INSTITUT FÜR INFORMATIK
Fachbereich Informatik und Mathematik

Bachelorarbeit

Automatisierte Bilderkennung von Monogrammen mit OpenCV

Huy Luong
Studiengang: B.Sc. Informatik
Matrikelnummer: 5727773

Frankfurt am Main
07.05.2019

Betreuer: Prof. Dr. Roberto Zicari, Dr. Karsten Tolle

Professur für Datenbanken und Informationssysteme
Goethe Universität Frankfurt

Erklärung zur Abschlussarbeit

Erklärung gemäß Bachelor-Ordnung Informatik 2011 §25 Abs. 11

Hiermit erkläre ich, dass ich die Bachelorarbeit selbständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst habe.

Frankfurt am Main, den 07.Mai.2019

Huy Luong

Zusammenfassung

Die Erkennung von Monogrammen ist von essentieller Bedeutung in der Bestimmung von antiken Münzen. Da die Menge bereits entdeckter und aufgezeichneter Monogramme wächst und eine große Menge an Monogrammen sich in vielen Hinsichten ähneln, nahezu deckungsgleich sind und gleiche Merkmale aufweisen, wird es immer schwieriger zu erkennen, ob diese bereits als Datenensatz vorhanden sind oder ein neuer Datensatz angelegt werden muss. In dieser Arbeit wird mithilfe der Bibliothek *OpenCV* und dem *SIFT*-Algorithmus die automatische Erkennung von gleichen und sich ähnelnden Monogrammen mittels dem Vergleich von robusten Schlüsselpunkten implementiert. Dabei steht das Auffinden von Merkmalen im Fokus, die einerseits gut unterscheidbar sind, andererseits aber auch mit hoher Wahrscheinlichkeit auf verschiedenen Bildern eines gleichen Monogramms wiederzufinden sind.

Inhaltsverzeichnis

1 Einleitung

1.1 Relevanz

Mit dem heutigen Stand der Digitalisierung ist es möglich die Arbeit vor der Erfassung eines Monogramms deutlich zu beschleunigen. Um zu überprüfen, ob ein Monogramm bereits in einer Datenbank vorhanden ist, benötigt es großen manuellen Aufwand, der folglich mit stetig wachsender Datenmenge steigt. Der ganze Prozess der Erkennung soll dabei möglichst automatisiert erfolgen. Im Vergleich zu Computern ist die Erkennung von Objekten für den Menschen einfach. Die Abstraktionsfähigkeit des Menschen erlaubt es auch Objekte klassifizieren zu können, die ein leicht verändertes Erscheinungsbild aufweisen. Wenn ein Objekt beispielsweise gedreht oder leicht verzerrt ist, würde der Mensch es trotzdem weiterhin als das Objekt erkennen, was es ursprünglich war.
Die automatisierte Erkennung der Monogramme soll genau mit dieser Problematik umgehen können und sicherstellen, dass leicht veränderte Monogramme trotzdem erkannt werden können.

1.2 Aufgabenstellung

Ziel dieser Arbeit ist es eine Anwendung zu entwerfen, die die bereits genannte Problematik löst, beziehungsweise den Aufwand reduziert. Dazu soll mittels der Open-Source-Bibliothek *OpenCV* und des *SIFT*-Algorithmus die Erkennung von Monogrammen, die bereits in einer Datenbank vorhanden sind, automatisiert werden.
Bei der Eingabe eines Monogramms soll nicht nur der beste Zuordnungsvorschlag berechnet werden, sondern es sollen stattdessen die fünf ähnlichsten Monogramme ausgegeben werden. Die Aufgabenstellung ist mittels einer Python-Anwendung zu lösen mit Zuhilfenahme der Bibliothek *OpenCV*.

1.3 Aufbau der Arbeit

Zunächst werden die theoretischen Grundlagen vermittelt, auf denen die erstellte Anwendung basiert. Anschließend wird auf die konkrete Implmenetierung des Algorithmus eingegangen. Im letzten Abschnitt wird die Anwendung getestet. Die daraus folgenden Resultate werden abschließend evaluiert und analysiert.

2 Theoretische Grundlagen

Zur Lösung des Problems befasst sich die vorliegende Arbeit zunächst mit den Grundlagen aus dem Bereich der *Computer Vision*. Objekt- und Formerkennung sind essentielle Bestandteile des in dieser Arbeit behandelten Ansatzes und werden mittels *Local Feature Detection* umgesetzt. Dabei werden markante Punkte (engl. *Features*) auf einem Bild gefunden und beschrieben. Diese markanten Punkte können dann auf zwei verschiedenen Bildern auf ihre Ähnlichkeit verglichen werden. Anhand dieser Merkmale ist es möglich zwei Bilder, mit gleichem oder sehr ähnlichem Inhalt, einander zuzuordnen. Im vorliegenden Fall besteht dieser Inhalt aus Monogrammen.

2.1 Monogramme

Monogramme sind Zeichen, die aus einem oder mehreren kunstvoll gestalteten Buchstaben zusammengefügt werden. In vielen Fällen stellen die Buchstaben die Anfangsbuchstaben eines Vor- und Nachnamen dar. Die graphischen Symbole wurden im Mittelalter und der frühen Neuzeit überwiegend auf Urkunden verwendet.

Abbildung 2.1: Monogramm, das sich aus den Buchstaben **H** und **R** zusammensetzt.

Um Monogramme in einer Datenbank zu speichern, ist es nötig sie inhaltlich unterscheiden zu können, damit ein Monogramm nicht versehentlich doppelt in einer Datenbank aufgenommen wird. Eine Herangehensweise wäre es, die Buchstaben die ein Monogramm enthält, als Charakteristikum zu nehmen, um Monogramme auseinanderzuhalten. Das Monogramm in (Abb 2.1) würde, dann als Charakteristikum die Buchstaben "HR"besitzen. Die Reihenfolge der Buchstaben ist intuitiv gewählt, da die Leserichtung der in dieser Arbeit verwendeten Sprache (deutsch), von links nach rechts geht.
Das Charakteristikum des Monogramms in (Abb 2.2 linkes Monogramm) ist im Gegensatz

dazu, nicht mehr intuitiv zu bestimmen. Es könnte zum einen als "XB"definiert werden, und zum anderen als "BX".
Noch schwieriger einzuordnen ist das Monogramm in (Abb 2.2 rechtes Monogramm). In diesem Fall ist es kaum möglich das Monogramm überhaupt in Buchstaben zu wandeln.

Abbildung 2.2: Monogramme die komplexere Strukturen aufweisen.

2.2 Begründung des methodischen Vorgehens

Im Gegensatz zum Menschen erkennen Computer nicht unmittelbar, aus welchen Buchstaben sich das Monogramm in (Abb 2.1) zusammensetzt.
Um Merkmale zu extrahieren, könnte das gesamte Monogramm in einzelne Komponenten aufgeteilt und klassifiziert werden. Hierbei treten jedoch bereits die ersten Probleme und Fragestellungen dieser Methode auf. Die Aufteilung der Komponenten stellt unter Umständen eine Schwierigkeit dar. Während in (Abb 2.1) die zwei Buchstaben durch eine vertikale Linie getrennt werden können, ist dies in (Abb 2.3) nicht in gleicher Weise möglich.

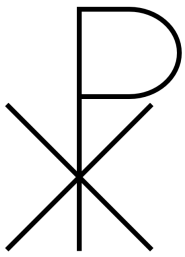

Abbildung 2.3: Monogramm bestehend aus den Buchstaben "P" und "X".

Dieser Ansatz geht in die Richtung der optischen Zeichen- und Texterkennung [QA09]. Optische Texterkennung wird häufig genutzt, um Inhalte auf schriftlichen Dokumenten in digitale Zeichenfolgen umzuwandeln.
Für Monogramme eignet sich der Ansatz allerdings nur bedingt, da wie in (Abb 2.3) die Aufteilung sehr komplex ausfallen kann. Monogramme können teilweise auch Symbole enthalten und sind nicht nur auf Buchstaben beschränkt.

Ein anderer Ansatz wären künstliche neuronale Netze. Diese könnten so trainiert werden, dass sie Monogramme klassifizieren können. Allerdings bedarf dies einer größeren Daten-

menge, als hier in dieser Arbeit zur Verfügung stehen [SHK+14, S. 1946]. Des Weiteren existieren viele Monogramme, die sich in nur sehr kleinen Details unterscheiden (Abb 2.4), welche vom neuronalen Netz gegebenenfalls nicht als ein solcher Unterschied wahrgenommen werden, abhängig von der Trainingsweise des Netzes. Zudem müsste bei jedem neu hinzukommenden Monogramm das gesamte künstliche neuronale Netz erneut trainiert werden.

Abbildung 2.4: Die erweiterte Linie stellt einen Unterschied zwischen den Monogrammen dar.

Diese Arbeit folgt dem Ansatz der Local Feature Detection. Die Erkennung der Monogramme soll mithilfe von *OpenCV* und des *SIFT*-Algorithmus erfolgen, der auf dem Gebiet der Bilderkennung bereits vielversprechende Ergebnisse erzielen konnte [LMT+07]. Basierend auf der Datenmenge, die in dieser Arbeit genutzt wird, ist der Algorithmus sehr schnell. Im Vergleich zu den anderen Verfahren fällt der Overhead des *SIFT*-Algorithmus viel geringer aus. Die Problematik, dass das Bild in Komponenten zu unterteilen ist, entfällt gänzlich, da das Bild im Gesamten betrachtet und klassifiziert wird. Der Aufwand des Trainings, das bei einem künstlichen neuronalen Netz erforderlich ist, entfällt ebenfalls. Denn bei einem neuen Monogramm sind lediglich Features zu berechnen. Hier ist der Aufwand im Vergleich zum Trainingsprozess eines neuronalen Netzes geringer.

2.3 SIFT-Algorithmus

In diesem Abschnitt wird der für diese Arbeit verwendete *SIFT*-Algorithmus (*Scale Invariant Feature Transform*) nach David Lowe [L+99, Low04] genauer erläutert und beschrieben. Die Grundlage des Algorithmus ist das Auffinden von robusten und vergleichbaren Schlüsselpunkten (engl. *Keypoints*) eines Bildes, um diese dann auf einem anderen Bild wiederzufinden. Robust bedeutet in diesem Fall, dass die Schlüsselpunkte auch dann als der Gleiche erkannt werden, falls sich die Ausrichtung oder auch Skalierung des Schlüsselpunkts ändert. Dabei stellt der Algorithmus ebenfalls in begrenztem Maße sicher, dass die Merkmale robust gegen Translation (Versetzung), Illumination (Änderung der Lichtverhältnisse) und gegen teilweiser affiner Verzerrung sind. Angewandt wird der Algorithmus ausschließlich auf Grauwertbildern. Der Algorithmus kommt in verschiedenen Gebieten zum Einsatz, wie der Objekterkennung, Gestenerkennung oder des Stitchings (Abb 2.5).

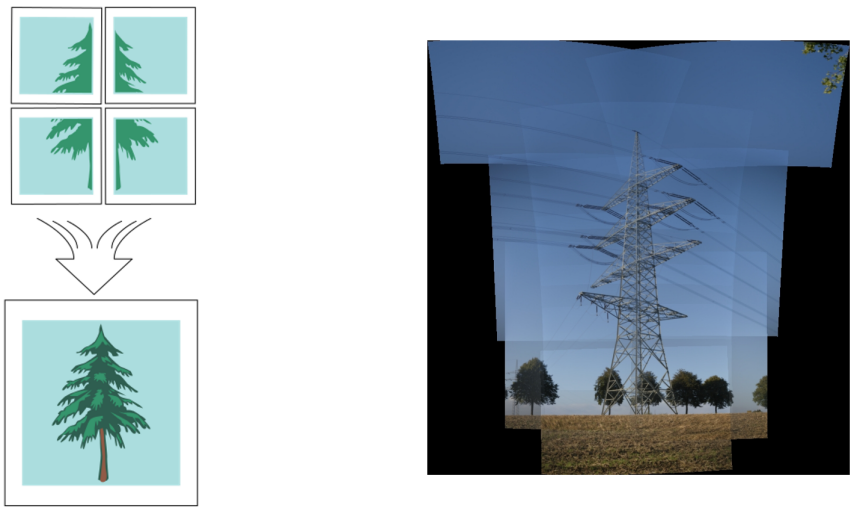

Abbildung 2.5: Beim Stitching werden Einzelbilder zu einem Gesamtbild zusammengefügt. Dies wird unter anderem bei der Erstellung von Panoramabildern genutzt.

Um Schlüsselpunkte miteinander vergleichen zu können, muss man ihnen einen eindeutigen Fingerabdruck geben und ihre Beschaffenheit definieren und festlegen. Dieser Fingerabdruck wird auch als Deskriptor (engl. *Keypointdescriptor* oder auch nur *Descriptor*) bezeichnet. Ein Schlüsselpunkt setzt sich daraus folgend aus drei Komponenten zusammen:

1.) Die Koordinate, auf der sich der Schlüsselpunkt befindet

2.) Seiner Hauptorientierung (Rotation)

3.) und dem Deskriptor

Im Folgenden wird beschrieben, wie man diese Komponenten nach Lowe bestimmt. Dabei wird auf die wichtigsten Aspekte bezüglich der Erstellung und Zuordnung der Schlüsselpunkte und Deskriptoren eingegangen. Zum tieferen Verständnis der mathematischen Berechnungen wird auf [Low04] verwiesen.

2.3.1 Bestimmung von Schlüsselpunktkandidaten

Um skaleninvariante Schlüsselpunkte zu finden, sind Punkte in unterschiedlichen Skalierungsebenen zu durchsuchen. Diese werden theoretisch mit dem Laplacian of Gaussian (LoG) bestimmt. Da diese Methode allerdings kostspielig ist, wird ein anderes Verfahren genutzt, um ein Ergebnis zu approximieren, das dem Ergebnis des LoG gleicht.
Die Annäherung wird mittels des Difference of Gaussian (DoG) erreicht. Dabei wird das Ursprungsbild abfallend runter skaliert. Jede Skalierungsebene wird als *Oktave* bezeichnet. Jede Oktave besteht aus mehreren Bildern (x_{layer} als Anzahl der Bilder in einer Oktave) gleicher Größe, die unterschiedlich stark (abhängig vom σ-Wert) mit einem Gauß-Filter geglättet werden [2.1].

$$\underbrace{L_\sigma(x,y)}_{Geglaettetes-Bild} = \underbrace{I(x,y)}_{Input-Bild} * \underbrace{G_\sigma(x,y)}_{Gauß-Funktion} = I(x,y) * \left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2+y^2}{2\sigma^2}} \right) \tag{2.1}$$

Die Größe der Bilder einer Oktave schrumpft von einer zur nächsten Oktave auf die Hälfte. Dabei dient das letzte Bild innerhalb einer Oktave als Ausgangsbild der nächsten Oktave. Die Verkleinerung wird erreicht, indem nur jeder zweite Pixel in X- und Y-Richtung übernommen wird. Durch die Gauß-Glättung werden Pixelinformationen an die Nachbarpixel übergeben, sodass der Informationsverlust der Verkleinerung nur gering ausfällt.
Um nun die DoGs zu bestimmen, wird in jeder Oktave die Differenz der Grauwerte zweier benachbarter Bilder errechnet. Daraus resultierend erhalten wir ($x_{layer} - 1$) DoG Bilder (Abb 2.6).

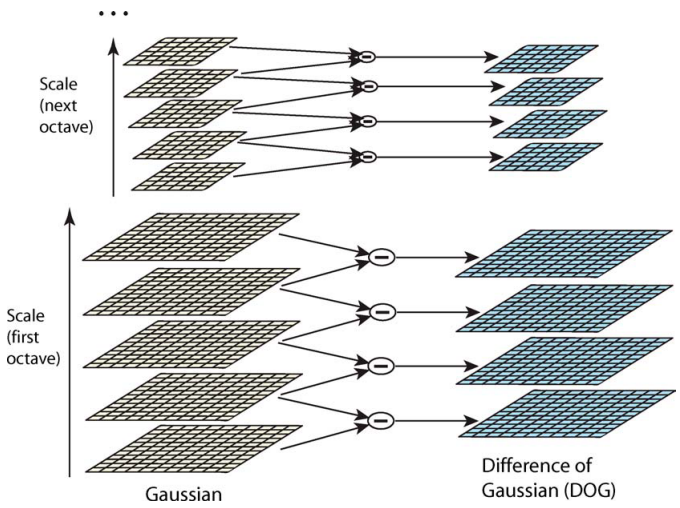

Abbildung 2.6: Darstellung einer Oktave und der daraus resultierende DoG.

Potentielle Schlüsselpunktkandidaten können nun bestimmt werden, indem man die Pixel der DoG Bilder auf lokale Extrema prüft. Ein Pixel wird als ein lokales Extrema identifiziert, wenn es den größten bzw. kleinsten Grauwert innerhalb seiner 26 Nachbarn hat. Die 26 Nachbarn ergeben sich aus den acht Nachbarn der selben Ebene sowie der neun Nachbarn in der vorherigen, sowie nächsten Ebene im Skalenraum (Abb 2.7).

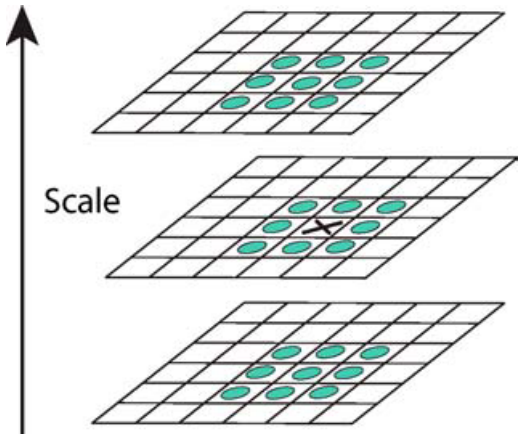

Abbildung 2.7: Prüfung auf lokale Extrema innerhalb der 26 Nachbarn.

2.3.2 Stabilitätsanalyse

Alle im vorherigen Schritt ermittelten Extremwerte sind Schlüsselpunktkandidaten.
Punkte auf ein und der selben Kante haben unter dem *SIFT*-Algorithmus oftmals die

gleichen Merkmalseigenschaften und sind daher nicht zu unterscheiden. Kanten sind deshalb weniger von Bedeutung als Ecken. Homogene Regionen und Kanten werden mithilfe einer Hesse-Matrix gefiltert, und als Schlüsselpunktkandidaten ausgeschlossen. Für den genaueren mathematischen Hintergrund wird auf [HS+88] verwiesen.

2.3.3 Die Hauptorientierung des Schlüsselpunktes

Um die Rotationsinvarianz gewährleisten zu können, wird für jeden Schlüsselpunkt eine Hauptorientierung bestimmt, um somit eine Drehung des Schlüsselpunktes erkennen zu können. Für die Bestimmung der Hauptorientierung wird, ausgehend von dem entsprechenden gaußgelätteten, sowie skalierten Bild in dem das Extrema aufgefunden wurde, für jeden Pixel in der Umgebung des Schlüsselpunktes, der Gradient, bestehend aus Gradientenrichtung (Richtung des größten Helligkeitsanstiegs) [2.2] und dessen Betrag [2.3], bestimmt.

$$\theta(x,y) = \tan^{-1} \frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)} \tag{2.2}$$

$$m(x,y) = \sqrt{\Big(L(x+1,y) - L(x-1,y)\Big)^2 + \Big(L(x,y+1) - L(x,y-1)\Big)^2} \tag{2.3}$$

Diese Gradienten sind als Winkel angegeben und werden in ein 36 Bin (deutsch *Klassen*) Histogramm (1 Bin pro 10°) gewichtet eingetragen und aufsummiert (Abb 2.8). Gewichtet werden die Gradienten zum einen nach ihrer Gradientenlänge m_i, und zum anderen nach ihrer Position in einem um den Schlüsselpunkt zentrierten konzentrischen Fenster, das nach der Gauß-Funktion gewichtet ist (Abb 2.10). Somit haben Gradienten, die weiter vom Schlüsselpunkt entfernt sind weniger Einfluss auf die Hauptorientierung des Schlüsselpunktes.
Das Maximum aus dem Histogramm stellt dann die Hauptorientierung des Schlüsselpunktes dar. Ein besonderer Fall der eintreten kann, ist, dass zu einem Schlüsselpunkt, ein weiterer Schlüsselpunkt erstellt werden kann mit der selben Position, allerdings mit anderer Hauptorientierung. Der Fall tritt ein, falls ein anderer Bin mindestens 80% des absoluten Maximums entspricht.

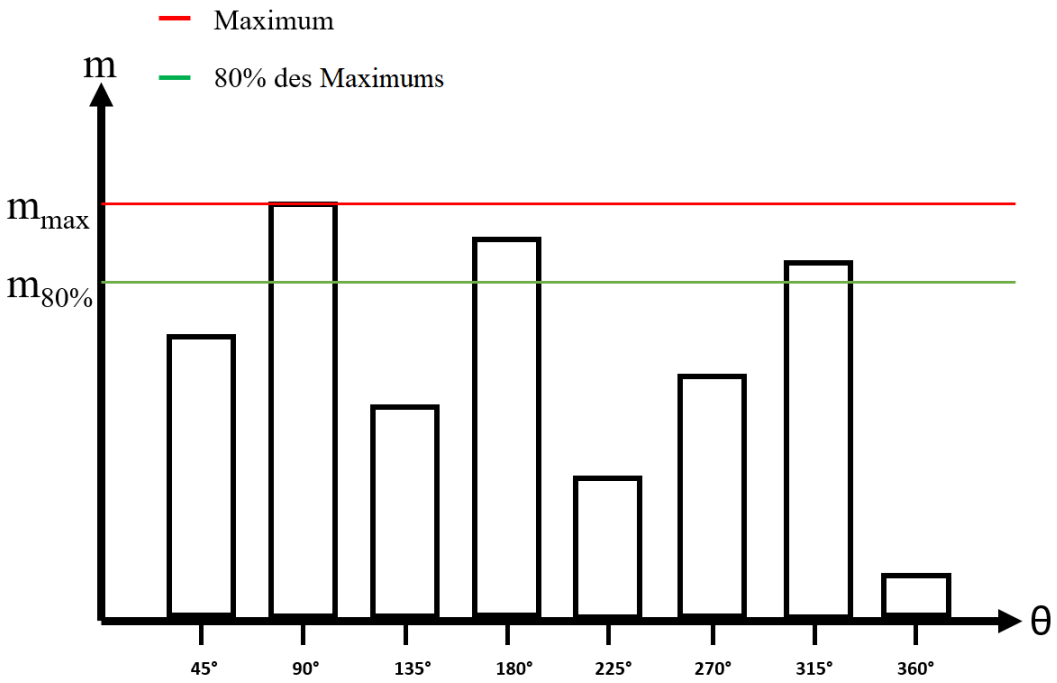

Abbildung 2.8: Bestimmung der Hauptorientierung mittels des Maximums im Histogramm. Der Schlüsselpunkt besitzt zudem zwei weitere Hauptorientierungen. (Zur Veranschaulichung wurde in der Grafik eine Klassenbreite von 45° gewählt, statt der üblichen 10°).

2.3.4 Erstellung des Deskriptor

Der Deskriptor ist der Fingerabdruck des Schlüsselpunktes. Er stellt die Beschaffenheit des Schlüsselpunktes dar und ist notwendig, um die Schlüsselpunkte vergleichbar zu machen, damit diese gegebenenfalls auf anderen Bildern gefunden werden können. Der Deskriptor wird als Vektor dargestellt und enthält Informationen über die Gradienten in der lokalen Umgebung des Schlüsselpunktes.
Ein rotierter Schlüsselpunkt teilt mit seinem nicht rotierten Äquivalent den selben Deskriptor, allerdings haben beide verschiedene Hauptorientierungen.
Betrachtet wird ein 16 x 16 Pixelraum mit dem Schlüsselpunkt als Zentrum. Dieser Raum ist zudem nach der Hauptorientierung des Schlüsselpunktes ausgerichtet und wird nun in 16 gleich große Felder (Subbereiche) (siehe Abb 2.9) aufgeteilt mit jeweils 4 x 4 Pixel.

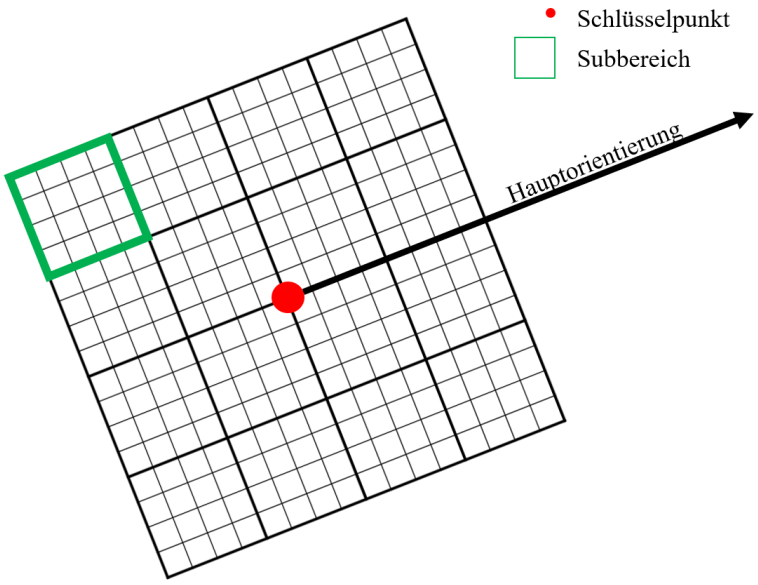

Abbildung 2.9: Schlüsselpunkt mit Umgebungspixel.

Im Abschnitt (Kapitel 2.3.3) wurde die Hauptorientierung des Schlüsselpunktes mittels der Gradienten der Umgebungspixel bestimmt. Diese Gradienten werden nun auch für den Deskriptor benötigt. Für jeden dieser 16 Subbereiche wird mittels der Aufsummierung seiner 16 Gradienten, analog zu (Kapitel 2.3.3 allerdings mit 8 statt 36 Klassen), ein 8-Bin Histogramm erstellt (8 Bins: 8 x 45° = 360°) (Abb 2.10). Alle 8-Bin Histogramme werden zusammen als ein Vektor gespeichert und bilden den Feature Vector/Deskriptor. Daraus ergibt sich, dass jeder Deskriptor aus 128 Daten besteht.
$16 Histogramme \times 8 Bins = 128$.

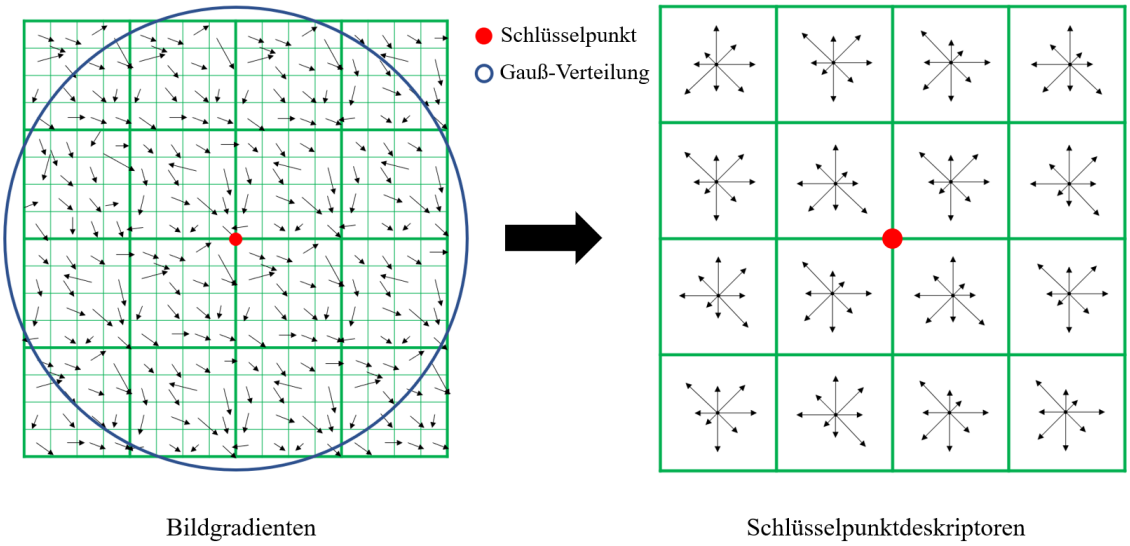

Abbildung 2.10: Bildgradienten, die zu Orientierungshistogrammen mit je 8 Bins zusammengefasst werden und den Schlüsselpunktdeskriptor bilden. (Die Histogramme dienen zur Veranschaulichung, Betrag sowie Richtung der 8-Bin Histogramme korrelieren nicht mit den Bildgradienten).

2.3.5 Matching der Schlüsselpunkte

Bis zu diesem Abschnitt wurden die Position, Hauptorientierung, sowie der Deskriptor von Schlüsselpunkten bestimmt. Um nun Schlüsselpunkte auf zwei verschiedenen Bildern einander zuordnen zu können, falls Gleichheit besteht, bedarf es ein Maß, mit dem dies beurteilt wird.
Lowe verwendet dazu den Euklidischen Abstand. Damit kann der Abstand zweier Deskriptoren bestimmt werden, die wir als Vektoren gespeichert haben.

$$d(x,y) = \|x-y\|^2 = \sqrt{|x_1-y_1|^2 + ... + |x_n-y_n|^2} \quad x,y \in \mathbb{K}^n \tag{2.4}$$

Der Euklidische Abstand erlaubt es uns nun, den nächsten Nachbarn eines Schlüsselpunktes K_{1_i} in der Menge der Schlüsselpunkte K_2 im zweiten Bild zu ermitteln. Der Schlüsselpunkt K_{2_i} aus der Menge K_2 dessen Deskriptor den geringsten Euklidischen Abstand zum Deskriptor von K_{1_i} hat, wird gematched. Dies wird mit jedem Schlüsselpunkt K_1 des Inputbildes durchgeführt.

Um die Qualität des Matchings der Schlüsselpunkte zu verbessern, nutzt *Lowe* den sogenannten *Ratio Test*. Der Abstand von K_{1_i} zu seinem nächsten Nachbarn muss signifikant geringer sein, als zu seinem übernächsten Nachbarn.
Mit dem *Ratio Test* werden weniger aussagekräftige Matches aussortiert. Dies geschieht, indem der Abstand der Deskriptoren des nächsten und übernächsten Nachbarn, K_{2_i} und $K_{2_{i+1}}$, des betrachteten Schlüsselpunktes K_{1_i} berechnet wird. Ist der Abstand zwischen den Deskriptoren groß genug, ist das Matching von K_{1_i} mit K_{2_i} aussagekräftig. Ist der Abstand dagegen zu klein, sind sich K_{2_i} und $K_{2_{i+1}}$ zu ähnlich und das Matching wird aussortiert, da es demnach nicht mehr aussagekräftig ist.

2.3.6 Problematische Faktoren

Die Stärken des SIFT-Algorithmus können unter bestimmten Umständen von Nachteil sein und dazu führen, dass unterschiedliche Objekte nicht differenziert werden können.
Die Rotationsinvarianz kann in einigen Fällen dazu führen, dass nicht identische Objekte als identisch deklariert werden. Beispielsweise könnte der Buchstabe „W“, bei Rotation auch fälschlicherweise als ein „M“ erkannt werden.
Ferner könnten durch die Skalierungsinvarianz, Objekte, die sich hinsichtlich ihrer Größe unterscheiden würden, als gleiches Objekt identifiziert werden.

Ein weiteres Problem liegt bei Bildern mit wenig Inhalt vor (z.B. nur eine Linie). Hierbei kann der Algorithmus keine Schlüsselpunkte generieren, sodass das zu prüfende Bild nicht vergleichbar ist.

2.4 Skelettierung

Monogramme können in Abhängigkeit von ihrer Quelle unterschiedliche Qualitäten aufweisen. Hierbei zählt beispielsweise, welches Zeichenprogramm beziehungsweise Eingabegerät verwendet wurde. Auch das Format, indem die Bilder anschließend gespeichert werden, hat Einfluss auf die spätere Qualität. Abfotografierte oder eingescannte Bilder können unter Bildrauschen leiden.
Bilder, die im SVG-Format (*Scalable Vector Graphics*) erstellt wurden, weisen im Vergleich dazu eine wesentlich höhere Qualität auf, da diese Bilder ohne Qualitätsverluste skalierbar sind. Somit können diese ohne Verlust auf eine vorgegebene Größe skaliert und beispielsweise in einer Datenbank normiert gespeichert werden.
Eine weitere Problematik stellt allerdings generell die Diversität der Quellen da. Da die Erstellung der Monogramme nicht standardisiert ist können Linien auch unterschiedlich breit sein, abhängig von der genutzten Strichstärke.
Mit dem Skelettieren der Bilder wird versucht letzteres Problem zu minimieren. Die Vor-

verarbeitung soll dabei nicht nur die Linien auf eine einheitliche Breite bringen, sondern hat zusätzlich auch den Effekt einer Reduzierung des Bildrauschens.

Im Folgenden wird erläutert, wie die Skelettierung im Allgemeinen funktioniert. Darauf aufbauend wird ausgeführt, wie die Skelettierung in dieser Arbeit implementiert wurde. In den noch folgenden Definitionen der morphologischen Basisoperationen wird im Skelettierungsprozess stets mit dem Negativ eines Bildes gearbeitet (weißes Objekt auf schwarzem Hintergrund).

2.4.1 Skelettierungsbeispiele

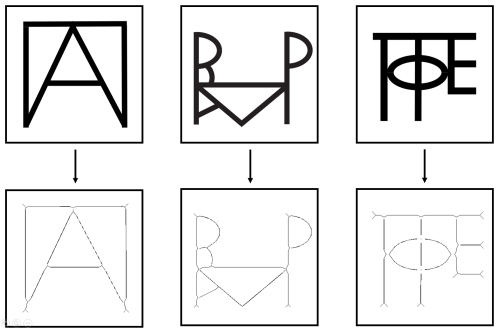

Abbildung 2.11: Monogramme vor und nach der Skelettierung.

2.4.2 Strukturmaske

Die Strukturmaske ist wie ein kleines Fenster, das über das Eingabebild in Pixelschritten verschoben wird. Sie wird einmal komplett über das Eingangsbild verschoben. Dabei wird in jeder Verschiebung/Iteration ausschließlich mit dem Fensterausschnitt gearbeitet. Abhängig von der morphologischen Transformation wird dann das Pixel im Zentrum auf schwarz oder weiß gesetzt (Abb 2.12). Die Strukturmaske kann dabei individuell geformt sein. Gängige Formen sind Rechtecke, Ellipsen und Kreuze (Abb 2.13).

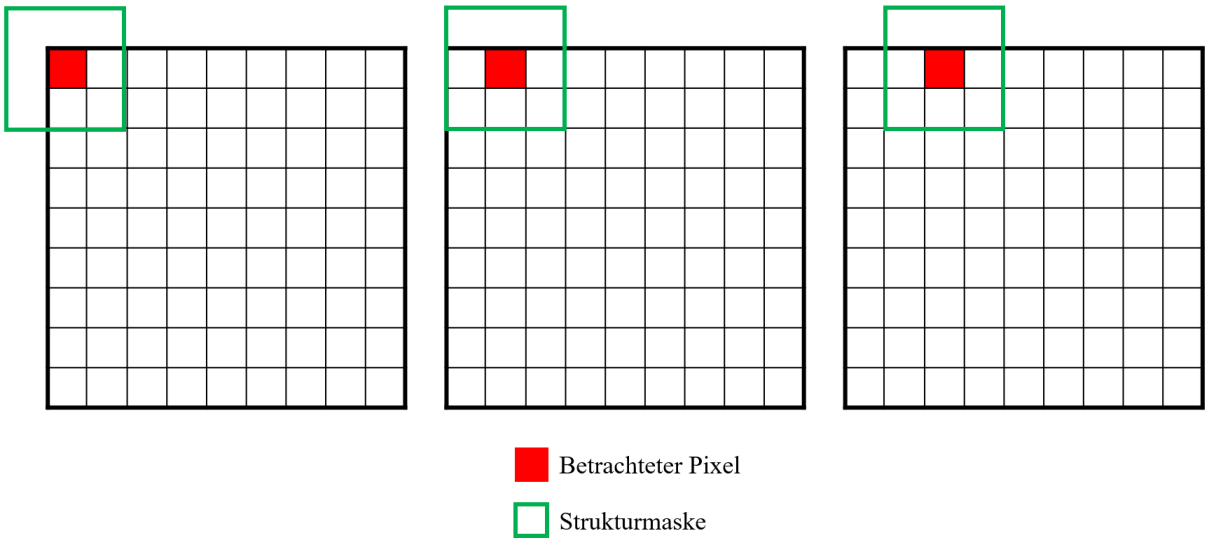

Abbildung 2.12: Verschiebung der Strukturmaske über das Eingabebild.

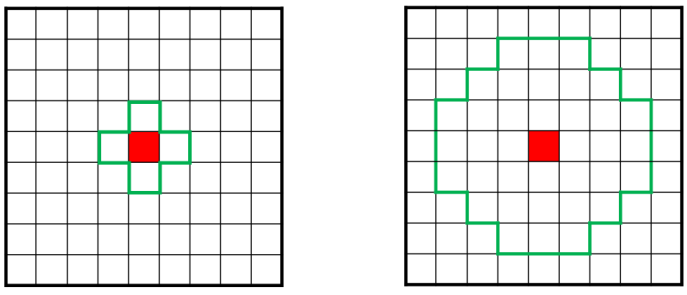

Abbildung 2.13: Strukturmaske in Form eines Kreuzes und einer Ellipse.

2.4.3 Morphologische Basisoperationen

Morphologische Basisoperationen sind Grundoperationen, die bei der Bildverarbeitung mit binären Bildern genutzt werden [ope, Tra92]. Im Folgenden werden weiße Objekte bzw. Buchstaben auf schwarzem Hintergrund betrachtet. Weiße Pixel haben den Wert 1 und schwarze den Wert 0.

Erosion

Bei der Erosion wird bei jeder Iteration, also der Verschiebung der Strukturmaske, betrachtet, von welchen Pixelwerten der momentan betrachtete Pixel im Originalbild umgeben ist. Falls alle Pixel in der Umgebung den Wert 1 haben, wird der Wert des betrachteten Pixels ebenfalls auf 1 gesetzt. In jedem anderen Fall wird der Wert des betrachteten Pixels auf 0 gesetzt.

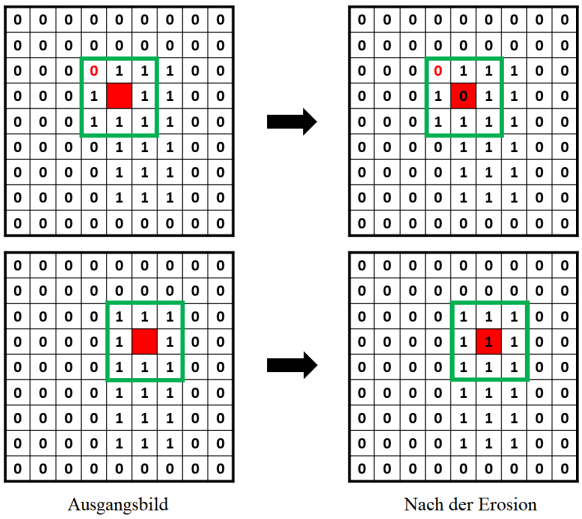

Abbildung 2.14: Im Ausgangsbild wurde absichtlich der Wert des betrachteten Pixels ausgelassen, da dieser keine Rolle spielt.

Daraus folgt, dass das Objekt kleiner wird und Linien schmaler, da alle Pixel am Rand ausgelöscht werden. Ein zusätzlicher Nebeneffekt der Erosion ist, dass das Bildrauschen reduziert wird (Abb 2.15).

Abbildung 2.15: Reduzierung des Rauschens: Die weißen Pixel außerhalb des Buchstaben werden durch die Erosion entfernt.

Dilation

Die Dilation ist die entgegengesetzte Operation zur Erosion. Dabei nimmt der betrachtete Pixel den Wert 1 an, falls mindestens 1 Pixel innerhalb der Strukturmaske den Wert 1 aufweist. Somit werden Objekte größer und Linien dicker.

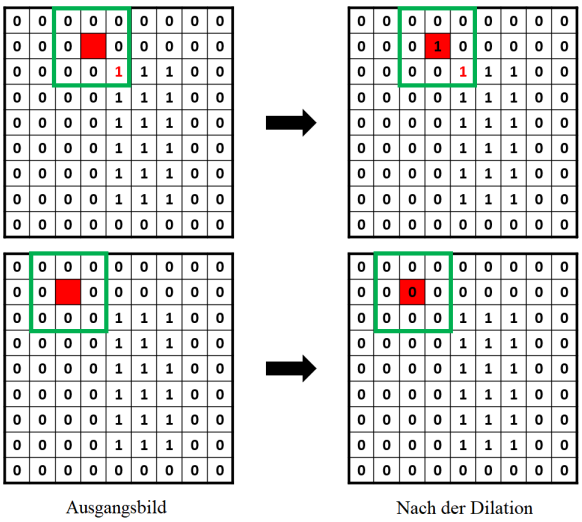

Abbildung 2.16: Die Dilation analog zu (Abb 2.14).

Beim Entfernen von Rauschen wird die Dilation nach der Erosion ausgeführt. Die Erosion entfernt vereinzelte weiße Pixel, wodurch das Objekt aber auch kleiner wird. Die Dilation bringt das Objekt wieder auf die ursprüngliche Größe, ohne dass das Rauschen wiederkehrt.

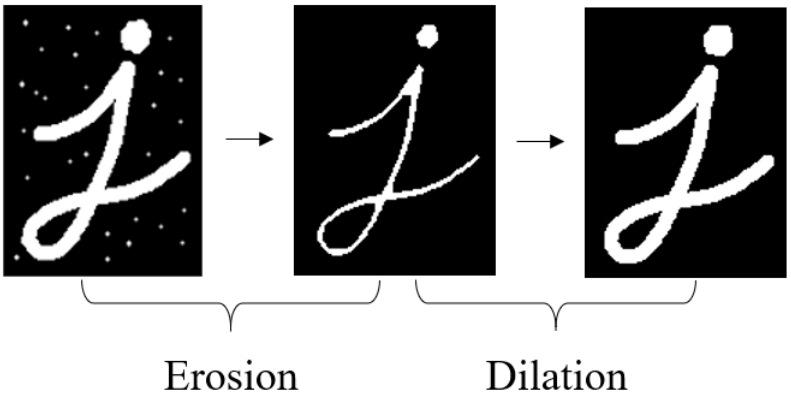

Abbildung 2.17: Durch die Dilation wird der Buchstabe wieder dicker.

3 Konzept und Implementierung

3.1 Vorbetrachtung

Nachdem im ersten Teil dieser Arbeit die theoretischen Grundlagen behandelt wurden, steht nun die eigene Forschung im Vordergrund. Um die Frage zu beantworten und herauszufinden, ob sich *OpenCV* und der *SIFT*-Algorithmus für die effiziente Erkennung von Monogrammen eignet, wurde der Algorithmus mit verschiedenen Parametern, sowie der Skelettierung als Vorverarbeitung implementiert. Es folgen nun detaillierte Erläuterungen zur konkreten Umsetzung der Anwendung, der Vorbereitung der Daten, sowie welche Software und Bibliotheken verwendet wurden.

3.2 Analyse geeigneter Software

3.2.1 IDE

Als IDE wurde in dieser Arbeit PyCharm 2018.2.4 (Community Edition) verwendet.

3.2.2 OpenCV und Numpy

OpenCV (Open Source Computer Vision Library) ist eine freie Programmbibliothek, die eine Vielzahl von Algorithmen für maschinelles Sehen und für die Bildverarbeitung zur Verfügung stellt. In dieser Arbeit wird hauptsächlich der *SIFT*-Algorithmus genutzt.
Die Bibliothek ist für die Programmiersprachen C, C++, Python sowie Java geschrieben.
Anwendungsfelder sind unter anderem:

- Gesichtserkennung

- Gestenerkennung

- Stereoskopisches Sehen

Nach eigenen Angaben wurde *OpenCV* unter dem Fokus von Recheneffizienz und der Umsetzung von Echtzeitanwendungen entwickelt. *OpenCV* ist in C/C++ geschrieben und unterstützt die parallele Verarbeitung mehrerer Prozessorkerne. *Numpy* stellt dabei Datenstrukturen zur Verfügung, auf denen *OpenCV* arbeitet.
Numpy ist eine Python Bibliothek für wissenschaftliches Rechnen. Seine Stärken liegen in

der effizienten Verarbeitung von Vektoren, Matrizen und mehrdimensionalen Arrays. *OpenCV* nutzt die mehrdimensionalen Arrays von *Numpy* zum Einlesen der Bilder.
Für die kommerzielle Nutzung des *SIFT*-Algorithmus bedarf es einer Lizenz. Die Nutzung des Algorithmus für Forschungszwecke ist allerdings kostenfrei.

In dieser Arbeit wird *OpenCV* in der Version 3.3.0 verwendet und *Numpy* in der Version 1.15.2.

3.2.3 QTCreator und PyQT5

Für die automatische Erkennung der Monogramme in dieser Arbeit ist eine GUI (Graphical User Interface) mithilfe der Python-Bibliothek *PyQT5* (in der Version 5.12.1) erstellt worden. Dabei wurde der QTCreator zur Hilfe genommen, der die Umsetzung des Interfaces vereinfacht und übersichtlicher gestaltet.

3.2.4 Serialisierung der Daten mit Pickle

Die Monogramme werden als Python-Objekte instanziiert, was im späteren Abschnitt noch genauer erläutert wird. Der Prozess der Speicherung dieser Objekte wird als Serialisierung bezeichnet. Die Serialisierung der Daten erfolgt mittels der Python-Bibliothek *Pickle*. *Pickle* konvertiert die Struktur eines Python-Objekts in einen Byte-Stream und speichert diesen dann in einer Binärdatei (engl. *Binary file*) ab. Dadurch müssen die Monogramme nicht jedes mal instanziiert werden und können beim Neustart der Anwendung mittels der Binärdatei wieder aufgerufen werden.

Pickle wird in der Version 4.0 verwendet.

3.3 Implementierung

In diesem Abschnitt wird auf die konkrete Implementierung eingegangen. Dabei wird zuerst das Konzept vorgestellt. Anschließend wird darauf eingegangen wie die Daten erhoben und für die Anwendung aufbereitet wurden. Es folgen Erläuterungen zur Implementierung von grundlegenden Elementen der Anwendung und im letzten Teil des Abschnitts wird abschließend die GUI präsentiert.

3.3.1 Konzeptdiagramm

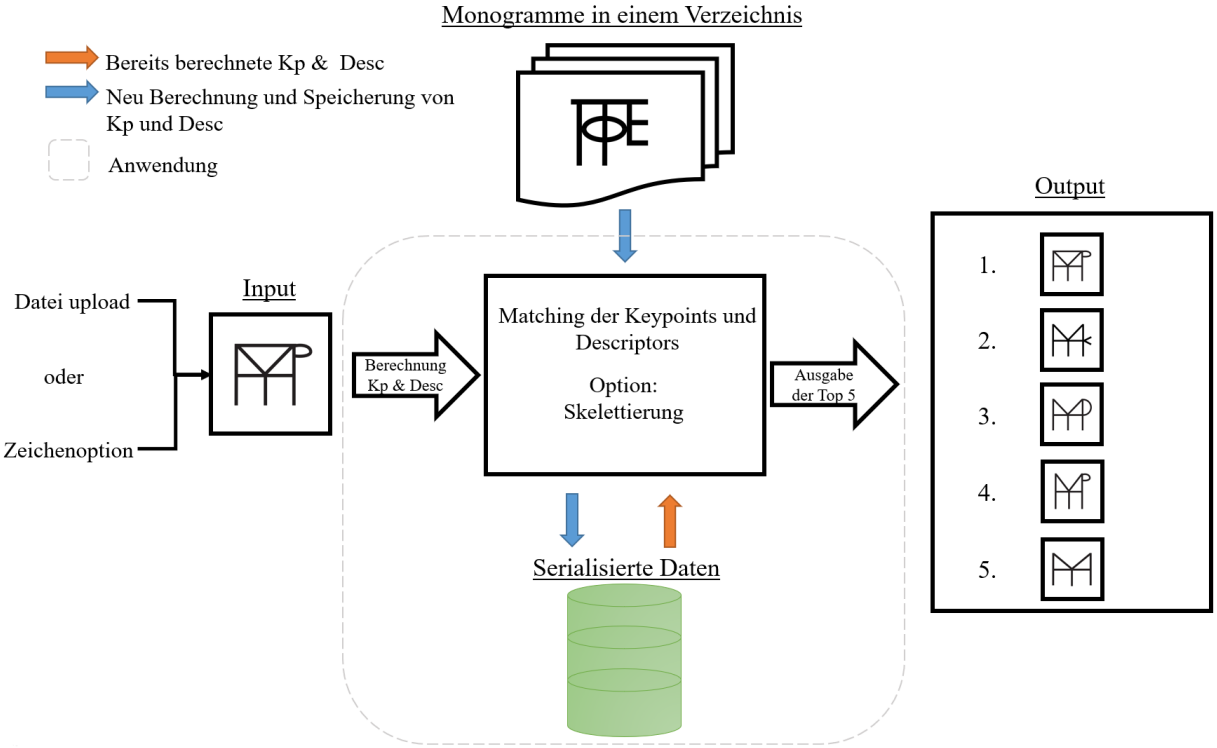

Abbildung 3.1: Ablauf der Erkennung.

In (Abb 3.1) wird schematisch der Aufbau der Anwendung dargestellt. Die Anwendung lässt sich in drei Teile gliedern. Der erste Teil befasst sich mit dem Input, also auf welche Art und Weise das Monogramm eingegeben wird. Der zweite Teil befasst sich mit der Bereitstellung der bereits vorhandenen Daten. Diese können entweder als eine Sammlung von Bildern vorliegen oder auch in einer bereits verarbeiteten Form gespeichert sein. Zu guter Letzt setzt sich der dritte Teil mit der Zuordnung und Rückgabe der Monogramme auseinander.

3.3.2 Erhebung und Aufbereitung der Daten

Die in dieser Arbeit verwendeten Daten sind vom DFG (Deutsche Forschungsgemeinschaft) geförderten CNT-Projekt (*CORPUS NUMMORUM THRACORUM*) [cnt] und teilweise vom *Forvm Ancient Coins* [for] bereitgestellt.
Die Daten, die vom *Forvm Ancient Coins* übernommen wurden, mussten dabei vorbearbeitet werden, da diese im GIF (*Graphics Interchange Format*) vorlagen und einen transparenten Hintergrund aufwiesen. Dieser führte zu Fehlern beim Extrahieren der Features mittels *OpenCV*.
Mittels der Python-Bibliothek *Pillow* wurden alle Bilder ins PNG-Format konvertiert und zusätzlich der Alpha Channel, der für den Undurchlässigkeitswert eines Pixel steht, entfernt (RGBA $\rightarrow$ RGB).
Die Bilder des CNT-Projekts, die im SVG-Format vorlagen, mussten auch ins PNG-Format konvertiert und anschließend vom transparenten Hintergrund befreit werden.[sh]
Für die Konvertierung der Bilder wurde ein separates Pythonskript geschrieben, das alle in einem Ordner befindlichen Bilder ins PNG-Format konvertiert
Die Anwendung unterstützt als Input-Formate: PNG, JPEG sowie TIFF. Das PNG-Format wurde als Outputformat des Konvertierungsskripts gewählt, da im Gegensatz zum JPEG-Format keine Qualitätsverluste bei der Komprimierung entstehen.
Das TIFF-Format eignet sich zum nachträglichen bearbeiten von Bildern, da auch die Ebenen eines Bildes gespeichert werden, allerdings benötigen Bilder im TIFF-Format auch mehr Speicherplatz als Bilder im PNG-Format. Da eine nachträgliche Änderung der Zeichnungen nicht vorgesehen ist, fiel die Auswahl des Outputformats auf das PNG-Format.

3.3.3 Die Klasse: SIFTObject

Die Klasse *SIFTObject* beschreibt die Instanz eines Monogramms. Sie enthält alle relevanten Informationen über das Monogramm, wie zum Beispiel den Namen, das Bild des Monogramms als Grauwerte in Form eines Numpy-Arrays, die durch den *SIFT*-Algorithmus extrahierten Schlüsselpunkte und deren Deskriptoren, und weitere wichtige Attribute, die im unteren Klassendiagramm aufgeführt sind. Es wird für jedes bereits bekannte Monogramm, sowie für das Inputmonogramm ein SIFTObjekt erstellt.

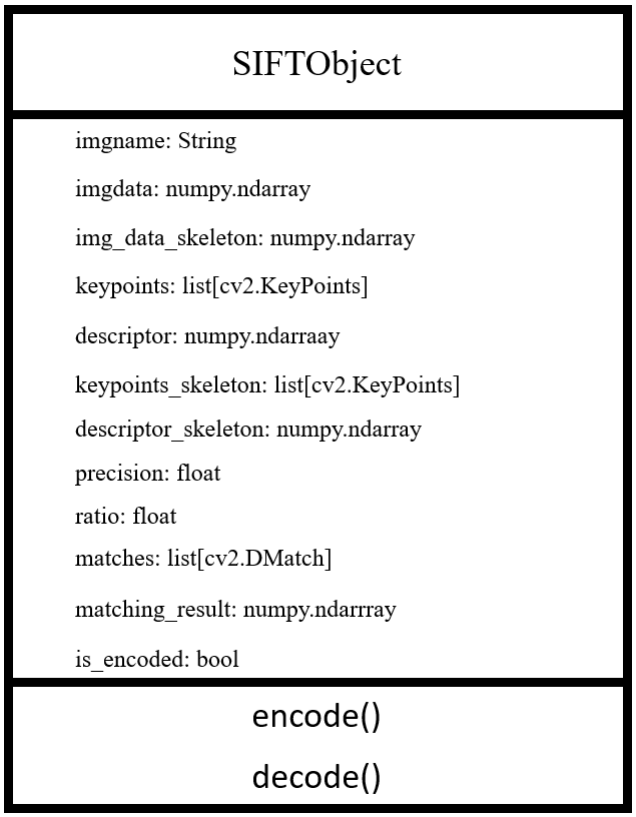

Abbildung 3.2: Klasseninstanz, die ein Monogramm und seine für das Matching relevante Attribute darstellt. Die Funktionen *encode*, *decode* und das Attribut *is_encoded* werden für die Serialisierung der Instanzen benötigt.

3.3.4 Skeleton Implementierung

In vielen Bibliotheken ist bereits ein *Thinning-Algorithmus* implmentiert wie beispielsweise der schnelle parallele *Zhang-Suen Thinning Algorithm*[ZS84] in *Open-CV* und in der Bildverarbeitungsbibliothek *Scikit-Image*. Allerdings liefern die bereits implementierten Thinning-Algorithmen, mit dem in dieser Arbeit verwendeten Datensatz, unbrauchbare Ergebnisse (Abb 3.3). Die Algorithmen entfernen zu viele Pixel, sodass wichtige Kanten verloren gehen können. Daher wird ein selbst implementierter Thinning-Algorithmus verwendet, der die grundlegenden morphologischen Basisoperationen nutzt und anpassbar ist.

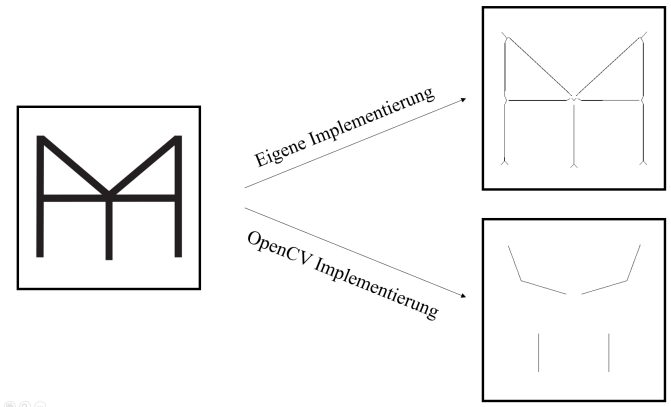

Abbildung 3.3: Resultate der Skelettierung.

Der selbst implementierte Algorithmus ist simpel. Es wird solange eine Erosion ausgeführt bis in der nächsten Iteration kein weißer Pixel mehr verbleiben würde. Anpassbar ist hier die Anzahl der Iterationen bis zur kompletten Auslöschung des Objektes, bevor der Skelettierungsprozess gestoppt wird.

Algorithm 1: Skelettierung [ske, Abe]

Input: Grauwertbild I als numpy.ndarray

Output: Skelett O des Grauwertbild als numpy.ndarray

```
1  Kehre die Pixelwerte (0, 255) von I um;
2  Initalisiere Strukturelement S;
3  Initialisiere Outputbild O;
4  while Nicht alle Pixel schwarz do
5  |  E =Erodiere I mittels S;
6  |  D =Dilation von E mittels S;
7  |  Diff = I − D;
8  |  I = E;
9  |  O += Diff;
10 end
11 Kehre die Pixelwerte von O um;
12 return (O)
```

Als erstes wird das Negativbild des Inputbildes erstellt, da die morphologischen Operationen mit weißen Objekten auf schwarzem Hintergrund arbeiten. Anschließend wird das Strukturelement S definiert, sowie die Platzhaltervariable O, die am Ende den Output enthält. Nun wird der Algorithmus solange ausgeführt, bis durch die Erosion alle weißen Pixel ausgelöscht werden.
Innerhalb der Schleife wird auf dem Bild eine Erosion und Dilation ausgeführt. Die Differenz des Inputbildes und des Bildes nach der Dilation enthält die Linien, die nur noch ein Pixel breit sind. Diese Differenz wird dem Outputbild O in jeder Iteration hinzugefügt. Somit baut sich das Outputbild in jeder Iteration auf. Anzumerken ist, dass in jeder Iteration das Ausgangsbild I durch die erodierte Version in der vorherigen Iteration überschrieben wird.

3.3.5 Einlesen der Daten

In (Abb 3.1) ist der Ablauf der Anwendung schematisch dargestellt. Im ersten Schritt wird ein Bild ausgewählt das Untersucht werden soll. Alternativ besteht auch die Möglichkeit ein Zeichenfenster zu öffnen, um eine eigene Zeichnung als Input zu kreieren.
Nach Auswahl des Inputs werden dessen Schlüsselpunkte und Deskriptoren berechnet. Anschließend werden für das Matching entweder bereits serialisierte SIFTObject-Instanzen aus einer Datei geladen, wodurch eine Berechnung der Schlüsselpunkte und Deskriptoren entfällt oder ein Verzeichnis ausgewählt, in dem sich die Monogramme befinden. Letztere Option erfordert die Berechnung der Schlüsselpunkte und Deskriptoren aller im Verzeichnis befindlichen Monogramme. Dabei werden entsprechende SIFTObject-Instanzen erstellt, serialisiert und als Datei zum Abruf gespeichert.
Die Serialisierung bringt dabei einen Speedup-factor von 45.
Die Rechenzeit wurde mit der Python-Standardbibliothek *time* ermittelt, indem ein Kontrollpunkt vor- und nach der Berechnung des Matchings gesetzt wurde. Die Zeitdifferenz bei dem Durchlauf mit serialisierten Daten wurde anschließend mit der Zeitdifferenz des Durchlaufs ohne serialisierte Daten verglichen. Bei einem Durchlauf ohne serialisierte Daten wurde eine Zeit von 43.4 Sekunden gemessen. Im Gegensatz dazu wurde bei einem Durchlauf mit serialisierten Daten eine Rechenzeit von 0.97 Sekunden gemessen.
Durchgeführt wurde die Messung auf einem Rechner mit den folgenden Spezifikationen:

- OS: Windows 10 64-Bit
- CPU: Intel Core i7-8750H @ 2.20GHz
- RAM: 16GB DDR4

3.3.6 Das Matching

Da für jedes bereits bekannte Monogramm, sowie ebenfalls für das Inputmonogramm die Schlüsselpunkte und Deskriptoren vorliegen, wird im nächsten Schritt ein Matching vom Inputmonogramm zu jedem bereits bekannten Monogramm erstellt. Dabei erhalten wir für jedes Matching eine Liste der erfolgreich zugeordneten Schlüsselpunkte.
Es ist allerdings nicht ausreichend das beste Ergebnis unter den bekannten Monogrammen nur anhand der Anzahl der erfolgreich zugeordneten Schlüsselpunkte festzulegen. Beispielsweise kann ein Monogramm mit vielen Schlüsselpunkten viele falsche Matches erzeugen, das dann besser bewertet wird als ein Monogramm mit vergleichsweise weniger Matches, die allerdings korrekt wären.
Deshalb wird zu jedem SIFTObject das Attribut *ratio* berechnet, das die Anzahl der Schlüsselpunkte des jeweiligen Monogramms bei der Beurteilung der Ähnlichkeit zum Inputmonogramm berücksichtigt. Das Attribut *precision* gibt zum Schluss den Wert an, nachdem die Ähnlichkeit zwischen Inputmonogramm und bekanntem Monogramm bewertet wird.

$$ratio = \frac{\#SchluesselpunkteInputmonogramm}{\#SchluesselpunkteBekanntesMonogramm} \tag{3.1}$$

$$precision = \frac{\#Schluesselpunktmatches}{\#SchluesselpunkteInputmonogramm} \times ratio \tag{3.2}$$

3.3.7 GUI

Die GUI der Anwendung besteht aus drei Fenstern.
Das erste Fenster (Abb 3.4) ist das Hauptfenster. In ihm wird das Inputmonogramm, sowie die Quelle der bereits vorhandenen Monogramme ausgewählt. Außerdem kann der *Ratio test* sowie die Skelettierung aktiviert oder deaktiviert werden.

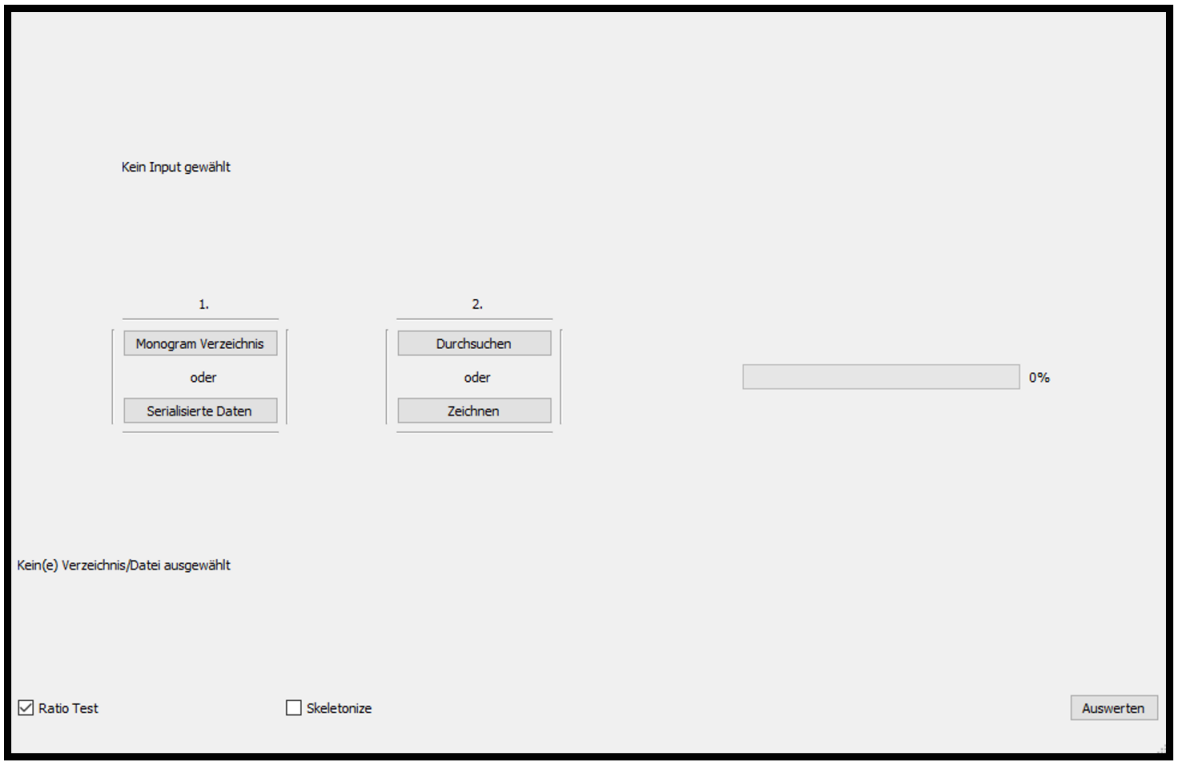

Abbildung 3.4: Dies ist das Hauptfenster.

Das zweite Fenster (Abb 3.5) zeigt die besten fünf Zuordnungen des Inputmonogramms an. Dabei sind die Zuordnungsvorschläge von oben nach unten sortiert, wobei sich ganz oben der beste und ganz unten der schlechteste Vorschlag befindet. Der Klick auf “show match”führt zum letzten Fenster.

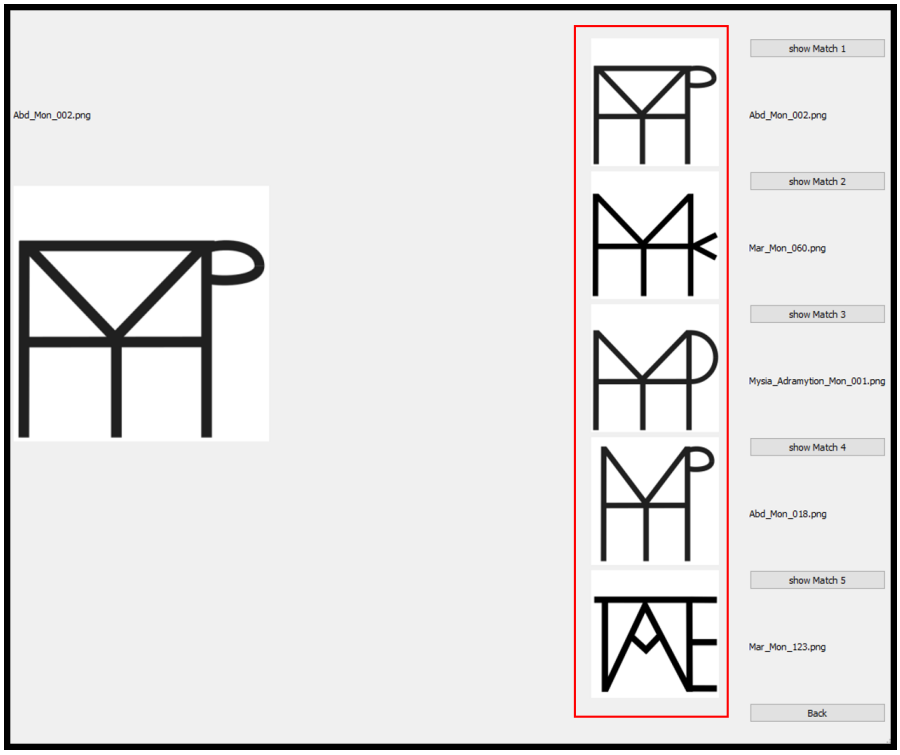

Abbildung 3.5: Das Fenster zeigt die besten fünf Zuordnungen.

Das letzte Fenster (Abb 3.6) zeigt die Schlüsselpunkte für das Inputmonogramm, sowie für das zugeordnete Monogramm. Eine Gerade verbindet die jeweils korrespondierenden Schlüsselpunkte.

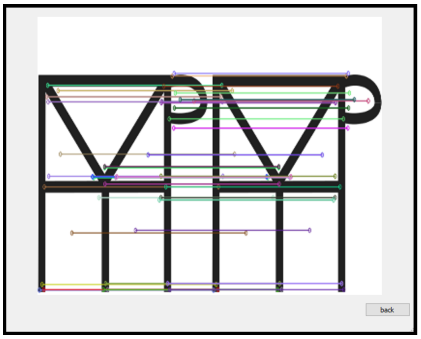

Abbildung 3.6: Das Fenster zeigt die korrespondierenden Schlüsselpunkte.

4 Evaluation & Resultate

Das folgende Kapitel befasst sich mit der Auswertung der entwickelten Anwendung.
Zunächst wird der Aufbau der Evaluation beschrieben, anschließend werden die Ergebnisse betrachtet. Zusätzlich wird der Einfluss von einer Vorverarbeitung der Bilder auf die Vorhersagegenauigkeit des automatisierten Prozesses festgestellt.

4.1 Testdurchführung

Als Standard gesetzt ist ein Matching mit *Ratio Test* (Kapitel 2.3.5). Wahlweise kann die Auswertung mit der Option *Skeletonize* gestartet werden. Die Option führt dazu, dass von jedem Monogramm das Skelett gebildet wird und danach die Schlüsselpunkte und Deskriptoren anhand des Skeletts neu errechnet und mit den neu errechneten Schlüsselpunkten und Deskriptoren des Inputskeletts zugeordnet werden.
Als Datenbasis wurden im ersten Durchgang 415 Monogramme genutzt, die vom SVG-Format ins PNG-Format konvertiert wurden (HD). Im zweiten Durchgang wurden zusätzlich 600 Monogramme mit schlechteren Auflösungen (SD) dazu genommen, die auch im PNG-Format vorlagen. Im dritten Durchgang wurde dann noch zusätzlich die Skelettierung aktiviert.

Getestet wurde demnach mit folgenden Optionen:

1.) ohne Vorverarbeitung (HD Bilder)

2.) ohne Vorverarbeitung (HD + SD Bilder)

3.) mit Skelettierung der Monogramme und Rauschreduzierung (HD + SD Bilder)

Der zweite Testdurchlauf soll zeigen, ob und um wie viel die Erkennungsrate sinkt wenn man Bilder unterschiedlicher Qualität vorliegen hat. Der dritte Testdurchlauf soll zeigen, ob die Skelettierung, durch die Vereinheitlichung der Linien, die Erkennungsrate im Vergleich zum zweiten Testdurchlauf verbessern kann.
Bewertet wurde nach zwei Kriterien: Zum einen, ob das Monogramm identifiziert wird, und zum anderen, wie viele der Bilder, die unter den besten fünf zugeordnet wurden, mindestens eine deckungsgleiche (Abb 4.1) Komponente enthalten.

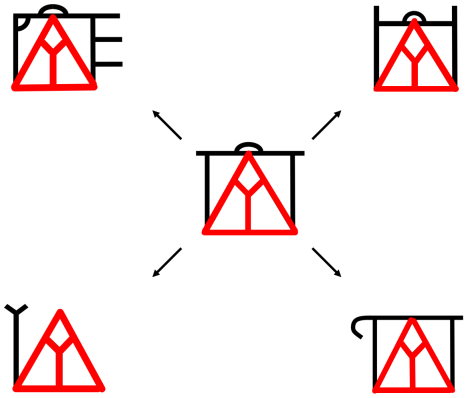

Abbildung 4.1: Rot markiert ist die Komponente, die auf allen Monogrammen deckungsgleich ist.

Getestet wurden in jedem Durchlauf jeweils 50 Monogramme. Im Rahmen dieser Arbeit wird das Matching der besten fünf Bilder als Klassifizierung bezeichnet. Wird ein Monogramm richtig identifiziert, impliziert dies, dass mindestens eines der Bilder deckungleiche Komponenten enthält.
Für die Berechnung der Identifikations- und Klassifikationsrate wurden folgende Gleichungen genutzt:

$$Identifizierungrate = \frac{\#Identifikationen}{50} \tag{4.1}$$

$$Klassifizierungsrate = \frac{\#Deckungsgleichebilder}{5 \times 50} \tag{4.2}$$

Eine Klassifizierungsrate von 80 Prozent bedeutet in dem Fall, dass bei einer Auswertung unter den fünf besten Ergebnissen, in Erwartung vier Bilder davon deckungsgleiche Komponenten besitzen.

4.2 Testergebnisse

Im Folgenden werden die Ergebnisse betrachtet, die sich nach den drei Testdurchläufen ergeben haben.

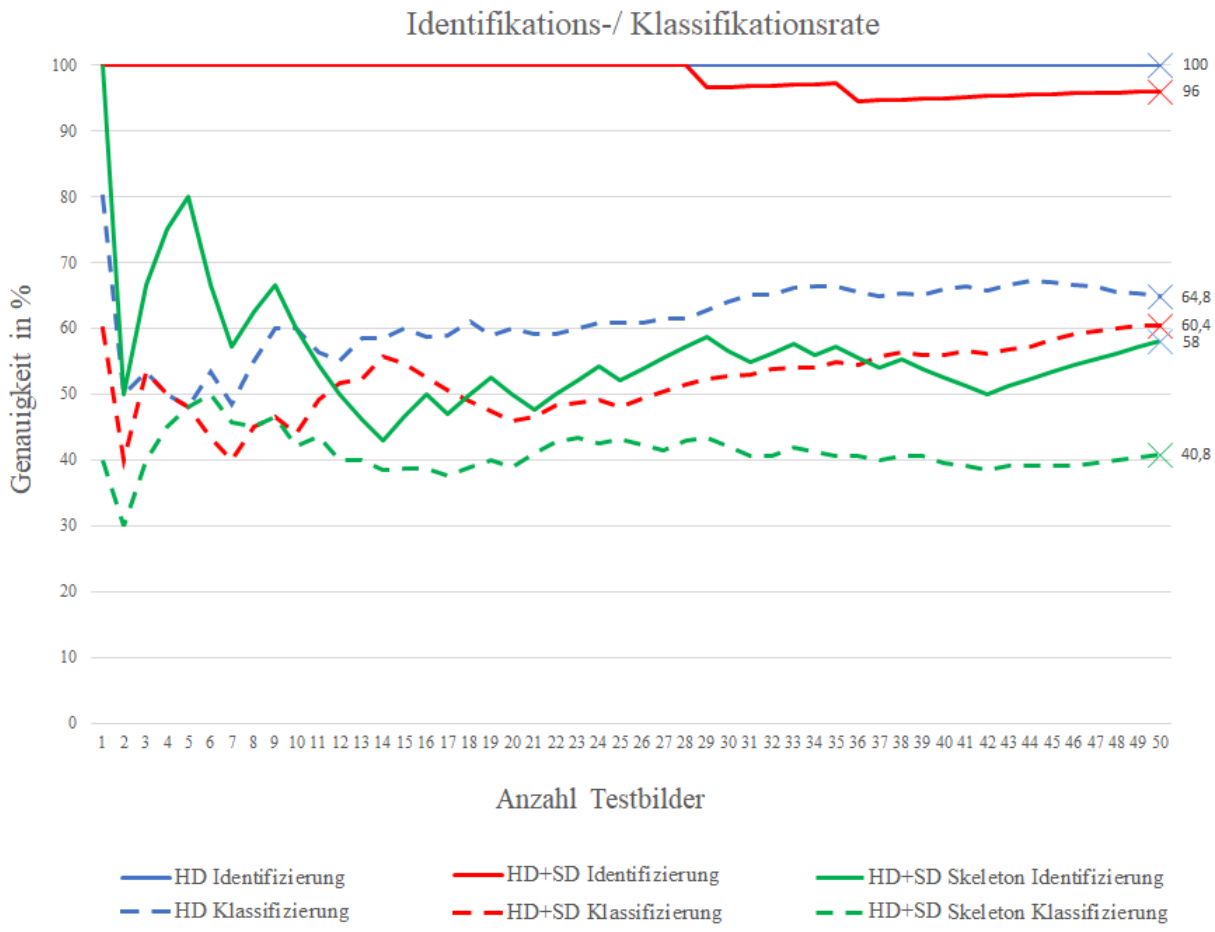

Abbildung 4.2: Die × markierten Werte sind die Genauigkeiten nach dem 50. Testbild.

Die Grafik (Abb 4.2) zeigt deutliche Unterschiede in der Genauigkeit, abhängig von der Qualität und Vorverarbeitung des Inputs. Die Monogramme, die aus den SVG-Daten erstellt wurden, zeigen eine Identifizierungsgenauigkeit von 100 Prozent. Die Identifikations- und Klassifikationsrate bei Hinzunahme der schlechter aufgelösten Bilder sinkt nur circa um vier Prozent. Die vorverarbeiteten Bilder wiederum weisen eine deutlich schlechtere Identifikations- und Klassifikationsrate auf. Am auffälligsten ist, dass die Identifikationsrate im Vergleich zur Auswertung ohne Vorverarbeitung auf fast die Hälfte sinkt.

5 Diskussion

5.1 Analyse der Ergebnisse

Die Testreihe hat gezeigt, dass die Anwendung die besten Ergebnisse mit den hochauflösenden Bildern erzielt. Der Testlauf mit Bildern gemischter Auflösung hat ein minimal schlechteres Ergebnis erzielt. Die Skelettierung hatte allerdings eine signifikante Verschlechterung in der Identifizierungs- sowie Klassifizierungsrate zufolge. Die Verschlechterung legt nahe, dass die Skelettierung neben der Reduzierung der Linienstärke und des Bildrauschens auch die Auslöschung von Schlüsselpunkten als Nebeneffekt hat.

5.2 Diskussion der Methode

Die Resultate zeigen, dass der SIFT-Algorithmus für die automatisierte Erkennung von Monogrammen geeignet ist. Der Versuch der Vereinheitlichung der Bilder mittels der Skelettierung hat zwar zu keiner Verbesserung des Erkennungsprozesses geführt, dennoch liegt die Identifizierungsrate der Anwendung ohne Vorverarbeitung der Bilder bei über 95 Prozent und die Klassifizierungsrate bei über 60 Prozent. Das positive Ergebnis ist wahrscheinlich auch zu einem großen Teil auf die Qualität des Datensatzes zurückzuführen. Dies ändert aber nicht die Tatsache, dass die Anwendung funktioniert und zur Lösung der Aufgabenstellung dieser Arbeit führt.
Für Daten schlechterer Qualität könnten anderweitige Vorverarbeitungsalgorithmen genutzt werden, die bessere Ergebnisse liefern. Denkbar wäre auch gegebenenfalls die Skelettierung anzupassen und eine weiterführende Studie zu führen, bis zu welchem Maße eine Skelettierung ohne den Verlust von wichtigen Merkmalen durchführbar ist. Ebenfalls könnte man prüfen, ob anders geformte Strukturelemente bessere Ergebnisse liefern.

6 Fazit und Ausblick

Ziel dieser Arbeit war es eine Anwendung zu entwickeln, welche Monogramme automatisiert erkennen soll. Der Grundgedanke dahinter bestand darin, in einer Datenbank bereits vorhandene Monogramme zu erkennen oder alternativ strukturell ähnliche Monogramme vorzuschlagen. In Erwägung gezogen wurden verschiedene Implementierungsansätze, unter welchen sich die Erkennung mittels Feature Extraction, mit Anwendung des *SIFT*-Algorithmus, als am geeignetsten beurteilt wurde. Mit der Implementierung des *SIFT*-Algorithmus konnten Ergebnisse erzielt werden, die einen Einsatz der Anwendung für die automatisierte Erkennung von Monogrammen, zumindest auf den in dieser Arbeit verwendeten Daten, befürworten. Ob sich die Anwendung auch im realen Einsatz tatsächlich bewährt, kann nicht garantiert werden. Die Ergebnisse dieser Arbeit zeigen zumindest ein großes Potential der Automatisierung durch *OpenCV* mit Anwendung des *SIFT*-Algorithmus.
Ein zukünftiges Forschungsthema könnte die weitere Verbesserung des Erkennungsprozesses sein. Es könnten geeignetere Vorverarbeitungsmodule entworfen werden, die die Erkennungsrate von qualitativ schlechteren Bildern erhöht.
Ein weiteres Thema wäre die Verbesserung der Recheneffizienz. Denkbar wäre eine Parallelisierung des Algorithmus. Zum einen könnten Schlüsselpunkte, sowie Deskriptoren auf mehreren Threads gleichzeitig berechnet werden und zum anderen könnte das paarweise Matching der einzelnen Bilder auch parallel erfolgen.
Ein weiterer Ausblick besteht in der Umsetzung mittels eines künstlichen neuronalen Netzes. Denkbar wäre es, dass bei einer zukünftig größeren Datenmenge (10.000 - 100.000 Monogramme) ein künstliches neuronales Netz bessere Erkennungsraten erzielen kann, als der *SIFT*-Algorithmus, da ein künstliches neuronales Netz bessere Ergebnisse liefert, je mehr Daten zur Verfügung stehen.

Anhang

Das gesamt Projekt mit Quellcode und Executable ist auf Github hochgeladen.
URL: https://github.com/Hyu22/OpenCV

Abbildungsverzeichnis

Literaturverzeichnis

[Abe] Félix Abecassis. Opencv - morphological skeleton. http://felix.abecassis.me/2011/09/opencv-morphological-skeleton/. [Online, Accessed: 2019-05-06].

[cnt] Corpus nummorum thracorum. https://www.corpus-nummorum.eu. [Online, Accessed: 2019-05-06].

[for] Forvm ancient coins. https://www.forumancientcoins.com/numiswiki/view.asp?key=Monogram. [Online, Accessed: 2019-05-06].

[Fri10] Carsten Fries. Objekterkennung mit sift-merkmalen. https://users.informatik.haw-hamburg.de/~ubicomp/projekte/master09-10-aw1/fries/bericht.pdf, 2010. [Online, Accessed: 2019-05-06].

[HS+88] Christopher G Harris, Mike Stephens, et al. A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244. Citeseer, 1988.

[L+99] David G Lowe et al. Object recognition from local scale-invariant features. In *iccv*, volume 99, pages 1150–1157, 1999.

[LMT+07] Jun Luo, Yong Ma, Erina Takikawa, Shihong Lao, Masato Kawade, and Bao-Liang Lu. Person-specific sift features for face recognition. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*, volume 2, pages II–593. IEEE, 2007.

[Low04] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[MC14] Hauke Martens and Jören Carstens. Scale invariant feature transfor. https://kogs-www.informatik.uni-hamburg.de/~seppke/content/teaching/sose16/bvproj/CM-SIFT15.pdf, 2014. [Online, Accessed: 2019-05-06].

[ope] Morphological transformations. https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html. [Online, Accessed: 2019-05-06].

[QA09] Muhammad Tahir Qadri and Muhammad Asif. Automatic number plate recognition system for vehicle identification using optical character recognition. In *2009 International Conference on Education Technology and Computer*, pages 335–338. IEEE, 2009.

[sh] streetlogics (https://stackoverflow.com/users/542550/streetlogics). Convert svg to png in python. `https://stackoverflow.com/questions/6589358/convert-svg-to-png-in-python/19718153#19718153`. [Online, Accessed: 2019-05-07].

[SHK+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[ske] Skeletonization using opencv-python. `http://opencvpython.blogspot.com/2012/05/skeletonization-using-opencv-python.html`. [Online, Accessed: 2019-05-06].

[Tra92] Panos E Trahanias. Binary shape recognition using the morphological skeleton transform. *Pattern recognition*, 25(11):1277–1288, 1992.

[ZS84] TY Zhang and Ching Y Suen. A fast parallel algorithm for thinning digital patterns. *Communications of the ACM*, 27(3):236–239, 1984.