Ostbayerische Technische Hochschule Regensburg
Fakultät für Informatik und Mathematik

Inhaltsbasierte Bildvergleiche

Möglichkeiten zur Bestimmung der visuellen Ähnlichkeit zwischen zwei Bildern

Vorgelegt von: Andreas Huber <andreas.huber@st.oth-regensburg.de>
Matrikelnummer: 3370380
Studiengang: Master Informatik (Schwp. Software Engineering)

Betreuer: Prof. Dr. techn., Dipl.-Ing. Markus Kucera

Abgabedatum: 12. Dezember 2022

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Vorwort

Visuelle Ähnlichkeitsbestimmung von Bildern spielt in der heutigen Zeit eine immer wichtigere Rolle. Anders als bei der Klassifikation von Bildern, geht es dabei nicht um die korrekte Klassenzuordnung des Bildes, sondern um die konkrete visuelle Ähnlichkeit zwischen zweier Bilder (TensorFlow, 2022). Inhaltsbasierte Bildvergleichsalgorithmen berücksichtigen Parameter, wie Farbe, Form, Textur und Struktur. Aus diesen Informationen wird die Überschneidung und somit die Ähnlichkeit zweier Bilder bestimmt (baeldung, 2022).

Inhaltsbasierte Algorithmen werden neben der umgekehrten Bildersuche auch beispielsweise für Dublettenerkennung, Anwendung des Urheberrechts-Diensteanbieter-Gesetzes, Qualitätsprüfung und vieles mehr verwendet. Da diese Problematik noch immer ein offenes Forschungsthema ist und es bisher keine allgemeingültige Lösung gibt, beschäftigt sich dieses Paper mit einer Übersicht aller aktuell gängigen Kategorien an inhaltsbasierten Bildvergleichsalgorithmen.

2 Übersicht visueller Vergleichsalgorithmen

2.1 Mean squared error (MSE)

Der naive Ansatz zwei Bilder auf visuelle Ähnlichkeit zu bewerten, ist es iterativ alle Pixelintensitäten miteinander zu vergleichen. Als Werkzeug zum Vergleich dient hierbei die *mittlere quadratische Abweichung*[1] (Mean squared error) (Monsters, 2020). Je niedriger die Abweichung zwischen den Bildern, desto mehr stimmen sie überein. Obwohl dieses Verfahren verhältnismäßig leicht zu implementieren ist, ist es nur für Fälle geeignet, bei denen sowohl das Referenz- als auch das Suchbild fast identisch sind. Bereits kleinste Helligkeitsänderungen, bei gleichbleibendem Inhalt, ergeben nach Berechnung, des *euklidischen Abstands*[2], eine hohe mittlere quadratische Abweichung (siehe Abbildung 1). (Antoniadis, 2022)

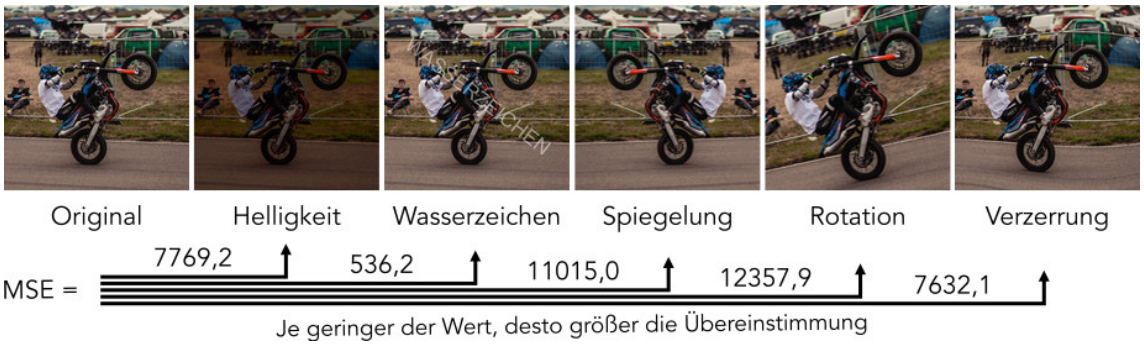


Abbildung 1: MSE: Anwendung an verschiedenen Testbildern
Quelle: Eigene Darstellung

2.2 Perceptual Hashing

Neben der mittleren quadratischen Abweichung ist eine der einfachsten, aber auch anfälligsten Methoden zur Bestimmung der Ähnlichkeit von zwei Bildern das Hashing. Dabei gibt es viele verschiedene Ansätze und Vorgehensmodelle. Die gängigste Methode ist die Anwendung des pHashes, auch genannt Perceptual Hashing. (Ramos, 2022)

Beim Perceptual Hashing werden sowohl für das Referenz- als auch für Suchbild ein Hash berechnet. Die aus deren Differenz resultierende *Hamming-Distanz*[3] ergibt die Übereinstimmung der Bilder. Je geringer die Distanz, desto ähnlicher. Das Verfahren ist nicht normiert und noch ein offenes Forschungsthema. (Klinger & Starkweather, 2010)

[1] https://www.spektrum.de/lexikon/mathematik/mittlere-quadratische-abweichung/7033 [09.12.2022]
[2] https://www.spektrum.de/lexikon/mathematik/euklidischer-abstand/4584 [09.12.2022]
[3] https://www.spektrum.de/lexikon/mathematik/hamming-abstand/3770 [09.12.2022]

Im Folgenden wird ein beispielhaftes pHash-Verfahren erläutert. Zuerst wird sowohl das Referenz- als auch das Suchbild in eine Graustufen-Grafik umgewandelt. Schließlich wird die Grafik auf 32x32 Pixel skaliert. Auf das entstandene Grauwertbild folgen zwei *Diskrete Kosinus Transformationen*[4] (1. Pro Zeile, 2. Pro Spalte). Die hochfrequenten Abschnitte befinden sich nun links oben in einer 8x8 Matrix. Daraufhin wird der Median-Grauwert der 64 Pixel berechnet. Jeder Pixel, dessen Grauwert unter dem Durchschnitt liegt wird weiß eingefärbt, der Rest schwarz. Daraus ergibt sich ein 64-Bit langer Hashwert (schwarz: 0, weiß: 1). Abbildung 2 verdeutlicht den Ablauf zur Generierung des Hashes anhand einer Testgrafik. Zuletzt wird zwischen den beiden generierten Bitfolgen der Hamming-Abstand berechnet. Je niedriger der Abstand der beiden Bitfolgen ist, desto größer ist die Übereinstimmung zwischen den Eingabebildern. (Ramos, 2022)

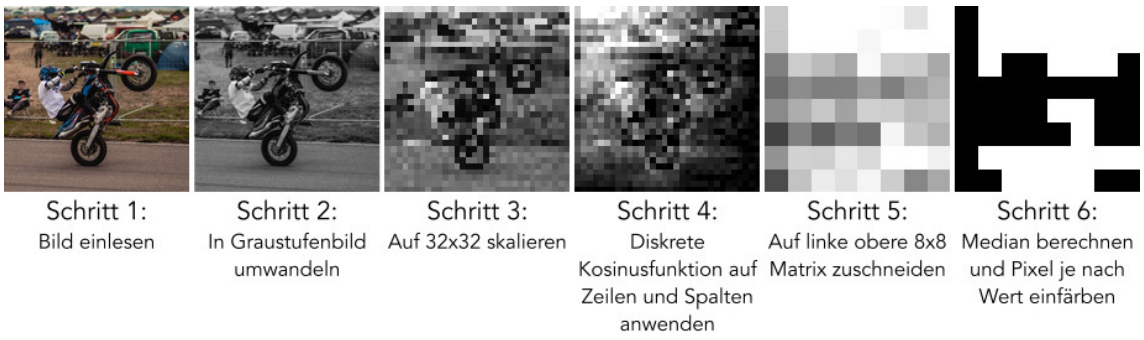


Abbildung 2: PHASH: Schrittweise Berechnung des Hashes
Quelle: Eigene Darstellung; Zugehöriger Programmcode im Anhang

Vorteile

- Verhältnismäßig einfache Implementierung möglich
- Schnelle Suchperformance, wenn die Hashes der Referenzbilder bereits in einer dafür geeigneten Datenstruktur (z.B. *k-d-Bäume*[5], *VP-Bäume*[6] oder *Kugelbäume*[7]) vorliegen (Christina, 2020)
- Robust gegen Wasserzeichen, Farbfilter, leichte Helligkeits- und Kontraständerungen, Gammakorrekturen, Skalierungen sowie Komprimierungen (siehe Abbildung 3) (Klinger & Starkweather, 2010)

Nachteile

- Nicht robust gegen Spiegelungen, Rotierungen und Verzerrungen (siehe Abbildung 3)

[4]https://de-academic.com/dic.nsf/dewiki/339357 [09.12.2022]
[5]https://www.cs.cmu.edu/ ckingsf/bioinfo-lectures/kdtrees.pdf [09.12.2022]
[6]https://fribbels.github.io/vptree/writeup [09.12.2022]
[7]https://towardsdatascience.com/tree-algorithms-explained-ball-tree-algorithm-vs-kd-tree-vs-brute-force-9746debcd940 [09.12.2022]

- Nicht robust gegen Zuschneidungen, neu eingefügte Elemente oder Änderungen des Blickwinkels

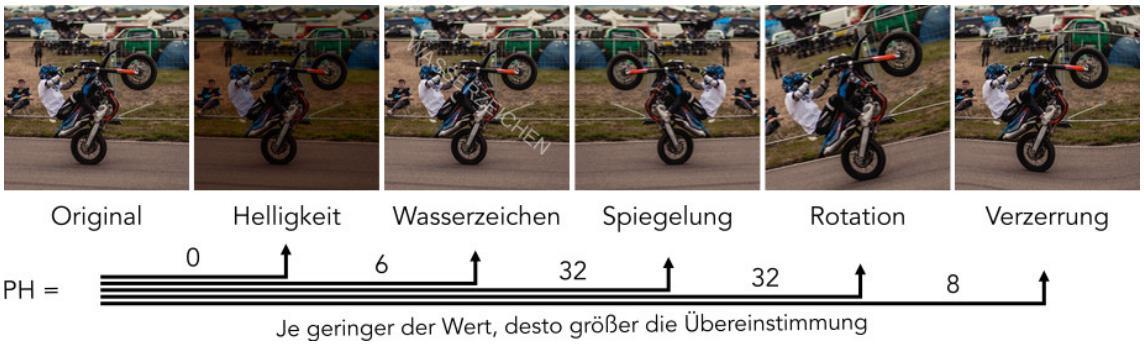


Abbildung 3: PHASH: Anwendung an verschiedenen Testbildern
Quelle: Eigene Darstellung

2.3 Histogram Intersection

Ein robusteres, aber auch rechenaufwendigeres Verfahren ist der Histogram Intersection Algorithmus von Swain und Ballard. Farbhistogramme sind stabil gegenüber Spiegelungen, Verschiebungen und leichten Rotationen um die Betrachtungsachse (siehe Abbildung 4). Außerdem ändern sie sich bei Änderungen des Blickwinkels, des Maßstabs und bei Verdeckung von Elementen nur langsam. Gegen Belichtungsänderungen können stark reduzierte Farbhistogramme jedoch Probleme bereiten. (Swain & Ballard, 1991)

Im Standardfall wird die Histogram Intersection auf Farbhistogramme angewandt. Zuerst muss die Menge an zu diskretisierenden Farben im Histogramm festgelegt werden - zum Beispiel 200. Folglich gibt es 200 mögliche „Farbeimer“ (Bins) in die jeder Pixel der Bilder einsortiert wird. Nach dem Übereinanderlegen der durch das Referenz- und das Suchbild berechneten Histogrammen, wird schließlich deren Überschneidung berechnet. Je stärker sich die Histogramme überschneiden, desto ähnlicher sind die Bilder (Lee et al., 2004). Zusätzlich wird wie beim Hashing ein gewisser Schwellenwert vorher definiert. Der Vorgang kann beispielsweise auch auf die einzelnen Farbkanäle aufgeteilt werden. Bei RGB würde das bedeuten, dass man jeweils den Rot-, Grün-, und Blau-Kanal einzeln betrachtet. (Swain & Ballard, 1991)

Eine weitere Untersuchung hat ergeben, dass sowohl die Wahl des Farbraums als auch die Festlegung des Quantization Levels (Anzahl der Bins) eine große Rolle bei der Erfolgsschance dieses Vorgehensmodells spielen. In dieser Studie lieferte der CIELab-Farbraum im Allgemeinen die besten Ergebnisse. (Lee et al., 2004)

Um die Genauigkeit des Algorithmus zusätzlich zu verbessern, können neben Farbhistogrammen weitere Histogramme erstellt und verglichen werden. Ein mögliches Beispiel wäre ein „Texture Direction“ oder „Texture Scale“ Histogramm. In

diesen Fällen wird nicht versucht anhand der Farben eine Ähnlichkeit zu erkennen, sondern durch die Struktur des Bildes. Hier werden beispielsweise aus dem Bild extrahierte Ecken oder Kanten in ein Histogramm sortiert und schließlich auf Überschneidungen analysiert (Simek, 2009). Natürlich wird jedoch jeder weitere zusätzlicher Berechnungsschritt die allgemeine Performance des Systems beeinträchtigen.

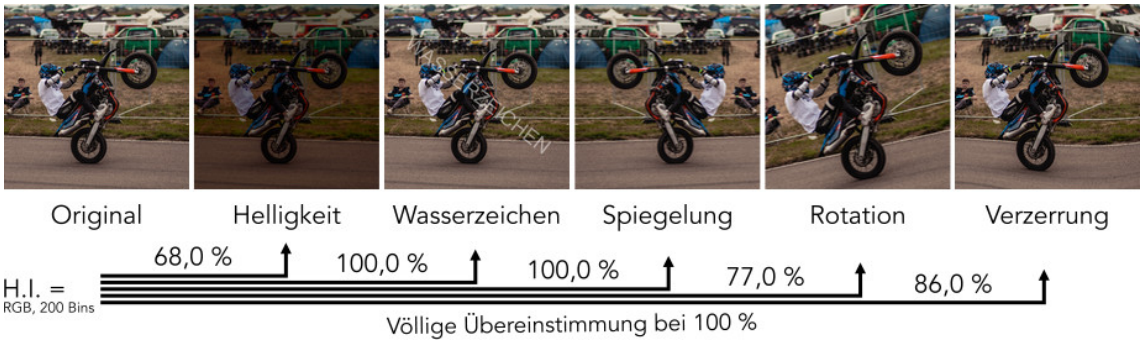


Abbildung 4: Histogram Intersection: Anwendung an verschiedenen Testbildern
Quelle: Eigene Darstellung

2.4 Scale-Invariant Feature Transform (SIFT)

David G. Lowe hat mit der Einführung des SIFT-Algorithmus einen für die Bildverarbeitung wertvollen Beitrag geleistet. Der SIFT-Algoritmus findet lokale Merkmale in Bildern. Diese Merkmale sind invariant gegenüber Bildskalierung und -drehung und teilweise invariant gegenüber Änderungen der Beleuchtung sowie des Betrachtungswinkels. Die Wahrscheinlichkeit einer Störung durch Verdeckung einzelner Elemente oder auftretenden Bildrauschens wird, durch räumliche Streuung und einem vielseitigen Frequenzbereich, erreicht. Im Folgenden wird die grobe Vorgehensweise zur Findung der lokalen Merkmale erläutert. (Lowe, 2004)

1. **Scale-space extrema detection**
   Zu Beginn wird aus einem Bild durch repetetives Unschärfen (mittels *Gaußschen Weichzeichner*[8]) und Halbieren der Größe eine Serie an Bildern erzeugt (siehe Abbildung 5). Auf die Bilder der Bilderserie folgt die Anwendung der *Gaußschen Differenzfunktion*[9] (siehe Abbildung 6). (Sinha, n. d. a)

[8]https://datacarpentry.org/image-processing/06-blurring/ [09.12.2022]
[9]https://www.sciencedirect.com/science/article/pii/B9780123694072500097 [09.12.2022]

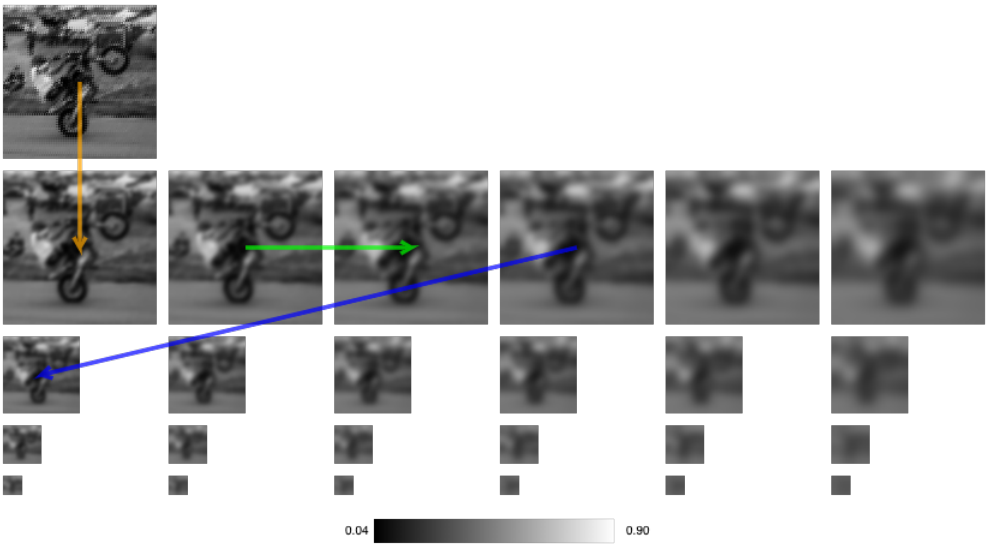


Abbildung 5: SIFT: Erzeugung der Bilderserie
Quelle: Eigene Darstellung; http://weitz.de/sift/

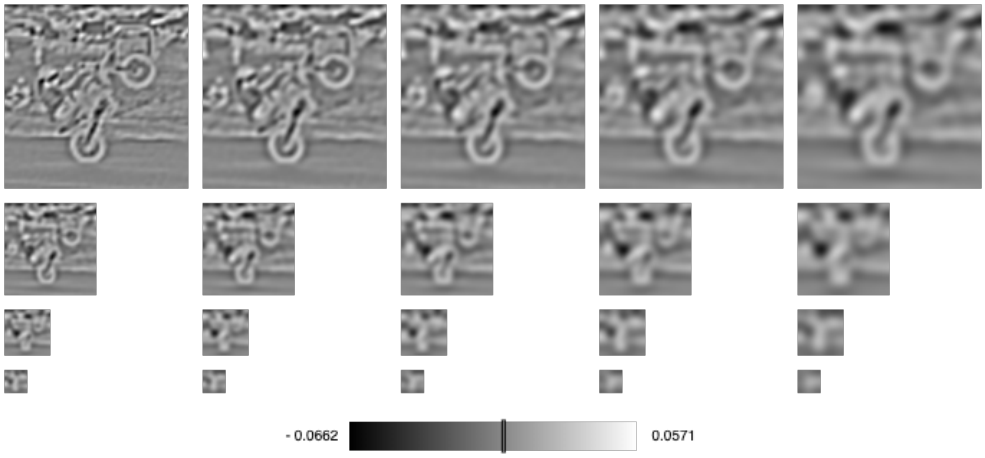
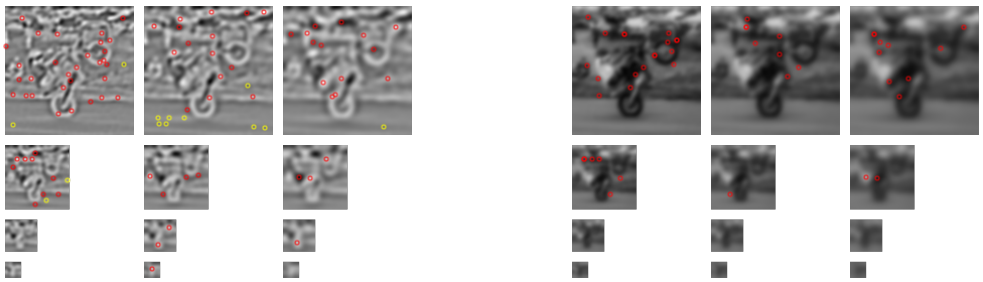


Abbildung 6: SIFT: Anwendung Gaußscher Differenzfunktion
Quelle: Eigene Darstellung; http://weitz.de/sift/

2. **Keypoint localization**
   Aus den vorher differenzierten Grafiken werden mit dem *Marr-Hildreth-Operator*[10] interessante Schlüsselpunkte bzw. Merkmale herausgefiltert (siehe Abbildung 7a) (Sinha, n. d. b). Eine dem *Harris-Corner-Detector*[11] ähnliche Prozedur wird angewandt, um unwichtige Punkte herauszusieben (siehe Abbildung 7b) (Sinha, n. d. c).

[10] https://theailearner.com/2019/05/25/laplacian-of-gaussian-log [09.12.2022]
[11] https://docs.opencv.org/3.4/dc/d0d/tutorial_py_features_harris.html [09.12.2022]



(a) Extrahierte Schlüsselpunkte (b) Filterung unwichtiger Schlüsselpunkte

Abbildung 7: SIFT: Schlüsselpunkt-Lokalisierung
Quelle: Eigene Darstellung; http://weitz.de/sift/

3. **Orientation assignment**
   Mithilfe eines Histogramms kann den Schlüsselpunkten nun eine Orientierung zugewiesen werden (siehe Abbildung 8). Dadurch werden die Punkte neben der Robustheit gegenüber Skalierung und Verschiebung auch gegen Rotation invariant (siehe Abbildung 10). (Sinha, n. d. d)

(a) Orientierung des Merkmals (b) Zugehöriges Farbhistogramm (normalisiert)

Abbildung 8: SIFT: Zuordnung der Orientierung
Quelle: Eigene Darstellung; http://weitz.de/sift/

4. **Keypoint descriptor**
   Zuletzt wird nach Anwendung weiterer mathematischer Verfahren eine Art Fingerabdruck des Merkmals berechnet (siehe Abbildung 9). Dadurch wird die Stabilität gegenüber begrenzter lokaler Formverzerrungen und Beleuchtungsänderungen erreicht (siehe Abbildung 10). (Sinha, n. d. e)

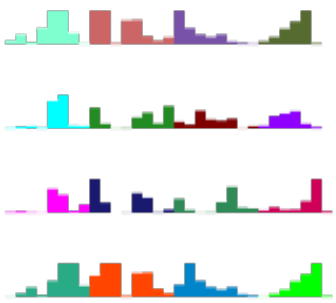


Abbildung 9: SIFT: Pro Schlüsselpunkt erzeugte Deskriptoren (Fingerabdruck)
Quelle: Eigene Darstellung; http://weitz.de/sift/

Nach Anwendung dieses Algorithmus sollte man bei einem 500x500 Pixel großen Bild mit rund 2000 robusten Merkmalen rechnen. Werden die gefundenen Merkmale in einer Datenbank gespeichert, können sie in nahezu Echtzeit auf die Schlüsselpunkte eines Suchbilds verglichen werden. (Lowe, 2004)

Ein ähnlicher, jedoch bis 2033 patentierter[12], Algorithmus zur Erkennung von Bildmerkmalen ist der „Speeded up robust features (SURF)“ (Bay et al., 2006). Ferner gibt es viele ähnliche frei verfügbare Algorithmen der Gruppe „Bildmerkmalsfindung“.

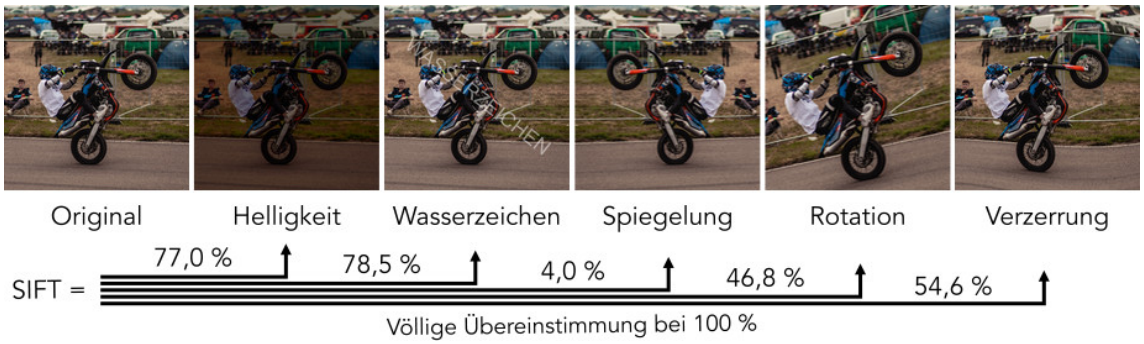


Abbildung 10: SIFT: Anwendung an verschiedenen Testbildern
Quelle: Eigene Darstellung

2.5 Structural Similarity Index (SSIM)

Anders als die meisten anderen Vorgehensmodelle versucht die menschliche Wahrnehmung nicht Fehler zu quantifizieren, sondern strukturelle Informationen aus einer Szene wiederzuerkennen. Um dieses Verhalten nachzuahmen, haben forschende Mitglieder der IEEE den Structual Similarity Index entworfen. (Wang et al., 2004)

[12]https://patents.google.com/patent/CN103640018A/en [09.12.2022]

Der Index extrahiert, vergleicht und kombiniert folgende drei Parameter:

- Leuchtdichte
- Kontrast
- Struktur

Im ersten Schritt wird die Leuchtdichte durch Mittelwertbildung über alle Pixelwerte gemessen. Anschließend wird der Kontrast durch Berechnung der Standardabweichung aller Werte ermittelt. Vereinfacht dargestellt wird die Struktur verglichen, indem das Eingangssignal durch seine Standardabweichung geteilt, sodass eine genormte Standardabweichung entsteht. (Datta, 2020)

Sobald die drei Werte berechnet wurden, werden diese anhand, im Original-Paper, veröffentlichten mathematischen Funktionen verglichen und schließlich kombiniert (siehe Abbildung 11). (Datta, 2020)

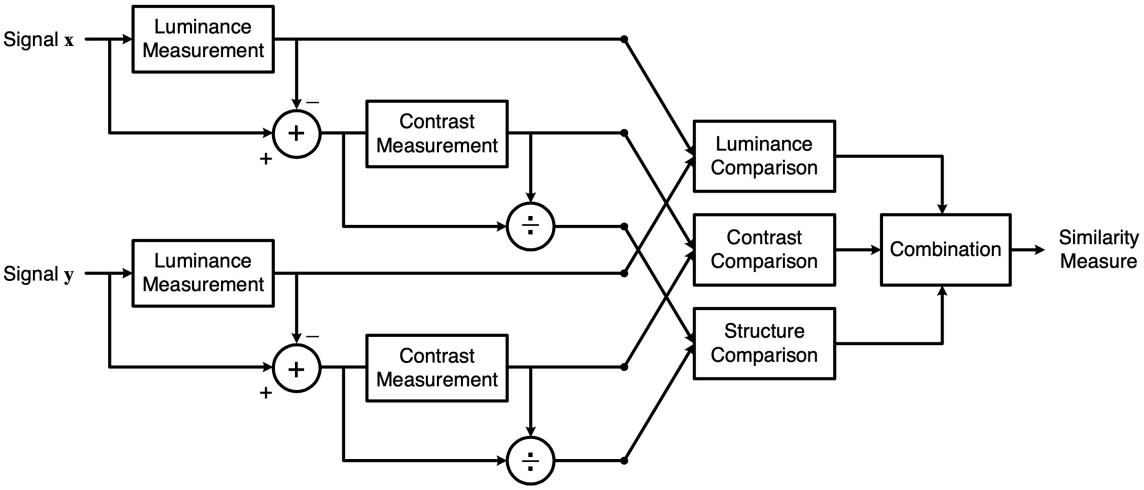


Abbildung 11: SSIM: Ablaufsdiagramm
Quelle: Wang, Z., Bovik, A. C., Sheikh, H. R. & Simoncelli, E. P. (2004). Image Quality Assessment: From Error Visibility to Structural Similarity

Obendrein sei es nützlich den Index nicht global auf das Bild anzuwenden. Das Paper gibt an den Algorithmus vorzugsweise auf verschiedene lokale Stellen einzusetzen. Erstens seien für die Analyse interessante Objekte oft instationär und zweitens können Bildverzerrungen auch räumlich variabel sein. Um die Referenz zur Nachahmung der menschlichen Wahrnehmung zu ziehen, ist ebenfalls festzustellen, dass Menschen ebenfalls nur lokale Bereiche einer Grafik in hoher Auflösung wahrnehmen können. (Datta, 2020)

Auch wenn der SSIM durch seine Einfachheit, gegenüber anderen Verfahren, eine überragende Performance aufweist, ist der Algorithmus sehr anfällig für Ska-

lierungen, Verschiebungen und Rotierungen (siehe Abbildung 12). (Zhang & Bull, 2014)

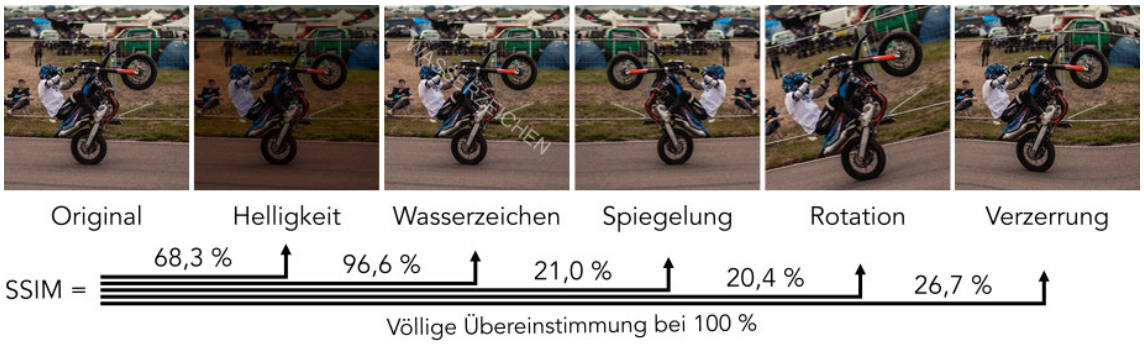


Abbildung 12: SSIM: Anwendung an verschiedenen Testbildern
Quelle: Eigene Darstellung

Aufbauend auf den SSIM-Algorithmus gibt es weitere optimierende Forschungen, wie zum Beispiel der Complex Wavelet Structural Similarity Index. Der CW-SSIM adressiert die vorher genannten Probleme und erzeugt durch Verwendung von Wavelet-Transformationen eine Robustheit gegenüber kleinen Verschiebungen und Rotierungen. (Sampat et al., 2019)

2.6 Siamese Network (Deep Learning)

Zu guter Letzt liefert der Image Matching Algorithmus auf Basis des Siamesischen Neuronalen Netzwerks die besten Ergebnisse. Dabei besteht das neuronale Netzwerk aus zwei identischen Subnetzwerken. Meist kommt hierfür das sehr verbreitete *Convolutional Neural Network*[13] als Subnetzwerk zum Einsatz, welches die Merkmalsvektoren aus den Bildern extrahiert. (Melekhov et al., 2016)

Die beiden entstehenden Vektoren dienen als Eingabe für die *Contrastive Loss Function*[14]. Im Falle eines Matches minimiert dieser Vorgang die euklidische Distanz, andernfalls maximiert er die Distanz bei inkomparabelen Bilderpaaren. (Melekhov et al., 2016)

Wie bei vielen Machine Learning Ansätzen ist bei der Verwendung einer der größten Nachteile das Training. Damit das Siamese Network optimal funktioniert, muss das Netzwerk mit einem sehr großen Datensatz an klassifizierten Bilderpaaren trainiert werden (Antoniadis, 2022). Das ist außerdem der Grund, wieso im Rahmen dieser Abhandlung keine Anwendung an Testbildern durchgeführt werden konnte.

[13] https://www.ibm.com/cloud/learn/convolutional-neural-networks [09.12.2022]
[14] https://towardsdatascience.com/contrastive-loss-explaned-159f2d4a87ec [09.12.2022]

3 Fazit und Ausblick

Zusammenfassend bietet dieses Paper einen groben Überblick über den aktuellen Forschungsstand bei der Ähnlichkeitsbestimmung von Bildern. Je nach Größe und Art des Projekts eignen sich manche Algorithmen mehr und manche weniger. Eine allgemeingültige Lösung gibt es bisher nicht.

Der feingranulare inhaltsbasierte Vergleich zwischen Bildern wird auch sicherlich in Zukunft noch ein wesentlicher Bestandteil der Forschung sein. Neben den in der Einleitung 1 erläuterten Anwendungsfälle spielen Image Matching Algorithmen insbesondere in der Medizin eine immer größere Rolle. Beispielsweise sind computergestützte Diagnosen durch Bildvergleichsalgorithmen in der Lage Radiologen bei der Erkennung von Krankheiten zu helfen (Muramatsu, 2018).

Quellenverzeichnis

Antoniadis, P. (2022). *Algorithms for Image Comparison*. Verfügbar 4. Dezember 2022 unter https://www.baeldung.com/cs/image-comparison-algorithm

baeldung. (2022). *What Is Content-Based Image Retrieval?* Verfügbar 9. Dezember 2022 unter https://www.baeldung.com/cs/cbir-tbir

Bay, H., Tuytelaars, T. & Van Gool, L. (2006). SURF: Speeded Up Robust Features (A. Leonardis, H. Bischof & A. Pinz, Hrsg.).

Christina. (2020). *Determining how similar two images are with Python + Perceptual Hashing*. Verfügbar 4. Dezember 2022 unter https://lvngd.com/blog/determining-how-similar-two-images-are-python-perceptual-hashing/

Datta, P. (2020). *All about Structural Similarity Index (SSIM): Theory + Code in PyTorch*. Verfügbar 4. Dezember 2022 unter https://medium.com/srm-mic/all-about-structural-similarity-index-ssim-theory-code-in-pytorch-6551b455541e

Klinger, E. & Starkweather, D. (2010). *pHash - The open source perceptual hash library*. Verfügbar 4. Dezember 2022 unter https://www.phash.org/

Lee, S. M., Xin, J. H. & Westland, S. (2004). Evaluation of Image Similarity by Histogram Intersection.

Lowe, D. G. (2004). Distinctive Image Features from Scale-Invariant Keypoints.

Melekhov, I., Kannala, J. & Rahtu, E. (2016). Siamese network features for image matching. https://doi.org/10.1109/ICPR.2016.7899663

Monsters, D. (2020). *A Quick Overview of Methods to Measure the Similarity Between Images*. Verfügbar 4. Dezember 2022 unter https://medium.com/@datamonsters/a-quick-overview-of-methods-to-measure-the-similarity-between-images-f907166694ee

Muramatsu, C. (2018). Overview on subjective similarity of images for content-based medical image retrieval.

Ramos, A. (2022). *Introduction To Perceptual Hashes: Measuring Similarity*. Verfügbar 4. Dezember 2022 unter https://apiumhub.com/tech-blog-barcelona/introduction-perceptual-hashes-measuring-similarity/

Sampat, M. P., Wang, Z., Gupta, S., Bovik, A. C. & Markey, M. K. (2019). Complex Wavelet Structural Similarity: A New Image Similarity Index.

Simek, K. (2009). *Image comparison - fast algorithm*. Verfügbar 4. Dezember 2022 unter https://stackoverflow.com/a/844113/17726654

Sinha, U. (n. d. a). SIFT: Theory and Practice. Verfügbar 4. Dezember 2022 unter https://aishack.in/tutorials/sift-scale-invariant-feature-transform-scale-space/

Sinha, U. (n. d. b). SIFT: Theory and Practice. Verfügbar 4. Dezember 2022 unter https://aishack.in/tutorials/sift-scale-invariant-feature-transform-keypoints/

Sinha, U. (n. d. c). SIFT: Theory and Practice. Verfügbar 4. Dezember 2022 unter https://aishack.in/tutorials/sift-scale-invariant-feature-transform-eliminate-low-contrast/

Sinha, U. (n. d. d). SIFT: Theory and Practice. Verfügbar 4. Dezember 2022 unter https://aishack.in/tutorials/sift-scale-invariant-feature-transform-keypoint-orientation/

Sinha, U. (n. d. e). SIFT: Theory and Practice. Verfügbar 4. Dezember 2022 unter https://aishack.in/tutorials/sift-scale-invariant-feature-transform-features/

Swain, M. J. & Ballard, D. H. (1991). Color Indexing.

TensorFlow, G. I. (2022). *Image classification*. Verfügbar 9. Dezember 2022 unter https://www.tensorflow.org/lite/examples/image_classification/overview

Wang, Z., Bovik, A. C., Sheikh, H. R. & Simoncelli, E. P. (2004). Image Quality Assessment: From Error Visibility to Structural Similarity.

Zhang, F. & Bull, D. R. (2014). Academic Press Library in signal Processing.

Anhang

Perceptual Hashing: Implementierung

Schrittweise Generierung eines Perceptual Hashes in Python unter der Verwendung der OpenCV-Bibliothek.

```python
# Perceptual Hash Implementation by Andreas Huber
# Date: 2022-12-10

import numpy as np
import cv2

# Generates Perceptual Hash
def generatePerceptualHash(path, export=False):
    # Step 1: Read Image
    image = cv2.imread(path)
    if (export):
        cv2.imwrite('phash-results/phash-step-1.png', image)

    # Step 2: To grayscale
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    if (export):
        cv2.imwrite('phash-results/phash-step-2.png', image)

    # Step 3: To 32x32
    image = cv2.resize(image, (32, 32), interpolation = cv2.INTER_NEAREST)
    if (export):
        cv2.imwrite('phash-results/phash-step-3.png', image)

    # Step 4a: Discrete Cosinus Transform: Rows
    image = np.float32(image) / 255.0
    rows = cv2.dct(image, cv2.DCT_ROWS)

    # Step 4b: Discrete Cosinus Transform: Columns
    rows = cv2.rotate(rows, cv2.ROTATE_90_COUNTERCLOCKWISE)
    rows = cv2.flip(rows, 0)
    cols = cv2.dct(rows, cv2.DCT_ROWS)
    image = cv2.flip(cols, 0)
    image = cv2.rotate(image, cv2.ROTATE_90_CLOCKWISE)

    if (export):
        cv2.imwrite('phash-results/phash-step-4.png', image * 255)

```

```
    # Step 5: Crop to 8x8
    image = image[0:8, 0:8]

    if (export):
        cv2.imwrite('phash-results/phash-step-5.png', image * 255)

    # Step 6a: Calculate median
    imageValues = image.flatten()
    median = np.median(imageValues)

    # Step 6b: intensity < median ? black : white
    for i in range(8):
        for j in range(8):
            if (image[i][j] < median):
                image[i][j] = 0.0
            else:
                image[i][j] = 1.0

    if (export):
        cv2.imshow('resultHash', image)
        cv2.imwrite('phash-results/phash-step-6.png', image * 255)

    # Build string binary perceptual hash
    hash = image.flatten()
    hash = map(int, hash)

    if (export):
        cv2.waitKey(0)
        cv2.destroyAllWindows()

    return ''.join(map(str, hash))
```