehend aus Gesichtern realer Menschen entwickelt werden und darauf ausgelegt sind, Gesichter realer Personen zu erkennen. Ziel dieser Arbeit ist das Testen und Vergleichen bestehender Gesichtserkennungsmodelle. Es soll festgestellt werden, ob sich diese für den Einsatz auf historischen Gemälden eignen. Sollten sich mehrere Modelle eignen wird überprüft, welche für den bestimmten Anwendungskontext priorisiert werden. Hierfür werden die Modelle *Haar-Cascade*, *Caffe*, *MediaPipe*, *Dlib HOG*, *Dlib Landmark*, *Dlib CNN*, *MTCNN*, *Yunet* und *RetinaFace* verglichen. Aus den am besten geeigneten Modellen soll anschließend ein Python-Modul entwickelt werden, das die Gesichtsbereiche eines Bildes ausgibt. Mithilfe dieser Daten soll die Möglichkeit gegeben werden zu überprüfen, ob sich ein Overlay potenziell auf einem Gesicht befindet um so einen Automatisierungsprozess für dessen Platzierung zu ermöglichen.

Im Recherche-Kapitel wird kurz auf die getesteten Modelle eingegangen und ihre grobe Funktionsweise erläutert, um ihre Ergebnisse in einen Kontext einordnen zu können. Als erster Schritt der Testreihe werden in Stichprobentests die verschiedenen Modelle auf ihre grundsätzliche Eignung geprüft. Es soll festgestellt werden, ob ein getestetes Modell überhaupt in der Lage ist, mit historischen Gemälden zu arbeiten. Modelle, die diese Voraussetzung nicht erfüllen, werden bereits an dieser Stelle ausgeschlossen, um in späteren Tests Zeit und Aufwand zu sparen. Im Anschluss daran werden Tests mit Bildern unterschiedlicher Auflösungen durchgeführt. Ziel ist es, zu untersuchen, ob die Bildauflösung einen Einfluss auf die Ergebnisse der Modelle hat und welche Auswirkungen dabei zu beobachten sind. Zudem soll auf dieser Basis festgelegt werden, mit welcher Auflösung die Bilder in den folgenden Tests verwendet werden sollen, um möglichst gute Ergebnisse zu erzielen. Darauf folgend werden die Confidence-Grenzwerte der Modelle im nächsten Test optimiert. Dies soll sicherstellen, dass in der finalen Auswertung die jeweils besten Ergebnisse der Modelle miteinander verglichen werden können, um eine fundierte Entscheidung darüber zu treffen, welche Modelle für das zu entwickelnde Python-Modul verwendet werden sollen. Der finale Test vergleicht

die False-Negatives, False-Positives sowie die Anzahl der Gesichter, die jeweils nur von einem bestimmten Modell erkannt wurden. Ziel ist es, festzustellen, welche Modelle die zuverlässigsten Ergebnisse liefern und welche sich potenziell ergänzen lassen, um in möglichst allen Bildtypen eine vollständige Gesichtserkennung zu ermöglichen.

2 Recherche

Ziel dieses Kapitel ist es, die Funktionsweise der verschiedenen Gesichtserkennungsmodelle grob zu erläutern, um ihre Ergebnisse besser einordnen zu können. Dabei wird bewusst auf eine vertiefte technische Beschreibung verzichtet, da dies einerseits den Rahmen dieser Arbeit überschreiten würde. Andererseits haben erste Stichprobentests gezeigt, dass die Modelle bei Bildern historischer Werke messbar andere Ergebnisse liefern als bei realen Fotos.

- **Haar-Cascade:** Der **Haar-Cascade-Detektor** basiert auf dem von Viola und Jones entwickelten Verfahren zur schnellen Objekterkennung mittels einer kaskadierten Boosting-Architektur (Viola & Jones, 2001). Haar-ähnliche Merkmale werden ermittelt, indem die Differenz der Pixel-Summen benachbarter rechteckiger Regionen berechnet wird. Mithilfe eines Integralbilds lassen sich diese Merkmale in konstanter Zeit auswerten. Anschließend erfolgt die Merkmalsauswahl durch den AdaBoost-Algorithmus, der eine Vielzahl schwacher Klassifikatoren zu einem starken Klassifikator kombiniert. Die Klassifikatoren sind in mehreren Stufen kaskadiert, wobei jede Stufe unpassende Bildbereiche frühzeitig verwirft, um die Erkennungszeit zu verkürzen. In der ursprünglichen Implementierung umfasste die Kaskade 38 Stufen, beginnend mit nur einem Merkmal in der ersten Stufe und steigend auf bis zu 6000 Merkmale in späteren Stufen. In dieser Arbeit wird OpenCV verwendet, um mit der Klasse `CascadeClassifier` das vortrainierte XML-Modell zu laden. Das verwendete Modell `haarcascade_frontalface_default.xml` wurde mit dem **Caltech Frontal Face Dataset** von 1999 trainiert (Howse, 2019, 101–102).
- **Caffe: Convolutional Architecture for Fast Feature Embedding (Caffe)** ist ein Deep-Learning-Framework, das zur Entwicklung und Ausführung von Modellen dient (Jia et al., 2014). In dieser Arbeit wird das Modell `res10_300x300_ssd_iter_140000.caffemodel` zusammen mit `deploy.prototxt` verwendet, um die Parameter der Netzwerkarchitektur zu definieren. Wie der Modellname bereits andeutet, basiert es auf dem Single Shot MultiBox Detector (SSD)-Framework mit einem **ResNet10**-Backbone. SSD-Modelle bestehen aus einem SSD-Head und einem Backbone (Esri Developer, o.D.). Häufig wird ein Netzwerk wie ResNet verwendet, wobei die finale Klassifikationsebene entfernt wird, um daraus das Backbone zu erzeugen. SSD nutzt mehrere Feature-Maps,

auf denen sogenannte Default-Boxes platziert werden, welche auf Merkmale untersucht werden (Liu et al., 2016). Die Angabe „300x300“ im Modellnamen zeigt an, dass das Modell mit Bildern der Größe 300×300 Pixel trainiert wurde. Damit das Modell auch größere Eingaben verarbeiten kann, wird die Funktion `blobFromImage` von OpenCV genutzt, um das Bild für die Gesichtserkennung auf 300×300 zu skalieren (Serengil, 2020). Die Trainingsdaten des Modells sind nicht öffentlich dokumentiert

- **MediaPipe:** **MediaPipe** ist ein Open-Source-Framework von Google, das zur Erstellung von On-Device-Machine-Learning-Pipelines entwickelt wurde (Google AI Edge, 2024, 2025b). MediaPipe bietet neben der Gesichtserkennung auch weitere Funktionen wie Hand- und Körpererkennung. In dieser Arbeit wird ausschließlich das Teilmodul *MediaPipe Face Detection* betrachtet, welches ein Teil von *MediaPipe Solutions* ist. Als Ergebnis liefert MediaPipe sowohl die Position des Gesichts als auch die von Gesichtsmerkmalen wie Augen, Nasenspitze und Mund (Google AI Edge, 2025a). *MediaPipe Face Detection* nutzt **BlazeFace** als Modell in seiner Pipeline, welches auf 128×128 Pixel trainiert wurde (Google AI Edge, 2025a). *BlazeFace* verwendet ein SSD-Framework, das für eine schnellere Performanz modifiziert wurde, sodass eine Geschwindigkeit von 200 bis zu 1000 FPS erreicht werden kann. Es wurde explizit für den Einsatz auf mobilen Endgeräten optimiert, um effizienter mit GPU-Ressourcen umzugehen (Bazarevsky et al., 2019).
- **Dlib HOG:** **Dlib** ist ein Open-Source-C++-Toolkit, das unter anderem Gesichtserkennung mit vorgefertigten Methoden ermöglicht (dlib, 2022). Die hier verwendete Methode `get_frontal_face_detector` nutzt den Histogramm of Oriented Gradients (HOG). HOG ist ein Feature-Deskriptor, der Bilder in Feature-Vektoren überführt, indem er lokale Gradienten berechnet, daraus Histogramme erstellt und diese blockweise normalisiert (Dalal & Triggs, 2005). Dlib kombiniert HOG mit einem Sliding-Window-Verfahren über eine Bildpyramide sowie einer Support Vector Machine (SVM) (dlib, o.D. d, o.D. e). Beim Sliding-Window-Verfahren wird mit einer festen Fenstergröße über das Bild iteriert und jeder so erzeugte Bildausschnitt analysiert (Esri Developer, o.D.). Die SVM dient als Klassifikator, um die von HOG erzeugten Feature-Vektoren aus den verschiedenen Bildausschnitten zu bewerten und festzustellen, ob diese Gesichter enthalten (Dalal & Triggs, 2005).
- **Dlib HOG Landmark:** Verwendet wird in dieser Arbeit das `shape_predictor_68_face_landmarks.dat`-Modell von Davis King (King, 2024). Das Modell basiert auf der Methode aus dem Paper “One Millisecond Face Alignment with an Ensemble of Regression Trees” von Kazemi und Sullivan, 2014 (dlib, o.D. c). Darin wird gezeigt, wie Gesicht-Landmarks innerhalb von Millisekunden

mittels Regressionsbäumen und Boosting geschätzt werden können. Wie im Namen abzulesen ist, markiert das Modell 68 Landmarks in einer vordefinierten Bounding-Box (dlib, o.D. b). Landmarks sind dabei unter anderem Punkte im Gesicht wie Augen, Nase und Lippen (dlib, o.D. c). Das Modell nimmt Bounding-Boxes und markiert in diesen die Landmarks. Es braucht zusätzlich die Hilfe eines Detektors, der die benötigten Bounding-Boxes zur Verfügung stellen kann. Im Fall von **shape_predictor_68_face_landmarks.dat** wurde dieses darauf ausgelegt, zusammen mit Dlib HOG verwendet zu werden, weswegen in dieser Arbeit Dlib HOG für die Erstellung der Bounding-Boxes verwendet wird (King, 2024). Das Modell wurde mithilfe des iBUG 300-W-Datensatzes trainiert (dlib, o.D. c; King, 2024).

- **Dlib CNN: mmod_human_face_detector.dat** ist ein Modell von Dlib, das Convolutional Neural Network (CNN) verwendet. Dlib selbst beschreibt das Modell als genauer im Vergleich zu HOG, weist aber auch darauf hin, dass die höhere Genauigkeit auf Kosten eines höheren Rechenaufwands und damit verbundener Latenz erreicht wird (dlib, o.D. a). Wie aus dem Namen des Modells ersichtlich, wurde es mithilfe von Max-Margin Object Detection (MMOD) trainiert. MMOD ist ein Trainingsverfahren, bei dem alle möglichen Sub-Fenster in einem Bild berücksichtigt werden – im Gegensatz zum Sliding-Window-Verfahren, das nur über Stichproben möglicher Fensteroptionen iteriert (King, 2015). Ein CNN erkennt Gesichter, indem es das Eingabebild schrittweise mit kleinen Matrizen, sogenannten Filtern oder Kernels, faltet (GeeksforGeeks, 2025). Dabei gleiten die Filter über das Bild und erzeugen durch die Faltung für jeden Filter eine eigene Feature-Map. Anschließend wird Pooling angewendet, welches benachbarte Bereiche zusammenfasst und so die Auflösung sowie den Rechenaufwand reduziert (GeeksforGeeks, 2025). Durch die Beibehaltung der zweidimensionalen Struktur kann ein CNN räumliche Zusammenhänge im Bild ausnutzen und so Merkmale wie Kanten zuverlässig erkennen.
- **MTCNN: Multi-task Cascaded Convolutional Networks (MTCNN)** kombiniert Gesichtserkennung und Gesichtsausrichtung in einem Modell (Zhang et al., 2016). Dafür werden drei CNNs kaskadiert, um sowohl effizient als auch effektiv zu arbeiten. Das erste CNN wird Proposal Network (P-Net) genannt (Zhang et al., 2016). Das P-Net ist ein schnelles, weniger tiefes CNN, das mithilfe einer Bildpyramide die ersten möglichen Bounding-Boxes markiert. Anschließend wird Non-Maximum Suppression (NMS) verwendet, um überlappende Bounding-Boxes zusammenzufügen. Im zweiten Schritt werden weitere falsche Bounding-Boxes vom sogenannten Refine Network (R-Net) verworfen, die keine Gesichter enthalten. Dieses ist ein komplexeres CNN im Vergleich zum P-Net (Zhang et al., 2016). Zuletzt wird das stärkste CNN eingesetzt, nachdem nur noch

wenige Bounding-Boxes verbleiben. Das Output Network (O-Net) markiert fünf Landmark-Punkte und generiert so das Endergebnis (Zhang et al., 2016).

- **Yunet:** **Yunet** ist ein leichtgewichtiger Detektor und CNN, der speziell für den Einsatz auf ressourcenbeschränkten Geräten wie Smartphones entwickelt wurde (Wu et al., 2023). Anders als andere Detektoren wie SSD nutzt Yunet einen ankerfreien Ansatz (Wu et al., 2023). Indem auf Anker oder Default-Boxes verzichtet wird, benötigt Yunet weniger Rechenleistung. Allerdings haben ankerfreie Ansätze Probleme mit kleineren Gesichtern oder wenn in einem Bild unterschiedlich große Gesichter vorkommen (Wang et al., 2018).
- **RetinaFace (CPU):** **RetinaFace** ist ein einstufiger, pixelweiser Gesichtsdetektor, welcher je nach Variante ResNet oder MobileNet als Backbone verwendet (Deng et al., 2019). Einstufig bedeutet, dass Klassifikation und Lokalisierung in einem einzigen Durchlauf gemeinsam ausgeführt werden, ähnlich wie beim SSD. Pixelweise bedeutet hier, dass für jede Gitterzelle der Feature-Map eine Vorhersage getroffen wird. Pro Zelle gibt RetinaFace einen Face-Score, eine Bounding-Box, fünf Landmark-Koordinaten sowie zusätzliche Informationen für ein 3D-Mesh des Gesichts aus (Deng et al., 2019). Um Gesichter verschiedener Größen zuverlässig zu erkennen, nutzt RetinaFace Feature-Pyramiden, bei denen Feature-Maps unterschiedlicher Auflösungen kombiniert werden. Trainiert wurde das Modell auf dem **WIDER FACE**-Datensatz (Deng et al., 2019). Zusätzlich zu klassischen Trainingsansätzen wurden auf allen erkennbaren Gesichtern des Datensatzes fünf Landmark-Punkte händisch annotiert, um die Lokalisierungsgenauigkeit zu erhöhen. Ein zusätzlicher Zweig des Modells erzeugt außerdem ein 3D-Gesichts-Mesh, welches für selbstüberwachtes Lernen verwendet wird (Deng et al., 2019).

3 Tests

Viele Gesichtserkennungsmodelle wurden mit Datensätzen trainiert, die ausschließlich Gesichter von realen Personen enthalten. Hinzu kommt, dass diese Modelle für die Erkennung von Gesichtern realer Personen optimiert wurden. Modelle, die zuverlässig reale Gesichter detektieren, weisen bei der Anwendung auf Bilder historischer Gemälde häufig eine deutlich geringere Erkennungsgenauigkeit auf. Ziel der folgenden Tests ist es, die Eignung und Leistungsfähigkeit verschiedener Gesichtserkennungsmodelle für die Anwendung auf historischen Gemälden zu evaluieren. Begonnen wird mit den Stichprobentests, in welchen ungeeignete Modelle aussortiert werden sollen. Es folgen die Größentests, die dazu dienen, die Ergebnisse bei verschiedenen Auflösungen zu vergleichen. So soll sichergestellt werden, dass weitere Tests nur Bilder mit einer für die jeweiligen Modelle geeigneten Auflösung verwenden. Im Anschluss werden die Confidence-Grenzwerte der Modelle optimiert. Dafür werden die Confidence-Werte für erkannte Gesichter und False-Positives verglichen, um den bestmöglichen Grenzwert zu ermitteln. Im letzten Test werden alle Modelle direkt miteinander verglichen. Hierbei werden die False-Negatives, False-Positives sowie die Gesichter gezählt, die nur von einem Modell erkannt wurden.

3.1 Stichprobentests

Um Zeit und Aufwand zu sparen, werden in Stichprobentests die Eignung der verschiedenen Gesichtserkennungsmodelle überprüft. Ungeeignete Modelle werden am Ende dieser Stichprobentests ausgeschlossen. Hauptkriterium für die Eignung eines Modells ist die Fähigkeit, Gesichter auf historischen Gemälden zuverlässig zu erkennen. Als sekundäres Kriterium sollte die Anzahl an False Positives in einem vertretbaren Verhältnis stehen. Ein Modell, das zwar alle Gesichter erkennt, jedoch auch große Teile des restlichen Bildes fälschlicherweise als Gesichter markiert, erlaubt keine präzise Aussage, ob sich ein Overlay auf einem Gesicht befindet. Weitgehend vernachlässigt wird der Aspekt der Rechenzeit. Im Kontext des CDA gilt es als wichtiger, Gesichter möglichst zuverlässig zu erkennen, als dies möglichst schnell zu tun. Hierfür wurden vier Werke ausgewählt: ein Einzelporträt, ein Zwei-Personen-Porträt, ein Drei-Personen-Porträt sowie ein Gruppenbild. Jedes Modell wird auf jedes der vier Werke angewendet, wodurch sich direkt ein erster Eindruck ergibt, wie gut sich die Modelle im Vergleich

eignen. Durch die verschiedenen Bildtypen ist es zudem möglich, dass sich bereits Eignungen für bestimmte Bildkategorien abzeichnen. Schließlich dienen die Tests auch dazu, erste Confidence-Grenzwerte für die Modelle zu definieren.

- **Haar-Cascade:** Haar-Cascade, genauer `haarcascade_frontalface_default.xml`, eignet sich in keiner Weise. Es scheitert sowohl daran, zuverlässig das Gesicht in einem Einzelporträt zu erkennen, als auch daran, mehrere Gesichter in einem Gruppenbild zu identifizieren. Zudem kommt es häufig zu False-Positives – teilweise werden mehr falsche als tatsächliche Gesichter markiert. Haar-Cascade besitzt keinen direkt einstellbaren Confidence-Wert, jedoch ein Äquivalent. Selbst bei Variation dieses Wertes liefert das Modell keine zuverlässigen Ergebnisse und scheidet somit bereits an dieser Stelle für weitere Versuche aus. Auffällig ist zudem, dass Haar-Cascade bei identischen Motiven mit höherer Auflösung noch schlechter abschneidet, da vermehrt False-Positives entstehen.
- **Caffe:** Das Caffe-Modell besteht den Stichprobentest. In Einzel-, Zwei- und Drei-Personen-Porträts werden jeweils alle Hauptgesichter erkannt – zuverlässig und ohne ein einziges False-Positive. Allerdings zeigt das Modell Schwächen bei kleineren Gesichtern oder solchen, die nicht im Fokus des Porträts stehen. Im Gruppenbild wurde kein einziges Gesicht erkannt – auch keine False-Positives. Senkt man den Confidence-Wert auf 0.2, werden im Drei-Personen-Porträt alle Gesichter erkannt, weiterhin ohne False-Positives. Das Gruppenbild bleibt jedoch eine Schwachstelle: Um Gesichter zu erkennen, müsste der Confidence-Wert so stark reduziert werden, dass die resultierenden False-Positives die Ergebnisse unbrauchbar machen würden. Bei höherer Auflösung ändern sich die Resultate kaum – nur im Gruppenbild treten mehr False-Positives auf, was die Problematik unterstreicht. Bei geringerer Auflösung erkennt Caffe weiterhin alle Gesichter korrekt und dies ohne False-Positives. Im Drei-Personen-Porträt wurde ein Nebengesicht nur halb erkannt. Überraschenderweise wurden im Gruppenbild mit geringer Auflösung tatsächlich Bereiche markiert. Dabei wurden einige Gesichter korrekt erkannt, jedoch waren die Bounding-Boxes zu groß, um sinnvoll nutzbar zu sein. Wiederholte Tests mit mittlerer Auflösung (600×1130 Pixel) zeigten, dass ein Nebengesicht im Drei-Personen-Porträt nicht erkannt wurde, obwohl dies bei höheren und niedrigeren Auflösungen gelang. Diese Inkonsistenz erschwert die Auswahl einer geeigneten Bildauflösung und könnte ein Grund sein, Caffe letztlich auszuschließen.
- **MediaPipe:** MediaPipe erkennt das Gesicht im Einzelporträt zuverlässig und ohne False-Positives. Dies ist jedoch der einzige bestandene Stichprobentest. In allen anderen Tests – inklusive Zwei-, Drei-Personen-Porträts und Gruppenbild – werden weder Gesichter noch False-Positives markiert. Das deutet darauf hin, dass MediaPipe nur große Gesichter erkennt. Da das Modell im gewünschten

Kontext kaum brauchbar ist, wird es nicht weiter getestet. Auch bei höher aufgelösten Bildern bleiben die Ergebnisse identisch.

- **Dlib HOG:** Das HOG-basierte Modell von dlib zeigt gute Leistungen bei der Erkennung kleiner Gesichter, hat jedoch Probleme mit größeren. Das Gesicht im Einzelporträt wird nur erkannt, wenn der Confidence-Wert stark reduziert wird – was wiederum zu 0–3 False-Positives führen kann. Die Gesichter in Zwei- und Drei-Personen-Porträts sowie größtenteils im Gruppenbild werden hingegen gut erkannt. Trotz offensichtlicher Schwächen sind die Ergebnisse solide genug für weiterführende Tests. Bei hochauflösenden Bildern erkennt das Modell auch das Einzelporträt, allerdings steigt die Zahl der False-Positives, ähnlich wie bei geringem Confidence-Wert und niedriger Auflösung. Dies deutet darauf hin, dass der Confidence-Wert abhängig von der Bildauflösung angepasst werden muss, was den Einsatzaufwand erhöht. Bei geringer Auflösung gibt es im Einzelporträt ein False-Positive, im Zwei- und Drei-Personen-Porträt jedoch keine. Im Gruppenbild verschlechtern sich die Ergebnisse deutlich: Statt 6 Gesichtern mit 3 False-Positives werden nur noch 1 Gesicht und 2 False-Positives erkannt. Das zeigt, dass Gruppenbilder und Porträts separat betrachtet werden sollten. Bei mittlerer Auflösung erkennt das Modell bei Einzel-, Zwei- und Drei-Personen-Porträts alle Gesichter mit 1–2 False-Positives pro Bild.
- **Dlib HOG Landmark:** Anfangs wurde geplant, das Landmark-Modell ergänzend zur Überprüfung der Ergebnisse von Dlib HOG zu verwenden. Ziel war es, False-Positives auszusortieren und dadurch präzisere Ergebnisse zu erzielen. Diese Annahme bestätigt sich nicht. Zwar platziert das Modell die Landmarks korrekt, jedoch führt das Aussortieren basierend auf den Landmark-Daten häufig zur Entfernung korrekter Gesichter statt False-Positives. Das Modell liefert somit keinen Vorteil gegenüber Dlib HOG und wird daher nicht weiter getestet. Auch hochauflösende Bilder verbessern die Ergebnisse nicht. Im Gegenteil, es entstehen häufig mehr False-Positives. Im Drei-Personen-Porträt wurde zwar ein zusätzliches Gesicht erkannt, die Gesamtleistung liegt dennoch unter der vom einfachen Dlib HOG.
- **Dlib CNN:** Dlib CNN, konkret `mmod_human_face_detector.dat`, ist das langsamste aller getesteten Modelle. Bereits bei Bildern mit einer Größe von 998×1314 Pixeln liegt die Bearbeitungszeit bei etwa 20 Sekunden, während alle anderen Modelle unter einer Sekunde bleiben. Die Ergebnisse der Stichprobentests sind jedoch fehlerfrei: Alle Gesichter in Einzel-, Zwei- und Drei-Personen-Porträts wurden erkannt, ohne False-Positives. Im Gruppenbild wurde hingegen nichts erkannt. Aufgrund dieser hohen Zuverlässigkeit wird Dlib CNN weiter getestet. Ob sich die lange Bearbeitungszeit durch eine entsprechend hohe Genauigkeit rechtfertigt, muss sich noch zeigen. Bei hochauflösenden Bildern (1200×1593

Pixel) bleiben die Ergebnisse gleich, allerdings verdoppelt sich die Bearbeitungszeit auf ca. 40 Sekunden. Bei geringer Auflösung (400×531 Pixel) arbeitet das Modell deutlich schneller, erkennt jedoch nur das Gesicht im Einzelporträt. Bei mittlerer Auflösung werden wieder alle Gesichter erkannt, mit Ausnahme des Nebengesichts im Drei-Personen-Porträt. Anders als viele andere Modelle lässt Dlib CNN vermuten, mit höheren Auflösungen besser zu funktionieren, wobei es zu einer erheblich höheren Rechenzeit kommt.

- **MTCNN:** MTCNN besteht die Stichprobentests für Einzel-, Zwei- und Drei-Personen-Porträts fehlerfrei. Im Gruppenbild werden einige Gesichter erkannt, allerdings nur solche mit Confidence > 0.5 . Es gibt keine False-Positives. Somit wird MTCNN weiter getestet. Bei höherer Auflösung bleibt der durchschnittliche Confidence-Wert gleich, es treten jedoch in einzelnen Fällen 1–3 False-Positives auf. Im Gruppenbild werden mehr Gesichter erkannt (6 von 13), sowie keine neuen False-Positives. Einige dieser False-Positives haben hohe Confidence-Werte (0.85), was die Auswahl eines optimalen Schwellwerts erschwert. Bei geringer Auflösung erkennt MTCNN weiterhin zuverlässig alle Gesichter in Einzel-, Zwei- und Drei-Personen-Porträts mit einer Confidence von 1.0 und ohne False-Positives. Im Gruppenbild wird nichts markiert. Bei mittlerer Auflösung bleibt das Verhalten stabil – im Gruppenbild werden vier Gesichter mit ca. 0.85 Confidence erkannt.
- **Yunet:** Das Yunet-Modell (`face_detection_yunet_2023mar.onnx`) ist ohne Konfiguration sehr strikt. Es erkennt Gesichter mit hoher Präzision, aber auch sehr selektiv. Nach Einstellung des Confidence-Wertes auf 0.8 werden alle Gesichter in Einzel-, Zwei- und Drei-Personen-Porträts erkannt, jedoch nur wenige im Gruppenbild. Ob 0.8 optimal ist, wird sich in weiteren Tests zeigen. Bei höherer Auflösung verbessert sich die Confidence der erkannten Gesichter leicht. Im Gruppenbild verändert sich die Erkennung – ein vorher erkanntes Gesicht fehlt, ein anderes wird zusätzlich erkannt. Es gibt weiterhin keine False-Positives. Bei geringer Auflösung erkennt Yunet alle Gesichter bis auf die im Gruppenbild, wo kein Gesicht markiert wird. Auch bei mittlerer Auflösung bleiben die Ergebnisse identisch zu jenen bei niedriger Auflösung.
- **RetinaFace (CPU):** Verwendet wird RetinaFace mit ResNet-50-Backbone und CPU-Nutzung, da GPU lediglich die Geschwindigkeit, nicht aber die Genauigkeit erhöht. So bleiben die Ergebnisse vergleichbar. RetinaFace erzielt in den Stichprobentests die besten Ergebnisse aller Modelle: Alle Gesichter in Einzel-, Zwei- und Drei-Personen-Porträts werden ohne False-Positives erkannt. Im Gruppenbild werden 10 von 13 Gesichtern identifiziert. Es treten zwei False-Positives auf, bei denen Pferdegesichter markiert wurden. Bei höherer Auflösung bleiben die Ergebnisse stabil, die Confidence-Werte ändern sich nur minimal.

Im Gruppenbild wurde ein schwach erkanntes Gesicht nicht erneut erkannt. Bei geringer Auflösung erkennt RetinaFace weiterhin alle Gesichter in Einzel-, Zwei- und Drei-Personen-Porträts korrekt. Die Anzahl erkannter Gesichter im Gruppenbild sinkt auf 8 von 13, bleibt damit jedoch konkurrenzlos hoch. Auch bei mittlerer Auflösung bleiben die Erkennungsergebnisse stabil und präzise.

3.2 Größentests

Das CDA speichert Bilder von Werken in drei Größen: Small, Medium und Large. Da jedes Werk sein eigenes Format hat, haben nicht alle Bilder die exakt gleiche Auflösung. Small-Bilder haben 400 px Breite, Medium-Bilder haben 600 px Breite, Large-Bilder haben 1200 px Breite. Während der Stichprobentests wurde bereits ein wenig mit Bildern verschiedener Auflösungen experimentiert, die Ergebnisse waren jedoch nicht sehr übersichtlich oder eindeutig, sodass nochmal alle verbliebenen Modelle mit allen drei Größen getestet werden.

Als Ergebnis wird festgehalten, dass Large-Bilder sich am besten eignen. Diese haben den Nachteil, dass sie die meisten False-Positives generieren. Dennoch werden auf ihnen die meisten Gesichter erkannt, was in diesem Anwendungskontext das relevantere Kriterium ist. Des Weiteren wird davon ausgegangen, dass einige dieser False-Positives darauf zurückzuführen sind, dass die Confidence-Grenzwerte noch nicht optimiert wurden. Dlib HOG wird verworfen, da es während der Tests im Vergleich deutlich mehr False-Positives gezeigt hat und es bereits genug Modelle mit akzeptablen Werten gibt, sodass das Modell als Kandidat nicht mehr relevant ist.

3.3 Confidencetests

Zur bestmöglichen Repräsentation der Modelle werden alle Confidence-Grenzwerte bzw. Score-Thresholds der Modelle angepasst. Zunächst wird kein Grenzwert gesetzt und die Ausgaben beobachtet. Dabei wird explizit auf die Werte des kleinsten Confidence-Wertes für ein erkanntes Gesicht und des höchsten Confidence-Wertes für ein False-Positiv geachtet. Sollten sich diese überschneiden, so wird mithilfe des Kontextes der anderen Ergebnisse bestimmt, welcher Confidence-Grenzwert sich für das jeweilige Modell am besten zur Gesichtserkennung auf historischen Gemälden eignet. Hier sei noch einmal angemerkt, dass jedes Modell seine eigene Implementierung für Confidence-Werte hat. Ein höherer Wert bedeutet jedoch nicht zwangsläufig, dass ein Modell besser ist – er gibt lediglich an, wie sicher sich das Modell ist, ein Gesicht erkannt zu haben.

- **Caffe:** Die Ergebnisse von Caffe waren bei -1 Confidence-Grenzwert viel zu unkenntlich, also wurde der Test wiederholt. Im zweiten Anlauf wurden alle Markierungen mit einem Confidence-Wert $> 0,1$ akzeptiert, da alle Ergebnisse unter diesem Wert aufgrund der riesigen Mengen an False-Positives unbrauchbar wären. Minimaler Confidence-Wert für erkannte Gesichter beim Einzelporträt war 0,11, während der maximale False-Positive-Wert 0,21 ist. Dieser Wert wurde bei einem Bild gemessen, alle anderen erreichten einen Wert von 1,00 oder ca. 0,80. Da nur ein Bruchteil des Cranach-Archivs aufgrund der limitierten Zeit getestet werden kann, wird der Wert nicht als Ausreißer, sondern als gleichwertig betrachtet, da es im restlichen Archiv bestimmt ähnliche Bilder zum 0,11-Kandidaten gibt. Ergebnisse des Tests mit Zwei-Personen-Porträts waren deutlich weniger konstant als im letzten Test. Geringster Wert für ein erkanntes Gesicht war 0,14, 0,13 wenn Nebengesichter mitgezählt werden. Allerdings gab es mit 0,4 ein großes False-Positive, sowohl vom Wert als auch von der Fläche her. Der Test mit Drei-Personen-Porträts verlief sehr schlecht für Caffe. Der geringste erkannte Wert ist 0,1, es ist aber auch sehr wahrscheinlich, dass der eigentliche Wert noch geringer ist, da in manchen Bildern nicht alle Gesichter erkannt wurden. Auch gab es eine sehr große Menge an False-Positives, höchster Wert in diesem Test: 0,28. Mit minimaler Gesichtserkennung bei Confidence von 0,1 und maximalem False-Positive-Confidence von 0,4 ist das Bestimmen eines optimalen Grenzwertes sehr schwierig. Nach mehreren weiteren Tests wurde entschieden, den Grenzwert auf 0,14 zu setzen. Leider werden damit einige Ergebnisse von erkannten Gesichtern verworfen, allerdings gibt es bis 0,13 noch große Mengen an False-Positives, was die Gesamtergebnisse brauchbarer macht.
- **Dlib CNN:** Dlib CNNs Tests mit Einzelporträts haben selbst bei einem Grenzwert von -1 keine False-Positives gezeigt. Der minimale Wert für erkannte Gesichter lag bei 1,03. Im Test mit Zwei-Personen-Porträts gab es weiterhin keine False-Positives, allerdings wurden auch einige Neben- sowie Hauptgesichter nicht erkannt. 0,70 war der kleinste gemessene Wert für ein Gesicht. Weiter wurden im Test mit Drei-Personen-Porträts keine False-Positives erkannt, aber auch teils manche Gesichter nicht erkannt. Der geringst gemessene Wert für ein erkanntes Gesicht war in diesem Test 0,97, womit 0,7 der gesamt niedrigste Wert für Dlib CNN ist. Da bisher keine False-Positives erkannt wurden und Dlib CNN sehr streng scheint, wird der verwendete Grenzwert für Dlib CNN 0,6 sein.
- **MTCNN:** MTCNN ist sehr sicher durch den Test mit Einzelporträts gekommen. Keine False-Positives und ein minimaler Confidence-Wert bei der Gesichtserkennung von 0,99. Generell sind alle Werte konstant zwischen 0,99 und 1,00. Im Test mit Zwei-Personen-Porträts wurden weiterhin keine False-Positives erkannt, allerdings ebenso einige Gesichter nicht. Der geringst gemessene Wert bei der

Gesichtserkennung beträgt 0,82. Die Ergebnisse des Tests mit Drei-Personen-Porträts erschweren das Bestimmen eines endgültigen Grenzwertes für MTCNN. Der höchst gemessene Wert für False-Positives beträgt 0,95, generell haben – wie zuvor erwähnt – alle False-Positives von MTCNN sehr hohe Confidence-Werte. Die Entscheidung, wo der Grenzwert gesetzt werden soll, fällt sehr schwer. Die beiden Möglichkeiten sind 0,82 für den minimalen Wert eines erkannten Gesichts oder 0,91, um den größten Teil der False-Positives auszusortieren. Der Grund, warum 0,82 überhaupt in Erwägung gezogen werden kann, liegt daran, dass die markierten False-Positives bisher immer kleine Bereiche waren, die ignoriert werden könnten. Der Grenzwert wird vorläufig auf 0,82 gesetzt; je nach Ergebnis wird das Modell mit 0,91 Grenzwert getestet und entschieden, welcher Wert verwendet werden soll.

- **Yunet:** Yunet hat bei seinem Einzelporträt-Test keine False-Positives gehabt und ein Gesicht übersehen, ansonsten betrug der minimale Wert für ein erkanntes Gesicht 389,12. Der Test mit Zwei-Personen-Porträts verlief mit einem ähnlichen Ergebnis und einem minimalen Wert von 233,79. Der letzte Confidence-Test von Yunet mit Drei-Personen-Porträts ergab weiterhin keine False-Positives und auch den gesamt minimalen Wert für erkannte Gesichter von 154,46. Da Yunet wie Dlib CNN eher streng zu sein scheint, wird der Grenzwert für Yunet auf 125 gesetzt, um weitere mögliche erkannte Gesichter zuzulassen.
- **RetinaFace (CPU):** RetinaFace hat als minimalen Confidence-Wert 0,74 für Gesichter im Test mit Einzelporträts, ohne Fehler. Die Ergebnisse für Zwei-Personen-Porträts zeigen einen minimalen Wert von 0,71 und keine False-Positives, es wurden aber auch manche Gesichter nicht erkannt. Das Ergebnis des Tests mit Drei-Personen-Porträts entspricht dem gesamt minimalen Wert von RetinaFace: 0,68 Confidence für Gesichter und 0,52 für Nebengesichter. Da in keinem Test False-Positives markiert wurden, wird der absolut geringst gemessene Wert 0,52 als Grenzwert verwendet.

3.4 Vergleich

Keines der Modelle hat es geschafft, jedes Gesicht aus den bisherigen Tests fehlerfrei zu erkennen. Somit werden im letzten Test alle fünf verbliebenen Modelle gleichzeitig getestet. Dabei werden für jedes Modell die Anzahl von False-Negatives, False-Positives und die Anzahl von Gesichtern, die nur von diesem Modell erkannt wurden, gezählt. Ziel ist es, ein Modell oder eine kleine Auswahl von Modellen herauszuarbeiten, die genutzt werden kann, um so viele Gesichter wie möglich auf historischen Gemälden zu erkennen.

Die Entscheidung, welches oder welche Modelle sich zur Weiterentwicklung eignen, ist keine einfache, da es kein fehlerfreies Modell gibt, das einfach ausgewählt werden könnte. Auch ist es so, dass es in manchen Bildern vorkommen kann, dass mehrere Gesichter von unterschiedlichen einzelnen Modellen erkannt werden. Der ursprüngliche Ansatz war, je nach Bild den Confidence-Grenzwert und das Modell zu wählen. Hierbei bleibt jedoch weiterhin das Problem bestehen, dass kein einzelnes Modell alle Gesichter eines Bildes erkennt. Der neue Ansatz ist nun, je nach Bild mehrere Modelle zu- und abzuschalten, um auf diese Weise mehr Gesichter zu erkennen oder False-Positives zu vermeiden. Unter Berücksichtigung des angestrebten Anwendungszwecks fällt die finale Wahl auf: RetinaFace (CPU), MTCNN und CNN. RetinaFace ist das Modell, mit den wenigsten False-Negatives von allen Modellen. Zusätzlich hat es auch keine False-Positives generiert, womit es als eine gute Basis dient. MTCNN wurde gewählt, da es zusammen mit CNN eines der Modelle ist, welches Gesichter markiert hat, die von keinem anderen Modell markiert wurden. Dabei wird der Confidence-Grenzwert von 0.82 beibehalten, da viele der besonders kleinen Gesichter nur von MTCNN erkannt werden und unter den alternativen Wert von 0.91 fallen. Da MTCNN abgeschaltet werden kann, falls die False-Positives stören, ist es dennoch eine gute Ergänzung zur Modellauswahl. CNN wird aus dem gleichen Grund wie MTCNN hinzugezogen. Es hat zwar keine False-Positives, die zu Problemen werden könnten, sollte jedoch nur bei Bedarf eingesetzt werden, da es die Rechenzeit deutlich erhöht. Weiter enthalten die Ergebnisse des Modells viele False-Negatives.

4 Implementierung

Ursprünglich war der Ansatz, ein eigenständiges Programm zu entwickeln, welches selbstständig Wasserzeichen auf Bildern platziert. Dabei sollten die Gesichter ausgespart werden, um die Ästhetik des Bildes nicht zu stören. Aufgrund der limitierten Zeit wird dieser Ansatz auf das Markieren von Gesichtsbereichen reduziert. Diese Reduktion erlaubt eine flexiblere Implementierung, indem statt eines eigenständigen Programms ein Modul erstellt wird. Damit ergeben sich mehr Anwendungsmöglichkeiten, die über das einfache Platzieren von Wasserzeichen hinaus gehen.

Bisher wurde für das Laden der Bilder in dieser Arbeit `cv2` genutzt, genauer die Funktion `cv2.imread()`, welche Bilder im BGR-Format lädt. Bei der Implementierung ist allerdings aufgefallen, dass diese Funktion Probleme mit Umlauten hat. Aus diesem Grund wurde auf die Funktion `Image.open().convert("RGB")` von PIL gewechselt. Bemerkenswert ist, dass RetinaFace mit RGB-Input weniger False-Negatives generiert. Dieses Ergebnis ist insofern unerwartet, da BGR als Input-Format angegeben wurde (lounging-lizard & nttstar, 2025). Die Abweichung der Confidence-Werte war mit etwa 0,02 gering, weshalb RGB nun als Eingabeformat für RetinaFace verwendet wird.

Hauptfunktion des Moduls ist `cranach_detector()`, welche gleichzeitig mit RetinaFace, MTCNN und Dlib CNN Bereiche mit Gesichtern auf Bildern markiert und diese in einer Liste zurückgibt. Das Modul liefert Funktionen mit, die überprüfen lassen, ob sich eine Position oder Fläche auf einem Gesicht befindet. Sollten diese allerdings nicht ausreichen, so kann man die ausgegebene Liste für eigene Funktionen verwenden.

Um möglichst viele verschiedene Anwendungszwecke abzudecken, akzeptiert die Funktion `cranach_detector()` verschiedene Arten von Eingaben:




- None: Lässt Nutzer*innen einen Ordner manuell über eine grafische Benutzeroberfläche auswählen, dessen Inhalte dann iteriert werden.
- File: Es kann ein Pfad zu einer einzelnen Bilddatei hinterlegt werden, welche bearbeitet werden soll.
- Path: Wird stattdessen der Pfad eines Ordners übergeben, so werden alle Bilder dieses Ordners zu einer Liste verarbeitet, durch die dann iteriert wird.

- List: Falls die interne Funktion für den jeweiligen Anwendungskontext nicht ausreicht oder bereits eine Liste mit Bildpfaden vorliegt, kann diese direkt übergeben werden.

Weiter können auch alle Modelle einzeln zu- oder abgeschaltet werden – sowohl beim Funktionsaufruf als auch während der Laufzeit. RetinaFace ist dabei standardmäßig aktiviert und alle anderen Modelle deaktiviert. Da RetinaFace bereits gute Ergebnisse liefert, reicht es in den meisten Fällen aus. Dies spart Rechenzeit und verringert die Anzahl von False-Positives.

Auch wenn die Confidence-Grenzwerte optimiert wurden, kann es dennoch sein, dass bestimmte Bilder andere Werte benötigen. Aus diesem Grund wurde die Möglichkeit hinzugefügt, die Confidence-Werte während der Laufzeit zu konfigurieren, um für jedes Bild und jedes Modell einzeln Grenzwerte festzulegen, sofern dies erforderlich ist.

Die von den Modellen erkannten Bereiche werden als Dict in einer Liste gespeichert, mit den Attributen `x`, `y`, `w` (width), `h` (height), `model`, Confidence und `image_name`. Diese wird von der Funktion `cranach_detector()` nach Durchlaufen aller übergebenen Bilder zurückgegeben. Dabei wird die Liste auch intern in einer Variable gespeichert, um sie mit anderen internen Funktionen wie `position_isIntersecting()` und `area_isIntersecting()` direkt zu nutzen. Sollte dies nicht ausreichen, kann auf die ausgegebene Liste zurückgegriffen werden. Diese kann dann beispielsweise in eigenen Python-Funktionen weiterverarbeitet werden oder in das JSON-Format überführt werden für externe Anwendungen. Das Format, in dem RetinaFace seine Ergebnisse ausgibt, unterscheidet sich leicht von MTCNN und Dlib CNN. RetinaFace gibt als Ergebnis die Eckpunkte eines Rechtecks, während die beiden anderen Modelle einen Startpunkt inklusive Breite und Höhe angeben. Um die Arbeit mit den gemeinsamen Ergebnissen der Modelle zu erleichtern, werden die Ergebnisse von RetinaFace in das Format von MTCNN und Dlib CNN überführt, auch wenn das Format von RetinaFace leichter weiterzuverarbeiten ist. Der Grund, warum Funktionen in diesem Modul dennoch Eingaben im Format `x`, `y`, Breite und Höhe erwarten, liegt darin, dass im Frontend eher mit Werten in diesem Format gearbeitet wird. Zudem sind nicht alle Overlays rechteckig, sodass Eckpunkte nur schwer bestimmt werden könnten.

Die Farben der markierten Gesichter wurden für das Package angepasst, um auch für Menschen mit Farbsehschwäche unterscheidbar zu bleiben. Als Basis für die Auswahl der Farben diente das Paper von Petroff, 2021. Gewählt wurden (87, 144, 252)  Blau für RetinaFace, (248, 156, 32)  Orange für MTCNN und (228, 37, 54)  Rot für Dlib CNN. Blau ist die Farbe, die sich am besten eignet, um von anderen unterschieden werden zu können. Sie wurde bewusst für RetinaFace gewählt, da das Modell als Basis dient und so am häufigsten mit den anderen Farben in Kontakt kommt. Allerdings können die gewählten Farben auf bestimmten Bildtypen eine

reduzierte Erkennbarkeit aufweisen. Aus diesem Grund wurde die Option hinzugefügt, Markierungen in dunkleren Farben anzeigen zu lassen.

Sollten mehrere Modelle dasselbe Gesicht markieren, wird nur der Bereich mit der geringsten Fläche als gültige Gesichtserkennung beibehalten. Dafür wird überprüft, ob sich zwei markierte Bereiche überschneiden, und die Fläche mit der Fläche des kleineren Bereichs verglichen. Wenn der überschneidende Bereich 75% oder mehr der Fläche des kleineren Bereichs ausmacht, wird der größere Bereich verworfen. So werden möglichst genaue Ergebnisse behalten und vermieden, dass Markierungen von Gesichtern, die nah beieinander liegen, verworfen werden.

Um zu überprüfen, ob eine Position oder ein Bereich auf einem Gesicht liegt, wurden die Funktionen `position_isIntersecting()` und `area_isIntersecting()` implementiert. Als Argumente erwarten beide Funktionen ein Tupel: (x, y) im Fall von `position_isIntersecting()` und $(x, y, \text{width}, \text{height})$ bei `area_isIntersecting()`, zusammen mit dem Dateinamen eines Bildes, welches zuvor schon einmal von `cranach_detector()` bearbeitet wurde, damit intern die markierten Bereiche zur Verfügung stehen. Die Funktionen geben `True` zurück, sollte sich die Position oder der Bereich auf einem Gesicht befinden. Optional kann auch eine Margin angegeben werden, falls ein gewisser Abstand zu Gesichtern eingehalten werden soll. Befindet sich ein Gesicht innerhalb der Margin um den Bereich oder Punkt, gibt die Funktion `True` zurück.

5 Fazit

Das Ziel der Arbeit, Gesichtserkennungsmodelle zu finden, welche für historische Gemälde geeignet sind, konnte größtenteils erfüllt werden. Das Modell RetinaFace alleine ist in der Lage, beinahe alle Gesichter auf getesteten Einzel-, Zwei- und Drei-Personen-Porträts zu finden. Zusammen mit MTCNN und Dlib CNN konnte ein Python-Modul entwickelt werden, welches in der Lage ist, alle Gesichter auf den getesteten Porträts zu markieren. Dieses bietet die Möglichkeit, Modelle je nach Bild an- oder abzuschalten sowie ihre Confidence-Grenzwerte anzupassen. Damit ist das Modul in der Lage, flexibel auf verschiedene Werke angewendet zu werden, um bestmögliche Ergebnisse zu erzielen. Des Weiteren konnten Funktionen implementiert werden, die die prüfen, ob sich ein Overlay auf einem Gesicht befindet. Alternativ kann die ausgegebene Liste von Gesichtsbereichen verwendet werden, um eigene Funktionen zu implementieren. Sollen die Werte außerhalb von Python verwendet werden, kann die Liste zu einer JSON-Datei formatiert werden, um sie in anderen Anwendungen zu nutzen.

Als mögliche Weiterführung der Arbeit kann das GUI des Moduls überarbeitet werden, um die Nutzerfreundlichkeit zu optimieren. Ein mögliches Feature, das hinzugefügt werden könnte, wäre ein Zähler, welcher zeigt, wie viele Bereiche vom jeweiligen Modell markiert wurden. So kann schneller von Nutzer*innen erkannt werden, ob sich auf einem Bild False-Positives befinden, wenn die Zahl der Bereiche die tatsächliche Anzahl der Gesichter auf dem Bild übersteigt. Die dafür notwendigen Funktionen sind bereits implementiert, jedoch wurde aus Zeitmangel noch keine grafische Darstellung dafür umgesetzt. Auch könnte eine Funktion implementiert werden, welche die Liste der markierten Bereiche direkt zu einer JSON-Datei formatiert, um sie auch in anderen Programmen nutzen zu können.

Eine weitere mögliche Fortführung der Arbeit wäre ein intensiveres Testen mit Fokus auf Gruppenbilder. Die Tests an Gruppenbildern haben bereits gezeigt, dass es deutlich schwieriger ist, auf ihnen zuverlässig Gesichter zu finden. Aufgrund des Mangels an Ressourcen (Zeit) wurden spätere Tests an Gruppenbildern verworfen, um den Fokus auf Porträts setzen zu können. Dies geschah mit der Überlegung, dass das Platzieren eines Overlays auf dem Gesicht eines Porträts die Ästhetik des Bildes stärker beeinflusst. Würde man die Forschung mit Gruppenbildern fortsetzen wollen, so müssten weitere Modelle, die nicht in dieser Versuchsreihe getestet wurden,

herangezogen werden. Alternativ könnte in Erwägung gezogen werden, ein eigenes Modell für diesen Anwendungszweck zu trainieren oder zu entwickeln.

6 Formaler Aufbau

In diesem Kapitel finden Sie grundlegende Hinweise zum formalen Aufbau Ihrer Arbeit.

6.1 Reihenfolge

Eine wissenschaftliche Arbeit besteht in der Regel aus den folgenden Teilen:

1. Deckblatt
2. Kurzfassung/Abstract (optional)
3. Inhaltsverzeichnis
4. Abbildungs- und Tabellenverzeichnis (auch am Ende üblich)
5. Abkürzungsverzeichnis (auch am Ende üblich)
6. Einleitung
7. Hauptteil
8. Zusammenfassung/Fazit
9. Literaturverzeichnis
10. Anhänge (optional)
11. Erklärung

6.2 Deckblatt

Das Deckblatt beinhaltet: Titel der Arbeit, Art der Arbeit, Verfasser*in, Matrikelnummer, Abgabetermin, Betreuer*in sowie Zweitgutachter*in. Das Deckblatt wird bei Arbeiten, die länger sind als 15 Seiten, bei der Seitenanzahl zwar mitgezählt, jedoch nicht nummeriert.

6.3 Inhaltsverzeichnis

Wir empfehlen eine Dezimalgliederung wie in diesem Dokument angelegt. Werden innerhalb eines Kapitels Unterüberschriften verwendet, müssen mindestens zwei vorhanden sein: wo ein 2.1 ist, muss es ein 2.2 geben.

Das Inhaltsverzeichnis enthält immer die Seitenangaben zu den aufgelisteten Gliederungspunkten; es wird dabei aber selbst nicht im Inhaltsverzeichnis aufgelistet. Die Seiten, die das Inhaltsverzeichnis selbst einnimmt, können römisch gezählt werden.

Für eine Abschlussarbeit ist eine Gliederungstiefe von wenigstens drei Ebenen üblich. In der Regel werden nur bis zu vier Ebenen vorne im Inhaltsverzeichnis abgebildet. Hier sollten Sie aber unbedingt die Gepflogenheiten in Ihrem Fach berücksichtigen und ggf. in Erfahrung bringen.

6.4 Abbildungsverzeichnis und Tabellenverzeichnis

Abbildungen und Tabellen werden in entsprechenden Verzeichnissen gelistet. In dieser Vorlage erscheinen sie direkt nach dem Inhaltsverzeichnis. Dann können die entsprechenden Seiten römisch gezählt werden. Die Verzeichnisse können jedoch auch am Ende der Arbeit vor oder hinter dem Literaturverzeichnis stehen. Dann werden sie regulär mit Seitenzahlen versehen.

6.5 Abkürzungsverzeichnis

Die Zahl der Abkürzungen sollte übersichtlich bleiben. Das Abkürzungsverzeichnis enthält lediglich wichtige fachspezifischen Abkürzungen in alphabetischer Reihenfolge, insbesondere Abkürzungen von Organisationen, Verbänden oder Gesetzen. Gängige Abkürzungen wie „u. a.“, „z. B.“, „etc.“ werden nicht aufgenommen.

Zur technischen Umsetzung mit L^AT_EX vergleiche auch Abschnitt 7.18.

6.6 Literaturverzeichnis

Das Literaturverzeichnis wird alphabetisch nach Autorennamen geordnet. Es enthält alle im Text zitierten Quellen – und nur diese. Mehrere Schriften einer Person werden nach Erscheinungsjahr geordnet. Schriften derselben Person aus einem Erscheinungsjahr müssen Sie selbst unterscheidbar machen. In den Ingenieurwissenschaften wird

zusätzlich häufig ein Nummern- oder Autorenkürzel dem Namen in eckigen Klammern voran-gestellt. Mehr hierzu und weitere wichtige Regeln des Zitierens lernen Sie in den E-Learning-Kursen des Schreibzentrums¹ kennen.

Zur Verwaltung der verwendeten Literatur eignen sich entsprechende Softwaretools wie Citavi oder Zotero, die mit verschiedenen Textverarbeitungsprogrammen kompatibel sind.

6.7 Rechtschreibung, Grammatik

Achten Sie bei der Abgabe Ihrer Arbeit auf ein einwandfreies Deutsch bzw. Englisch. Wenn Fehler die Lesbarkeit beeinträchtigen, kann sich dies durchaus negativ auf die Note auswirken. Nutzen Sie daher unbedingt die Rechtschreibprüfung Ihres Textverarbeitungsprogramms, auch wenn diese nicht alle Fehler erkennt.

Für alle, die sich bei diesem Thema unsicher fühlen, empfehlen wir die E-Learning-Kurse des Schreibzentrums². Wenden Sie sich ggf. auch an die Beauftragte für Studierende mit Beeinträchtigung³.

6.8 Umfang der Arbeit

Alle Fächer nennen verbindliche Angaben zu Unter- und Obergrenzen, die in der Regel eingehalten werden müssen. Verzeichnisse und Anhänge werden dabei in aller Regel nicht mitgezählt. In Einzelfällen – insbesondere bei empirischen Arbeiten – können abweichende Vereinbarungen mit der Betreuungsperson getroffen werden.

¹https://ilu.th-koeln.de/goto.php?target=cat_52109&client_id=thkilu

²https://ilu.th-koeln.de/goto.php?target=cat_52109&client_id=thkilu

³https://www.th-koeln.de/studium/studieren-mit-beeintraechtigung_169.php

7 Gestaltung: Textsatz mit \LaTeX

Mit \LaTeX ist es verhältnismäßig einfach, Dokumente zu erstellen, die professionellen Ansprüchen genügen. Ein entscheidender Vorteil ist, dass der Nutzer fast nur den Inhalt beisteuert, während die korrekte äußere Form dann automatisch erzeugt wird. \LaTeX basiert auf \TeX , das von Donald Knuth entwickelt wurde **knuth:tex**. Einige weitere Vorteile gegenüber gängiger Textverarbeitung:

Frei/Plattformunabhängig: Bei \LaTeX handelt es sich um freie Software. Es wird kein proprietärer Editor benötigt, um \LaTeX -Dokumente zu schreiben. Tatsächlich können die Dokumente auf *jedem* Rechner mit *jedem* beliebigen Editor bearbeitet werden.

Reines Textformat: Der Quelltext – die **tex**-Datei – ist ein reines Textformat. Dadurch eignen sich \LaTeX -Dokumente auch hervorragend zur Versionskontrolle mit beispielsweise git. Dies wiederum ermöglicht eine effiziente Zusammenarbeit mehrerer Autor*innen.

Aufteilen großer Dokumente: Der Quelltext großer Dokumente, wie beispielsweise von Projektarbeiten, kann auf mehrere Dateien aufgeteilt werden. So können beispielsweise mehrere Personen an jeweils einem eigenen Kapitel arbeiten. Aufgrund der beiden oberen Punkte wird es auch nicht zu Kompatibilitätsproblemen kommen.

Trennen von Layout/Inhalt: Mit \LaTeX kann man explizit das Layout für das gesamte Dokument festlegen – oder die verwendete Dokumentklasse kümmert sich implizit darum. Zeitgemäße Textverarbeitung bietet mit Formatvorlagen zwar entsprechende Funktionalitäten; aber durch den programmatischen Ansatz mit \LaTeX kann noch genauer Einfluss auf das Layout genommen werden. Anschließend kann die ganze Konzentration auf das Schreiben gelegt werden.

Professionelles Ergebnis: Ein mit \LaTeX erzeugtes Dokument schaut professioneller aus, als ein entsprechendes, mit Textverarbeitung erzeugtes Dokument. Das gilt vor allem für mathematiklastige Dokumente. Aber auch andere Dokumente können von einem einheitlichen Layout, gleichmäßigem Grauwert des Fließtexts, stimmigeren Seitenumbrüchen und hochwertigen Vektorgraphiken profitieren – um nur mal einige Punkte zu nennen.

Vielseitig einsetzbar: Mit L^AT_EX können nicht nur „einfache“ Dokumente erzeugt werden. Es existieren unzählige Dokumentklassen, die beispielsweise auch das Erstellen von Präsentationen oder Postern ermöglichen.

In den folgenden Abschnitten 7.1 bis 7.18 wird auf diverse Aspekte eingegangen, die Sie beim Erstellen Ihres Dokuments berücksichtigen sollten.

7.1 Unter der Haube

Sie definieren in Ihren TEX-Dokumenten, was Ihre Inhalte sind (Text mit Gliederung, Bilder, Tabellen, Literaturverweise, ...) und wie diese jeweils grob aussehen sollen (z. B. Platzierung von Abbildungen mittels *Gleitumgebungen*, vgl. Abschnitt 7.6).

Beim Erstellen des endgültigen Dokuments wendet L^AT_EX „unter der Haube“ eine ganze Menge Regeln an, die festlegen, wie das alles bestmöglich umgesetzt werden kann. Diese Regeln betreffen z. B. den Anteil von Text und Bildern pro Seite, Abstände innerhalb von Zeilen, aber auch Sonderfälle wie das Vermeiden einzelner Zeilen eines Abschnitts alleine auf einer Seite (sog. „Schusterjungen“ oder „Hurenkinder“).

Im Ergebnis kann es also passieren, dass z. B. Ihre Abbildungen „springen“, während Sie an Ihrem Text arbeiten. Das hat im Zweifel alles seine Richtigkeit und kann im Notfall am Ende noch optimiert werden.

In diesem Zusammenhang ist zu vermeiden, in den Gestaltungsprozess einzugreifen, indem z. B. manuell Zeilenumbrüche („`\newline`“ oder „`\\`“) eingefügt werden oder Abstände. Ausnahmen bitte nur in begründeten Fällen wie in dieser Vorlage bei der Gestaltung des Deckblatts.

Weitere Infos dazu, wie Sie mit dieser Vorlage hier weiterarbeiten können, finden Sie in Abschnitt 7.18.

7.2 Überschriften

Wir nutzen in dieser Vorlage das Kapitel („`\chapter`“) als höchste Gliederungsebene. Danach kommen Abschnitte („`\section`“) und Unterabschnitte („`\subsection`“). Diese drei Ebenen werden nummeriert und erscheinen im Inhaltsverzeichnis. Falls Sie Ihren Text weiter gliedern wollen, gibt es noch den „`\paragraph`“-Befehl.

Bitte beachten Sie, dass im Text nie zwei Überschriften direkt aufeinander folgen sollten. Nach einer Überschrift kommt immer erst etwas Text (siehe z. B. die Kapitelanfänge hier auf Seite 20 und Seite 23). Für weitere Hinweise vgl. Abschnitt 6.3.

7.3 Absätze

Stellen im Text, an denen ein neuer Absatz beginnen soll, können im Quellcode durch „`\par`“ markiert werden. Wie diese Absätze im fertigen Dokument genau aussehen, wird durch den Parameter „`\parskip`“ in der Dokumentenklasse bestimmt – dazu mehr in Abschnitt 7.18. Das ist ein großer Vorteil von L^AT_EX: Der Stil kann jederzeit für das gesamte Dokument einfach verändert werden.

Hinweis: Sie erhalten das gleiche Verhalten auch, wenn Sie im Quellcode statt des „`\par`“-Befehls eine leere Zeile stehen lassen. Vielleicht gefällt Ihnen das sogar noch besser.

7.4 Silbentrennung

Die automatische Silbentrennung in L^AT_EX funktioniert grundsätzlich gut. Es kann aber immer mal kleinere Probleme und erwünschtes Verhalten geben. Wenn Sie die Trennung für ein bestimmtes Wort beeinflussen möchten, können Sie mit dem „`\hyphenation`“-Befehl manuell die erlaubten Trennstellen spezifizieren. So kann man insbesondere erreichen, dass bestimmte Wörter nie getrennt werden, was z. B. für Eigennamen unerwünscht sein könnte.

Zum Beispiel werden Wörter, die einen Bindestrich enthalten, *nur* dort getrennt, das kann dazu führen, dass Zeilen nicht richtig dargestellt werden können, was zu einer Warnung führt (siehe Abschnitt 7.17). In solche Fällen müssten Sie im Quellcode manuell zusätzlich Trennstellen angeben.

7.5 Aufzählungen

Nutzen Sie die Umgebungen

- „`\begin{itemize}`“ ... „`\end{itemize}`“
- „`\begin{enumerate}`“ ... „`\end{enumerate}`“
- „`\begin{description}`“ ... „`\end{description}`“
- „`\begin{labeling}`“ ... „`\end{labeling}`“

um schöne Listen zu erstellen. Auch hier gilt, dass das genaue Aussehen im Dokument global eingestellt wird, das können Sie jederzeit verändern, dazu mehr in Abschnitt 7.18.

7.6 Abbildungen

Wenn jemand Ihre fertige Arbeit in die Hände bekommt, kann es gut sein, dass sie/er zunächst grob durchblättert, dabei kaum Text liest, aber die Abbildungen anschaut. Aus dieser Erfahrung entstammt die „Regel“, dass man die wichtigsten Punkte der Arbeit auf diese Weise verstehen können sollte.

Abbildungen stehen nie alleine, sondern werden durch die Unterschrift (*caption*) beschrieben. Dabei sollte alles enthalten sein, was notwendig ist, um die Abbildung zu verstehen. Nur in Ausnahmefällen muss man in der Unterschrift auf den Text verweisen. Umgekehrt muss auf jede Abbildung mindestens ein Mal im Text verwiesen werden, dazu siehe auch Abschnitt 7.12.

In den folgenden beiden Abschnitten wird zwischen Bildern (in Abschnitt 7.6.1) und Vektorgrafiken (in Abschnitt 7.6.2) unterschieden, da es sich um ganz unterschiedliche Techniken handelt, die jeweils passend genutzt werden sollten.

Denken Sie daran, dass nicht alle Menschen alle Farben gleich gut sehen können. Etwa 10 % der Männer in Deutschland sind beispielsweise von einer Rot-Grün-Schwäche betroffen. Vielleicht wird Ihre Arbeit auch auf einem Schwarz-Weiß-Drucker gedruckt. Daher sollten Sie Abbildungen im besten Fall so gestalten, dass sie auch ohne Farben verständlich sind.

7.6.1 Bilder

Bilder können Sie mit „`\includegraphics`“ einbinden. Es reicht (und wird sogar empfohlen!), den Dateinamen ohne Endung und ohne Pfad anzugeben. Beim Kompilieren werden alle Verzeichnisse durchsucht, die im „`\graphicspath`“ angegeben sind. In aller Regel soll ein Bild nicht alleine im Dokument erscheinen, sondern in einer



Abbildung 7.1: Vielleicht handelt es sich hierbei um Kunst?

Umgebung, die die automatische Nummerierung sicherstellt, eine Bildunterschrift hinzufügt und schließlich ermöglicht, dass die Abbildung an einer optimalen Stelle platziert wird (daher auch die Bezeichnung „Gleitumgebung“. In diesem Fall ist das die „`\figure`“-Umgebung.

Für die Umgebung stellen wir ein, wo sie auftauchen darf (dazu siehe auch Abschnitt 7.1). Dabei steht `t` für ganz oben auf der Seite, `b` für ganz unten und `h` für „hier“, was also die Positionierung innerhalb des Texts meint. Falls Sie mal Platz sparen müssen, sind `t` und `b` zu bevorzugen.

Achtung: Viele Inhalte wie Formeln, Code, Diagramme, Visualisierung von Daten, usw. sollten *nicht* als Bild eingefügt werden, sondern in einer passenden Form. Dazu siehe den folgenden Abschnitt über Vektorgrafiken.

7.6.2 Vektorgrafiken

Einfache Abbildungen (z. B. Koordinatensysteme, Ablaufdiagramme, usw.) müssen Sie nicht als Bild einfügen. Stattdessen können diese im Quellcode direkt erzeugen können. Dafür bietet sich das mächtige „`tikz`“-Paket an.

Ein Vorteil ist, dass Ihr Dokument so kleiner bleibt. Aber auch, dass die Abbildungen i. d. R. hübscher aussehen. Das gilt insbesondere beim Betrachten am Bildschirm, da sich Vektorgrafiken beliebig skalieren lassen. Das erlaubt es Ihnen sogar, Ihre Daten,

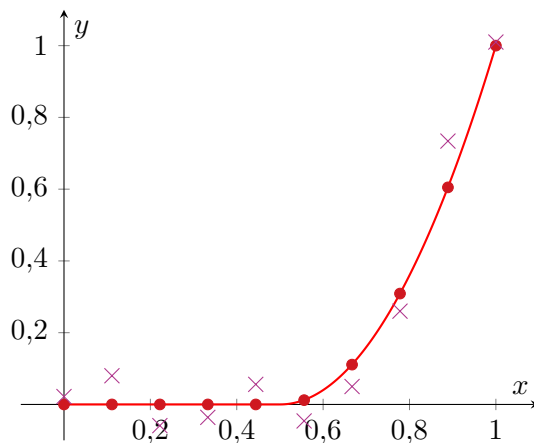


Abbildung 7.2: Eine schöne Grafik, die im Quellcode erzeugt wird!

z. B. aus Experimenten, separat zu halten und entsprechende Abbildungen dynamisch daraus zu generieren. Siehe dazu das Beispiel in Abbildung 7.2.

Sie finden ganz viel Beispiele zu TikZ natürlich im Internet. Außerdem gibt es ein aktuelles Buch **kottwitz:tikz**.

7.7 Tabellen

Grundsätzlich werden Tabellen in L^AT_EX mit der „`\tabular`“-Umgebung gebaut. Das ist dann nur die Tabelle selbst, ohne Nummerierung und ohne Bildunterschrift. Das Prinzip ist also das gleiche wie bei Abbildungen (s.o.): Erst die Umgebung (hier „`\table`“), darin die Tabelle selbst. Vielleicht sind die Befehle `rowcolor` oder

Überschrift links	Überschrift rechts
1	2222
10	222
100	22

Tabelle 7.1: Eine einfache Tabelle

`multicolumn` irgendwann für Sie nützlich. Es gibt noch viele weitere Pakete, die helfen, noch hübschere Tabellen zu gestalten, beispielhaft seien hier nur `array`, `booktabs` und `tabularx` genannt.

7.8 Abbildungs- und Tabellenverzeichnis

Mit L^AT_EX lassen sich Abbildungs- und Tabellenverzeichnis automatisch erstellen. Dabei tauchen alle Einträge entsprechend auf, für die Sie *Gleitumgebungen* korrekt angelegt haben (siehe Abschnitt 7.6.1 und 7.7).

Für diese Verzeichnisse wird standardmäßig der Text aus der `caption` übernommen. Dabei kommt es immer wieder vor, dass diese Beschreibung zu lang ist. Dafür kann mit in der `caption` in eckigen Klammern optional eine kürzere Version angeben. Dazu siehe auch Abschnitt 8.1.4.

7.9 Formeln

Eine der größten Stärken von L^AT_EX ist, dass man viele Möglichkeiten hat, Formeln einfach und schön aufzuschreiben. Das „`amsmath`“-Paket ist in diesem Zusammenhang

besonders beliebt, weil es ganz viele Möglichkeiten bietet. Hier nur ein kleines Beispiel mit der „align“-Umgebung:

$$\sum_{i=1}^n i = 1 + 2 + \dots + (n-1) + n \quad (7.1)$$

$$= (1 + n) + (2 + (n-1)) + \dots \quad (7.2)$$

$$= \frac{1}{2} \cdot (n+1) \quad (7.3)$$

Aber auch einfache Formeln im Text wie $x \in \mathbb{N}$ sind natürlich möglich. Ein häufiger Fehler dabei ist, dass der „Mathe-Modus“ im Text vergessen wird: Wir sprechen über den x -Wert und *nicht* den x-Wert.

Ebenso häufig gibt es den Fehler auch andersrum, also dass Text im Mathe-Modus geschrieben wird.: $a_{falsch} = 42$, aber $a_{richtig} = 42$, vielleicht auch $a_{\text{richtig}} = 42$.

Funktionen wie \sin werden automatisch gut dargestellt, wie in

$$\sin \alpha = \left(\frac{a}{c} \right) \quad (7.4)$$

Die Klammern wurden hier nur eingefügt, um den entsprechenden Mechanismus zu demonstrieren: automatisch wachsende Klammern!

Manchmal wollen Sie einen eigenen „Operator“ benutzen, der optisch gleich aussehen soll. Genau dafür ist der „`\DeclareMathOperator`“-Befehl da, damit kann man fehlende Funktionen wie etwa $\operatorname{sgn}(x)$ hinzufügen.

Es empfiehlt sich, allen mathematischen Symbole, die Sie in Ihrer Arbeit benutzen, im Quellcode sprechende Namen zu geben, das geht am einfachsten mit dem „`\newcommand`“-Befehl. Dann können Sie jederzeit anpassen, wie Sie den Gewichtsvektor \mathbf{w} im gesamten Dokument darstellen wollen oder wie die imaginäre Einheit i mit $i^2 = -1$ aussehen soll. Das setzt natürlich voraus, dass Sie die Bezeichnungen konsequent nutzen. Dadurch wird aber auch Ihr Quellcode besser lesbar!

7.10 Quellcode, Pseudocode

Soll in der Abschlussarbeit ein Ausschnitt vom Quelltext dargestellt werden, so ist die naheliegende Idee, einfach einen Screenshot davon aufzunehmen und via `\includegraphics` als Abbildung einzufügen. Allerdings entpuppt sich diese Idee als schlecht, sobald das fertige Dokument näher herangezoomt wird: Sofort verpixelt der dargestellte Quelltext. L^AT_EX bietet hierfür jedoch eine elegantere Alternative:

Das Paket `listings` zum Darstellen von Quelltext – direkt im Quelltext des L^AT_EX-Dokuments oder aber direkt aus einer externen Datei ausgelesen.

Mithilfe der Umgebung `lstlisting` lässt sich der Quelltext direkt im L^AT_EX-Dokument eingeben. Mit dem Befehl `\lstinputlisting{<Datei>}` lässt sich der Quelltext aus einer externen `<Datei>` auslesen und darstellen. Achtung: Der Pfad zu `<Datei>`, relativ zum L^AT_EX-Dokument, muss hierbei angegeben werden.

Außerdem kann das Layout des im fertigen Dokument dargestellten Quelltexts beeinflusst werden. Hierfür existiert ein *key-value-Interface*, über welches mithilfe spezieller *keys* Einfluss auf Dinge wie beispielsweise die zu verwendende Schriftart oder die Hintergrundfarbe genommen wird. Dazu wird der Befehl `\lstdefinestyle{<Stil>}` verwendet. Dabei ist `<Stil>` eine Liste mehrerer Paare der Form `<key>=<value>`, welche jeweils durch ein Komma voneinander getrennt werden. Ein Beispiel ist in der Präambel dieser Vorlage, in der Datei `definitions.tex` zu finden. Für nähere Informationen sei an dieser Stelle auf die Dokumentation des Paktes verwiesen. Ein Beispiel für solch ein mit der Umgebung `lstlisting` erzeugten Quelltext ist in Abbildung 7.3 gegeben.

```
\lstdefinestyle{myLaTeX}{
  language=TeX,
  basicstyle=\footnotesize\ttfamily,
  frame=single,
  backgroundcolor=\color{gray!10},
}
```

```
\begin{lstlisting}
\lstdefinestyle{myLaTeX}{
  language=TeX,
  basicstyle=\footnotesize\ttfamily,
  frame=single,
  backgroundcolor=\color{gray!10},
}
\end{lstlisting}
```

Abbildung 7.3: Beispiel für ein `listing`, welches mithilfe der Umgebung `lstlisting` erstellt worden ist. Links ist das fertige Listing zu sehen, rechts ist der entsprechende Quelltext dargestellt, der zu ebenjener Ausgabe führt. Zufälligerweise handelt es sich um einen Ausschnitt desjenigen Stils, der in dieser Vorlage verwendet wird.

7.11 Weitere Verzeichnisse

Mithilfe des Pakets `glossaries` lassen sich weitere Verzeichnisse erzeugen. Ein Glossar sowie ein Abkürzungs- oder Symbolverzeichnis lassen sich direkt erzeugen. Außerdem können auch weitere Verzeichnisse definiert werden. Wer das komplette Potential von

`glossaries` ausschöpfen möchte, benötigt Perl auf dem Rechner sowie das Perl-Skript `makeglossaries`. Allerdings existiert auch eine „eingedampfte“ Variante mit etwas eingeschränkter Funktionalität, welche komplett ohne Perl und externes Skript auskommt. Hierzu sei auf die Dokumentation des Pakets verwiesen.

Durch die Option `toc` beim Laden von `glossaries` erscheinen die zusätzlichen Verzeichnisse auch im Inhaltsverzeichnis. Wird weiterhin das Paket `hyperref` verwendet, so sind die im Text ausgegebenen Einträge dieser Verzeichnisse Links, die direkt in das entsprechende Verzeichnis führen. Die Verzeichnisse selbst können dann durch den Befehl `\printglossaries` an der gewünschten Stelle im Dokument ausgegeben werden. Auch hier wird für weiterführende Informationen wieder auf die Dokumentation des Pakets verwiesen.

7.11.1 Glossar erstellen

Ein Glossar kann ohne weitere Vorkehrungen direkt verwendet werden. Ein Eintrag im Glossar kann dann über den Befehl `\newglossaryentry{<Label>}{<Spezifikation>}` definiert werden. Dabei ist `<Spezifikation>` eine *key-value*-Liste. Die wichtigsten *keys* sind `name` und `description`, über welche der Name und die Beschreibung des zu definierenden Eintrags festgelegt werden. Über den Befehl `\gls{<Label>}` kann dann der zuvor definierte Begriff im Text ausgegeben werden. Beispiel gefällig? Der `glsdog` ist der beste Freund des Menschen.

7.11.2 Abkürzungsverzeichnis erstellen

Um ein Abkürzungsverzeichnis verwenden zu können, muss `glossaries` mit der Option `acronym` geladen werden. Eine Abkürzung kann dann in der Präambel über den Befehl `\newacronym{<Label>}{<Abkürzung>}{<Ausgeschrieben>}` definiert werden. Über den Befehl `\gls{<Label>}` kann dann die zuvor definierte Abkürzung im Text ausgegeben werden. Dabei stellt L^AT_EX dann automatisch sicher, dass die Abkürzung bei der ersten Erwähnung im Text ausgeschrieben wird. Bei allen späteren Erwähnungen wird dann nur noch die Abkürzung ausgegeben. Beispiel gefällig? Das ist eine SVM. Und dort ist gleich noch eine SVM.

7.11.3 Symbolverzeichnis erstellen

Um ein Symbolverzeichnis verwenden zu können, muss `glossaries` mit der Option `symbols` geladen werden. Ein Symbol kann dann in der Präambel über den Befehl `\newglossaryentry{<Label>}{<Spezifikation>}` definiert werden. Der Befehl

ist also genau derselbe wie beim Glossar. Zusätzlich muss für *Spezifikation* noch `type=symbols` angegeben werden. Über den Befehl `\gls{<Label>}` kann dann das zuvor definierte Symbol im Text ausgegeben werden. Beispiel gefällig? Die Kraft \vec{F} ist gemäß der folgenden Gleichung definiert:

$$\vec{F} = m \cdot \vec{a}$$

7.12 Verweise

Setzen Sie in Ihrem Quellcode Marken mit dem „`\label`“-Befehl. Aus der Platzierung geht hervor, auf welche Nummerierung sich die Marke bezieht, also etwa Gliederungsebene (siehe Abschnitt 7.2), Tabelle (siehe Abschnitt 7.7), Abbildung (siehe Abschnitt 7.6) oder Gleichung (siehe Abschnitt 7.9). Alle genannten werden nämlich separat nummeriert, das kann am Anfang etwas gewöhnungsbedürftig sein.

Auf die markierten Stellen können Sie dann mit dem „`\ref`“-Befehl verweisen, wobei der eben nur die passende Nummer liefert. Die passende Bezeichnung, z. B. „Abbildung“, müssten Sie dann selbst ergänzen. Daher haben wir hier das Paket `cleveref` eingebunden, das uns den zuletzt genannten Schritt automatisiert.

Auf jede Abbildung und jede Tabelle muss im Text verwiesen werden, es dürfen keine nummerierten Umgebungen einfach „in der Luft hängen“. Im Gegensatz dazu müssen Abschnitte und Gleichungen nicht alle explizit referenziert werden. Sie können aber Ihren Leser*innen helfen, wenn Sie über sinnvolle Verweise nachdenken.

7.13 Besondere Abstände und Zeichen

An Leerzeichen kann grundsätzlich ein Zeilenumbruch (oder sogar Seitenumbruch) erfolgen. In manchen Fällen möchte man das vermeiden, u. a., weil Zeilen nicht mit Zahlen beginnen sollten. Ein typisches Beispiel ist „Lange Straße 123“. Hier benötigt man hinter „Straße“ ein sog. geschütztes Leerzeichen, das mit einer Tilde erzeugt wird.

Genauso wie Gleitumgebungen optimal verteilt werden (vgl. Abschnitt 7.1), werden auch horizontale Abstände zwischen Wörtern und Sätzen in L^AT_EX in jeder Zeile dynamisch angepasst. Dabei werden alle Punkte als Satzende interpretiert. Bei Abkürzungen wie „z. B.“ sieht das nicht schön aus, der Leerzeichen-Abstand ist zu groß. Hier muss in der Mitte manuell ein halbes Leerzeichen erzeugt werden mit „\,“. Wenn Ihnen das Tippen solcher Konstrukte zu umständlich erscheint, können Sie

sich eigene Kommandos wie „\zb“ definieren. Zum Definieren eigener Befehle vgl. Abschnitt 7.9.

Auch bei waagerechten Strichen gibt es, wie bei Leerzeichen, unterschiedliche Längen. Für Gedankenstriche – solche hier – oder wenn „bis“ gemeint ist (wie in 14:00–16:00), reicht der einfache Bindestrich (Minuszeichen) nicht aus, das passende Teichen wird in L^AT_EX einfach durch ein Doppel-Minus erzeugt.

Problematisch im Quellcode sind alle Zeichen, die in L^AT_EX eine Funktion haben: Prozentzeichen %, kaufmännisches Und &, Unterstrich _, geschweifte Klammern { ... } sind typische Beispiele. Diese müssen im Quelltext mit einem *Backslash* eingegeben werden, sonst erhält man Fehlermeldungen.

7.14 Wahl der Grundschriftart

Standardmäßig verwendet L^AT_EX für den Fließtext eine serifenbehaftete Schriftart. Für gedruckte Arbeiten sind serifenbehaftete Schriften vorteilhaft, weil die Serifen die Grundlinie betonen und somit das Auge beim Rücksprung am Zeilenende zum Beginn der nächsten Zeile unterstützt. Außerdem führen die unterschiedlichen Strichstärken zu eindeutigeren Wortbildern und unterstützen somit den Leseprozess.

Wird solch ein Dokument jedoch an einem alten Monitor mit geringer Auflösung betrachtet, so kann es sein, dass die feinen Serifen nicht mehr vernünftig dargestellt werden. In solch einem Fall kann es vorteilhaft sein, eine serifenlose Schrift zu verwenden. Auch kann es sein, dass serifenlose Schriften aus Gründen der Barrierefreiheit bevorzugt werden.

In diesem Fall kann mit dem Befehl `\renewcommand{\familydefault}{\sfdefault}` eine serifenlose Schrift als Grundschriftart festgelegt werden. Wer `lualatex` zum Kompilieren sowie das Paket `fontspec` verwendet, kann außerdem auf alle verfügbaren Schriften zugreifen. Ist auf dem Rechner die Schriftart Arial vorhanden, so kann mit dem Befehl `\setsansfont{Arial}` die Schriftart Arial als serifenlose Schrift festgelegt werden. Am Ende der Datei `definitions.tex` sind die beiden besagten Zeilen zu finden und müssen bei Bedarf nur auskommentiert werden.

7.15 Metadaten für den pdf-Betrachter

Manche pdf-Betrachter können zusätzliche Metadaten, wie Name des Autors, Titel des Dokuments (auch abweichend vom Namen der Datei) oder Schlüsselbegriffe anzeigen. Mit dem Paket `hyperref` lassen sich diese Metadaten mit dem Befehl

`\hypersetup{⟨Einsellungen⟩}` konfigurieren. Dabei ist `⟨Einsellungen⟩` eine *key-value*-Liste. Die wesentlichsten *keys* sind `pdfauthor`, `pdftitle` und `pdfkeywords`. Die Bedeutung dieser *keys* ist selbsterklärend.

Außerdem werden Links im Dokument (bei Verwendung des Pakets `hyperref`) in manchen pdf-Betrachtern als farbige Kästchen hervorgehoben. Diese farbigen Kästchen erscheinen natürlich nicht im gedruckten Dokument. Sie dienen lediglich als Hilfe, dass man nicht „auf gut Glück“ mit dem Cursor über das Dokument fahren muss, bis man den Link gefunden hat. Wenn die farbigen Kästchen stören, so können diese in `\hypersetup` mit `hidelinks` deaktiviert werden.

7.16 Wechsel zwischen ein- und doppelseitigem Layout

Diese Vorlage ist für ein einseitiges Layout optimiert. Dabei sind die linken und rechten Ränder jeweils gleich groß auf allen Seiten, neue Kapitel beginnen unmittelbar auf der nächsten Seite. Wird das fertige pdf-Dokument am Computer betrachtet, sieht das genau richtig aus. Soll das Dokument hingegen doppelseitig ausgedruckt werden, so kann das Layout noch etwas angepasst werden: Typischerweise sind die inneren Ränder dann etwas schmaler als die äußeren Ränder. Und neue Kapitel beginnen jeweils auf einer neuen, rechten Seite – was zu einzelnen Vakantseiten zwischen den Kapiteln führen kann. Für doppelseitig ausgedruckte Dokumente sieht das dann besser aus.

Um das doppelseite Layout zu aktivieren, genügt es bereits, die Auskommentierung der Option `twoside` im optionalen Argument von `\documentclass` zu entfernen.

7.17 Kompilieren

Das Erstellen (Kompilieren) von großen Dokumenten mit L^AT_EX kann verhältnismäßig lange dauern. Da man i. d. R. nur an wenigen Stellen gleichzeitig arbeitet, kann es daher sinnvoll sein, übrige Teile auszukommentieren. Das geht besonders leicht, wenn man Text in getrennte Dateien auslagert und mit dem „`\input`“- und/oder „`\include`“-Befehl einbindet. So bleibt auch das Hauptdokument übersichtlich.

Grundsätzlich sollte das Ziel sein, dass Ihr Dokument ohne Warnungen kompiliert. Am besten kümmert man sich regelmäßig darum, entsprechende Probleme zu beheben.

Eine typische Warnung ist „*Reference ... undefined*“. Vielleicht verschwindet sie beim nochmaligen Erstellen, denn erst dann sind ggf. neue Positionen bekannt. Wenn

diese Warnung bleibt, muss das Problem unbedingt behoben werden, sonst haben Sie irgendwo Fragezeichen im Text stehen.

Warnungen, die sich auf zu volle Boxen beziehen, sind teilweise schwieriger zu verstehen und / oder zu beheben. Im **draft**-Modus (vgl. Abschnitt 7.18) werden die zugehörigen Stellen genau markiert, das kann eine große Hilfe sein. Gegen zu lange Zeilen hilft teilweise, Trennstellen zu markieren (vgl. Abschnitt 7.4). Sonst muss ggf. ein Satz minimal umformuliert werden.

Der **draft**-Modus hat drüber hinaus den Vorteil, dass das Kompilieren schneller geht (s. o.), dafür werden für Abbildungen nur Platzhalter eingefügt.

7.18 Diese Vorlage

In dieser Vorlage wird KOMA-Script verwendet, eine „Sammlung von Klassen und Paketen für L^AT_EX“¹, die insbesondere das Erstellen von deutschen Texten mit den entsprechenden üblichen typographischen Standards unterstützt.

Dokumente mit L^AT_EX zu erstellen ist ganz ähnlich wie Programmieren. Ein Beispiel: Überall dort, wo ein Absatz entstehen soll, haben wir in unserem „Quellcode“ den Befehl „**\par**“ benutzt. Was dieser Befehl genau tut, wird durch dessen Implementierung festgelegt. Und diese ergibt sich hier sozusagen aus dem Parameter **parskip** der Dokumentklasse.

Andere Einstellungen, die direkt in der Dokumentklasse erfolgen können, betreffen z. B. die Schriftgröße, die Bindungskorrektur (**BCOR**) und die Größe von Überschriften (**headings**). Im Prinzip kann man auch die Größe der Ränder mit dem **DIV**-Parameter beeinflussen, davon wird aber abgeraten. Die Ränder werden automatisch so eingestellt, dass Zeilen eine Länge haben, die gut zu lesen ist.

Hier haben wir außerdem die Option **twoside** gewählt, für beidseitigen Druck. Daher sind die Ränder außen auf geraden und ungeraden Seiten unterschiedlich. Falls Sie Ihr Dokument am Ende einseitig drucken wollen, stellen Sie das bitte um.

Nach der Festlegung der Dokumentklasse haben wir in der sog. Präambel einige Pakete eingebunden. Zum Beispiel das **scrlayer-scrpage**-Paket, mit dem wir das Aussehen der Fuß- und Kopfzeile definieren können. Diese Vorlage wurde so eingerichtet, dass in den Kopfzeilen einer Doppelseite oben links immer die aktuelle Kapitel-Überschrift und oben rechts die aktuelle Abschnitt-Überschrift angezeigt wird (siehe **definitions.tex**).

¹<https://komascript.de>

In der vorliegenden Vorlage werden einige Pakete eingebunden. Im Folgenden wird die Funktion der wichtigsten davon kurz erläutert.

fontspec Erlaubt die freie Wahl der Schriftart (funktioniert aber nur bei Kompilation mit **lualatex**)

babel Erlaubt das Umstellen der Standard-Sprache auf Deutsch

selnolig Sorgt für automatisch korrekt gesetzte Ligaturen (funktioniert aber nur bei Verwendung von **fontspec** und somit auch **lualatex**, übernimmt automatisch die Spracheinstellung von **babel**)

microtype Optimiert das Aussehen des Textes (Satzspiegel)

csquotes Sorgt für automatisch korrekt gesetzte Anführungszeichen (übernimmt automatisch die Spracheinstellung von **babel**)

tikz und pgfplots Damit können Abbildungen direkt im Quellcode erzeugt werden, vgl. Abschnitt 7.6.2

hyperref Anklickbare Links im PDF

biblatex Verbesserte Quellenangaben und -verzeichnis

amsmath und amssymb Große Erweiterung der Möglichkeiten, mathematische Inhalte darzustellen

listings Zur Darstellung von Quellcode, vgl. Abschnitt 7.10

cleveref Vereinfacht das Einfügen von Verweisen (siehe Abschnitt 7.12)

glossaries Komfortables Erstellen eines Abkürzungsverzeichnis

8 Zitate und Literaturangaben

Fremdes Gedankengut muss immer kenntlich gemacht werden. Vor allem muss es überprüfbar und auffindbar sein. Hierzu dient die Technik des Zitierens und Belegens.

Verschiedene Fachrichtungen und Studiengänge folgen spezifischen Zitierkonventionen. Wichtige Hinweise zu gängigen Zitationssystem und -stilen finden Sie in den E-Learning-Kursen des Schreibzentrums¹.

8.1 Zitieren

Wörtlich übernommene Textpassagen werden durch Anführungszeichen unten und oben („...“) kenntlich gemacht.

Wenn Ihr Zitat bereits ein Zitat enthält, müssen Sie die „doppelten Anführungszeichen“ im Text durch ‚einfache Anführungszeichen‘ ersetzen. Dazu vergleiche auch Abschnitt 7.13.

8.1.1 Quellenverweise

Für alle Zitate muss ein Quellenverweis erstellt werden. Der Quellenverweis ist eine im Zitationsstil festgelegte Kurznotation, die auf die vollständige Literaturangabe im Literaturverzeichnis verweist. Quellenverweise können *entweder* im laufenden Text (anglo-amerikanische Zitierweise bzw. Harvard-Stil) *oder* über eine Fußnote am unteren Ende der Seite *oder* in einer Endnote am Ende des gesamten Textes erfolgen. Hier gelten unterschiedliche fachliche Konventionen, die unbedingt beachtet werden müssen.

Entscheidet man sich für Kurzverweise im Textfluss oder für Endnoten, kann der Fußnotenbereich für Kommentare und für Verweise auf Stellen im eigenen Text genutzt werden.

Beachten Sie die Positionen von Hochzahl und Satzzeichen:

¹https://ilu.th-koeln.de/goto.php?target=cat_52109&client_id=thkilu

Wenn das wörtliche Zitat selbst mit einem Punkt endet, steht dieser vor dem beendenden Anführungszeichen. Die Hochzahl folgt dann direkt danach ohne Leerzeichen. Ein Punkt für den eigenen Satz entfällt dann.

Wenn das wörtliche Zitat nicht mit einem Punkt endet gibt es zwei Fälle: Steht das Zitat mitten im eigenen Satz, folgt die Hochzahl direkt nach den Anführungszeichen. Steht das Zitat am Ende des eigenen Satzes, notiert man zuerst die Anführungszeichen, dann den eigenen Satzpunkt und erst dann die Hochzahl.

Damit die Quellenverweise auch bei mehreren gleichlautenden Kurztiteln oder Jahreszahlen eines Autors eindeutig bleiben, erhalten die Jahreszahlen einen zusätzlichen kleinen Buchstaben.

8.1.2 Derselbe und Ebenda

Textverarbeitungsprogramme machen das Verweisen über *derselbe* und *ebenda* heute überflüssig, da man nicht mehr gezwungen ist, die gleichen Angaben wieder und wieder abzutippen. Sollten Sie sich (zum Beispiel auf Anraten Ihres Prüfenden) dennoch für dieses Verfahren entscheiden, empfehlen wir dringend, *ebenda* und *derselbe* etc. *erst in der redaktionellen Endphase* einzufügen, weil die Bezüge erst dann klar sind.

8.1.3 Zitate aus zweiter Hand

Bei Zitaten aus zweiter Hand, sog. Sekundärzitaten, übernimmt man ein Zitat eines Autors oder einer Autorin, ohne sich in der Originalquelle über den Sinnzusammenhang informiert zu haben. Zitate aus zweiter Hand sind nur zulässig, wenn die Originalquelle nicht beschaffbar ist. Bei allgemein zugänglicher wissenschaftlicher Literatur sind Zitate aus zweiter Hand unbedingt zu vermeiden. Ist es nicht möglich, ein Zitat mit dem Originaltext zu vergleichen, dann notiert man zitiert nach (es folgt die Quelle, der man das Zitat entnommen hat) oder zitiert in. Dies kommt zum Beispiel vor, wenn man wissenschaftsgeschichtliche Sachverhalte aus Lehrbüchern zitiert.

8.1.4 Abbildungen und Tabellen zitieren

Hier gelten die gleichen Richtlinien wie für Textzitate, d. h. auch hier gibt es getreue und abgeänderte Übernahmen. Beim originalgetreuen Zitat können Sie eine Abbildung z. B. in PowerPoint oder einem Zeichen-/Vektorprogramm genau nachbilden. Beim Scannen ist das Copyright zu berücksichtigen; es ist nur dann erlaubt, wenn der Autor bzw. der Verlag die – zumeist kostenpflichtige – Erlaubnis dazu erteilt hat. Der Quellenverweis ist in diesem Fall wie beim wörtlichen Zitat zu gestalten. Bei eigenen

Veränderungen, muss dem Quellenverweis ein Zusatz angehängt werden (z. B. *mit geringfügigen Veränderungen, mit eigenen Berechnungen*, usw.).

Weil Grafiken häufig übernommen werden, empfiehlt es sich, eigene Grafiken oder Bebilderungen auch als solche zu kennzeichnen.

In Tabellen- und Abbildungsunterschriften wird die Quelle immer direkt unter der Abbildung angegeben und nicht in einer Fuß- oder Endnote. Ins Abbildungs- und Tabellenverzeichnis gehören die Quellenangaben allerdings nicht. Dazu siehe Abschnitt 7.8.

8.2 Literaturverzeichnis

Nachfolgend wird die Gestaltung des obligatorischen Literaturverzeichnisses erläutert.

8.2.1 Inhalt und Anordnung der Literaturangaben

Im Literaturverzeichnis werden alle verwendeten Schriften in alphabetischer Reihenfolge nach Autorennamen gelistet. Das Literaturverzeichnis muss alle im Text zitierten Quellen beinhalten, aber auch keine darüber hinaus gehenden. Sind Werke nicht nur in gedruckter Form, sondern auch elektronisch publiziert, sollte die Literaturangabe der Druckfassung folgen. Es sei denn, das digitale Dokument hat eine feste Dokumentennummer (DOI). Wurde im Textteil in den Quellenverweisen statt des Titels ein Buchstabenkürzel verwendet, so ist diese Angabe auch im Literaturverzeichnis kenntlich zu machen.

Mehrere Werke, die ein Autor innerhalb eines Jahres veröffentlicht hat, müssen differenziert werden. Für die Kurztitel im Text sind die Jahreszahlen daher durch angehängte Kleinbuchstaben zu unterscheiden. Die Jahreszahl in der Quellenangabe bleibt jedoch unverändert ohne angehängte Buchstaben. Die Reihenfolge von a, b, c etc. richtet sich nach der Reihenfolge der Quellenverweise. Ob Vornamen abgekürzt oder ausgeschrieben werden, hängt von den Konventionen Ihres Faches ab. Bleiben Sie jedoch einheitlich.

8.2.2 Bibliographische Angaben im Literaturverzeichnis

Die Reihenfolge in der Notation der Literaturangabe hängt vom Dokumententyp und von fachlichen Konventionen ab. Bei Monografien sieht sie also anders aus als bei Zeitschriftenaufsätzen, in der Chemie wieder anders als in den Geisteswissenschaften oder der Mathematik. Auch für die *Interpunktion* gibt es keine einheitlichen Vorgaben.

Die Daten einer Literaturangabe entnehmen Sie bei Büchern dem so genannten Titelblatt. Dies ist nicht der Buchdeckel, sondern ein bedrucktes Blatt am Buchbeginn mit den wichtigsten werk- und buchidentifizierenden Angaben. Zunächst werden Verfasser oder Herausgeber zusammen mit dem Titel genannt, darauf folgen Erscheinungs- und Druckvermerke wie Verlag, Ort und Erscheinungsjahr. Es gibt zwar eine DIN-Regel für die Titelbeschriftung, die aber sehr unterschiedlich ausgelegt wird.

In den E-Learning-Kursen des Schreibzentrums² finden Sie weitere Informationen zu den bibliographischen Angaben für verschiedene Publikationstypen. Die hier folgende Literaturliste ist nur ein Beispiel für ein mögliches Format.

²https://ilu.th-koeln.de/goto.php?target=cat_52109&client_id=thkilu

Literatur

- Bazarevsky, V., Kartynnik, Y., Vakunov, A., Raveendran, K., & Grundmann, M. (2019). BlazeFace: Sub-millisecond Neural Face Detection on Mobile GPUs. *CoRR*, *abs/1907.05047*. <http://arxiv.org/abs/1907.05047>
- Dalal, N., & Triggs, B. (2005). Histograms of oriented gradients for human detection. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, 1, 886–893 vol. 1. <https://doi.org/10.1109/CVPR.2005.177>
- Deng, J., Guo, J., Zhou, Y., Yu, J., Kotsia, I., & Zafeiriou, S. (2019). RetinaFace: Single-stage Dense Face Localisation in the Wild. *CoRR*, *abs/1905.00641*. <http://arxiv.org/abs/1905.00641>
- dlib. (2022). *Dlib C++ Library* [Developer Documentation für dlib]. Verfügbar 28. April 2025 unter <https://dlib.net>
- dlib. (o.D. a). *cnn-face-detector* [Developer Documentation für cnn-face-detector.py von dlib]. Verfügbar 3. Mai 2025 unter dlib.net/cnn_face_detector.py.html
- dlib. (o.D. b). *Dlib C++ Library* [Dlib Python API Documentation]. Verfügbar 2. Mai 2025 unter dlib.sourceforge.net/python/index.html
- dlib. (o.D. c). *face-landmark-detection.py* [Developer Documentation für face-landmark-detection.py von dlib]. Verfügbar 2. Mai 2025 unter https://dlib.net/face_landmark_detection.py.html
- dlib. (o.D. d). *get-frontal-face-detector* [Developer Documentation für get-frontal-face-detector von dlib]. Verfügbar 28. April 2025 unter https://dlib.net/face_detector.py.html
- dlib. (o.D. e). *scan-fhog-pyramid* [Developer Documentation für scan-fhog-pyramid von dlib]. Verfügbar 28. April 2025 unter https://dlib.net/train_object_detector.py.html
- Esri Developer. (o.D.). *How single-shot detector (SSD) works?* [ArcGIS API for Python Documentation]. Verfügbar 26. April 2025 unter <https://developers.arcgis.com/python/latest/guide/how-ssd-works/>
- GeeksforGeeks. (2025). *Introduction to Convolution Neural Network* [Erklärung von CNNs]. Verfügbar 3. Mai 2025 unter www.geeksforgeeks.org/introduction-convolution-neural-network/
- Google AI Edge. (2024). *MediaPipe Framework* [Developer Documentation für MediaPipe]. Verfügbar 26. April 2025 unter ai.google.dev/edge/mediapipe/framework

- Google AI Edge. (2025a). *Face detection guide* [Developer Documentation für MediaPipe]. Verfügbar 26. April 2025 unter https://ai.google.dev/edge/mediapipe/solutions/vision/face_detector
- Google AI Edge. (2025b). *MediaPipe Solutions guide* [Developer Documentation für MediaPipe]. Verfügbar 26. April 2025 unter <https://ai.google.dev/edge/mediapipe/solutions/guide.md>
- Howse, J. (2019). *OpenCV 4 for Secret Agents: Use OpenCV 4 in secret projects to classify cats, reveal the unseen, and react to rogue drivers, 2nd Edition*. Packt Publishing. <https://books.google.de/books?id=b1qWDwAAQBAJ>
- Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R. B., Guadarrama, S., & Darrell, T. (2014). Caffe: Convolutional Architecture for Fast Feature Embedding. *CoRR*, *abs/1408.5093*. <http://arxiv.org/abs/1408.5093>
- Kazemi, V., & Sullivan, J. (2014). One Millisecond Face Alignment with an Ensemble of Regression Trees. <https://doi.org/10.13140/2.1.1212.2243>
- King, D. E. (2009). Dlib-ml: A Machine Learning Toolkit. *Journal of Machine Learning Research*, *10*, 1755–1758.
- King, D. E. (2015). Max-Margin Object Detection. *CoRR*, *abs/1502.00046*. <http://arxiv.org/abs/1502.00046>
- King, D. E. (2024). *dlib-models* [GitHub Page von dlib-models]. Verfügbar 2. Mai 2025 unter github.com/davisking/dlib-models
- Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. C. (2016). SSD: Single Shot MultiBox Detector. In B. Leibe, J. Matas, N. Sebe & M. Welling (Hrsg.), *Computer Vision – ECCV 2016* (S. 21–37). Springer International Publishing.
- lounging-lizard & nttstar. (2025). *Does FacialAnalysis.get expect BGR or RGB?* [GitHub Issue zu FacialAnalysis.get]. Verfügbar 27. Mai 2025 unter <https://github.com/deepinsight/insightface/issues/2741>
- Petroff, M. A. (2021). Accessible Color Cycles for Data Visualization. *CoRR*, *abs/2107.02270*. <https://arxiv.org/abs/2107.02270>
- Serengil, S. I. (2020). *Deep Face Detection with OpenCV in Python* [Online; accessed 2025-04-26]. Verfügbar 26. April 2025 unter <https://sefiks.com/2020/08/25/deep-face-detection-with-opencv-in-python/>
- Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, *1*, I–I. <https://doi.org/10.1109/CVPR.2001.990517>
- Wang, C., Luo, Z., Lian, S., & Li, S. (2018). Anchor Free Network for Multi-Scale Face Detection, 1554–1559. <https://doi.org/10.1109/ICPR.2018.8545814>
- Wu, W., Peng, H., & Yu, S. (2023). YuNet: A Tiny Millisecond-level Face Detector. *Machine Intelligence Research*, *20*, 656–665. <https://doi.org/10.1007/s11633-023-1423-y>

Zhang, K., Zhang, Z., Li, Z., & Qiao, Y. (2016). Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks. *CoRR*, *abs/1604.02878*. <http://arxiv.org/abs/1604.02878>

Anhang

Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer oder der Verfasserin/des Verfassers selbst entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Anmerkung: In einigen Studiengängen findet sich die Erklärung unmittelbar hinter dem Deckblatt der Arbeit.

Ort, Datum

Unterschrift