Universität des Saarlandes
Fachbereich Informatik

Masterarbeit

Klassifikation von Werkstoffgefügebildern aufgrund teilchenbasierter Kenngrößen

vorgelegt von:

Corinna Richter

Erstprüfer:
Prof. Dr. Gerhard Weikum

Zweitprüfer:
Prof. Dr. Frank Mücklich

Saarbrücken, den 31. März 2005

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Als Gusseisen bezeichnet man eine Eisenlegierung mit einem hohen Anteil von Kohlenstoff und Silizium sowie weiteren Bestandteilen wie Mangan, Chrom oder Nickel. Die bekannteste und am weitesten verbreitete Sorte des Gusseisens ist der sogenannte Grauguss, bei dem der Kohlenstoffüberschuss in Form von Graphitlamellen vorliegt. Betrachtet man die Produktionsmenge in den führenden Industrienationen und weltweit, so nimmt die Werkstoffgruppe des Gusseisens einen der führenden Plätze ein [15] und findet in verschiedenen Industriezweigen Verwendung. Dabei unterscheidet man nach DIN EN ISO 945 verschiedene Typen von Gusseisen (vgl. Abbildung 1.1). Diese unterscheiden sich nicht nur in der Form und Anordnung der jeweiligen Graphiteinlagerungen voneinander. Sie differieren auch grundlegend bezüglich ihrer mechanischen und physikalischen Eigenschaften. Insbesondere hängen diese (Zugfestigkeit, Dauerfestigkeit, Bruchsicherheit, etc.) davon ab, welche Spannungskonzentrationen bei Belastung an den Graphiteinschlüssen entstehen, was wiederum stark von der Form der Einlagerungen bestimmt wird. Für die einzelnen Industriezweige ist die Verwendung von Gusseisen des geeigneten Typs in einer spezifizierten Qualität von größter Bedeutung. Was für einen Verwendungszweck ein sehr geeignetes Material darstellt, kann in einer anderen Branche verheerende Folgen haben. Beispielsweise werden in der Fahrzeugindustrie Bremsscheiben aus einer bestimmten Sorte von Grauguss hergestellt. Schon geringe qualitative Schwankungen können das Verhalten der Bauteile stark beeinflussen, was unter Umständen katastrophale Folgen für den Fahrer hat. Die Anforderungen, die an eine korrekte Klassifikation des verwendeten Materials gestellt werden, sind also hoch. Der Prozess der Klassifikation liegt vielfach in Händen von Spezialisten, die das Material genau begutachten und dann auf der Basis von Richtreihenbildern ei-

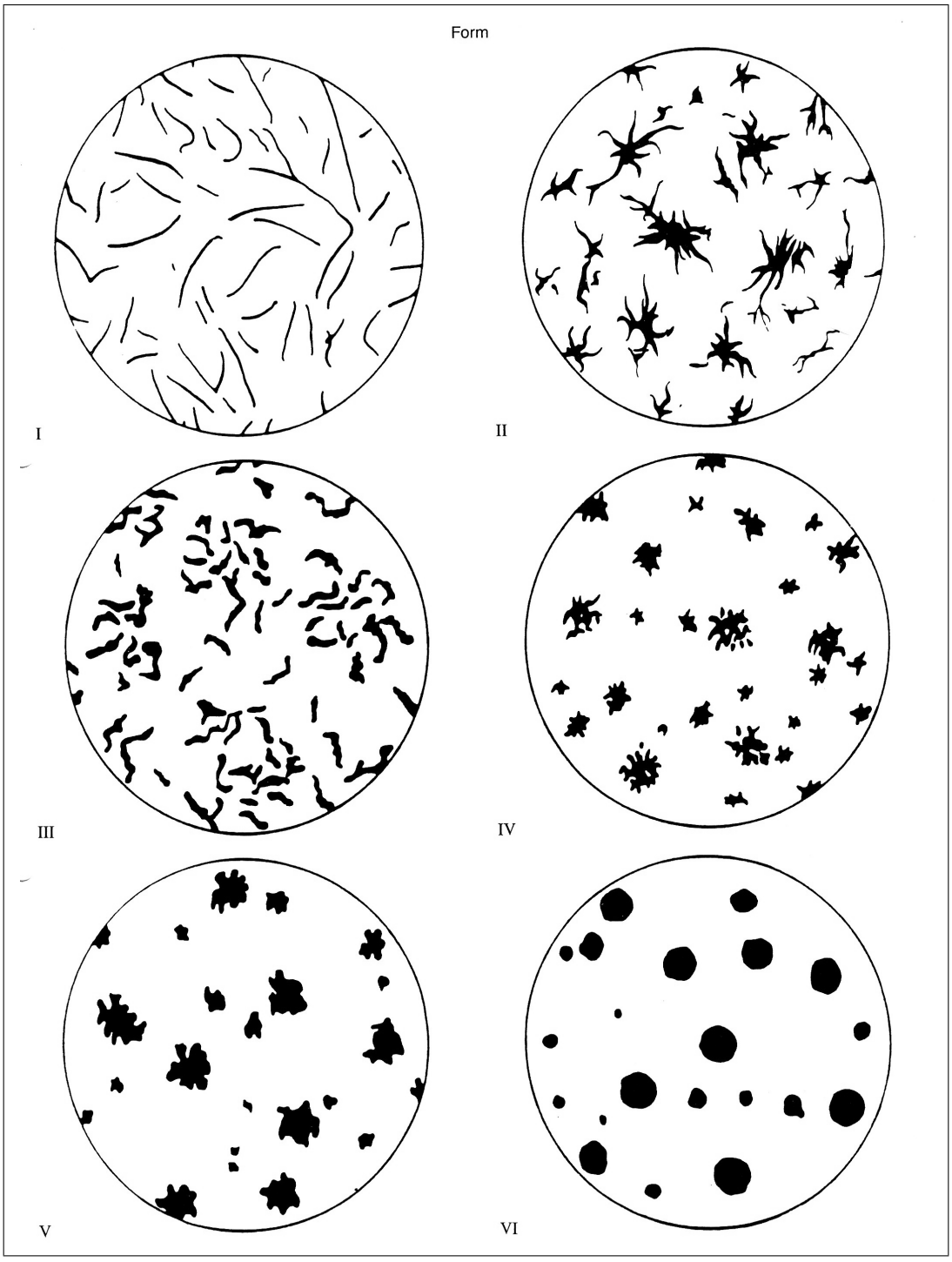

Abbildung 1.1: Richtreihenbilder nach DIN EN ISO 945

nem Graphittyp zuordnen. Eine solche - rein visuelle - Klassifikation ist naturgemäß sehr fehleranfällig, zeitaufwendig und lässt breiten Interpretationsspielraum [15]. Daher wuchs schon bald das Interesse an Verfahren, die moderne Techniken der Bildverarbeitung und der automatischen Klassifikation nutzen.

Vor diesem Hintergrund entstand 2003 die Arbeit von Kathrin Roberts [21], auf deren Ergebnissen ich mit der vorliegenden Arbeit aufbaue. Roberts entwickelte ein System zur Klassifikation der verschiedenen Anordnungen des Lamellengraphits (ebenfalls spezifiziert in Form von idealisierten Richtreihenbildern in DIN EN ISO 945). Dabei werden Gefügebilder des Werkstoffes über ein Webinterface geladen, analysiert und mittels einer Support Vector Machine (SVM) klassifiziert. Da sich die Unterklassen des Lamellengraphits durch die *Anordnung* der einzelnen Lamellen zueinander auszeichnen, wurden dabei messfeldbasierte gefügecharakteristische Kenngrößen verwendet, d.h. es wurde jeweils die gesamte Struktur des Bildes betrachtet. Das Ziel der vorliegenden Arbeit war es, die SVM-Klassifikation auch für die anderen Graphittypen zur Verfügung zu stellen. Da hierbei jedoch v.a. die Parameter der *einzelnen Graphitteilchen* eine Rolle spielen, sollten einzelne Partikel des Bildes auf der Basis von teilchenbasierten Methoden und Kenngrößen klassifziert werden. Eine solche teilchenbasierte Klassifikation ist nicht grundsätzlich neu (vgl. [14], [3], [22] und [13]). Die Auswahl der Parameter, die für die Klassifikation bisher heran gezogen werden, ist jedoch keineswegs einheitlich. Ziel dieser Arbeit war es nun, eine möglichst breite Palette teilweise wenig untersuchter Parameter mit in die Klassifikation einzubeziehen. In einem zweiten Schritt sollte dann der Einfluss der einzelnen Parameter auf das Ergebnis der Klassifikation analysiert werden.

Dabei sollten nach DIN EN ISO 945 folgende Graphitmorphologien in Gusseisen berücksichtigt werden:

Typ I (Lamellengraphit) Im Gusseisen mit Lamellengraphit liegen die Graphiteinlagerungen in Form von dünnen unregelmäßigen Plättchen vor (vgl. Abbildung 1.1, oben links). Diese Lamellen wirken bei Zugbelastung als vorgegebene Einkerbung, so dass das Material an diesen Stellen leicht bricht. Lamellengraphit ist insgesamt relativ spröde. Doch ist dieser Graphittyp effizient in der Wärmeleitung, verfügt über gute Dämpfungseigenschaften und gute Bearbeitbarkeit. Innerhalb des Lamellengraphits unterscheidet man wiederum nach DIN EN ISO 945 je nach Anordnung und Größe der Lamellen die (Sub-)Typen A, B, C, D und E. Die Abgrenzung dieser Klassen wurde in verschiedenen Veröffentlichungen diskutiert (u.a. [21] und [15]) und soll in dieser Arbeit nicht behandelt werden.

Typ III (Vermikulargraphit) Beim Gusseisen mit Vermikulargraphit (vgl. Abbildung 1.1, Mitte links) handelt es sich um einen Eisengusswerksoff mit Graphiteinschlüssen, die in korallenbaumartiger Graphitverzweigung, aber auch in Form von einzelnen Vermikeln (Würmchen) vorliegen. Die Spannungskonzentrationen an den Graphiteinlagerungen bei Belastung sind im Vergleich zum Typ I niedriger, jedoch höher als bei Typ IV bis VI.

Typ IV/V Typ IV (vgl. Abbildung 1.1, Mitte rechts) liegt normalerweise in Temperguss vor. Temperguss bezeichnet einen Eisengusswerkstoff, der in Sandformen abgegossen wird und zunächst üblicherweise hart und spröde ist. Anschließend wird er in einem Glühprozess, dem sogenannten 'Tempern', gedehnt und bearbeitbar gemacht. Temperguss ist somit ein Werkstoff, der plastisch verformbar ist und Belastungen in Form von Schwingungen und in gewissem Umfang auch in Form von Stößen oder Schlägen abfangen kann, ohne zu reißen oder zu brechen. Er verfügt über eine Knautschzone [12]. Typ V (vgl. Abbildung 1.1, unten links) liegt aufgrund seiner Charakteristika zwischen Graphit in Temperguss und Kugelgraphit. Da er Typ IV stark ähnelt, wurden beide Typen in dieser Arbeit als kombinierte Klasse behandelt (Typ IV/V).

Typ VI (Kugelgraphit) Bei dieser Art des Gusseisens liegen die Graphiteinlagerungen in kugelartiger Form vor (vgl. Abbildung 1.1, unten rechts). Gusseisen mit Kugelgraphit ist kein einzelner Werkstoff, sondern eine Familie von Werkstoffen, die durch die Möglichkeit der Steuerung des Feingefüges ein breites Spektrum an Eigenschaften bietet. Das charakteristische Merkmal, das alle Arten von Gusseisen mit Kugelgraphit verbindet, ist die mehr oder weniger sphärolithische Form der Graphitkugeln. Die Kugeln verhindern die Rissausbreitung aufgrund des stumpfen Rissgrunds und machen das Gusseisen mit Kugelgraphit duktil (zäh). Dadurch besitzt das Material sehr günstige Eigenschaften bzgl. Bruchsicherheit, Zugfestigkeit und Dauerfestigkeit.

1.2 Problemstellung

Ziel der vorliegenden Arbeit war die Entwicklung eines Systems, das die vier genannten Graphittypen aufgrund teilchenbasierter Kenngrößen klassifiziert. Dabei waren folgende Teilprobleme zu lösen:

- Umwandlung der Gefügebilder in ein internes, binäres Format
- Bestimmung der Lage der Teilchen im Bild

- Extraktion der Teilchen und Berechnung der grundlegenden Parameter (Umfang, Fläche, Konvexe Hülle, Feretsche Durchmesser)
- Berechnung der gewählten Formparameter
- Anbindung einer Support Vector Machine (Training und Klassifikation)
- Bestimmung der für die Klassifikation relevanten Merkmale im Rahmen einer Sensitivitätsanalyse
- Persistente Speicherung der erstellten Klassifikatoren
- Anbindung an ein webbasiertes Benutzerinterface

In den folgenden Kapiteln wird die Lösung dieser Teilprobleme beschrieben. Kapitel 2 stellt zunächst die Methoden zur Bildanalyse und zur Parameterberechnung vor. In Kapitel 3 wird der Prozess der Klassifikation näher beschrieben. Kapitel 4 geht anschließend auf die Sensitivitätsanalyse ein, mittels derer die Gewichtung der einzelnen Klassifikationsparameter errechnet wurde. Ein Überblick über die Software-Architektur und das Datenmodell wird in Kapitel 5 gegeben. Kapitel 6 stellt die Ergebnisse und den entwickelten Prototypen POCA (**P**art-**O**riented **C**lassification and **A**nalysis) vor. Eine kurze Zusammenfassung und ein Ausblick auf mögliche Erweiterungen in Kapitel 7 runden die Arbeit ab.

Kapitel 2

Analyse des Bildes und Berechnung der Parameter

In diesem Kapitel wird das verwendete Verfahren der Bildanalyse näher erläutert. Dabei sollen in Abschnitt 2.1 zunächst einige grundlegende Begriffe geklärt werden, die für digitalisierte Bilder und für das weitere Verständnis der Arbeit von Bedeutung sind. Anschließend folgt in Absatz 2.2 eine Beschreibung, wie die Teilchen aus dem Bild isoliert und wie ihre Grundparameter berechnet werden. Unter Grundparameter sei hier eine Eigenschaft des Partikels verstanden, die zwar nicht direkt in den Klassifikationsprozess eingeht, die aber benötigt wird, um die Klassifikationsparameter zu berechnen. Letztere werden detailliert in Abschnitt 2.3 dargelegt.

2.1 Bildanalyse

Die zu analysierenden Bilder sind Ergebnisse digitaler Aufnahmen von lichtmikroskopischen Abbildungen[1] und liegen zunächst als Grauwertbilder vor. Um verschiedene Parameter der Partikel messen zu können, muss das Grauwertbild in ein Binärbild umgewandelt werden. Auf den Prozess der Binarisierung soll im Rahmen dieser Arbeit nicht gesondert eingegangen werden; er wird u.a. von Roberts in ihrer Diplomarbeit [21] diskutiert. Für die vorliegende Arbeit werden binarisierte Gefügebilder in den Formaten .png, .gif oder .tif vorausgesetzt. Diese Bilder werden von der Anwendung in ein binäres, nicht komprimiertes Format konvertiert. Analog zur Arbeit von Roberts wird auch hier das PBM-Format verwendet [21], das leicht maschinenlesbar ist. Nach der Konvertierung liegt das Bild als Bitmap vor und kann als Bytearray weiter verarbeitet werden.

[1] genauere Hinweise hierzu finden sich u.a. in [15]

```
P1
# info.pbm
24 7
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 0 1 0 0 0 1 0 0 1 1 1 1 0 1 1 1 1 0
0 0 0 1 0 0 0 1 1 0 0 1 0 0 1 0 0 0 0 1 0 0 1 0
0 0 0 1 0 0 0 1 0 1 0 1 0 0 1 1 1 0 0 1 0 0 1 0
0 0 0 1 0 0 0 1 0 0 1 1 0 0 1 0 0 0 0 1 0 0 1 0
0 1 1 1 1 1 0 1 0 0 0 1 0 0 1 0 0 0 0 1 1 1 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Abbildung 2.1: Binärbild im PBM-Format [21]

2.1.1 Das Bild im PBM-Format

Das PBM-Format soll an dieser Stelle nur sehr kurz vorgestellt werden, da es schon in der Arbeit von Roberts [21] ausführlich erläutert wurde. PBM (Portable Bit Map) ist ein binäres Bildformat, das in seiner Rohform jedes Pixel als '0' (für einen weißen Bildpunkt) oder '1' (für einen schwarzen Bildpunkt) speichert. Abbildung 2.1 zeigt ein Beispiel einer PBM-Datei. Dabei wird in der ersten Zeile das verwendete Format näher präzisiert: P1 steht für das rohe, nicht kodierte PBM-Format (bei anderen Typen muss der Inhalt noch einem Dekodierungsprozess unterzogen werden). Anschließend folgen beliebige Kommentarzeilen, jeweils eingeleitet durch '#'. Verpflichtend ist die Angabe der Auflösung des Bildes, ausgedrückt durch die Höhe und Breite des Pixelrasters. Aufgrund dieser Angabe ist es möglich, die Position der einzelnen Pixel richtig zu interpretieren: Die Zeilen der Datei sind in ihrer Breite beschränkt, so dass Zeilenumbrüche nicht analog zum Pixelraster erfolgen.

2.1.2 Nachbarschaft, Zusammenhang und Kontur

In diesem Abschnitt sollen einige grundlegende Definitionen, auf denen die weitere Arbeit aufbaut, festgelegt werden. Dabei beziehe ich mich zunächst auf die Festlegungen von Roberts (vgl. [21], S.27f) bezüglich des Binärbildes:

Definition 1. Eine bestimmte Menge an Pixeln mit der Farbe F1 (üblicherweise schwarz) stellt die relevante Information dar und wird als *Bildvordergrund* bezeichnet. Die restlichen Pixel des Bildes mit der Farbe F2 (üblicherweise weiß) bilden den *Bildhintergrund*. Dabei ist F1 ungleich F2.

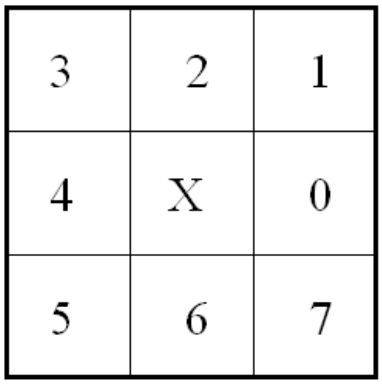

Abbildung 2.2: Pixel X mit den 8 möglichen Nachbarpixeln

Definition 2. Es wird ein Bild L mit der Auflösung $b \cdot h$ betrachtet: Sei $L_b = 0, 1, 2, .., b-1$ die Numerierung der Pixel in horizontaler Richtung und $L_h = 0, 1, 2, .., h-1$ die Numerierung der Pixel in vertikaler Richtung. Dann gibt das Paar (b_i, h_i) mit $b_i \in L_b$ und $h_i \in L_h$ die *Pixelposition* an.

Definition 3. Die *k-Nachbarschaft* eines gegebenen Pixels X bilden k der an X angrenzenden Pixel.

Dabei sind insbesondere die 4er- und die 8er-Nachbarschaft von Bedeutung. Abbildung 2.2 zeigt alle 8 möglichen Nachbarpixel von X bezüglich der 8er-Nachbarschaft. Die 4er-Nachbarschaft schränkt die Nachbarpixel auf die Pixel 0,2,4,6 ein.[2]

Definition 4. Ein Pfad $(p_0, p_1, .., p_n)$ von Pixelposition $p_0 = (b_0, h_0)$ wird als *Pixelsequenz basierend auf k-Nachbarschaft* bezeichnet, wenn für alle i mit $1 \leq i \leq n$ ein Pixel p_i in der Nachbarschaft von p_{i-1} liegt.

Definition 5. Eine Pixelmenge S wird als *Pixelmenge basierend auf k-Nachbarschaft* bezeichnet, wenn

1. zwei beliebige Pixel von S über eine vollständig in S liegende Pixelsequenz basierend auf k-Nachbarschaft verbunden sind oder

2. S nur ein Pixel enthält.

[2]Abweichend von Roberts [21] verwende ich an dieser Stelle eine etwas andere Numerierung der Pixel, entsprechend den Richtungscodes von Freeman, die in 2.2.1 eine Rolle spielen werden.

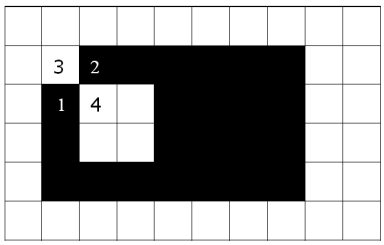

Abbildung 2.3: Unterschiedliche Nachbarschaftsbeziehungen für Vorder- und Hintergrund

Definition 6. Eine Pixelmenge S wird als *Zusammenhangskomponente (ZHK)* auf Grundlage einer k-Nachbarschaft bezeichnet, wenn

- S eine Pixelmenge basierend auf k-Nachbarschaft ist
- die Farbe der Bildpunkte der Pixelmenge S gleich ist und
- es keine andere Pixelmenge S' derselben Farbe mit den beiden genannten Eigenschaften und $S \subset S'$ gibt.

Definition 7. Ein *Binärobjekt* - oder *Teilchen* im vorliegenden Anwendungskontext - ist eine ZHK des Bildvordergrundes.

Definition 8. Ein *Loch* ist eine ZHK des Bildhintergrundes, die nicht mit dem Rand des Bildes verbunden ist.

Da in der vorliegenden Arbeit teilchenbasierte Messmethoden zur Anwendung kommen und die Teilchen zunächst in ihrer Kontur, d.h. ihrem Umriss, erfasst werden, soll an dieser Stelle noch folgendes präzisiert werden [10]:

Definition 9. Innerhalb eines Binärobjektes gilt die 8er-Nachbarschaft (d.h. auch Punkte, die nur über eine Ecke mit anderen Objektpunkten verbunden sind, hängen mit dem Rest des Objektes zusammen), im Hintergrund gilt dagegen die 4er-Nachbarschaft als Zusammenhangskriterium.

Diese Festlegung ist wichtig, um innere Löcher von Einstülpungen unterscheiden zu können. Würde man z.B. für das Teilchen in Abbildung 2.3 für Vorder- und Hintergrund die 8er-Nachbarschaft voraussetzen, so würde dies zu einem Widerspruch führen. Aufgrund der 8er-Nachbarschaft für den Vordergrund müssen die mit 1 und 2 markierten Pixel als durchgehende Linie gewertet werden, hinter der somit ein Loch beginnt. Aufgrund der

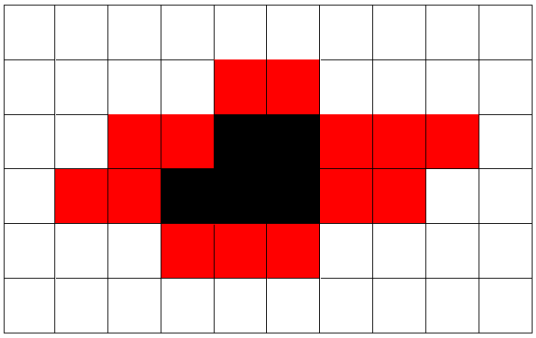

Abbildung 2.4: Kontur eines Objektes

8er-Nachbarschaft für den Hintergrund dürfte jedoch diese Verbindung nicht als Linie gewertet werden, da Pixel 3 und 4 zusammenhängen und somit kein Loch sondern nur eine Einstülpung vorliegt. Um diesen Widerspruch zu vermeiden, definiert man für den Hintergrund nur die 4er-Nachbarschaft.

Definition 10. Die *Kontur eines Objektes* ist eine Pixelsequenz aus zum Objekt gehörenden Pixeln, basierend auf 8er-Nachbarschaft, wobei jedes Pixel mindestens einen, zum Bildhintergrund gehörenden Nachbarn bzgl. der 4er-Nachbarschaft hat (vgl. Abbildung 2.4).

2.1.3 Bildvorbereitung und Extraktion der Teilchen

Um die einzelnen Binärobjekte näher analysieren zu können, müssen sie zunächst aus dem Bild isoliert werden. Dabei sind folgende Eigenschaften des Binärobjektes von Bedeutung:

- Startpunkt: Um den Startpunkt eines Binärpobjektes zu finden, wird das Bytearray des Bildes zeilenweise von links nach rechts geparst. Sobald man dabei auf ein schwarzes Pixel stößt, wird dieses als Startpunkt eines neuen Binärobjektes gekennzeichnet und eine Unterprozedur zum Abtasten der Kontur gestartet.

- Kontur des Objektes: Ausgehend vom gefundenen Startpunkt wird die Kontur des Objektes nach einem speziellen Verfahren so abgetastet, dass anschließend ein lückenloser, geschlossener Umriss in Form einer Pixelsequenz vorliegt (vgl. 2.2.1).

- Pixelmenge des Binärobjektes: Da das Binärobjekt auch 'Löcher' haben kann, müssen alle Pixel im Inneren der Kontur abgetastet werden. Der angewandte Algorithmus wird in den Absätzen 2.2.3 genauer erläutert.

2.2 Grundlegende Parameter der Partikel und ihre Berechnungsgrundlage

Grundlegende Parameter eines Partikels sind Eigenschaften, die zwar nicht direkt in die Klassifikation eingehen, die aber benötigt werden, um die zur Klassifikation verwendeten Formparameter zu berechnen. Dies sind insbesondere der Umfang, die Fläche, die konvexe Hülle und sein Durchmesser in verschiedene Richtungen (Feretsche Durchmesser).

2.2.1 Länge der äußeren Kontur

Die Länge der äußeren Kontur[3] eines Binärobjektes ist naturgemäß eine Eigenschaft, die - je nach Berechnungsmethode - starken Ungenauigkeiten unterworfen sein kann. Durch die Darstellung in Pixeln kann der Randverlauf immer nur näherungsweise dargestellt und berechnet werden (vgl. [6] und [23]). Dabei erscheint eine Vektordarstellung (ein Vektor aus Punkten und verbindenden Linien) in Form einer Kontur als eine der präzisesten Möglichkeiten der Berechnung (u.a. [23], S.185).

Freeman-Code

Um die Kontur eines Objektes effizient zu speichern, exisitieren verschiedene kompakte Darstellungen. Eine davon basiert auf der Arbeit von H.Freeman [9], der die Kontur in Form von Richtungsequenzen erfasst. Dabei verfolgt man den Umriss eines Objektes entgegen dem Uhrzeigersinn und fügt der Sequenz für jeden Schritt einen Richtungscode hinzu. Die Richtung wird in Form von Zahlen, wie in Abbildung 2.2 dargestellt, kodiert. Die beiden Figuren in Abbildung 2.5 sollen hier als Beispiel dienen. Der jeweilige Startpunkt der Kontur ist in der Grafik rot, der Verlauf der Kontur gelb markiert. Für die linke Figur ergibt sich somit ein Konturcode von (6 6 0 0 0 2 2 4 4 4), für die rechte (5 5 0 0 0 7 7 2 4 3 2 2 4). Der Startpunkt wurde durch Antasten von oben links gefunden, die Kontur wird entgegen dem Uhrzeigersinn durchlaufen.

[3] Da sich der Umfang eines Teilchens aus der Länge der äußeren Kontur sowie aus den Umfangslängen der im Objekt enthaltenen Löcher zusammensetzt, wähle ich an dieser Stelle nicht den zunächst naheliegenden Begriff 'Umfang'.

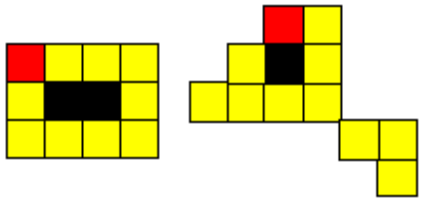

Abbildung 2.5: Freeman-Code: Beispiel (Bilder aus [10])

Konturverfolgung nach Pavlidis

Wenn die einzelnen Pixel der Kontur entlang eines geschlossenen Pfades abgetastet werden, so hat dies den Vorteil, dass genaue Informationen über den *Verlauf* (z.B. Einbuchtungen oder Ausstülpungen) der Kontur vorliegen, die spätere Berechnungen deutlich vereinfachen (z.B. die Berechnung des Umfangs oder der Eulerzahl). Pavlidis hat hierfür einen Algorithmus zur Verfügung gestellt, der für die vorliegende Anwendung implementiert wurde [19]. Bildhaft ausgedrückt entspricht der Algorithmus einem Betrachter, der um das Binärobjekt herumwandert und dabei als nächstes Pixel jeweils das von ihm aus gesehen äußerste rechte Pixel auswählt. Dabei dreht er sich in diese Richtung und fügt den dafür notwendigen Richtungscode der bisherigen Codesequenz hinzu. Die Umrundung des Objektes ist beendet, wenn der Startpunkt wieder erreicht ist (d.h. die Kontur ist geschlossen) und das nächste Pixel, das ausgewählt wurde, schon besucht wurde.

Algorithmus [19] Sei A der Ausgangspunkt der Kontur eines Objektes O, C der aktuelle Punkt, dessen Nachbarschaft gerade untersucht wird, S die aktuelle (Blick-) Richtung (vgl. Abbildung 2.2), *first* eine boolesche Variable, die markiert, wann der Startpunkt erneut gefunden ist, *found* eine boolesche Variable, die angibt, wann der nächste Punkt der Kontur gefunden ist und *count* ein Zähler, der angibt, wie oft sich der Betrachter auf einer Stelle schon um sich selbst gedreht hat. Sei außerdem N(P,S) eine Funktion, die den Nachbarn von P in Richtung S berechnet.

```
0. Suche den Punkt A durch Antastung von links oben an das Objekt;
1. Setze C := A, S:= 6, first := true;
2. While (C != A or first) do Schritte 3 - 9.
        Begin:
3.      Setze found := false, count = 0;
4.      While (!found and count < 3) do Schritte 5 - 8.
            Begin:
```

```
5.          If (N(C,S-1)liegt in O) then
                Setze C:=(N(C,S-1), S:=S-2, found:=true;
6.          Else if (N(C,S) liegt in O) then
                Setze C:=(N(C,S), found:=true;
7.          Else if (N(C,S+1) liegt in O) then
                Setze C:=(N(C,S+1), found:=true;
8.          Else S:=(S+2)mod 8, count++;
            End
9.      Setze first:=false;
        End.
10. Ende des Algorithmus
```

Bemerkung: Hierbei ist das Ende des Algorithmus verkürzt angegeben. Wenn der Ausgangspunkt wieder erreicht ist, muss die Kontur nicht zwangsläufig wieder geschlossen sein - es könnte eine Fortsetzung in eine andere Richtung geben. Daher muss noch ein nachfolgender Richtungscode berechnet werden. Erst wenn dieser mit dem ersten Richtungscode übereinstimmt, ist die Konturverfolgung beendet.

Umfangsberechnung nach Vossepoel und Smeulders

Intuitiv leuchtet ein, dass sich aus dem erhaltenen Konturcode die Länge der äußeren Kontur berechnen lässt. Dabei müssen jedoch Richtungsschritte parallel zur x- oder y-Achse (geradzahlige Richtungscodes) anders kalkuliert werden als Schritte in der Diagonalen (ungeradzahlige Richtungscodes). Während Schritte parallel zu den Achsen genau der Länge l eines Pixels entsprechen[4], sind diagonale Schritte um den Faktor $\sqrt{2}$ länger (vgl. Abbildung 2.6). Beim Abtasten der Kontur wird daher mitgezählt, wie oft jeder der acht Richtungscodes der Sequenz hinzugefügt wurde. Hat der Konturcode insgesamt n Schritte, wovon m nicht achsenparallel verlaufen, so kann die Länge des Umrisses folgendermaßen abgeschätzt werden: $K = n + m(\sqrt{2} - 1)$. Experimente zeigen jedoch, dass diese Art der Berechnung immer noch recht ungenau ist, da sie letztendlich nur die Länge des digtalisierten Umrisses berechnet und somit stark von der Auflösung und von der Art der Nachbarschaft abhängt. Vossepoel und Smeulders entwickelten in ihrer Arbeit eine wesentlich genauere Abschätzung des Umfangs [1]. Sie bauten dabei auf Untersuchungen von Groen und Verbeek [5] sowie von Proffitt und Rosen [7] auf.

[4] unter der Annahme von quadratischen Pixeln

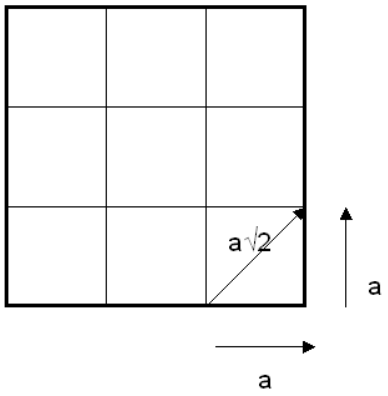

Abbildung 2.6: Länge eines Pixelschrittes parallel, bzw. diagonal zu den Achsen

Groen und Verbeek [5] zeigten in ihrer Arbeit, dass die naive Berechnung des Umfangs auf die oben beschriebene Art stark durch die relative Position des Objektes im Verhältnis zum Pixelgitter beeinflusst wird. Insbesondere stellte sich dabei heraus, dass die Einstiegshöhe e und der Winkel τ des Objektes von besonderer Bedeutung für den ermittelten Umfang sind. Groen und Verbeek untersuchten nun die Wahrscheinlichkeitsdichtefunktion über ein Liniensegment, abhängig von e und τ. Durch Integration dieser Funktion über eine Zelle des Pixelgitters fanden sie Koeffizienten, die eine genauere Abschätzung der Länge eines Liniensegementes ergaben. Für eine Kontur bestehend aus n Konturcodes, wovon m nicht parallel zu den Achsen verlaufen, ergab sich eine Längenabschätzung von $1.059n + 0.124m$.

Proffitt und Rosen [7] untersuchten insbesondere die Bedeutung der Eckenzahl in einem Liniensegment. Die Zahl der Ecken ergibt sich aus der Anzahl aufeinanderfolgender, nicht identischer Konturcodes. Intuitiv wird deutlich, dass die Abrundung solcher Ecken für ein digitalisiertes Bild zu einer genaueren Abschätzung der Länge eines Liniensegmentes führen kann.

Vossepoel und Smeulders kombinierten beide Konzepte in ihren Untersuchungen. Die genaue Beweisführung würde den Rahmen dieser Arbeit sprengen; daher sei an dieser Stelle auf ihren Artikel [1] verwiesen. Die Autoren konnten darin zeigen, dass folgende Abschätzung der Länge eines Liniensegmentes in einer 8er-Nachbarschaft eine Genauigkeit von 99,3 % erreicht:

Definition 11. Sei eine Kontur gegeben durch n Konturschritte, wovon m nicht parallel zu den Achsen verlaufen, sowie durch n_c Ecken. Dann kann die *Länge K des Liniensegmentes*

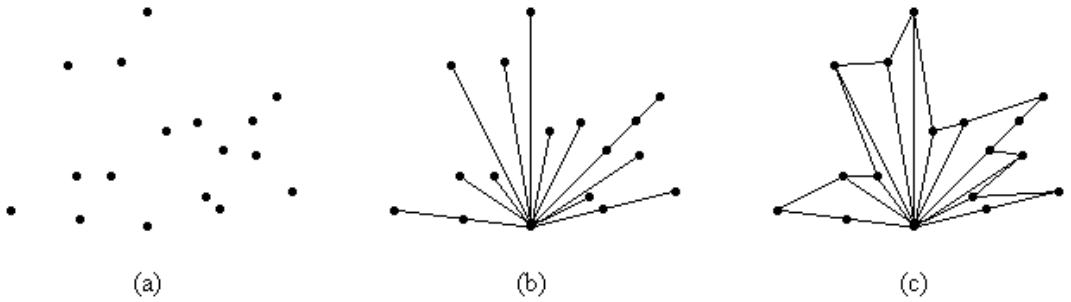

Abbildung 2.7: Graham-Scan-Algorithmus: Phase 1 und 2

folgendermaßen abgeschätzt werden: $K = 0.980n + 0.426m - 0,091n_c$

2.2.2 Konvexe Hülle

Eine Menge von Punkten heißt konvex, wenn sie für je zwei Punkte auch deren Verbindungsstrecke enthält. Die konvexe Hülle einer Punktmenge M ist die kleinste konvexe Menge, die M enthält. Bildlich gesprochen legt man quasi ein Gummiband um eine Punktmenge, so dass alle Punkte eingeschlossen werden. Für die Lösung dieses Problems existieren in der Literatur zahlreiche Algorithmen. Einer davon ist der Graham-Scan-Algorithmus, den ich in der vorliegenden Arbeit verwendet habe. Als Eingabe erhält der Algorithmus den Umriss des Objektes, wie er zuvor über das in 2.2.1 beschriebene Verfahren bestimmt wurde.

Graham-Scan-Algorithmus

Der Graham-Scan-Algorithmus verläuft in drei Phasen.

1. Zunächst wird aus der Punktmenge der Eingabe ein extremer Punkt p (z.B. mit minimaler y-Koordinate) bestimmt, der somit sicher zur konvexen Hülle gehört.

2. Anschließend werden die übrigen Punkte nach ihrem Winkel zu p aufsteigend sortiert. Punkte gleichen Winkels werden nach ihrem Abstand zu p aufsteigend sortiert. Ergebnis ist ein sternförmiges Polygon (vgl. Abbildung 2.7). [5]

3. In der dritten Phase wird nun das Polygon ausgehend von p gegen den Uhrzeigersinn durchlaufen, wobei konkave Ecken überbrückt werden. Um festzustellen, ob eine Ecke p_i eines Polygonzugs konvex oder konkav ist, betrachtet man das Dreieck der Ecken

[5] Bilder zum Graham-Scan-Algorithmus aus: http://www.iti.fh-flensburg.de/lang/algorithmen/geo/graham.htm

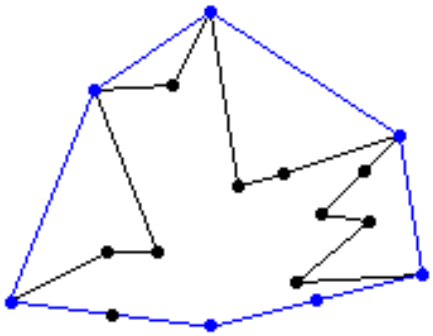

Abbildung 2.8: Graham Scan Algorithmus: Phase 3

p_{i-1}, p_i, p_{i+1}. Bei einer konvexen Ecke ergibt die Flächenberechnung dieser drei Punkte eine positive Dreiecksfläche, bzw. 0, falls alle drei Punkte auf einer Geraden liegen. Bei einer konkaven Ecke ergibt die Berechnung eine negative Fläche. Wird eine konkave Ecke gefunden, so wird p_i aus der konvexen Hülle entfernt. Dies könnte jedoch auch Konsequenzen für die schon durchlaufenen Punkte haben, da eine Ecke, die vorher konvex war, durch die veränderte Zusammensetzung der (vorläufigen) konvexen Hülle nun konkav ist. Daher muss an dieser Stelle ein Backtracking erfolgen. Das Ergebnis der dritten Phase zeigt Abbildung 2.8.

Komplexität Die Komplexität des Graham-Scan-Algorithmus liegt bei n Punkten im schlechtesten Fall in O(n log(n)). Da jeweils ein Punkt entfernt wird, wenn es zum Backtracking kommt, kann es maximal n Rückschritte geben. Somit erhält man n Schleifendurchläufe + n Rückschritte = 2n Aufrufe, an denen ein Punkt ausgewertet werden muss. Jeder dieser Aufrufe benötigt konstante Zeit (den nächsten Punkt suchen, die Position berechnen und evtl. die konvexe Hülle aktualisieren), so dass die Laufzeit hiervon in O(n) liegt. Die vorangehende Sortierung liegt (wie Sortierprobleme generell) in O(n log(n)), so dass die Gesamtlaufzeit letztlich hiervon bestimmt wird.

2.2.3 Fläche und Innere Konturen des Teilchens

Die *Fläche* eines Objektes ergibt sich aus der Summe der Fläche seiner Pixel. Da innerhalb des Teilchens Löcher vorhanden sein können, greift ein Algorithmus, der die Kontur einfach ausfüllt, an dieser Stelle zu kurz. Daher wurde ein Verfahren gewählt, das ausgehend von einem Startpunkt die acht möglichen Nachbarpixel untersucht. Wird dabei ein schwarzes Pixel gefunden, so wird es auf einen Stack S gelegt, da das Objekt an dieser Stelle noch

eine Fortsetzung haben könnte. Jedes besuchte schwarze Pixel wird markiert, so dass es nur einmal betrachtet werden muss. In den nächsten Iterationen wird nun jeweils ein Punkt aus S untersucht. Der Algorithmus terminiert, sobald S vollständig geleert wurde. Auf diese Weise kann die Anzahl m der Pixel, die zum Objekt gehören, einfach ermittelt werden. Da nur schwarze Pixel auf den Stack gelegt werden, behandelt die Flächeberechnung automatisch innere Löcher korrekt. Die Fläche ergibt sich dann als $A = m * pixelsize^2$, wobei *pixelsize* die Länge eines Pixels in horizontaler bzw. vertikaler Richtung ist.

Innere Löcher sind jedoch nicht nur kritisch für die Flächenberechnung. Auch für die Umfangsberechnung des Teilchens spielt ihre korrekte Abtastung eine Rolle, denn der Umfang der inneren Löcher wird zur Länge der äußeren Kontur addiert, wodurch der Gesamtumfang des Teilchens bestimmt wird. Aus diesem Grunde wird bei der oben beschriebenen Abtastung der acht Nachbarpixel auch ein Stack mit weißen Pixeln mitgeführt. Wird bei der Untersuchung der Nachbarpixel des Pixels X (bzgl. der 4er-Nachbarschaft) ein weißes Pixel gefunden und ist X kein Pixel der äußeren Kontur, so muss das weiße Pixel zu einem Loch gehören. Die vorliegende Implementierung sammelt zunächst alle weißen Pixel, bis das Objekt ganz erfasst ist, d.h. bis alle schwarzen Pixel markiert wurden. Anschließend wird von den weißen Pixeln dasjenige gesucht, das am weitesten links oben liegt. Ausgehend von diesem Punkt wird wiederum eine Konturverfolgung durchgeführt - diesmal allerdings basierend auf 4er-Nachbarschaft, (vgl. Definition 9). Dieses Verfahren lässt sich sicherlich noch effizienter gestalten. Da bei den untersuchten Bildern die Anzahl und Größe der Löcher innerhalb der Teilchen klein ist, schien das Verfahren für einen ersten Prototyp ausreichend. Problematisch bei der Suche nach einem effizienteren Algorithmus erwies sich die Tatsache, dass die Teilchen oft sehr ineinander verschachtelt sind, so dass die Bestimmung, ob ein Punkt zu einem Teilchen gehört oder nicht, relativ aufwendig ist. Daran scheiterte letztendlich die Implementierung eines performanteren Algorithmus zum Abtasten der inneren Konturen, den Pavlidis vorstellt[19]. Hier können weitere Implementierungen sicherlich noch einiges an Effizienz verbessern.

2.2.4 Feretsche Durchmesser

Feretsche Durchmesser bezeichnen die maximale Ausdehnung eines Teilchens in eine bestimmte Richtung (vgl. Abbildung 2.9). Der Feretsche Durchmesser für 0° ist somit die maximale Ausdehnung des Teilchens parallel zur x-Achse, der für 90° analog die maximale Ausdehnung des Teilchens parallel zur y-Achse. Die Berechnung der Klassifikationsparameter benötigt im Allgemeinen den *maximalen* Feretschen Durchmesser (MaxFeret).

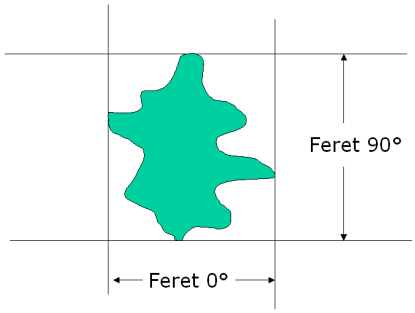

Abbildung 2.9: Feretsche Durchmesser eines Teilchens in 0° und 90°

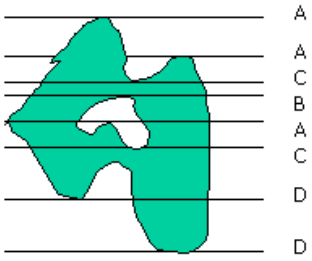

Abbildung 2.10: Tangenten vom Typ A, B, C und D

Hierzu werden zunächst die Feretschen Durchmesser in 36 Richtungen (0°,5°,10°,...175°) berechnet. Der MaxFeret ergibt sich dann als Maximum dieser Werte. Die Berechnung des Feretschen Durchmessers für 0° ist einfach zu lösen: Hierfür werden die Punkte der äußeren Kontur nach ihren x-Koordinaten aufsteigend sortiert. Der Feretsche Durchmesser für 0° errechnet sich dann als Differenz zwischen der minimalen und maximalen x-Koordinate. Analog lässt sich der Feret für 90° aus einer Sortierung der y-Koordinaten gewinnen. Um die übrigen Ferets zu berechnen, wird nun das Achsenkreuz in Schritten von 5° 17 mal um den Ursprung rotiert (oder anders ausgedrückt: die äußere Kontur des Teilchens wird punktweise um 5° um den Ursprung rotiert). Nach jeder Rotation werden die minimalen und maximalen x- bzw. y-Koordinaten und daraus die Feretschen Durchmesser in zwei aufeinander senkrecht stehenden Richtungen ermittelt.

2.2.5 A-, B-, C-, und D-Tangenten

Für jedes Teilchen lassen sich Tangenten bestimmen, die zur Charakterisierung der Struktur sowie für die Berechnung weiterer Parameter eine Rolle spielen [8]. Abbildung 2.10

verdeutlicht die verschiedenen Tangententypen. Formal lässt sich folgendes definieren:

Definition 12. Sei $p_i.y$ die y-Koordinate eines Punktes p_i. Sei k die gesamte Kontur eines Teilchens (innere und äußere Konturen), bestehend aus der Pixel- (=Punkt-)Sequenz $p_0, p_1, .., p_n$ und Punkt p_i ein lokales Maximum bzgl. der y-Koordinate (d.h. $p_i.y > p_{i-1}.y$ und $p_i.y > p_{i+1}.y$. Dann ist p Ansatzpunkt für

- eine *Tangente vom Typ A*, falls das Pixel unterhalb von p ein Vordergrundpixel und das oberhalb von p liegende Pixel ein Hintergrundpixel ist, und
- eine *Tangente vom Typ B*, falls das Pixel oberhalb von p ein Vordergrundpixel und das unterhalb von p liegende Pixel ein Hintergrundpixel ist.

Definition 13. Sei $p_i.y$ die y-Koordinate eines Punktes p_i. Sei k die gesamte Kontur eines Teilchens (innere und äußere Konturen), bestehend aus der Pixel- (=Punkt-)Sequenz $p_0, p_1, .., p_n$ und Punkt p_i ein lokales Minimum bzgl. der y-Koordinate (d.h. $p_i.y < p_{i-1}.y$ und $p_i.y < p_{i+1}.y$. Dann ist p Ansatzpunkt für

- eine *Tangente vom Typ C*, falls das Pixel unterhalb von p ein Vordergrundpixel und das oberhalb von p liegende Pixel ein Hintergrundpixel ist, und
- eine *Tangente vom Typ D*, falls das Pixel oberhalb von p ein Vordergrundpixel und das unterhalb von p liegende Pixel ein Hintergrundpixel ist.

Die Ansatzpunkte für die genannten Tangententypen lassen sich bestimmen, indem die Kontur des Teilchens einmal sequenziell durchlaufen wird, wobei die y-Koordinate beobachtet wird.

2.3 Berechnung der zur Klassifikation verwendeten Parameter

Basierend auf den vorgestellten Grundparametern wurden für jedes Teilchen 13 Formparameter berechnet und in den Klassifikationsprozess einbezogen[6].

[6]Der Übersichtlichkeit halber gebe ich an dieser Stelle auch die englischen Bezeichnungen an, wie sie sich in der Webanwendung finden.

2.3.1 Berechnung der teilchenbasierten Kenngrößen

Rundheit und Kreisform (Roundness und Circularity)

Der Faktor Rundheit setzt die Fläche des Teilchens in Bezug zu dem umgebenden Kreis. Der umgebende Kreis ergibt sich aus dem maximalen Durchmesser des Teilchens (MaxFeret). Der Faktor Kreisform berechnet sich als Wurzel aus dem Faktor Rundheit.

Definition 14. Sei A der Messwert der Fläche eines Teilchens und D der maximale Durchmesser (MaxFeret). Dann berechnet sich der Faktor *Rundheit* durch die Formel $\frac{4 \cdot A}{\pi \cdot D^2}$ und der Faktor *Kreisform* als $\sqrt{Rundheit}$.

Formfaktor (Area Shape)

Der Formfaktor gibt Aufschluss über das Verhältnis von der Fläche eines Teilchens zu der Fläche eines Kreises, der denselben Umfang wie das Teilchen hätte.

Definition 15. Sei A der Messwert der Fläche eines Teilchens und p die Länge seines Umfangs. Dann berechnet sich der *Formfaktor* durch die Formel $\frac{4\pi \cdot A}{p^2}$.

Kompaktheit (Compactness)

Der Faktor Kompaktheit basiert auf einer ähnlichen Kalkulation wie der Formfaktor. Jedoch wird als Bezugsgröße hier der konvexe Umfang des Teilchens genommen.

Definition 16. Sei A die Größe der Fläche eines Teilchens und u die Länge seines konvexen Umfangs. Dann berechnet sich die *Kompaktheit* des Teilchens über die Formel $\frac{4\pi \cdot A}{u^2}$.

Fraktale Dimension (Fractal Dimension)

Hierbei geht man davon aus, dass die Oberfläche eines Teilchens ein Fraktal bildet, d.h. sie konvergiert bei zunehmender Vergrößerung gegen unendlich. Die Fraktale Dimension ist ein Maß dafür, wie stark der Messwert für die Oberfläche bei zunehmender optischer Vergrößerung wächst [21]. Dieser Parameter gibt daher Auskunft darüber, wie glatt oder rauh die Oberfläche des Teilchens ist. Aus [17] stand ein Algorithmus zur Verfügung, der die fraktale Dimension eines Gefüges berechnet. Um diesen Algorithmus verwenden zu können, wird das einzelne Teilchen isoliert und dem Algorithmus, quasi als neues Teilbild, übergeben.

Abweichungsfaktor (Excursion Ratio)

Dieser Parameter setzt den Umfang des Teilchens in Beziehung zum umgebenden Rechteck, genauer: zu der Diagonalen des umgebenden Rechtecks.

Definition 17. Sei H der maximale Durchmesser eines Teilchens (MaxFeret), V der auf diesem Durchmesser senkrecht stehende Feret und p die Länge seines Umfangs. Dann berechnet sich der *Abweichungsfaktor* über die Formel $\frac{2\sqrt{H^2+V^2}}{p}$.

Elliptischer Formfaktor 1 und 2 (Elliptic Area Shape 1/2)

Bei den beiden elliptischen Formfaktoren wird jeweils die umgebende Ellipse des Teilchens betrachtet. Diese hat als Achsen den maximalen Durchmesser des Teilchens und den darauf senkrecht stehenden Feret. Beim Elliptischen Formfaktor 1 wird die Fläche der Ellipse in Relation zum umgebenden Kreis gesetzt: für kreisförmige Teilchen erreicht dieser Faktor den Wert 1.

Definition 18. Sei A der Messwert der Fläche eines Teilchens und H,V die Längen der Achsen der umgebenden Ellipse (MaxFeret und der darauf senkrecht stehende Feret). Dann berechnet sich der *Elliptische Formfaktor 1* über die Formel $\frac{8A}{\pi\cdot(a^2+b^2)}$.

Der Elliptische Formfaktor 2 vergleicht dagegen die Fläche der Ellipse mit der realen Fläche des Teilchens:

Definition 19. Sei A der Messwert der Fläche eines Teilchens und E die Fläche der Ellipse. Dann berechnet sich der *Elliptische Formfaktor 2* über die Formel $\frac{A}{E}$.

Konvexität des Umfangs (Convexity)

Wie der Name des Parameters schon andeutet, wird hier der konvexe Umfang des Teilchens verglichen mit dem realen Umfang des Teilchens. Für konvexe Teilchen ist dieser Faktor 1.

Definition 20. Sei p die Länge des Umfangs des Teilchens und u die Länge des Umfangs seiner konvexen Hülle. Dann berechnet sich die *Konvexität des Umfangs* als $(\frac{u}{p})^2$.

Konvexität der Fläche (ConvexityF)

Bei diesem zweiten Konvexitätsfaktor wird die Fläche der konvexen Hülle in Beziehung gesetzt zur Fläche des Teilchens. Für konvexe Teilchen ist dieser Faktor 1.

Definition 21. Sei A der Messwert der Fläche eines Teilchens und A_k der Messwert der Fläche der konvexen Hülle. Dann berechnet sich die *Konvexität der Fläche* als $\frac{A}{A_k}$.

Eulerzahl (Eulernumber)

Dieser Parameter ergibt sich aus der Anzahl der A-, B-, C- und D-Tangenten (vgl. Absatz 2.2.5). Die Eulerzahl charakterisiert topologische Eigenschaften eines Teilchens. Für ein abgeschlossenes konvexes Teilchen ist die Eulerzahl immer 1. Für ein allgemeines Teilchen ist die Eulerzahl 1 - (Anzahl der Löcher des Teilchens) [8].

Definition 22. Seien $tang_A, tang_B, tang_C, tang_D$ die Anzahl der A-, B-, C- und D-Tangenten eines Teilchens. Dann berechnet sich die *Eulerzahl* E folgendermaßen:

$$E = \frac{tang_A - tang_B - tang_C + tang_D}{2}$$

.

Rechteckmaß (RectMeasure)

Das Rechteckmaß beschreibt den Grad der Abweichung vom Rechteck bezüglich der vier Hauptrichtungen.

Definition 23. Sei A der Messwert der Fläche eines Teilchens und $F_0, F_{45}, F_{90}, F_{135}$ die Feretschen Durchmesser in die Richtungen 0°, 45°, 90° und 135°. Dann berechnet sich das *Rechteckmaß* nach der Formel $\frac{A}{min[(F_0 \cdot F_{90}),(F_{45} \cdot F_{135})]}$.

Streckung (Aspect Ratio)

Dieser Parameter setzt den maximalen und den minimalen Feretschen Durchmesser eines Teilchens zueinander in Beziehung.

Definition 24. Sei D_{max} der maximale Durchmesser eines Teilchens und D_{min} sein minimaler Feretscher Durchmesser. Dann bezeichnet man deren Quotienten $\frac{D_{min}}{D_{max}}$ als *Streckung* des Teilchens.

2.3.2 Sonstige Klassifikationsparameter

Neben den genannten Formparametern gehen zwei weitere Größen in die Klassifikation mit ein. Dies sind die **Pixelgröße** und die **Richtzahl** des Teilchens. Zwar wurde vorläufig nur mit *einer* Pixelgröße gearbeitet, so dass dieser Parameter ohne Einfluss blieb. Aus werkstoffwissenschaftlicher Sicht war es jedoch wichtig, in einem späteren Stadium den Einfluss verschiedener Pixelgrößen (und somit verschiedener Auflösungen) mit untersuchen zu können. Die Richtzahl ordnet die einzelnen Teilchen, abhängig von ihrem maximalen

Durchmesser, festen Klassen zu. Dies ist wie folgt spezifiziert in der Europäischen Norm EN ISO 945:

MaxFeret des Teilchens in μm	Richtzahl
>1000	1
500-1000	2
250-500	3
120-250	4
60-120	5
30-60	6
15-30	7
<15	8

Untersuchungen haben gezeigt, dass nur Teilchen gleicher Richtzahl in ihren Formparametern vergleichbar sind. Daher wurde die Richtzahl des Teilchens in den Klassifikationsprozess mit eingebracht.

Kapitel 3

Automatische Klassifikation mit SVM

In diesem Kapitel soll der Prozess der Klassifikation näher erläutert werden. Dabei geht es in Absatz 3.1.1 zunächst um die Grundlagen der verwendeten Klasssifikationsmethode, Support Vector Machines (SVM, Stützvektorverfahren) und darauf aufbauend in Absatz 3.1.3 um eine genauere Beschreibung einiger relevanter Details. In Absatz 3.1.4 werden dann die Software-Bibliotheken beschrieben, die in der vorliegenden Arbeit verwendet wurden.

3.1 SVM - Support Vector Machines

SVM ist ein Verfahren, das im Kontext des maschinellen Lernens entstanden ist. Vor dem Hintergrund der täglich wachsenden Anzahl an Informationen, die manuell kaum mehr zu erfassen sind, wuchs der Wunsch nach einer automatischen Kategorisierung der angebotenen Ressourcen. Support Vector Machines sollten die einzelnen Informationsobjekte (Webseiten, Textdokumente, Bilder, Sprachaufzeichnungen, etc.) automatisch einer oder mehreren Kategorien zuordnen und so eine systematische Filterung ermöglichen. Generell sind Support Vector Machines anwendbar auf verschiedenste Anwendungsgebiete der automatischen Klassifikation. Dabei wird ein Klassifikator mittels Daten zu einem bestimmten Problem trainiert und ist dann aufgrund dieses Trainings in der Lage, für neue Objekte Klassifikationsentscheidungen zu treffen. Die folgenden Abschnitte sollen dieses Verfahren näher erläutern. Dabei geht es nicht um eine detaillierte Darstellung der theoretischen Hintergründe der SVM - diese sind in der Literatur reichlich diskutiert. Der interessierte Leser sei hier z.B. auf [4] verwiesen. Ziel des Kapitels ist vielmehr die Darstellung der grundlegenden Idee der SVM, sofern dies zum Verständnis der vorliegenden Arbeit notwendig ist.

3.1.1 Grundlagen

Die SVM wird mittels einer Menge von Trainingsdaten hinsichtlich eines bestimmten Anwendungsgebietes trainiert. Diese Trainingsdaten bestehen aus N Beispielen der Form $(\vec{x_1}, y_1), (\vec{x_2}, y_2), \cdots, (\vec{x_N}, y_N)$, wobei $\vec{x} \in \Re^p$ dem Merkmalsvektor eines Trainingsobjektes und $y \in \{-1, 1\}$ seiner Klassifikation entspricht. Ein Training wird für genau eine Klasse durchgeführt: $y = 1$ besagt, dass es sich bei dem Objekt mit dem Merkmalsvektor $\vec{x}$ um ein positives Trainingsbeispiel handelt, $y = -1$ dagegen drückt aus, dass das Objekt keine Instanz der entsprechenden Klasse ist. Der Merkmalsvektor $\vec{x}^T = (x_1, x_2, ..., x_p)$ umfasst alle p Merkmalswerte des betrachteten Objektes. Ziel des Trainings ist nun, dass die SVM eine Klassifikationsregel lernt und aufgrund derer in der Lage ist, unbekannte Beispiele möglichst fehlerfrei zu klassifizieren. Hierzu versucht die SVM, eine Hyperebene zu finden, d.h. eine lineare Funktion, die positive und negative Trainingsdaten voneinander trennt. Positive Trainingsbeispiele liegen dann auf der einen, negative auf der anderen Seite der Hyperebene. Diese Hyperebene hat folgende Form:

$$\vec{w} \cdot \vec{x} + b = 0$$

Dabei sind $\vec{w}$ und b Komponenten, die von der SVM möglichst optimal bestimmt werden müssen. $\vec{w}$ ist ein sog. Gewichtsvektor, der den einzelnen Merkmalen eine bestimmte Wertigkeit zuordnet, b ist ein Schwellwert. Für alle positiven Trainingsbeispiele hat $\vec{w} \cdot \vec{x} + b$ einen Wert größer 0, für alle negativen Trainingsbeispiele einen Wert kleiner 0:

$$\begin{aligned} &(\vec{w} \cdot \vec{x} + b > 0 \textit{ für } y = +1 \textit{ und } \vec{w} \cdot \vec{x} + b < 0 \textit{ für } y = -1) \\ &\Leftrightarrow y(\vec{w} \cdot \vec{x} + b) > 0 \end{aligned} \tag{3.1}$$

Die eigentliche Klassifikationsfunktion wäre damit wie folgt definiert:

$$y = sign(\vec{w} \cdot \vec{x} + b)$$

Im Folgenden gehen wir zunächst davon aus, dass die Trainingsdaten linear separabel sind, d.h. es lässt sich eine Hyperebene finden, die negative von positiven Trainingsbeispielen separiert[1]. Im Allgemeinen gibt es dann nicht nur eine sondern viele Hyperebenen, die diese Bedingung erfüllen (vgl. Abbildung 3.1). Intuitiv ist ersichtlich, dass ein Klassifikator umso stabiler funktioniert, je größer der Abstand ist, der zwischen negativen und positiven Trainingsbeispielen liegt. Um diese Stabilität zu gewährleisten, versucht die SVM, die Hyperebene zu finden, die positive und negative Trainingsbeispiele mit *maximalem Abstand*

[1] Der Fall, dass die Trainingsdaten linear nicht separabel sind, wird in Absatz 3.1.3 erläutert.

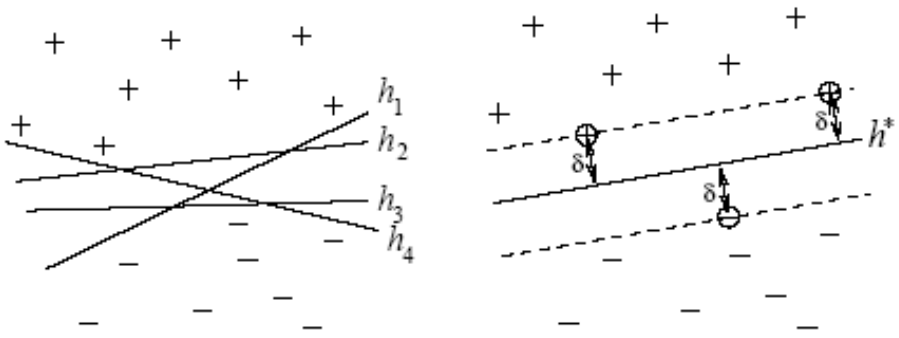

Abbildung 3.1: Mögliche Hyperebenen (links) und eine optimale Hyperebene (rechts)

trennt. In der Abbildung 3.1 ist dies die Hyperebene h*, die zu jedem Trainingsbeispiel mindestens den Abstand δ hat. Der Abstand eines Punktes $\vec{x}$ zur Hyperebene lässt sich ausdrücken durch $\frac{1}{\|w\|}|(\vec{w} \cdot \vec{x} + b)|$ Daher kann man das Problem folgendermaßen zusammenfassen:

$$\begin{gathered} \textit{Finde } \vec{w}, b \\ \textit{so dass } \delta \textit{ maximal ist} \\ \textit{und es gilt: } y\frac{1}{\|w\|}|(\vec{w} \cdot \vec{x} + b)| \geq \delta \textit{ für alle Trainingsvektoren } \vec{x} \end{gathered} \tag{3.2}$$

Die Trainingsbeispiele, deren Abstand zur Hyperebene genau δ ist, werden Support Vectors genannt. Sie sind ausschlaggebend für spätere 'Entscheidungen' der SVM.

Optimierung Zur Berechnung der oben beschriebenen Maximierung von δ sind einige Umformungen des Grundproblems notwendig. Da die Länge von $\vec{w}$ für die Lage der Hyperebene unerheblich ist, kann sie beliebig festgelegt werden, z.B. mit $\frac{1}{\|w\|} = \delta$. Substitution von δ durch $\frac{1}{\|w\|}$ ergibt:

$$\begin{gathered} \textit{Finde } \vec{w}, b \\ \textit{so dass } \frac{1}{\|w\|} \textit{ maximal ist} \\ \textit{und es gilt: } y|(\vec{w} \cdot \vec{x} + b)| \geq 1 \end{gathered} \tag{3.3}$$

Die Maximierung von $\frac{1}{\|w\|}$ ist gleichbedeutend mit der Minimierung von $\vec{w} \cdot \vec{w}$:

$$\begin{gathered} \textit{Finde } \vec{w}, b \\ \textit{so dass } \vec{w} \cdot \vec{w} \textit{ minimal ist} \end{gathered} \tag{3.4}$$

$$\textit{und es gilt: } y|(\vec{w} \cdot \vec{x} + b)| \geq 1 \tag{3.5}$$

Doch ist auch dieses Problem mit numerischen Methoden nur schwer lösbar. Daher sind weitere Umformungen notwendig (Lagrange-Multiplikatoren, Formulierung des dualen Problems nach Wolfe, Herleitung siehe u.a. [21] und [4]). So erhält man folgende Problemstellung:

$$\textit{Finde } \alpha_1, ..., \alpha_l \tag{3.6}$$

$$\textit{so dass } -\sum_{i=1}^{l} \alpha_i + \tfrac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} y_i y_j \alpha_i \alpha_j (\vec{x}_i \cdot \vec{x}_j) \textit{ minimal ist}$$

$$\textit{und es gilt: } \sum_{i=1}^{l} y_i \alpha_i = 0 \textit{ und } \alpha_i \geq 0 \ \forall i \tag{3.7}$$

Hierbei handelt es sich um ein quadratisches Optimierungsproblem. Um Probleme dieser Art zu lösen, existieren eine Reihe von Algorithmen. Hat man die Lösung $\alpha_1^*, ..., \alpha_l^*$ des Problems berechnet, so kann man den Gewichtsvektor $\vec{w}$ wie folgt daraus berechnen (siehe hierzu [21] und [4]):

$$\vec{w} = \sum_{i=0}^{l} \alpha_i^* y_i \vec{x_i}$$

Der Schwellwert b kann mit Hilfe eines beliebigen Support Vectors $\vec{x}_i$ bestimmt werden: $b = y_i - \vec{w} \cdot \vec{x_i}$.

3.1.2 SVM mit weicher Trennung

Für viele reale Trainingsdaten ist eine fehlerfreie Trennung auch mit komplexen Klassifikationsfunktionen nicht möglich. Hier ist es wünschenswert, beim Training Fehler, sog. Ausreißer, zuzulassen. Je mehr Fehler zugelassen werden, desto größer kann der Abstand δ gewählt werden. Um Fehler und Separationsbreite sinnvoll gegeneinander abzuwägen, wird ein Parameter C in die Berechnung eingeführt. Gleichzeitig wird nicht mehr nur die Länge von $\vec{w}$ minimiert sondern auch noch die Summe der Fehler ξ_i. Das neue Optimierungsproblem lautet somit:

$$\textit{Finde } \vec{w}, b \tag{3.8}$$

$$\textit{so dass } \vec{w} \cdot \vec{w} + \sum_{i=1}^{l} \xi_i \textit{ minimal ist}$$

$$\textit{und es gilt: } y|(\vec{w} \cdot \vec{x} + b)| \geq 1 - \xi_i \tag{3.9}$$

Die Lösung dieses Problems gleicht dem in Absatz 3.1.1 beschriebenen Vorgehen. In der Gleichung 3.7 erscheint nach den Umformungen C als obere Grenze für α_i: $0 \leq \alpha_i \leq C \ \forall i$. Bezüglich einer genaueren Herleitung sei auch hier auf [4] verwiesen.

3.1.3 Kernfunktion und Parameter

Wenn die Trainingsdaten nicht linear separabel sind, so kann das oben beschriebene Verfahren nicht angewandt werden, weil es für 3.4 keine Lösung gibt. Für diesen Fall bietet sich die Verwendung von Kernfunktionen an, die auf der Tatsache aufbauen, dass linear nichtseparable Daten in einem höherdimensionalen Raum unter Umständen doch noch linear separabel sind. Hierzu werden die Merkmalsvektoren $\vec{x_i}$ durch eine nicht-lineare Funktion ϕ in einen höherdimensionalen Raum X' abgebildet. Dort wird dann die Hyperebene berechnet. Die trennende Klassifikationsregel ist in X' linear, im Ursprungsraum jedoch nicht linear. Die Transformation ϕ in den höherdimensionalen Raum ist im allgemeinen nicht effizient zu berechnen. Boser et.al. [2] zeigten jedoch, dass die Berechnung des Skalarproduktes in X', also $\vec{x_i} \cdot \vec{x_j}$, sowohl in der Lern- als auch in der Klassifikationsphase ausreicht. Dies wiederum kann für bestimmte Abbildungen ϕ sehr effizient mittels einer Kernfunktion $K(\vec{x_i}, \vec{x_j}) = \phi(\vec{x_i}) \cdot \phi(\vec{x_j})$ berechnet werden. So kann z.B. ein polynomialer Klassifizierer angewandt werden, basierend auf der Kernfunktion

$$K_{poly}(\vec{x_i}, \vec{x_j}) = (\vec{x_i} \cdot \vec{x_j})^d.$$

Ein RBF-Klassifikator verwendet die radiale Basis Funktion

$$K_{rbf}(\vec{x_i}, \vec{x_j}) = exp(-\gamma(\vec{x_i} - \vec{x_j})^2).$$

Zweilagige neuronale Netze entstehen durch Anwendung der Sigmoid-Funktion

$$K_{sigmoid}(\vec{x_i}, \vec{x_j}) = tanh(s(\vec{x_i} \cdot \vec{x_j}) + c).$$

Ein einfaches Beispiel für eine Menge von Trainingsdaten, die erst nach Transformation in einen höher dimensionalen Raum linear separabel ist, zeigt Abbildung 3.2.

Als Default wurde für den Prototypen ein RBF-Kern gewählt, weil er sich bei der Anwendung als stabil und im Training als performanter (vgl. Kapitel 6.6) erwies. Seine Parameter (γ sowie der in 3.1.2 eingeführte Wert C) wurden experimentell über Cross-Validation ermittelt. Dabei werden einzelne Trainingsbeispiele mit verschiedenen Kerneinstellungen anhand der übrigen Trainingsdaten klassifziert und so die günstigsten Parameter ausgewählt. LIBSVM bietet hierzu ein automatisiertes Skript in Python an, das diese Arbeit übernimmt. Um die Leistung des Klassifikators weiter zu verbessern, wurden zudem die Trainingsdaten vorher skaliert (auf ein Intervall von - 1 bis 1), so dass Unterschiede im Wertebereich der einzelnen Merkmale ausgeglichen werden. Die Einstellung für die Skalierung wurde für jedes Training abgespeichert, so dass auf die zu klassifizierenden Daten dieselben Skalierungsparameter angewandt werden konnten.

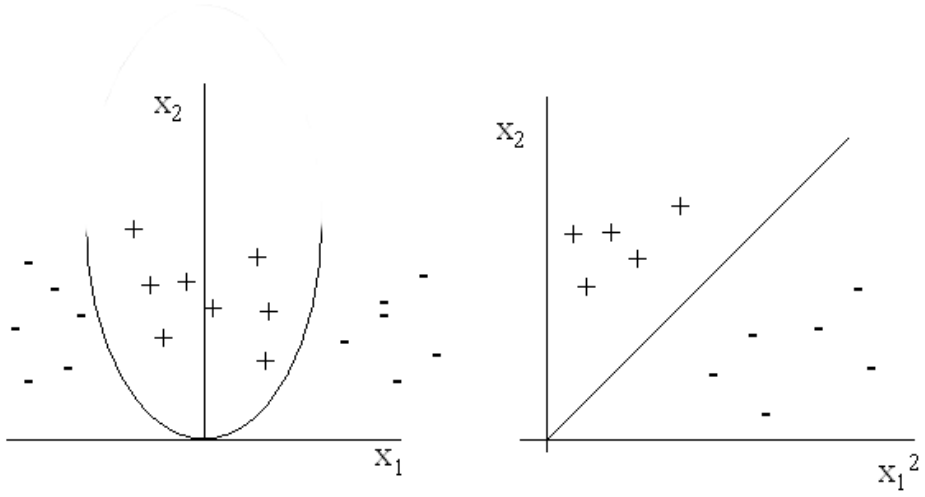

Abbildung 3.2: Linear nicht separable Daten, links vor und rechts nach der Transformation, projiziert auf x_1^2, x_2

3.1.4 Verwendete Bibliotheken

Der Prototyp arbeitet mit zwei verschiedenen Bibliotheken, einer SVM-Bibliothek in C (SVM-Light[2]) und einer in Java (LIBSVM[3]). Da die Applikation insgesamt in Java geschrieben ist, bot die Einbindung der Java-Bibliothek viele Vorteile. Insbesondere konnten die Klassen zur Darstellung der Merkmalsvektoren, SVM-Parameter, etc. direkt aus der Bibliothek verwendet werden. Es stellte sich jedoch die Frage, ob die Java-Implementierung wirklich ähnlich effizient sein kann wie die lang erprobte Bibliothek SVM-Light. Aus diesem Grunde wurden testweise beide Bibliotheken eingebunden. Beide sind durch einen Konfigurationsparameter aktivierbar oder deaktivierbar. Bei einem Training können also generell zwei Klassifikatoren trainiert werden. Bei der Klassifikation kann der Benutzer auswählen, welche Bibliothek er verwenden möchte. Im folgenden sollen die wichtigsten Funktionen beider Bibliotheken kurz vorgestellt werden.

SVM-Light

SVM-Light bietet in seiner neuesten Version auch die Funktionalität an, ein Training für mehrere Klassen gleichzeitig durchzuführen (Multiclassing). Dieses Feature wurde jedoch

[2]http://svmlight.joachims.org/

[3]http://www.csie.ntu.edu.tw/c̃jlin/libsvm/

im Prototyp noch nicht verwendet. Daher wird für *jede* Klasse separat ein Training durchgeführt. Folgende Kommandos von SVM-Light wurden verwendet:

- Training: `svm_learn [options] example_file model_file`
- Klassifikation: `svm_classify [options] example_file model_file output_file`

Für eine genauere Beschreibung der Bibliothek sei auf die Dokumentation der Software unter http://svmlight.joachims.org/ verwiesen.

LIBSVM

LIBSVM arbeitet standardmäßig mit Multiclassing, so dass diese Funktionalität auch ausgenutzt wurde. Die API bietet eine Modellierung der wichtigsten SVM-Konstrukte:

- `svm_model`: Ergebnis eines Trainings; speichert sowohl Informationen über die Kernparameter als auch über die anzuwendende Entscheidungfunktion (Hyperebene).
- `svm_parameter`: Parameter, die für ein Training verwendet werden (Kerneltyp, gamma, C, etc.)
- `svm_node`: Ein Merkmal und seine Ausprägung; analog dazu entspricht `svm_node[]` dem Merkmalsvektor eines Objektes.
- `svm_problem`: Eine Menge von Trainingsdaten, d.h. eine bestimmte Anzahl von Merkmalsvektoren und die zugehörigen Klassifizierungen (Labels).

Die vorliegende Anwendung verwendet die folgenden Funktionen der API:

- Training der SVM:
  `public static svm_model svm_train (svm_problem prob, svm_parameter param);`
- Klassifiziert ein Objekt anhand eines Modells:
  `public static double svm_predict (svm_model model, svm_node[] x);`
- Ermittelt die Wahrscheinlichkeiten für die Zugehörigkeit eines Objektes zu den Klassen anhand eines gegebenen Modells:
  `public static double svm_predict_probability (svm_model model, svm_node[] x, double[] prob_estimates);`
- Speichert ein Modell dateibasiert ab:
  `public static void svm_save_model (String model_file_name, svm_model model) throws IOException`

- Lädt ein Modell aus einer Datei:
  `public static svm_model svm_load_model (String model_file_name)`
  `throws IOException`
- Überprüft die Kernparameter für ein Training auf Inkonsistenzen:
  `public static String svm_check_parameter (svm_problem prob, svm_parameter param);`
- Überprüft, ob für ein gegebenes Modell die Berechnung von Wahrscheinlichkeiten bei der Klassifikation möglich ist:
  `public static int svm_check_probability_model (svm_model model);`

Kapitel 4

Sensitivitäts- und Faktorenanalyse

Ziel der vorliegenden Arbeit war neben der korrekten Klassifikation auch eine Einschätzung, welche der verwendeten Merkmale der Teilchen besonders aussagekräftig sind. Dabei sollte am Ende ein - auch für Nicht-Informatiker - gut interpretierbares Ergebnis stehen. Zunächst musste ein geeignetes Verfahren der multivariaten Analyse gefunden werden. Die Entscheidung für das Faktorenmodell bzw. die Hauptkomponentenanlyse (PCA - Principal Component Analysis) lag nahe, da dies gut erprobte und dokumentierte Methoden sind, um spezifische Einflussfaktoren einer statistischen Untersuchung zu ermitteln. Das allgemeine Faktorenmodell bildet dabei den umfassenden theoretischen Rahmen. In Absatz 4.1 werden zunächst einige statistische und algebraische Grundlagen eingeführt, die für das weitere Kapitel vorausgesetzt werden. In Absatz 4.2 wird das zugrunde liegende theoretische Modell beschrieben, wonach in Absatz 4.3 die konkrete Berechnung der Hauptfaktoren folgt. Als problematisch erweist sich bei der Hauptkomponentenanalyse die Tatsache, dass ihr Ergebnis unter Umständen schwer interpretierbar ist: Die ermittelten Hauptkomponenten sind sog. 'latente' Faktoren, d.h. es sind neue Faktoren, die als *Linearkombination* aus den ursprünglich verwendeten Parametern gebildet werden, so dass der direkte Einfluss der ursprünglichen Features nicht mehr ersichtlich ist. Um diesem Problem zu begegnen, wurde auf das Ergebnis der Hauptkomponentenanalyse eine Transformation angewandt, die in Absatz 4.4 näher erläutert wird. Absatz 4.5 gibt abschließend einen kurzen Überblick über die vorliegende Implementierung.

4.1 Statistische Grundlagen

Zunächst sollen nun die verwendeten statistischen Grundlagen und einige Sätze aus der Linearen Algebra, sofern sie für das Verständnis der Arbeit wichtig sind, kurz erläutert

werden[1]. Für die folgenden Definitionen nehmen wir an, dass für N Objekte die Beobachtungen für p Zufallsvariablen $X_1, X_2, ..., X_p$ vorliegen. Übertragen auf das Anwendungsgebiet dieser Arbeit betrachten wir also N Teilchen, für die jeweils p Merkmale berechnet wurden. Die Daten sind in Form einer (N x p)-Datenmatrix **A** angeordnet, so dass

$$\mathbf{A} = \begin{pmatrix} \mathbf{a_1^T} \\ \vdots \\ \mathbf{a_n^T} \\ \vdots \\ \mathbf{a_N^T} \end{pmatrix} = \begin{pmatrix} a_{11} & \cdots & a_{1j} & \cdots & a_{1p} \\ \vdots & & & & \vdots \\ a_{n1} & \cdots & a_{nj} & \cdots & a_{np} \\ \vdots & & & & \vdots \\ a_{N1} & \cdots & a_{Nj} & \cdots & a_{Np} \end{pmatrix}$$

Definition 25. Der *Erwartungswert E[X] einer Zufallsvariablen X* mit Wahrscheinlichkeitsdichtefunktion $f_x(x)$ ist folgendermaßen definiert:

$$E[X] = \begin{cases} \sum x f_x(x) \text{ falls X eine diskrete Zufallsvariable ist} \\ \int_{-\infty}^{+\infty} x f_x(x) dx \text{ falls X eine kontinuierliche Zufallsvariable ist} \end{cases} \tag{4.1}$$

E[X] gibt also an, welchen Wert X bei einer oftmaligen Wiederholung des Experimentes (d.h. in unserem Fall bei der Messung an 'vielen' Teilchen) im 'Mittel' annimmt. Der *Erwartungswert des Vektors* $\mathbf{X} \in \Re^p$ besteht aus den p Erwartungswerten der einzelnen Zufallsvariablen:

$$E(\mathbf{X}) = \begin{pmatrix} E(X_1) \\ \vdots \\ E(X_p) \end{pmatrix} = \begin{pmatrix} \mu_1 \\ \vdots \\ \mu_p \end{pmatrix} = \mu$$

Definition 26. Das *arithmetische Mittel* von **X** ist ein p-dimensionaler Vektor

$$\overline{\mathbf{x}} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{a_i} = \frac{1}{N} \begin{pmatrix} \sum_{i=1}^{N} a_{i1} \\ \vdots \\ \sum_{i=1}^{N} a_{ip} \end{pmatrix} = \begin{pmatrix} \overline{\mathbf{x}}_1 \\ \vdots \\ \overline{\mathbf{x}}_p \end{pmatrix}$$

und stellt eine Schätzung für den Erwartungswert $E(\mathbf{X}) = \mu$ dar.

Definition 27. Unter der (empirischen) *Varianz* versteht man ein Streuungsmaß, d.h. ein Maß für die durchschnittliche Abweichung einer Zufallsvariable X_k von ihrem Erwartungswert μ_k, abgeschätzt durch $\overline{x}_k$. Sie berechnet sich aus der Summe der quadrierten

[1]Für eine ausführlichere Darstellung sei u.a. auf [20] oder [16] verwiesen.

Abweichungen der einzelnen Werte von ihrem Mittelwert, dividiert durch die Anzahl der Beobachtungen:

$$var(X_k) = \frac{1}{N}\sum_{i=1}^{N}(a_{ik} - \mu_k)^2$$

Die Quadratwurzel aus der Varianz bezeichnet man als *Standardabweichung*.

Definition 28. Die (empirische) *Kovarianz* zweier Zufallsvariablen X_j und X_k gibt an, wie stark der Zusammenhang zwischen den beiden Variablen ist. Sie wird folgendermaßen berechnet:

$$Cov(X_j, X_k) = \sigma_{jk} = \frac{1}{N}\sum_{i=1}^{N}(a_{ik} - \mu_k) \cdot (a_{ij} - \mu_j)$$

Sie ist 0, wenn die Variablen X_j und X_k voneinander unabhängig sind. Ein positiver Wert σ_{jk} besagt, dass mit einer großen Wahrscheinlichkeit entweder sowohl X_j also auch X_k hohe oder aber beide niedrige Werte annehmen. Ein negativer Wert dagegen drückt aus, dass mit großer Wahrscheinlichkeit ein hoher X_j-Wert mit einem niedrigen X_k-Wert verbunden ist und umgekehrt. Intuitiv ist deutlich, dass $\sigma_{jk} = \sigma_{kj}$. Aus der Definition der Varianz (27) ergibt sich $var(X_k) = \sigma_{kk}$. Die symmetrische Kovarianzmatrix Σ besteht aus den Kovarianzen der einzelnen Zufallsvariablen miteinander:

$$\Sigma = \begin{pmatrix} \sigma_{11} & \sigma_{12} & \cdots & \sigma_{1p} \\ \sigma_{21} & \sigma_{22} & \cdots & \sigma_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{p1} & \sigma_{p2} & \cdots & \sigma_{pp} \end{pmatrix}$$

Die Aussagekraft der Kovarianz ist jedoch noch relativ, da sie von den Maßeinheiten der einzelnen Variablen abhängt. Von daher betrachtet man oft die aus der Kovarianzmatrix resultierende Korrelationsmatrix.

Definition 29. Der Korrelationskoeffizient ρ_{jk} berechnet sich aus der Kovarianz σ_{jk} und den Varianzen σ_{jj} und σ_{kk}:

$$\rho_{jk} = \frac{\sigma_{jk}}{\sqrt{\sigma_{jj}}\sqrt{\sigma_{kk}}}$$

Es gilt:

$$\rho_{jj} = \frac{\sigma_{jj}}{\sqrt{\sigma_{jj}}\sqrt{\sigma_{jj}}} = 1$$

Die Korrelationsmatrix ist eine symmetrische p x p-Matrix der Form:

$$Cor(\mathbf{X}) = \begin{pmatrix} 1 & \rho_{12} & \cdots & \rho_{1p} \\ \rho_{21} & 1 & \cdots & \sigma_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ \rho_{p1} & \rho_{p2} & \cdots & 1 \end{pmatrix}$$

Satz 1. Sei $\mathbf{Z}$ ein q-dimensionaler Vektor, $\mathbf{C}$ eine q x p-Matrix und $\mathbf{X}$ ein p-dimensionaler Vektor, so dass $\mathbf{Z}$ als Linearkombination aus $\mathbf{CX}$ entsteht ($\mathbf{Z} = \mathbf{CX}$):

$$Z_1 = c_{11}X_1 + c_{12}X_2 + \cdots + c_{1p}X_p$$

$$Z_2 = c_{21}X_1 + c_{22}X_2 + \cdots + c_{2p}X_p$$

$$\vdots$$

$$Z_q = c_{q1}X_1 + c_{q2}X_2 + \cdots + c_{qp}X_p$$

Dann gilt für den Erwartungswert und die Kovarianzmatrix von $\mathbf{Z}$:

$$\mu_z = E(\mathbf{Z}) = E(\mathbf{CX}) = C\mu_x$$

$$\Sigma_z = Cov(\mathbf{Z}) = Cov(\mathbf{CX}) = \mathbf{C}\Sigma_x\mathbf{C^T}$$

Satz 2. Sei $\mathbf{x} \in \Re^p, \mathbf{x} \neq \mathbf{0}$ und B eine positiv definite p x p - Matrix mit Eigenwerten $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p \geq 0$ und den zugehörigen normalisierten Eigenvektoren $\mathbf{e_1}, \mathbf{e_2}, ..., \mathbf{e_p}$. Dann gilt:

$$max\frac{\mathbf{x^T B x}}{\mathbf{x^T x}} = \lambda_1$$

Dieses Maximum wird genau dann erreicht, wenn $\mathbf{x} = \mathbf{e_1}$.

$$min\frac{\mathbf{x^T B x}}{\mathbf{x^T x}} = \lambda_p$$

Dieses Minimum wird genau dann erreicht, wenn $\mathbf{x} = \mathbf{e_p}$.

$$max\frac{\mathbf{x^T B x}}{\mathbf{x^T x}} = \lambda_{k+1} \text{ für } x \perp e_1, e_2, .., e_k$$

Dieses Maximum wird genau dann erreicht, wenn $\mathbf{x} = \mathbf{e_{k+1}}, k = 1, 2, ..., p-1$.[2]

Satz 3. Sei B eine positiv definite p x p - Matrix mit Eigenwerten $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p \geq 0$ und den zugehörigen normalisierten Eigenvektoren $\mathbf{e_1}, \mathbf{e_2}, ..., \mathbf{e_p}$. Dann ist die spektrale Dekomposition von B gegeben durch:

$$B = \lambda_1\mathbf{e_1}\mathbf{e_1^T} + \lambda_2\mathbf{e_2}\mathbf{e_2^T} + \cdots + \lambda_p\mathbf{e_p}\mathbf{e_p^T}$$

[2](Beweis siehe [20], S.81f)

4.2 Das Faktorenmodell

Das grundlegende Ziel der Faktorenanalyse ist es, die Kovarianzbeziehungen von p beobachtbaren korrelierten Variablen in Form einiger weniger latenter, d.h. nicht beobachtbarer Variablen auszudrücken. Man geht dabei davon aus, dass Variablen, die untereinander stark korrelieren, als Gruppe zusammengefasst werden können, und dass Variablen der einen Gruppe kaum Korrelationen mit den Variablen anderer Gruppen aufweisen. Jede Gruppe von Variablen entspricht dann quasi einem neuen latenten Faktor. Die Faktorenanalyse basiert auf dem Faktorenmodell, das im Folgenden kurz skizziert wird. Eine ausführlichere Darstellung findet sich u.a. in [20] und [16].

Es sei $\mathbf{X} = X_1, X_2, ..., X_p$ ein p-dimensionaler Vektor mit p beobachtbaren Zufallsvariablen, Σ die zugehörige Kovarianzmatrix und μ der Erwartungswert $\mathbf{E(X)}$. Das Faktorenmodell geht davon aus, dass $\mathbf{X}$ ausgedrückt werden kann als Linearkombination mehrerer latenter Faktoren (sog. gemeinsamen Faktoren) $F_1, F_2, ..., F_m$ sowie einigen spezifischen Faktoren (Einzelrestfaktoren) $\varepsilon_1, \varepsilon_2, ..., \varepsilon_p$, die sich nur auf einzelne Variablen auswirken . Genauer lässt sich diese Annahme wie folgt beschreiben:

$$\mathbf{X} = \mu + \mathbf{LF} + \varepsilon \tag{4.2}$$

Dabei ist $\mathbf{F}$ die (m x 1) - Matrix der latenten Faktoren, ε die (p x 1) - Matrix der Einzelrestfaktoren und $\mathbf{L}$ die (p x m) - Ladungsmatrix, die den Einfluss der ursprünglichen Variablen auf die latenten Faktoren angibt: l_{ij} beschreibt somit den Einfluss der i-ten Variable auf den j-ten Faktor. Da die Einzelrestfaktoren ε und die gemeinsamen Faktoren $\mathbf{F}$ nicht beobachtbar sind, kann ihr Nullpunkt frei gewählt werden. Daher kann o.B.d.A. angenommen werden, dass sie den Erwartungswert 0 besitzen:

$$E(\mathbf{F}) = 0, E(\varepsilon) = 0 \tag{4.3}$$

Von den spezifischen Faktoren ε wird angenommen, dass sie unkorreliert sind. Ihre Varianzen nehmen jedoch beliebige Werte an, so dass gilt:

$$Cov(\varepsilon) = E(\varepsilon\varepsilon^T) = V = diag\{v_1^2, ..., v_p^2\} \tag{4.4}$$

Für die Faktorvariablen $\mathbf{F}$ wird zusätzlich vorausgesetzt, dass sie bezüglich der spezifischen Faktoren unkorreliert sind:

$$Cov(\mathbf{F}, \varepsilon) = E(\mathbf{F}\varepsilon^T) = 0 \tag{4.5}$$

Beim orthogonalen Faktormodell[3] wird darüber hinaus angenommen, dass die Faktoren untereinander linear unabhängig sind:

(4.6)

$$Cov(\mathbf{F}) = E(\mathbf{F}\mathbf{F}^{\mathbf{T}}) = I \text{ (Identität)} \tag{4.7}$$

Das faktorenanalytische Modell ist auf der Basis der zu analysierenden Rohdaten schwer überprüfbar. Die Annahmen 4.4 bis 4.7 beinhalten jedoch auch eine Aussage über die Kovarianzmatrix Σ von X, die statistisch getestet werden kann *(Grundgleichung der Faktorenanalyse)*:

$$\begin{aligned}\Sigma &= E((\mathbf{X}-\mu)(\mathbf{X}-\mu)^T) = E((\mathbf{LF}+\varepsilon)(\mathbf{LF}+\varepsilon)^T) = \\ &\quad E(\mathbf{LF}\mathbf{F}^{\mathbf{T}}\mathbf{L}^{\mathbf{T}}) + E(\mathbf{LF}\varepsilon^{\mathbf{T}}) + E(\varepsilon\mathbf{F}^{\mathbf{T}}\mathbf{L}^{\mathbf{T}}) + E(\varepsilon\varepsilon^{\mathbf{T}}) = \\ &\quad \mathbf{L}\mathbf{L}^{\mathbf{T}} + 0 + 0 + \mathbf{V} = \\ &\quad \mathbf{L}\mathbf{L}^{\mathbf{T}} + \mathbf{V}\end{aligned}$$

Aus den Annahmen 4.4 bis 4.7 folgt weiterhin:

$$\begin{aligned}Cov(\mathbf{X},\mathbf{F}) &= E((\mathbf{X}-\mu)\mathbf{F}^{\mathbf{T}}) = \\ &\quad E((\mathbf{LF}+\varepsilon)\mathbf{F}^{\mathbf{T}}) = E(\mathbf{LF}\mathbf{F}^{\mathbf{T}} + \varepsilon\mathbf{F}^{\mathbf{T}}) = \\ &\quad \mathbf{L}E(\mathbf{F}\mathbf{F}^{\mathbf{T}}) + E(\varepsilon\mathbf{F}^{\mathbf{T}}) = \mathbf{L}\end{aligned}$$

Die Ladungsmatrix $\mathbf{L}$ stellt also im orthogonalen Faktorenmodell die Kovarianzmatrix zwischen den beobachtbaren und den latenten Faktoren dar.

Zur Abschätzung der latenten Faktoren $\mathbf{F}$ sowie der Ladungsmatrix $\mathbf{L}$ können verschiedene Methoden angewandt werden. Eine davon ist die Hauptkomponentenanalyse, die im folgenden Abschnitt erläutert wird.

4.3 Die Hauptkomponentenanalyse

Ziel einer Hauptkomponentenanalyse ist die Reduktion der p beobachtbaren korrelierten Variablen auf einige wenige latente Faktoren. Diese entstehen als Linearkombinationen der ursprünglichen Variablen und sollen einen möglichst großen Anteil von deren Gesamtvarianz in sich tragen. Daher wird zunächst eine lineare Transformation der beobachtbaren

[3]im Gegensatz zum obliquen Faktorenmodell, das auch korrelierte Faktoren zulässt

Variablen zu den neuen latenten Variablen, den sog. Hauptkomponenten, durchgeführt. Die gefundenen Faktoren werden absteigend nach ihrer Varianz sortiert. Daraus werden dann die wichtigsten k Faktoren ausgewählt (vgl. Absatz 4.3.3). Geometrisch stellen die latenten Faktoren ein neues, k-dimensionales Koordinatensystem dar, wobei die Achsen die Richtungen mit maximaler Varianz darstellen.

4.3.1 Berechnung der Hauptkomponenten

Es seien $X_1, X_2, ..., X_p$ p beobachtbare Zufallsvariablen mit der zugehörigen Kovarianzmatrix[4] Σ und deren Eigenwerten $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p \geq 0$ und Eigenvektoren $e_1, e_2, ..., e_p$. Es seien weiterhin $A = \begin{pmatrix} \mathbf{a_1^T X} \\ \mathbf{a_2^T X} \\ \vdots \\ \mathbf{a_p^T X} \end{pmatrix}$ eine p x p-Koeffizientenmatrix und $Y_1, Y_2, ..., Y_p$ Linearkombinationen der Form:

$$Y_1 = \mathbf{a_1^T X} = a_{11}X_1 + a_{12}X_2 + \cdots + a_{1p}X_p$$

$$Y_2 = \mathbf{a_2^T X} = a_{21}X_1 + a_{22}X_2 + \cdots + a_{2p}X_p$$

$$\vdots$$

$$Y_p = \mathbf{a_p^T X} = a_{p1}X_1 + a_{p2}X_2 + \cdots + a_{pp}X_p$$

Dann gilt nach Satz 1:

$$Var(Y_i) = Cov(Y_i, Y_i) = \sigma_{ii} = \mathbf{a_i^T \Sigma a_i} \text{ für } i = 1, 2, ..., p$$

$$Cov(Y_i, Y_k) = \mathbf{a_i^T \Sigma a_k} \text{ für } i, k = 1, 2, ..., p$$

Die Hauptkomponenten sind die linear unabhängigen $Y_1, Y_2, ..., Y_p$, deren Varianzen maximal sind. Die i-te Hauptkomponente ist dabei diejenige Linearkombination $\mathbf{a_i^T X}$, die $Var(Y_i)$ maximiert, wobei $Cov(a_i^T, a_k^T) = 0$ für $k < i$. Die folgende Definition besagt, dass genau die Eigenvektoren der Kovarianzmatrix die für die Hauptkomponenten geforderten Eigenschaften erfüllen, wobei die Eigenwerte dann die jeweiligen Varianzen reflektieren. Der Ansatz des Beweises wird anschließend kurz skizziert.

[4] In den folgenden Erläuterungen wird immer von der Kovarianzmatrix die Rede sein. Prinzipiell ist die Berechnung der Hauptkomponenten sowohl auf Basis der Kovarianz- als auch der Korrelationsmatrix möglich. Letztere stellt nur eine Skalierung der Kovarianzmatrix dar, die unterschiedliche Werteskalen für verschiedene Parameter ausgleicht.

Definition 30. Sei Σ die Kovarianzmatrix des p-dimensionalen Vektors $\mathbf{X}$ der beobachtbaren Variablen $X_1, X_2, ..., X_p$. Seien $(\lambda_1, \mathbf{e_1}), (\lambda_2, \mathbf{e_2}), ..., (\lambda_p, \mathbf{e_p})$ die Eigenwert-Eigenvektor-Paare von Σ, mit $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p \geq 0$. Dann ist die i-te Hauptkomponente gegeben durch:

$$Y_i = \mathbf{e_i^T X} = e_{i1}X_1 + e_{i2}X_2 + ... + e_{ip}X_p \text{ für } i = 1, 2, ..., p \tag{4.8}$$

Weiter gilt:

$$Var(Y_i) = \mathbf{e_i^T \Sigma e_i} = \lambda_i \text{ für } i = 1, 2, ..., p \tag{4.9}$$

$$Cov(Y_i, Y_k) = \mathbf{e_i^T \Sigma e_k} = 0 \text{ für } i \neq k \tag{4.10}$$

Beweis Aus Satz 2 erhalten wir für $B = \Sigma$:

$$max \frac{\mathbf{a^T \Sigma a}}{\mathbf{a^T a}} = \lambda_1 \text{ wobei dann gilt } \mathbf{a} = \mathbf{e_1}$$

Da es sich um normierte Eigenvektoren handelt, gilt $\mathbf{e_1^T e_1} = 1$. Daraus folgt (unter Verwendung von Satz 1):

$$max \frac{\mathbf{a^T \Sigma a}}{\mathbf{a^T a}} = \lambda_1 = \frac{\mathbf{e_1^T \Sigma e_1}}{\mathbf{e_1^T e_1}} = \mathbf{e_1^T \Sigma e_1} = \mathbf{a_1^T \Sigma a_1} = Var(Y_1)$$

Damit ist 4.9 erfüllt.

Weiterhin gilt nach Satz 2:

$$max \frac{\mathbf{a^T \Sigma a}}{\mathbf{a^T a}} = \lambda_{k+1} \text{ für } \mathbf{a} \perp \mathbf{e_1, e_2, .., e_k}$$

D.h. falls der k+1-te Eigenvektor senkrecht auf den vorangehenden k Eigenvektoren steht, ist seine Varianz genau λ_{k+1}.Für ein beliebiges $\mathbf{a} = \mathbf{e_{k+1}}$ mit $\mathbf{e_{k+1} e_i} = 0$ für i=1,2,...,k und k=1,2,...,p-1 erhalten wir somit:

$$\frac{\mathbf{e_{k+1}^T \Sigma e_{k+1}}}{\mathbf{e_{k+1}^T e_{k+1}}} = \mathbf{e_{k+1}^T \Sigma e_{k+1}} = Var(Y_{k+1}) = \lambda_{k+1}$$

Damit bleibt nur noch zu zeigen, dass die gefundenen Hauptkomponenten $Y_1, Y_2, ..., Y_p$ voneinander unabhängig sind, d.h. ihre Kovarianz ist 0. Dies entspricht dem Beweis, dass die Eigenvektoren $e_1, e_2, ..., e_p$ linear unabhängig sind, d.h. $\mathbf{e_i^T e_k} = 0$ für $i \neq k$. Hierzu betrachten wir eine spezifische Eigenschaft der Eigenvektoren:

Satz 4. Die zu verschiedenen Eigenwerten gehörenden Eigenvektoren sind paarweise orthogonal. Falls die Eigenwerte $\lambda_1, \lambda_2, ..., \lambda_p$ nicht alle verschieden sind, gibt es zu $\lambda_1, \lambda_2, ..., \lambda_p$ mindestens eine Menge von n paarweise orthogonalen Eigenvektoren $\mathbf{e_1}, \mathbf{e_2}, ..., \mathbf{e_p}$.

Mit $\mathbf{e_k^T}\Sigma\mathbf{e_k} = \lambda_k \Rightarrow \Sigma\mathbf{e_k} = \lambda_k\mathbf{e_k}$ folgt daraus:

$$Cov(Y_i, Y_k) = \mathbf{e_i^T}\Sigma\mathbf{e_k} = \mathbf{e_i^T}\lambda_k\mathbf{e_k} = \lambda_k\mathbf{e_i^T}\mathbf{e_k} = 0 \text{ für } i \neq k$$

Damit ist eine Abschätzung für die latenten Faktoren erreicht. Nun stellt sich die Frage, wie aus diesen p Faktoren die wesentlichen ausgewählt werden können und wie die Ladungsmatrix berechnet werden kann, die den Zusammenhang zwischen den latenten Faktoren und den ursprünglichen Variablen herstellt.

4.3.2 Berechnung der Ladungsmatrix

Sei Σ die Kovarianzmatrix des p-dimensionalen Vektors $\mathbf{X}$ der beobachtbaren Variablen $X_1, X_2, ..., X_p$. Seien $(\lambda_1, \mathbf{e_1}), (\lambda_2, \mathbf{e_2}), ..., (\lambda_p, \mathbf{e_p})$ die Eigenwert-Eigenvektor-Paare von Σ, mit $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p \geq 0$. Sei m <p die Anzahl der ausgewählten allgemeinen Faktoren. Dann lässt sich Σ nach Satz 3 als spektrale Dekomposition seiner Eigenwerte und Eigenvektoren wie folgt darstellen:

$$\begin{aligned}\Sigma &= \lambda_1\mathbf{e_1}\mathbf{e_1^T} + \lambda_2\mathbf{e_2}\mathbf{e_2^T} + \cdots + \lambda_p\mathbf{e_p}\mathbf{e_p^T} = \\ &[\sqrt{\lambda_1}\mathbf{e_1}\sqrt{\lambda_2}\mathbf{e_2}\vdots\cdots\vdots\sqrt{\lambda_p}\mathbf{e_p}] \begin{bmatrix} \sqrt{\lambda_1}\mathbf{e_1^T} \\ \sqrt{\lambda_2}\mathbf{e_2^T} \\ \vdots \\ \sqrt{\lambda_p}\mathbf{e_p^T} \end{bmatrix} = \mathbf{L}\mathbf{L^T}(\text{ nach 4.8})\end{aligned}$$

Daher berechnet sich die Ladungsmatrix für m ausgewählte Faktoren wie folgt:

$$L = [\sqrt{\lambda_1}\mathbf{e_1}\vdots\sqrt{\lambda_2}\mathbf{e_2}\vdots\cdots\vdots\sqrt{\lambda_m}\mathbf{e_m}] \tag{4.11}$$

4.3.3 Auswahl der Hauptkomponenten

Die ausgewählten Hauptkomponenten sollen einen möglichst großen Anteil der Gesamtvarianz in sich vereinigen. Daher muss zunächst betrachtet werden, wie der Anteil eines Faktors an der Gesamtvarianz abgeschätzt werden kann:

Definition 31. Sei Σ die Kovarianzmatrix des p-dimensionalen Vektors $\mathbf{X}$ der beobachtbaren Variablen $X_1, X_2, ..., X_p$. Seien $(\lambda_1, \mathbf{e_1}), (\lambda_2, \mathbf{e_2}), ..., (\lambda_p, \mathbf{e_p})$ die Eigenwert-Eigenvektor-Paare von Σ, mit $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p \geq 0$. Seien $Y_1 = \mathbf{e_1^T}\mathbf{X}, Y_2 = \mathbf{e_2^T}\mathbf{X}, ..., Y_p = \mathbf{e_p^T}\mathbf{X}$ die

errechneten Hauptkomponenten. Dann ist die Gesamtvarianz gegeben durch die Summe aller Eigenwerte λ_i und es gilt:

$$\sigma_{11} + \sigma_{22} + \cdots + \sigma_{pp} = \sum_{i=1}^{p} Var(X_i) = \lambda_1 + \lambda_2 + \cdots + \lambda_p = \sum_{i=1}^{p} Var(Y_i)$$

(Beweis siehe [20], S.429). Der Anteil der k-ten Hauptkomponenten an der Gesamtvarianz beträgt demnach $\frac{\lambda_k}{\lambda_1+\lambda_2+\cdots+\lambda_p}$, für k = 1,2,...,p.

Ein mögliches Kriterium für die Auswahl der Faktoren ist ein festgelegter Prozentsatz der Gesamtvarianz, der minimal von den ausgewählten Faktoren abgebildet werden muss. Ein anderer Ansatz ist, einen Schwellwert für den Eigenwert zu spezifizieren und diejenigen Eigenvektoren als Hauptkomponenten auszuwählen, deren zugehöriger Eigenwert über diesem Schwellwert liegt. Eine weitere Möglichkeit besteht darin, eine festgelegte Anzahl k von Faktoren auszuwählen, d.h. die k größten Eigenwerte. Im vorliegenden Anwendungsbeispiel zeigte sich experimentell, dass es nur zwei wirklich einflussreiche Faktoren gibt. Daher wurde für den ersten Prototypen die Vereinfachung akzeptiert, dass generell die beiden ersten Hauptkomponenten ausgewählt werden. Dies hat insbesondere den Vorteil, dass die nun folgende Rotation erheblich weniger komplex ist als bei einer beliebigen Anzahl von Komponenten. Für die weitere Entwicklung sollte jedoch im Hinblick auf anders strukturierte Anwendungsdaten hier ein flexiblerer Ansatz gewählt werden.

4.4 Orthogonale Rotation der Hauptkomponenten

Wie schon in der Einleitung erwähnt, ist die Interpretierbarkeit der gefundenen Hauptkomponenten und der zugehörigen Ladungen nicht immer einfach, da die ursprünglichen Variablen in den verschiedenen Faktoren als Teil einer Linearkombination erscheinen. Eine Möglichkeit, die Interpretation der Faktoren transparenter zu machen, ist die Transformation der Faktoren durch eine orthogonale Rotation. Dabei werden je zwei normierte Faktoren F_r, F_s um den Winkel α rotiert, so dass folgende neue Faktoren $\widetilde{F_r}, \widetilde{F_s}$ entstehen:

$$\begin{pmatrix} \widetilde{F_r} \\ \widetilde{F_s} \end{pmatrix} = \begin{pmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} F_r \\ F_s \end{pmatrix}$$

Gleichzeitig wird die Ladungsmatrix mittels der inversen Abbildung rotiert, so dass sich das Produkt aus Faktoren und Ladung LF und dessen Beitrag zur Gesamtvarianz der Variablen nicht ändert:

$$\widetilde{L} = L \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix}$$

Ziel der Rotation ist es, Faktoren so auszurichten, dass sie Variablen verwandter Gruppen beinhalten, so dass der neue Faktor durch die Eigenart dieser Gruppe charakterisiert werden kann. Um eine geignete Rotation zu bestimmen, bietet die Literatur eine Vielzahl von Kriterien. Eines davon ist die Varimax-Methode, die bei der Implementierung des Prototypen zur Anwendung kam und daher näher erläutert werden soll.

4.4.1 Varimax-Methode

Die Varimax-Methode wurde 1958 von Kaiser [11] entwickelt. Ihr Ziel ist, die Faktoren so zu rotieren, dass die zugehörige Ladungsmatrix in einigen der ursprünglichen Variablen sehr hohe, in anderen sehr niedrige Werte aufweist. Die Ladungsquadrate sollen entweder sehr große oder sehr kleine Werte annehmen, d.h. ihre Varianz soll maximiert werden. Von dieser Grundidee hat die Methode ihren Namen.

Definition 32. Sei $\widetilde{L}$ die Ladungsmatrix der transformierten Faktoren. Dann ist das Varimax-Kriterium wie folgt definiert:

$$v = \sum_{r=1}^{k}(\frac{1}{p}\sum_{i=1}^{p}\widetilde{l}_{ir}^{4} - (\frac{1}{p}\sum_{i=1}^{p}\widetilde{l}_{ir}^{2})^{2}) \rightarrow \max, \tag{4.12}$$

Die Rotation verläuft in mehreren Iterationen, da jeweils nur zwei Faktoren gleichzeitig rotiert werden können. Ein Konvergenzkriterium gibt an, wann das Ergebnis stabil ist und keine weiteren Iterationen mehr durchgeführt werden müssen. Gibt es nur zwei Faktoren, so lässt sich die Maximierung mittels einer einfachen Berechnung durchführen (vgl. [11] und [18]): Seien $\mathbf{l_k}$ und $\mathbf{l_l}$ die zu rotierenden Ladungsvektoren und t und b definiert wie folgt:

$$\begin{aligned} t &= 2(p\sum_{j=1}^{p}(l_{jk}^{2} - l_{jl}^{2})(2l_{jk}l_{jl}) - \sum_{j=1}^{p}(l_{jk}^{2} - l_{jl}^{2})\sum_{j=1}^{p}(2l_{jk}l_{jl})) \\ b &= p\sum_{j=1}^{p}((l_{jk}^{2} - l_{jl}^{2}) - (2l_{jk}l_{jl})^{2}) - (\sum_{j=1}^{p}(l_{jk}^{2} - l_{jl}^{2}) - (\sum_{j=1}^{p}2l_{jk}l_{jl}))^{2} \end{aligned} \tag{4.13}$$

Dann ist der optimale Winkel für die Rotation gegeben durch:

$$\phi = \frac{1}{4}\arctan(t, b) \tag{4.14}$$

4.5 Implementierung und verwendete Bibliotheken

Die Anwendung verwendet eine spezielle Bibliothek für die Vektorrechnung (Java Matrix Package[5]). Aus den skalierten Trainingsdaten, einer N x 15 - Matrix (N Teilchen, 15 Features), wird zunächst die Kovarianz- und Korrelationsmatrix berechnet. Ergebnis ist eine 15 x 15 - Matrix. Anschließend werden deren Eigenwerte und Eigenvektoren berechnet. Die beiden Eigenvektoren mit den höchsten Eigenwerten fungieren als Hauptkomponenten. Für sie wird die Ladungsmatrix berechnet (vgl. 4.11), eine 15 x 2 - Matrix. Für die beiden Ladungsvektoren kann eine Rotation durchgeführt werden, die die Varianz der Ladungsquadrate maximiert. Da nur zwei Hauptkomponenten ausgewählt wurden, kann der optimale Winkel für die Rotation über die numerische Gleichung gelöst werden, die in 4.13 und 4.14 vorgestellt wurde. Zunächst erfolgt die Berechnung von t und b aus den Werten der Ladungsmatrix. Diese werden anschließend in 4.14 eingesetzt. Würden mehr als zwei Hauptkomponenten ausgewählt, so müssten iterativ jeweils zwei der ausgewählten Ladungsvektoren rotiert werden, bis das Varianz gemäß einem vorgegebenen Konvergenzkriterium konvergiert. Die Auswahl einer beliebiger Anzahl von Hauptkomponenten ist für spätere Entwicklungen sicher eine wünschenswerte Erweiterung.

Nach der Rotation wird der Einfluss der Klassifikationsparameter direkt von der rotierten Ladungsmatrix abgelesen: je höher der Ladungswert, desto größer der Einfluss des Parameters. Mittels eines frei gewählten Schwellwertes können nun die relevanten von den weniger relevanten Klassifikationsparametern unterschieden werden. Dieser Schwellwert bestimmt in der vorliegenden Anwendung auch, welche der Parameter dem Benutzer gegenüber als besonders wichtig ausgewiesen werden (und so in die Benutzerführung mit eingehen).

[5] http://math.nist.gov/javanumerics/jama/

Kapitel 5

System-Architektur

In diesem Kapitel werden die Gesamtarchitektur des Systems und seine einzelnen Komponenten näher erläutert. Dabei geht es weniger um Implementierungsdetails als um einen allgemeinen Überblick über die Funktionsweise des Systems. Insbesondere sollen das Datenmodell und einige Entwurfsentscheidungen diskutiert werden. Von besonderer Bedeutung waren dabei im Hinblick auf das Anwendungsgebiet die folgenden Anforderungen:

- Flexibles Datenmodell, das die Integration zusätzlicher teilchenbasierter Kenngrößen ermöglicht (vgl. Abschnitt 5.1.1)
- Integration weiterer Graphitklassen auf Konfigurationsebene (vgl. Abschnitt 5.2.2)
- Erweiterbare Software-Architektur (vgl. Abschnitt 5.2.1 und 5.2.2)

5.1 Überblick über das Gesamtsystem

Ziel der Entwicklung war ein über das Internet zugreifbares System. Im Hinblick auf die angestrebte Plattformunabhängigkeit und Portierbarkeit lag die Entscheidung für Java als Programmiersprache und für Servlets als Webtechnologie nahe. Die Verwendung von JSP-Seiten bot dabei gegenüber reinen Servletklassen eine deutlichere Trennung von Programmlogik und HTML-Design. Alle *clientseitigen* Technologien sollten möglichst vermieden werden, um die Unabhängigkeit von Plattform, Browser, etc. zu gewährleisten. Die grobe Gesamtarchitektur der Anwendung entspricht quasi dem Standard einer Webanwendung; daher soll an dieser Stelle Abbildung 5.1 als Skizze genügen. Die Klassifikation mittels der verwendeten SVM-Bibliotheken geschieht in beiden Fällen dateibasiert, d.h. das Modell für die Klassifikation muss als Datei im Filesystem vorliegen. Die Speicherung dieser Modelldateien in einer Datenbank ist prinzipiell nicht notwendig. Daher schien

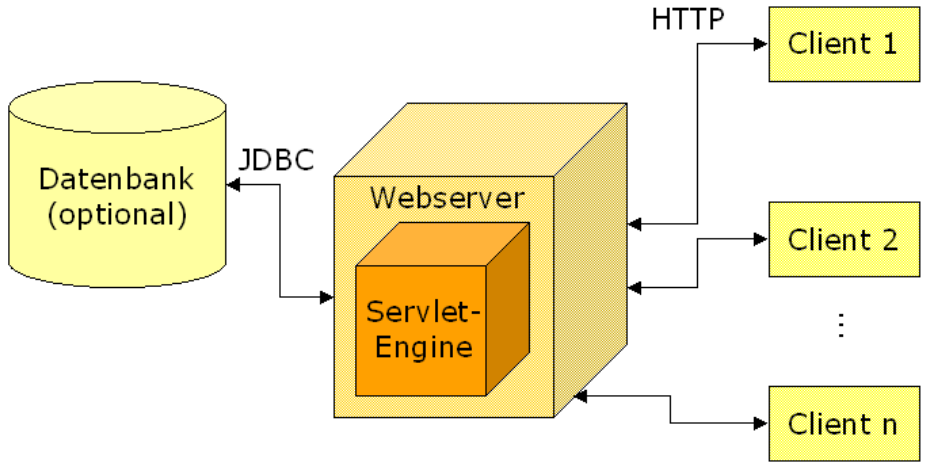

Abbildung 5.1: Überblick über die Systemarchitektur

es wünschenswert, die Art der Modellspeicherung konfigurierbar zu halten (vgl. Absatz 5.2.2). Aus diesem Grunde ist die Anbindung einer Datenbank nur optional für die Anwendung - alternativ können die Modelle auch serverseitig dateibasiert abgelegt werden. Unabhängig davon kann jedoch eine Datenbank genutzt werden, um bestimmte statistische Auswertungen über die einzelnen Teilchen vorzunehmen: Hier bringt die datenbankbasierte Speicherung einen deutlichen Mehrwert, da mit der strukturierten Form zusätzliche semantische Informationen verbunden sind. Das zugrundeliegende Datenmodell ist in Abbildung 5.2 dargestellt und soll im folgenden Abschnitt erläutert werden.

5.1.1 Datenmodell

Ein **Modell** ist Ergebnis eines Trainings und kann aus beliebig vielen Submodellen bestehen. Für jedes Modell werden die SVM-Parameter gespeichert, die beim Training verwendet wurden: der Kerneltyp (`kernel`), eine Datei, in der die Skalierungsparameter enthalten sind (`scalefile`), der Wert für γ und für C. Weiterhin wird vermerkt, welche Merkmale (`usedFeatures`) bei der Klassifikation verwendet wurden und für welche Klassen (`classes`) ein Training durchgeführt wurde. Das Ergebnis der Sensitivitätsanalyse wird zur raschen Anzeige über das Webinterface als Grafik (`sensitivitygraphics`) für jedes Modell gespeichert. Zudem wird gespeichert, welche Features im Rahmen der Faktorenanalyse als besonders wichtig eingeschätzt wurden (`importantFeatures`).

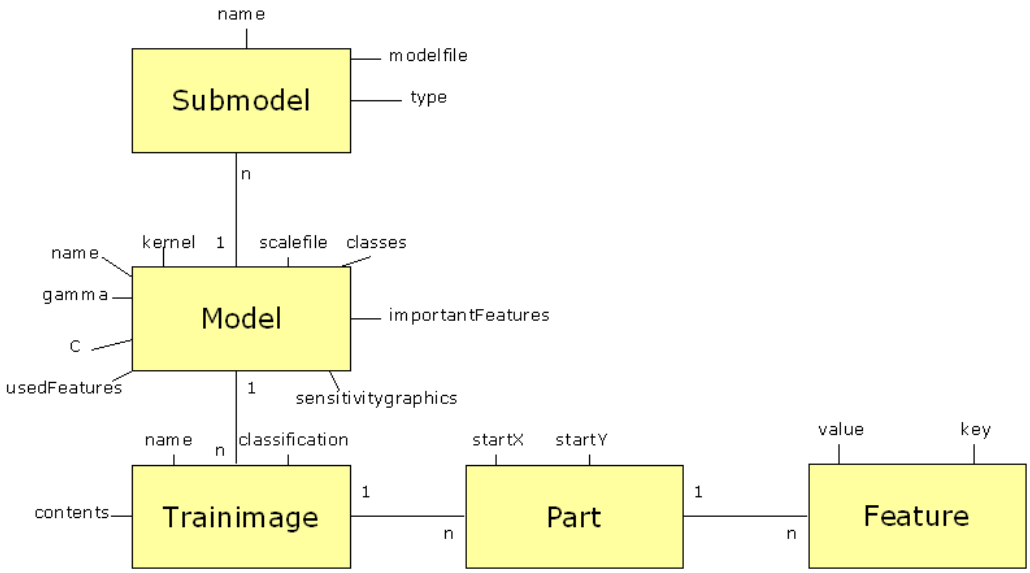

Abbildung 5.2: Datenmodell

Ein **Submodell** ist Teil eines Modells und bildet das Training einer konkreten SVM ab. Das Attribut `type` gibt an, um welche SVM es sich dabei handelt: 1 steht hier für SVM-Light, 0 für LIBSVM. Da beide SVM's dateibasiert arbeiten, muss die jeweilige Modelldatei gespeichert und beim Start der Webanwendung ins Filesystem geladen werden. Der Inhalt dieser Datei wird als BLOB in dem Attribut `modelfile` gespeichert, der Name der Datei in dem Attribut `name`.

Trainingsbilder (`Trainimage`) sind die Bilder, deren Daten für das Training eines bestimmten Modells verwendet wurden. Ein Trainingsbild wird daher immer mit dem Verweis auf das jeweilige Modell abgespeichert. Trainingsbilder werden als homogene Strukturen vorausgesetzt, daher kann pro Bild nur *eine* Klassifikation (`classification`) angegeben werden. Die Inhalte des Trainingsbildes werden im Attribut `contents`, der Name im Attribut `name` gespeichert. Sollen später auch heterogene Bilder erfasst werden, so bietet sich die Speicherung des Typs eines Teilchens im Rahmen der Feature-Tabelle als Key-Value-Struktur an (siehe unten).

Ein Trainingsbild besteht aus beliebig vielen **Teilchen** derselben Klasse. Für jedes Teilchen wird ein Verweis auf das zugehörige Bild gespeichert, zudem eine Startkoordinate (`startX, startY`), die das Pixel des Teilchens referenziert, das am weitesten oben links

liegt.

Ein Teilchen wird durch einen Merkmalsvektor charakterisiert. Um die Anzahl der Merkmale (**Features**) flexibel zu halten, wurden sie nicht als feste Attribute eines Teilchens modelliert, sondern als eigene Entities, die durch die beiden Attribute `key` und `value` näher bestimmt werden. Somit lassen sich pro Teilchen beliebig viele Kenngrößen und - im Falle von heterogenen Bildern - auch der Teilchentyp speichern. Gerade im Hinblick auf spätere Weiterentwicklungen, bei denen vielleicht mehr oder weniger und andere Merkmale berechnet werden sollen, schien ein solches flexibles Modell wünschenswert.

5.2 Software-Architektur

Die Software-Struktur lässt sich in mehrere Teile untergliedern:

- Das Paket `common` enthält Klassen, die von der gesamten Anwendung verwendet werden. Dies betrifft insbesondere den Speicherzugriff (`Storage, ModelStorage, DBStorage und FileStorage`) und das Logging der Anwendung (`Log`).
- Die anwendungsspezifischen Ausnahmebehandlungen stellt das Paket `exceptions` bereit.
- Das Paket `web` enthält JavaBeans (`RequestBean, ResultBean`, etc.), die in den JSP-Seiten eingebunden werden, und andere spezifische Klassen für die Weboberfläche.
- Die Umwandlung der Bilder in das binäre Format übernimmt das Paket `analyzer`.
- Die Pakete `parts` und `classifier` stellen die eigentliche Programmlogik bereit und sollen daher etwas genauer vorgestellt werden.

Einige der Pakete nutzen neben der Standard-Library und den schon erwähnten SVM-Komponenten externe Bibliotheken. Im einzelnen sind dies:

- Servlet-Klassen von Oreilly[1] (cos.jar)
- Java Imaging Utilities zur Konvertierung der Bildformate[2] (jiu.jar)
- Java Matrix Package zur Matrizenrechnung im Rahmen der Faktorenanalyse[3] (jama.jar)

[1] http://servlets.com/cos/
[2] http://jiu.sourceforge.net/
[3] http://math.nist.gov/javanumerics/jama/

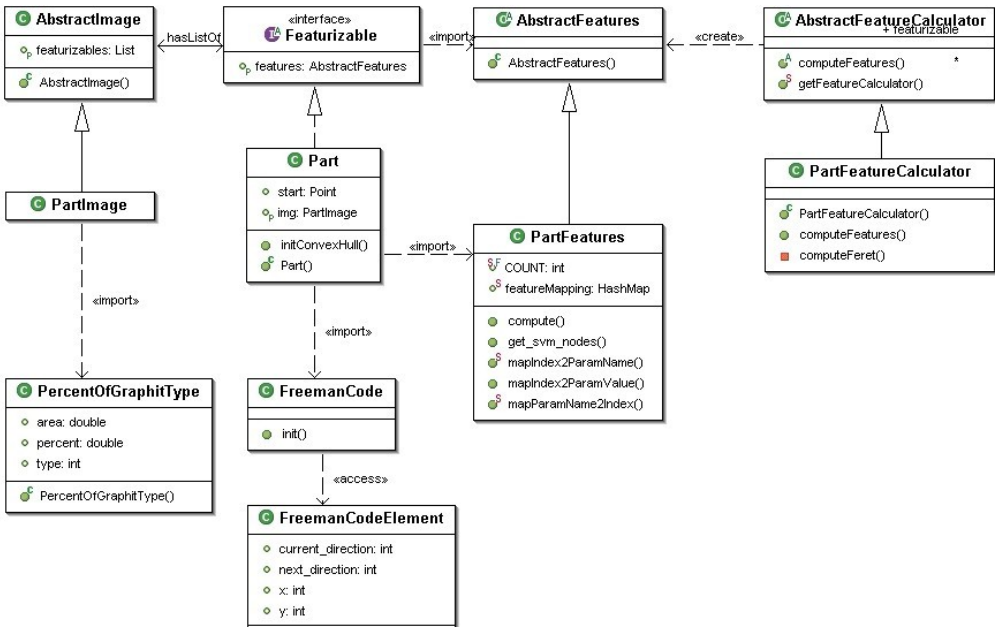

Abbildung 5.3: Das Paket `parts`

5.2.1 Klassenhierarchie

Die Klassenhierarchie sollte insbesondere die Möglichkeit der Erweiterbarkeit bieten. Daher wurde soweit möglich mit abstrakten Klassen und Interfaces gearbeitet, die verschiedene konkrete Instanziierungen erlauben. Dies sollen die folgenden beiden Beispiele erläutern.

Das Paket `parts`

Abbildung 5.3 zeigt einen Überblick über die Klassenhierarchie des Pakets `parts` in UML-Notation. Die einzelnen Klassen sind dabei mit ihren wichtigsten Methoden und Attributen sowie mit den wesentlichen Beziehungen untereinander dargestellt. Ein `AbstractImage` ist ein Bild, das nach bestimmten Gesichtspunkten analysiert werden soll. Im vorliegenden Fall sind das die einzelnen Teilchen (`Part`); allgemein könnten es jedoch auch andere Objekte des Bildes sein, sofern sie anhand ihrer Merkmale zu klassifizieren sind. Sie leiten sich daher von der abstrakten Klasse `Featurizable` ab. Die Merkmale der Objekte sind zusammengefasst in der Klasse `AbstractFeatures`. Da zur Berechnung einige komplexe Schritte notwendig sind, wurde hier noch eine Klasse eingefügt, die speziell dafür zuständig ist (`AbstractFeatureCalculator`).

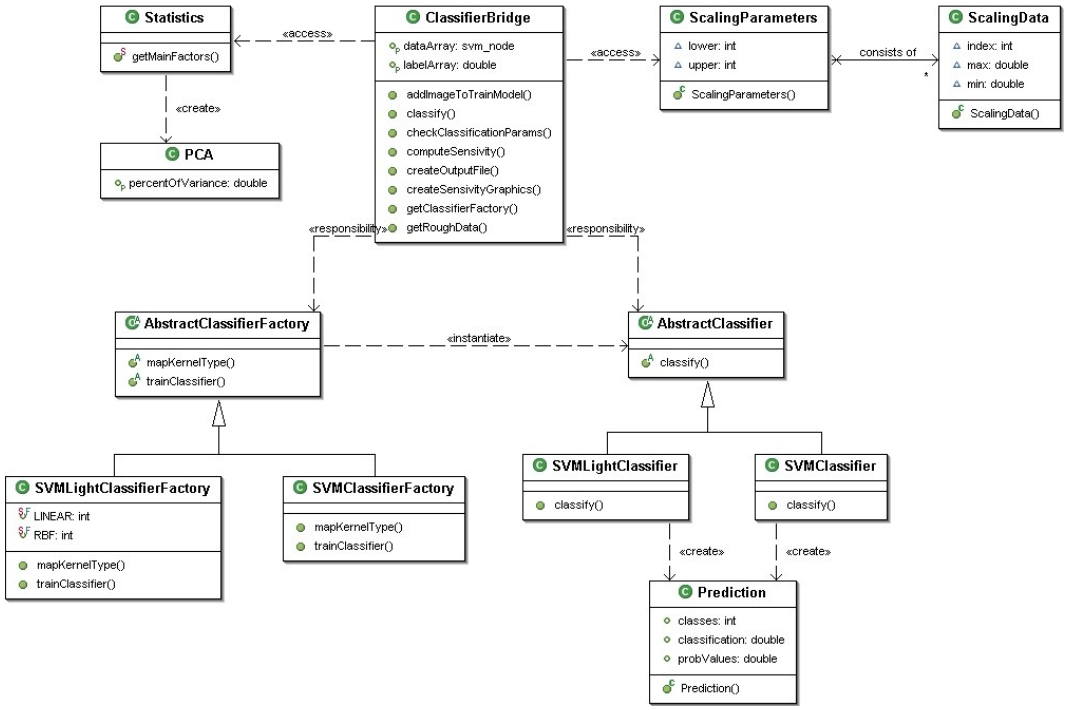

Abbildung 5.4: Das Paket `classifier`

Wenn ein Bild teilchenbasiert analysiert wird, dann handelt es sich um ein `PartImage`. Es enthält eine Liste von Teilchen, und jedes dieser Teilchen wird charakterisiert durch seine `PartFeatures`, die ihrerseits von einem `PartFeatureCalculator` berechnet werden. Ein Teilchen enthält jedoch neben den Merkmalen, die in die Klassifikation eingehen, auch einige Grundparameter, die vorab berechnet werden müssen. Dazu gehört z.B. der sog. `Freeman-Code` (vgl. Kapitel 2.2.1), der hier als eigene Klasse modelliert wurde.

Nach der Klassifikation eines Bildes verfügt es über Angaben, zu wieviel Prozent welcher Graphittyp in dem Bild vorkommt - diese Information kapselt die Klasse `PercentOf-GraphitType`.

Das Paket `classifier`

Im Zentrum des Paketes steht die sog. `ClassifierBridge` (vgl. Abbildung 5.4). Sie ist zuständig für die globale Kontrolle der Klassifikation, bzw. des Trainings. Sie stellt über ihre öffentlichen Methoden die Brücke zum Webinterface dar. Die `ClassifierBridge` koordiniert dabei drei grundlegende Funktionalitäten:

Einbindung der SVM's Die `ClassifierBridge` regelt die Ansteuerung der verschiedenen SVM's für den Trainings- und den Klassifkationsprozess. Die `AbstractClassifierFactory` erzeugt - abhängig vom gewählten Typ - einen neuen Klassifikator durch ihre Methode `train(...)`, die in den konkreten Unterklassen realisiert ist. Dieser Prozess entspricht dem Training einer SVM. Ein Klassifikator leitet sich von der abstracten Klasse `AbstractClassifier` ab und implementiert deren abstracte Methode `classify`. Ergebnis einer Klassifikation ist ein Objekt der Klasse `Prediction`, die sowohl den ermittelten Typ als auch Entscheidungswerte für alle bei der Klassifikation getesteten Graphittypen enthält. Diese Werte geben Aufschluss darüber, mit welcher Wahrscheinlichkeit ein Teilchen einer bestimmten Klasse zugeordnet wurde oder nicht. Durch die Abstraktion des Klassifikators von seiner konkreten Realisierung können beliebige andere Klassifikatoren eingebunden werden, die sich von denselben abstrakten Klassen ableiten.

Skalierung der Daten Die `ClassifierBridge` sorgt außerdem für die Skalierung der (Trainings- und Klassifikations-) Daten. Hierzu bedient sie sich der Klassen `ScalingParameters`, die neben der Unter- und Obergrenze des Zielintervalls (standardmäßig -1 und +1) eine Liste von `ScalingData` enthält: Dort wird das konkrete Minimum und Maximum für ein bestimmtes Merkmal gespeichert.

Sensitivitätsanalyse Nach jedem erfolgreichen Training startet die `ClassifierBridge` die Sensitivitätsanalyse der Daten (vgl. Kapitel 4). Hierzu verwendet sie die Klasse Statistics, die statische Methoden zur Berechnung der Hauptkomponentenanalyse und der Rotation der Faktoren enthält. Das Ergebnis der Sensitivitätsanalyse ist ein Objekt der Klasse `PCA`.

5.2.2 Erweiterbarkeit und Konfigurierbarkeit

Die Anwendung soll sowohl auf Implementierungs- als auch auf Konfigurationsebene im Hinblick auf Änderungen und Erweiterungen möglichst flexibel sein. Generell arbeitet die Anwendung daher mit einer Konfigurationsdatei (`config.properties`), die beim Start der Applikation in den Speicher geladen wird und danach in Form von `java.util.Properties` zur Verfügung steht. Die wichtigsten Aspekte sollen im Folgenden kurz beschrieben werden.

Hinzufügen neuer Graphitklassen Die Auswahl der beim Training berücksichtigten Klassen ist auf Konfigurationsebene möglich. Dies geschieht in der Konfigurationsdatei aufgrund folgender Parameter:

```
classifier.parts.available_classes = 1,3,4,6
```

In dieser Zeile wird zunächst über ganzzahlige Bezeichner angegeben, welche Klassen generell zur Verfügung stehen. Für jeden dieser Bezeicher muss dann ein Name konfiguriert werden, der in der Webanwendung sichtbar ist (ein sog. *Nice-Name*):

```
classifier.parts.available_classes.1 = Typ I
classifier.parts.available_classes.3 = Typ III
classifier.parts.available_classes.4 = Typ IV/V
classifier.parts.available_classes.6 = Typ VI
```

Damit die Teilchen aufgrund der für sie erzielten Klassifikation unterschiedlich eingefärbt werden, muss zudem für jede Klasse ein hexadezimaler RGB-Farbwert definiert werden:[4]

```
classifier.parts.color.1 = F58DDB
classifier.parts.color.3 = 0520A5
classifier.parts.color.4 = FFFF05
classifier.parts.color.6 = FEBF28
```

Konfiguration der Anwendungsumgebung, Auswahl der SVM's und Administrator-Modus Um die Portierbarkeit der Anwendung zu gewährleisten wurden neben den Datenbankparametern auch alle Pfade und betriebssystemspezifische externe Programme (z.B. SVM-Light) über Konfigurationsparameter eingebunden. Das Wurzelverzeichnis der Anwendung wird über folgenden Parameter referenziert:

```
common.AppInit.home = C:\\tomcat\\webapps\\classifier_neu\\
```

Alle anderen Verzeichnisse (Modelle, Benutzeranfragen, Klassifikationsergebnisse) sind dazu relativ konfigurierbar, z.B.

```
common.AppInit.uploadDir = upload\\
```

Die gewählte Form der Modellspeicherung (Datenbank oder Filesystem) wird über den Parameter

```
common.AppInit.ModelStorage.DB = 0
```

[4]Eine Auswahl von Beispielfarben enthält die begleitende Dokumentation der Anwendung, so dass die Konfiguration auch für Nicht-Informatiker kein Problem darstellen sollte.

konfiguriert. Eine 0 sagt hier aus, dass Modelle aus dem Dateisystem geladen werden, eine 1 steht für die Anbindung der Datenbank. Ob die globale Analyse der Merkmale über alle Teilchen angeboten werden soll, steuert der Parameter:

```
common.AppInit.FeatureAnalysis = 0
```

Für die erzeugte Excel-Datei im .csv-Format können das landesübliche Zahlenformat und der Separator der Daten über folgende Parameter gesteuert werden:

```
common.locale = de
clasifier.excel.separator = ;
```

Die verwendeten SVM's (LIBSVM und SVM-Light) können über folgende Parameter aktiviert oder deaktivert werden (Eine 0 steht hier für die Deaktivierung, die 1 für die Aktivierung der entsprechenden SVM):

```
classifier.parts.svmlight = 0
classifier.parts.svmlib = 1
```

Für SVM-Light müssen zudem die verwendeten Binaries (für Training und Klassifikation) referenziert werden:

```
classifier.parts.svmlight.bin.classify = ressources\\svmlight\\svm_classify
classifier.parts.svmlight.bin.train = ressources\\svmlight\\svm_learn
```

Die Logausgaben der Anwendung (hier ein datumsgenaues Logfile, der verwendete Logger, sowie eine Granularität für die Logausgaben) werden über folgende Zeilen konfiguriert:

```
common.Log.logfile = log\\log_%d.log
common.Log.logger = java.util.Logger
common.Log.level = FINEST
```

Normalerweise können schon erstellte Modelle vom Benutzer nicht wieder gelöscht werden. Dies schien nicht sinnvoll, da die Anwendung ja über Internet und vorläufig ohne abgestufte Benutzerrechte zugreifbar ist. Um dennoch Modelle löschen zu können, die vielleicht irrtümlich erstellt wurden oder veraltet sind, kann die Anwendung im Administratormodus gestartet werden. Hierfür wird folgender Parameter gesetzt:

```
common.AppInit.adminMode = 1
```

Nach einem Neustart der ServletEngine stehen nun in der Modellübersicht Buttons zum Löschen zur Verfügung.

Erweiterung der Implementierung um neue teilchenbasierte Kenngrößen Sollen bei der Klassifikation neue Merkmale berücksichtigt werden, so ist dies nicht ohne Implementierungsaufwand zu lösen: Die Berechnung der Merkmale muss in die entsprechende Klasse integriert werden. Als möglichst einfaches Vorgehen bietet die Anwendung hier folgende Möglichkeit an:

- Hinzufügen des Parameters in Form einer Ganzzahl und String-Konstante in der Klasse PartFeatures
- Inkrementierung der Konstante COUNT (Gesamtanzahl der Merkmale) in der Klasse PartFeatures
- Ergänzen des Merkmals in den entsprechenden Mappings der Klasse PartFeatures und im Layout des WebInterface (training_1.jsp und classification_1.jsp)
- Implementierung der Methode zur Berechnung des Merkmals in der Klasse PartFeatureCalculator und Aufruf in dessen Methode computeFeatures

Die meisten der Konfigurationsparameter können in ihren Default-Einstellungen übernommen werden, so dass sich der Administrationsaufwand in Grenzen hält. Lediglich die Datenbankparameter sowie der absolute Pfad zur Anwendung müssen einmalig gesetzt werden.

Kapitel 6

Experimentelle Ergebnisse

In diesem Kapitel werden das Web-Interface des Prototypen *POCA* (**P**art-**O**riented **C**lassification and **A**nalysis) sowie der Aufbau und die Ergebnisse der durchgeführten Tests beschrieben. Einige Bilder, die im Rahmen dieser Tests entstanden, sind im Anhang der Arbeit beigefügt.

6.1 Das Web-Interface

Das Web-Interface von *POCA* bietet neben Klassifikation und Training auch die Möglichkeit, eine globale Analyse über alle Parameter und alle Teilchen eines Typs durchzuführen. Weiterhin wurden Übersichtsseiten integriert, die dem Benutzer die Eigenschaften der Modelle und der verwendeten Parameter erläutern. Die genaue Bedienung des Web-Interfaces wird hier nur sehr verkürzt dargestellt. Der interessierte Anwender sei aber auf das Benutzerhandbuch verwiesen, das über die Webanwendung erreichbar ist.

Training Abbildung 6.1 zeigt den Aufbau der Webseite, über die der Benutzer ein neues Modell anlegen kann. Im oberen Teil der Seite wird zunächst ein Name für das Modell festgelegt. Anschließend werden die Bilder für das Training hochgeladen, jeweils mit der Angabe, welchen Teilchentyp das Bild enthält und mit welcher Pixelgröße es erstellt wurde. Wurden auf diese Weise alle Bilder hochgeladen, so können (optional) die zu berücksichtigenden Formparameter und die SVM-Parameter festgelegt werden. Anschließend wird das Training über den Button *Start Training* gestartet. Der Button *Reset* setzt alle Angaben für das Training zurück.

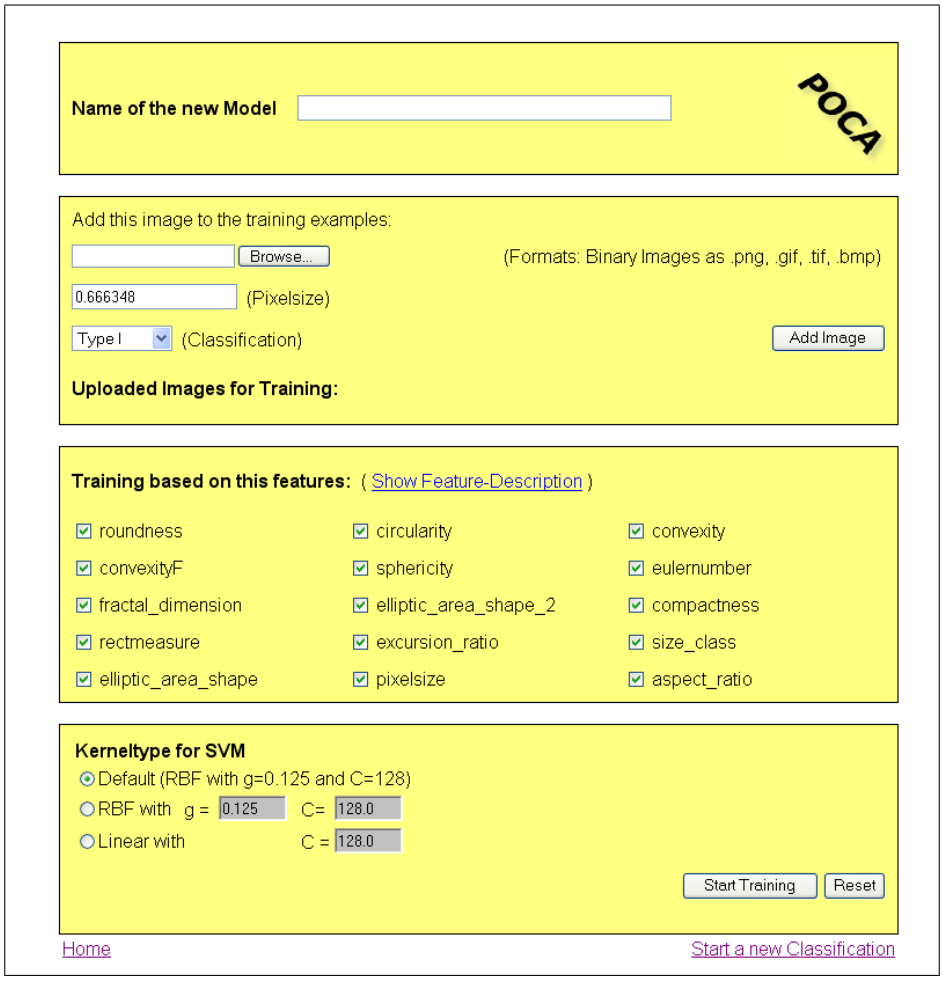

Abbildung 6.1: Trainingsseite von *POCA*

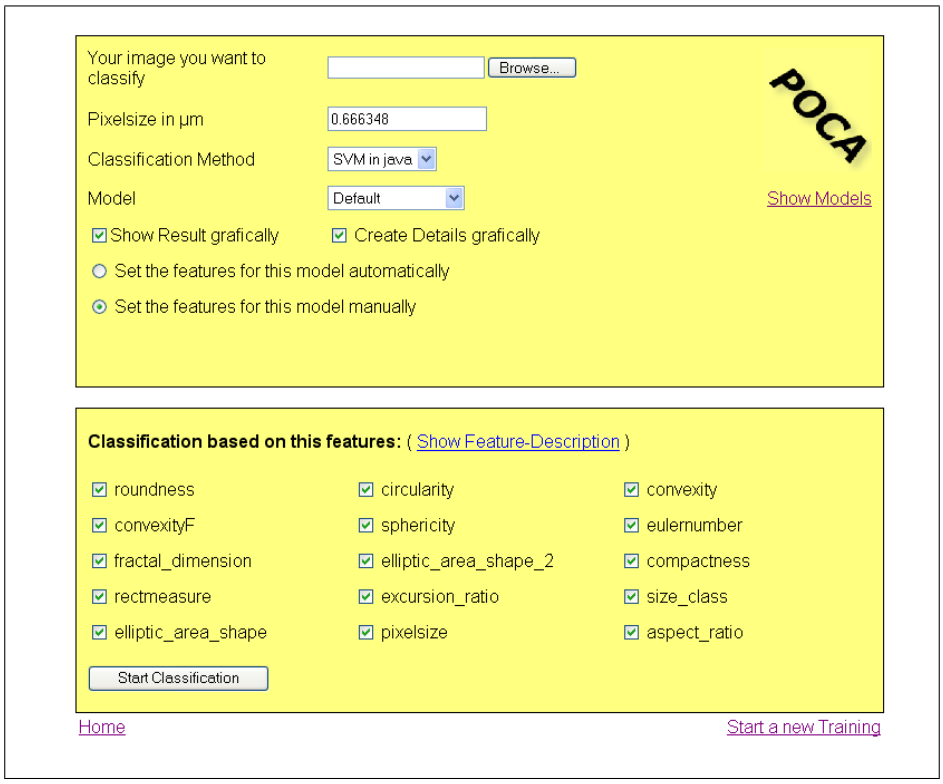

Abbildung 6.2: Klassifikationsseite von *POCA*

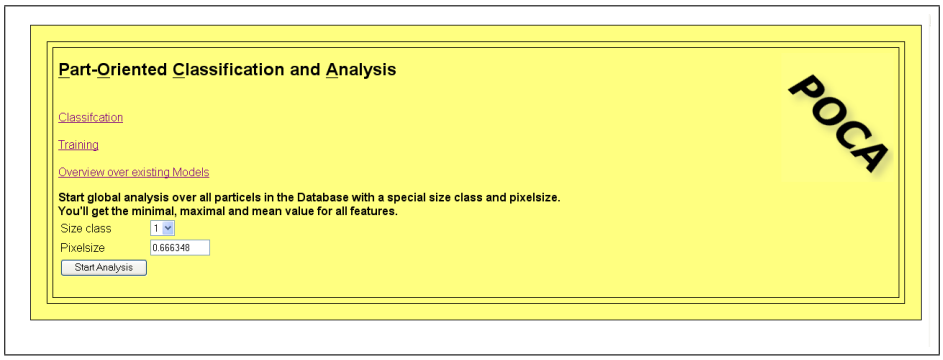

Abbildung 6.3: Startseite von *POCA* mit der globalen Featureanalyse

Klassifikation Wurde auf diese Weise mindestens ein Modell erstellt, so können Bilder anhand dieses Modells klassifiziert werden. Abbildung 6.2 zeigt die entsprechende Seite, über die der Benutzer ein Bild hochladen und die Klassifikation starten kann. Ergänzend kann hier angegeben werden, welche SVM-Bibliothek verwendet und welche der Klassifikationsparameter berücksichtigt werden sollen. Bezüglich der Darstellung kann zudem ausgewählt werden, ob das Ergebnis grafisch und mit Details für alle Teilchen präsentiert werden soll. Wird dies nicht gewählt, so erhält der Benutzer nur eine textuelle Information, wieviel Prozent von jedem Graphittyp in dem Bild enthalten sind. Ergänzend kann in beiden Fällen eine Übersicht über alle Teilchen und deren Parameter im .csv-Format heruntergeladen werden.

Die Startseite von *POCA* und die globale Featureanalyse Die Startseite von *POCA* bietet neben den Links zu Klassifikation und Training im unteren Teil auch die globale Featureanalyse an (vgl. Abbildung 6.3). Hier kann für Teilchen einer bestimmten Richtzahl und für Bilder einer bestimmten Pixelgröße eine Auswertung der einzelnen Parameter vorgenommen werden (Minimum, Maximum, Mittelwert und Standardabweichung).

Übersicht über die Modelle und Features Alle Seiten der Anwendung enthalten Links zu den Übersichten über die Modelle und Parameter. Abbildung 6.4 zeigt eine solche Modellübersicht, in der die Modelle mit ihren jeweiligen Eigenschaften vermerkt sind. Über einen Link ist zudem das grafische Ergebnis der Sensitivitätsanalyse erreichbar (vgl. Abbildung 6.5). Die Übersicht über die verwendeten Parameter wurde der Anwendung als pdf-Dokument beigefügt.

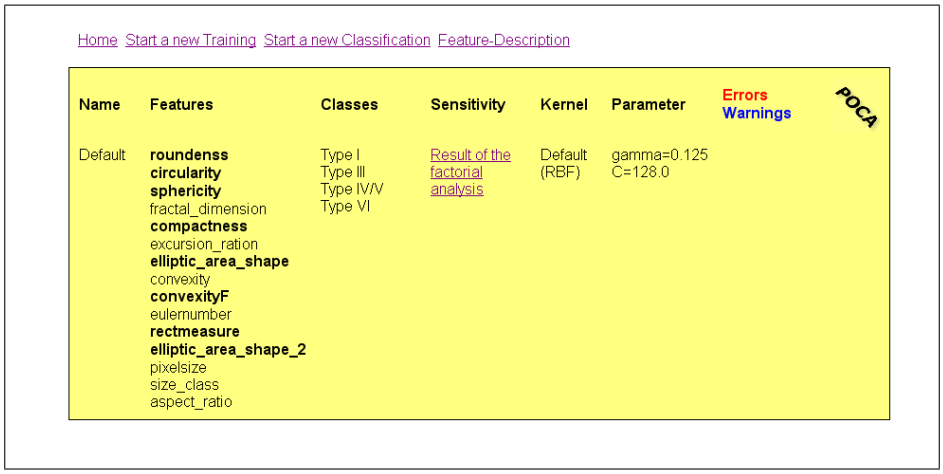

Abbildung 6.4: Modelübersicht von *POCA*

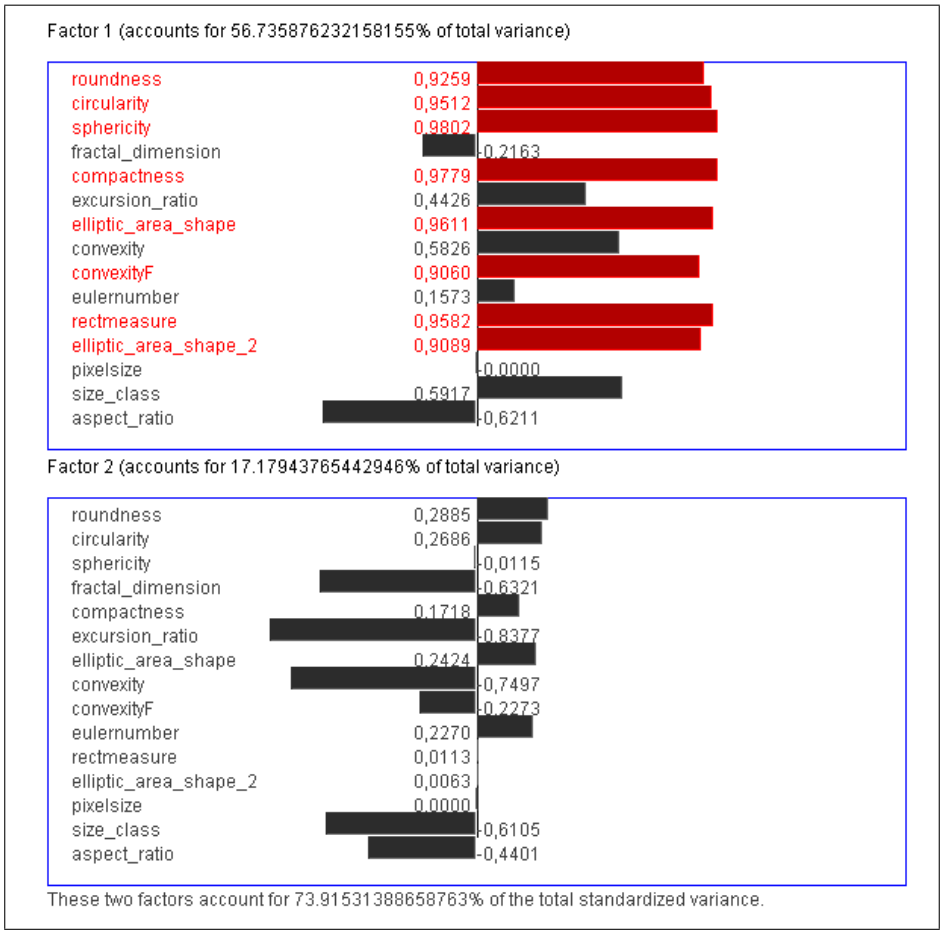

Abbildung 6.5: Sensitivitätsanalyse: zwei Hauptfaktoren mit ihren Ladungen aller Parameter

6.2 Test- und Trainingsdaten

Das Training der SVM's wurde mit homogenen Bildern durchgeführt, d.h. mit Bildern, die jeweils nur *einen* Teilchentyp enthielten. Da Gefügebilder üblicherweise nicht derart homogen sind, wurden vorab Teilchen manuell aus dem Bild herausgefiltert. Als Trainingsdaten standen danach zur Verfügung:

- 12 Bilder mit Teilchen von Typ I
- 10 Bilder mit Teilchen von Typ III
- 5 Bilder mit Teilchen von Typ IV/V
- 11 Bilder mit Teilchen von Typ VI

Zum Testen des Klassifikators dienten zusätzlich 40 heterogene Gefügebilder, d.h. Bilder, in denen mehr als eine Teilchenart enthalten ist. Um ihre Klassifikationsergebnisse beurteilen zu können, wurden sie mit den Ergebnissen eines anderen, in der Praxis erprobten Klassifikators - im Folgenden Vergleichsklassifikator genannt - verglichen. Dieser Vergleichsklassifikator entstand am Lehrstuhl für Funktionswerkstoffe (Prof. Dr. F. Mücklich) der Universität des Saarlandes. Er wurde dort unter Verwendung des Bildanalyseprogramms *ImageC* experimentell erstellt[1]. Dieser Klassifikator arbeitet regelbasiert mit zwei Formparametern, Rundheit und Kompaktheit, sowie der Richtzahl. Dabei wurden anhand der idealisierten Richtreihenbilder der DIN EN 945 für die verschiedenen Graphit-Typen bestimmte Zonen aus Mittelwert und Standardabweichung für Rundheit und Kompaktheit berechnet. Aufgrund dieser Wertekombination kann unter Beachtung der Richtzahl eine Klasse vorhergesagt werden. Die Ergebnisse des Vergleichsklassifikators lagen in Form eingefärbter Gefügebilder vor (vgl. Abschnitt 7).

Doch war auf diesem Wege keine exakte Bewertung der Klassifikationsgüte möglich. In der Arbeitsweise des Vergleichsklassifikator waren Fehler nicht ausgeschlossen. Zudem lagen die Ergebnisse in rein visueller Form vor, so dass ein exakter Vergleich fehleranfällig und zeitaufwendig gewesen wäre. Hier konnte es vorerst nur um einen ungefähren optischen Abgleich gehen - die exakte Beurteilung der Klassifikationsgüte der vorliegenden Anwendung im Hinblick auf heterogene Bilder ist somit ein wichtiger Forschungsgegenstand für die Zukunft. Um die Klassifikationsgüte dennoch beurteilen zu können, wurde der Klassifikator auf die homogenen Trainingsbilder angewandt. Diese waren von einem Spezialisten

[1]Leider gibt es noch keine offizielle Referenz zu diesem Klassifikator - Informationen sind jedoch erhältlich am Lehrstuhl für Funktionswerkstoffe (Prof. Dr. F. Mücklich) der Universität des Saarlandes

erstellt worden, so dass eine gute Qualität der Klassifikation vorausgesetzt werden konnte. Zudem war der Abgleich der Teilchen durch die Homogenität der Bilder einfacher. Das genaue Testszenario dazu wird in Absatz 6.4 beschrieben.

6.3 Erstellte Modelle

Zum Testen wurden zunächst mit *POCA* drei verschiedene Modelle erstellt, deren Eigenschaften im Folgenden kurz vorgestellt werden.

Das Modell *Default* Dieses Modell basiert auf ca. 3-4 Bildern pro Typ. Für Typ I gingen 1740, für Typ III 1117, für Typ IV/V 706 und für Typ VI 617 Teilchen in die Klassikfikation ein. Die unterschiedliche Anzahl der Teilchen pro Typ ergibt sich aus der Verschiedenartigkeit und der Verfügbarkeit der Bilder[2]. Das Modell verwendet die Default-Einstellungen für die SVM's (RBF-Kernel mit $\gamma = 0.125$ und $C = 128$) und berücksichtigt alle verfügbaren Klassifikationsparameter (vgl. Kapitel 2.3).

Das Modell *Linear* Dieses Modell wurde anhand derselben Testdaten, aber mit veränderten Einstellungen für die SVM's erstellt: Im Gegensatz zum *Default*-Modell wurde hier ein linearer Kernel mit $C = 1000$ verwendet. Auch hier wurden alle verfügbaren Parameter für das Training herangezogen.

Das Modell *NotAllFeatures* Bei diesem Modell wurden die Klassifikationsparameter auf diejenigen reduziert, die sich im Rahmen der Sensitivitätsanalyse des *Default*-Modells als besonders wichtig erwiesen hatten (Kreisform, Rundheit, Formfaktor, Kompaktheit, Konvexität der Fläche, Elliptischer Formfaktor 1 und 2 und das Rechteckmaß). Hier wurde also die Frage untersucht, inwieweit diese Hauptparameter alleine für eine korrekte Klassifikation ausreichend sind. Testdaten und SVM-Einstellungen waren dieselben wie bei dem *Default*-Modell.

Das Modell *NotAllFeatures2* Bei diesem Modell wurden die Klassifikationsparameter auf diejenigen reduziert, die sich im Rahmen der Sensitivitätsanalyse des *Default*-Modells als nicht wichtig erwiesen hatten (Konvexität des Umfangs, Fraktale Dimension, Eulerzahl,

[2]So waren z.B. sehr unterschiedliche Ausprägungen von Typ I, die sinnvollerweise in das Training einbezogen werden sollten, auf Bildern, die sehr viele Teilchen enthielten. Dagegen lagen von Typ IV/V insgesamt nur 740 Teilchen vor, von denen 34 aufgrund ihres zu geringen Durchmessers nicht als Trainingsobjekte geeignet waren.

Streckung, Pixelgröße, Richtzahl und der Abweichungsfaktor). Im Zentrum der Untersuchung stand die Frage, wie sich der Klassifikator bei einer Verringerung der Dimensionalität verhält, wenn genau die Dimensionen mit den größten Varianzen wegfallen.

6.4 Bewertung der Klassifikationsgüte

Zur Bewertung der Klassifikationsgüte wurden homogene Trainingsbilder verwendet. Dabei wurde soweit möglich auf Bilder zurückgegriffen, die nicht für das Training verwendet worden waren. Für Typ IV/V lag jedoch eine zu geringe Anzahl an homogenen Bilder vor, so dass hier Trainings- und Klassifikationsbilder identisch sind. Klassifiziert wurden 800 - 950[3] Teilchen je Klasse mit unterschiedlichen Modellen und Klassifikationsparametern. Dabei wurde in jedem Testlauf protokolliert, wieviele Teilchen den einzelnen Klassen zugeordnet wurden und welcher Klasse das Bild angehört (d.h. welches die *richtige* Klasse gewesen wäre). Ergänzend wurde auch festgehalten, wieviel Prozent des Bildvordergrundes den einzelnen Klassen zugeordnet wurde: Diese flächenorientierte Betrachtungsweise trägt der Tatsache Rechnung, dass die Fehlklassifizierung von sehr kleinen Teilchen aus werkstoffwissenschaftlicher Sicht weniger problematisch ist als die von großen Teilchen. Entscheidend für die Qualität des Materials ist der prozentuale Flächenanteil des Graphittyps. Da sich diese Untersuchung in den vorliegenden Beispielen weitgehend mit der teilchenorientierten Betrachtungsweise deckt, wird hier auf eine ausführliche Darstellung verzichtet. Die Ergebnisse sind jedoch überblicksartig in Abbildung 6.11 enthalten.

Aus diesen Fakten wurde für jede Klasse und jedes Testszenario die sog. Präzision (precision), Ausbeute (recall) und das F1-Maß berechnet. Dies sind gängige Maße zur Bewertung der Klassifikationsgüte. Sie beruhen auf der folgenden Kontingenztafel für einen Klassifikator h.

	Reale Klasse $y = +1$	Reale Klasse $y = -1$
Vorausgesagte Klasse $h(\vec{x}) = +1$	f^{++}	f^{+-}
Vorausgesagte Klasse $h(\vec{x}) = -1$	f^{-+}	f^{--}

[3]Eine exakte Übereinstimmung der Teilchenzahl jedes Typs ist aufgrund der unterschiedlich strukturierten Bilder kaum zu erreichen.

Dabei sind f^{++} die positiven Trainingsbeispiele einer Klasse, die der Klassifikator auch als positiv klassifiziert hat. Analog sind f^{--} alle negativen Beispiele, die korrekt als negativ erkannt wurden. f^{+-} sind die sog. *false positives*, d.h. negative Trainingsbeispiele, die der Klassifikator irrtümlicherweise positiv klassifiziert hat. f^{-+} sind analog die *false negatives*, d.h. positive Trainingsbeispiele, die nicht als solche erkannt wurden.

Definition 33. Die *Präzision (precision)* eines Klassifikators h gibt an, wieviel Prozent der durch h positiv klassifizierten Beispiele tatsächlich korrekt klassifiziert wurden.

$$Prec(h) = \frac{f^{++}}{f^{++} + f^{+-}}$$

Definition 34. Die *Ausbeute (recall)* eines Klassifikators h gibt an, wieviel Prozent der positiven Beispiele auch wirklich gefunden wurden.

$$Rec(h) = \frac{f^{++}}{f^{++} + f^{-+}}$$

Definition 35. Unter dem *F1*-Maß versteht man das harmonische Mittel aus Precision und Recall:

$$F1(h) = \frac{2}{\frac{1}{Prec(h)} + \frac{1}{Rec(h)}}$$

In den folgenden Absätzen wird Testaufbau und Ergebnis für die einzelnen Testläufe kurz beschrieben. Einen grafischen Vergleich von Präzision, Ausbeute und F1-Maß bzgl. der Teilchenanzahl für die einzelnen Graphittypen zeigen die Abbildungen 6.6 bis 6.9, ihre exakten Werte für alle Testläufe sind in Abbildung 6.11 zu sehen.

Testlauf 1: Das Modell Default Exemplarisch zeigt Abbildung 6.10 die Testdaten und Ergebnisse des Testlaufes mit dem Modell Default, wobei alle Klassifikationsparameter und die Bibliothek LIBSVM verwendet wurden. Dabei sind die Testdaten, die tatsächlich in die Auswertung eingingen, blau unterlegt. Um eine ungefähr gleiche Anzahl an Teilchen zu erhalten, musste für die Bilder vom Typ I eine Auswahl getroffen werden. Die Abbildungen 6.6 bis 6.9 zeigen einen Vergleich von Präzision, Ausbeute und F1 für die verschiedenen Testläufe und Graphittypen.

Testlauf 2: Das Modell Linear Der Testaufbau entsprach hier exakt dem von Testlauf 1, jedoch wurde anstelle des Modells Default das Modell mit dem linearen Kernel verwendet. Die Abbildungen 6.6 bis 6.9 zeigen, dass dies keine nennenswerten Auswirkungen auf die Klassifikationsgüte hat.

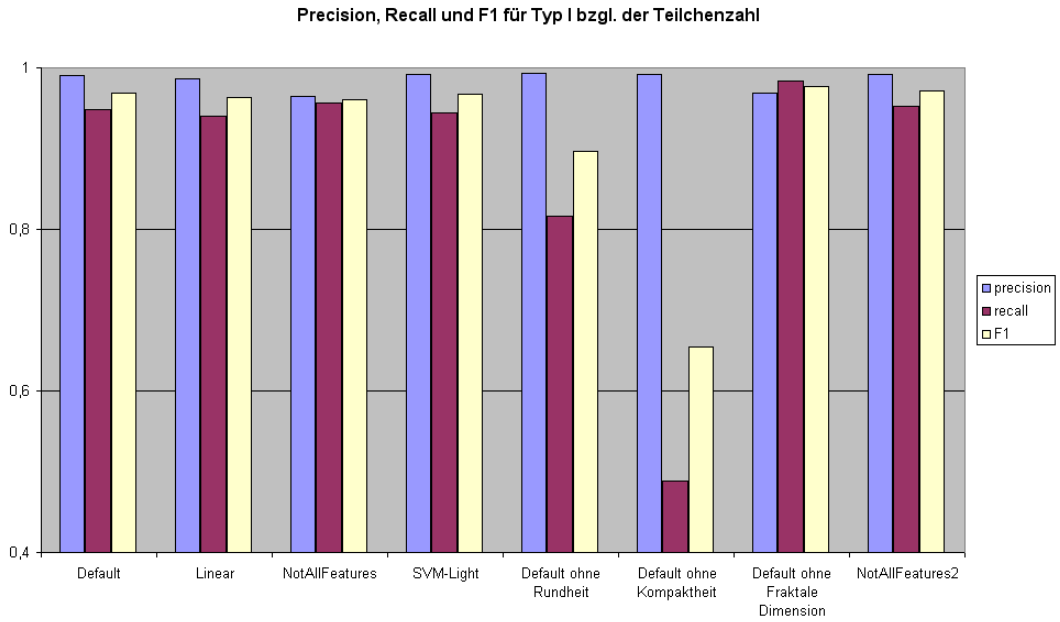

Abbildung 6.6: Präzision, Ausbeute und F1 für Typ I bzgl. der Teilchenzahl

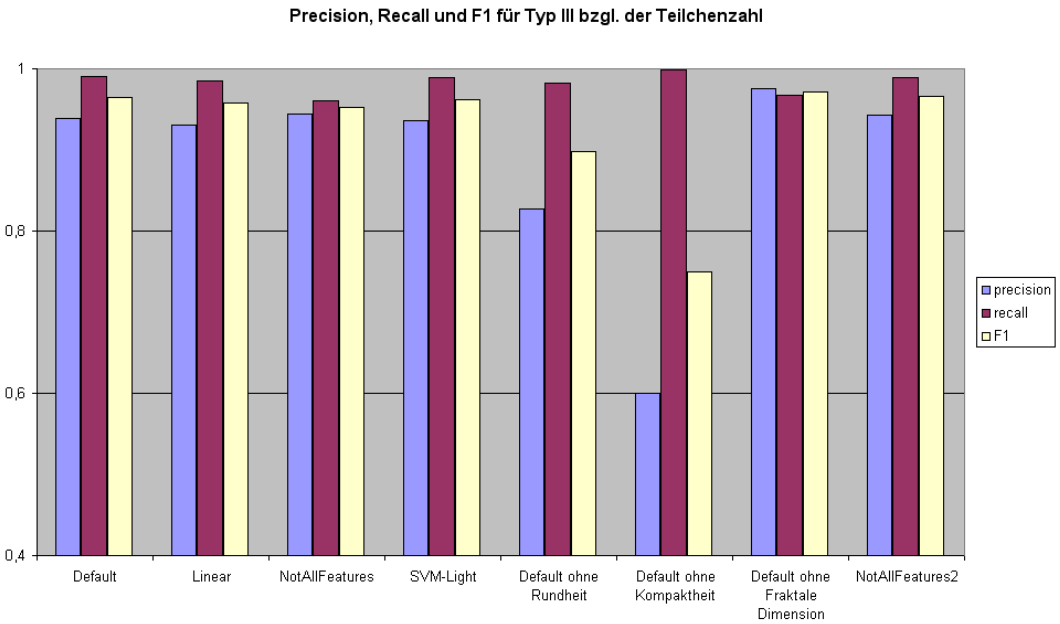

Abbildung 6.7: Präzision, Ausbeute und F1 für Typ III bzgl. der Teilchenzahl

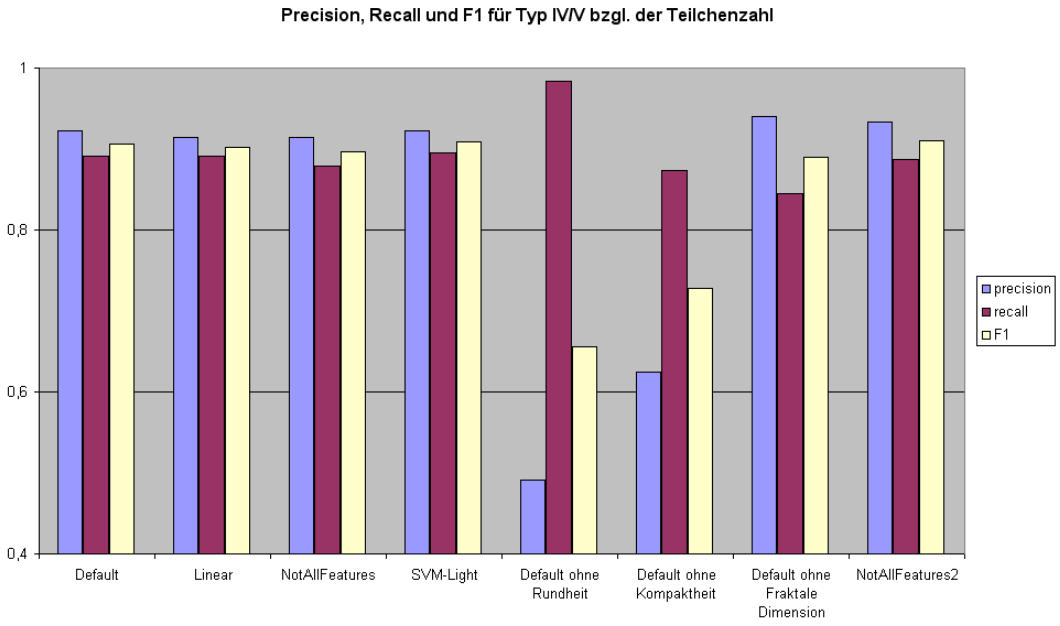

Abbildung 6.8: Präzision, Ausbeute und F1 für Typ IV/V bzgl. der Teilchenzahl

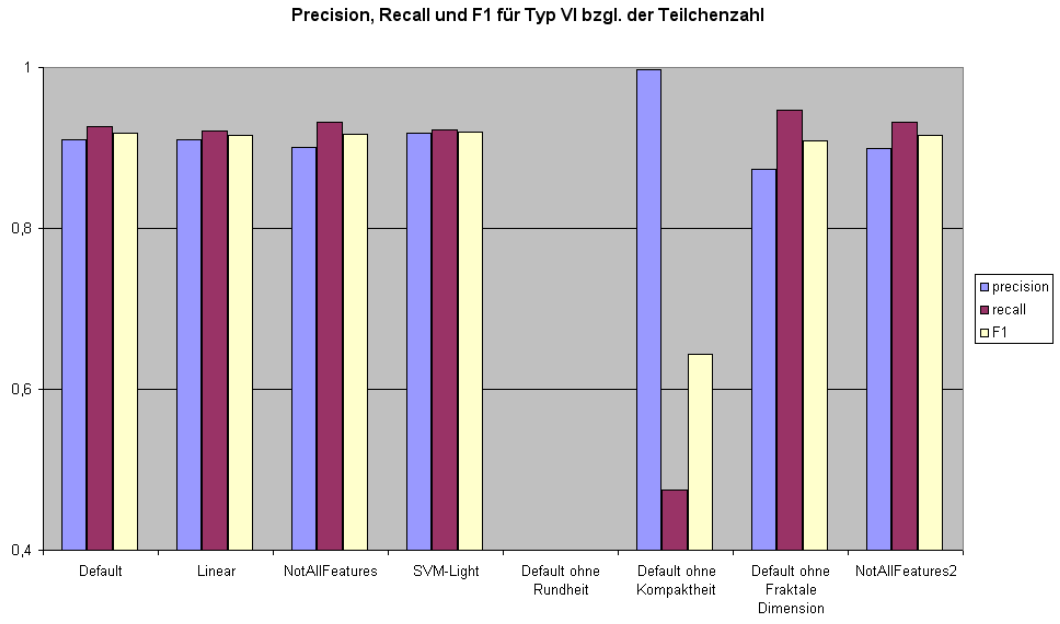

Abbildung 6.9: Präzision, Ausbeute und F1 für Typ VI bzgl. der Teilchenzahl

Modell Default (alle Parameter)											
			vom Klassifikator erkannte Typen					erkannter Bildvordergrund in Prozent			
Bildname	Teilchen gesamt	Bildtyp	Typ I	Typ III	Typ IV/V	Typ VI		Typ I	Typ III	Typ IV/V	Typ V
Typ_I_GJL_03.PNG	455	I	431	24	0	0		97,03	2,97	0	0
Typ_I_GJL_05.PNG	437	I	422	15	0	0		98,04	1,96	0	0
Typ_I_GJL_06.PNG	570	I	558	12	0	0		99	1	0	0
Typ_I_GJL_07.PNG	104	I	88	16	0	0		87,7	12,3	0	0
Typ_I_GJL_09.PNG	402	I	395	7	0	0		96,02	3,98	0	0
Typ_I_GJL_10.PNG	669	I	650	19	0	0		95,82	4,18	0	0
Typ_I_GJL_11.PNG	930	I	900	30	0	0		98,41	1,59	0	0
Typ_I_GJL_12.PNG	392	I	383	9	0	0		97,86	2,14	0	0
Summe Teilchen Typ I	951										
TYP_III_GJV_05.PNG	157	III	0	157	0	0		0	100	0	0
TYP_III_GJV_06.PNG	155	III	1	154	0	0		0,31	99,69	0	0
TYP_III_GJV_07.PNG	185	III	1	182	2	0		1,97	97,84	0,19	0
TYP_III_GJV_08.PNG	148	III	0	147	1	0		0	99,96	0,04	0
TYP_III_GJV_09.PNG	129	III	0	126	3	0		0	99,75	0,25	0
TYP_III_GJV_10.PNG	100	III	0	100	0	0		0	100	0	0
Summe Teilchen Typ III	874										
Typ_IV-V_GJS_01.PNG	227	IV/V	3	4	201	19		0,14	1,21	93,08	5,57
Typ_IV-V_GJS_02.PNG	196	IV/V	4	0	172	20		0,35	0	91,91	7,74
Typ_IV-V_GJS_03.PNG	96	IV/V	0	1	89	6		0	0,44	94,67	4,89
Typ_IV-V_GJS_04.PNG	151	IV/V	0	2	138	11		0	0,94	89,37	9,68
Typ_IV-V_GJS_05.PNG	126	IV/V	0	0	109	17		0	0	87,68	12,32
Summe Teilchen Typ IV/V	796										
Typ_VI_GJS_06.JPG	107	VI	0	0	24	78		0	0	14,52	85,48
Typ_VI_GJS_07.JPG	102	VI	0	0	4	98		0	0	2,02	97,98
Typ_VI_GJS_08.JPG	139	VI	0	0	5	134		0	0	0,97	99,03
Typ_VI_GJS_09.JPG	147	VI	0	0	7	140		0	0	1,68	98,32
Typ_VI_GJS_10.JPG	152	VI	0	0	6	146		0	0	1,67	98,33
Typ_VI_GJS_11.JPG	154	VI	0	0	8	146		0	0	1,12	98,88
Summe Teilchen Typ IV	801										
f++			902	866	709	742		282,59	597,24	456,71	578,02
f+-			9	56	60	73		2,77	20,00	22,46	40,2
f-+			49	8	87	59		17,41	2,76	43,29	21,98
precision			0,990121	0,939262	0,921977	0,910429		0,990292963	0,967598	0,953127	0,9349746
recall			0,948475	0,990847	0,890704	0,926342		0,941966667	0,9954	0,91342	0,9633667

Abbildung 6.10: Testdaten für das Modell Default

Testlauf 3: Das Modell NotAllFeatures In diesem Testlauf stand die Überprüfung der Faktorenanalyse im Vordergrund. Da das Modell NotAllFeatures nur mit den Klassifikationsparametern erstellt worden war, die sich im Rahmen der Faktorenanalyse als besonders aussagekräftig erwiesen hatten (Kreisform, Rundheit, Formfaktor, Kompaktheit, Konvexität der Fläche, Elliptischer Formfaktor 1 und 2 und das Rechteckmaß), durften die Ergebnisse bzgl. der Klassifikationsgüte nicht zu weit von der des Default-Modells abweichen. Dies belegen die Abbildungen 6.6 bis 6.9.

Testlauf 4: Das Modell Default ohne den Parameter der Rundheit Hierbei wurde von denselben Testdaten ausgegangen wie bei Testlauf 1. Jedoch wurde der Klassifikationsparameter der Rundheit nicht für die Klassifikation herangezogen. Abbildung 6.9 zeigt deutlich, dass damit die Unterscheidung von Typ IV/V und Typ VI unmöglich geworden ist, während er auf die Differenzierung der Klassen I und III nur eine geringfügige Auswirkung hat (vgl. Abbildungen 6.6 bis 6.7). Dies unterstützt das Ergebnis der Faktorenanalyse, die den Parameter Rundheit als einen der wichtigen identifiziert hatte.

Testlauf 5: Das Modell Default ohne den Parameter der Kompaktheit Hier wurde analog zu Testlauf 4 vorgegangen, jedoch diesmal der Parameter der Kompaktheit vom Klassifikationsprozess ausgeschlossen. Die Abbildungen 6.6 bis 6.9 zeigen, dass dies für alle Klassen zu einer deutlichen Verschlechterung der Klassifikationsgüte führt. Während die Präzision für die Typen I und VI weiterhin hoch bleibt, sinkt bei ihnen die Ausbeute drastisch ab, d.h. viele der Teilchen dieser Klassen werden nicht als solche erkannt. Für die Typen III und IV/V ist das Verhalten entgegengesetzt: Hier ist die Ausbeute nach wie vor sehr gut, allerdings werden viele Teilchen als Vertreter dieser Klassen erkannt, die es faktisch nicht sind.

Testlauf 6: Das Modell Default ohne den Parameter der Fraktalen Dimension Auch hier wurde analog zu Testlauf 4 verfahren, jedoch diesmal unter Auslassung eines Parameters, den die Faktorenanalyse als wenig relevant klassifiziert hatte, der Fraktalen Dimension. Die Abbildungen 6.6 bis 6.9 zeigen deutlich, dass hier die Klassifikationsgüte im Gegensatz zu Testlauf 4 und 5 kaum beeinträchtigt wird.

Testlauf 7: Das Modell NotAllFeatures2 Dieses Modell wurde nur mit den Parametern erstellt, die nicht zu den in der Faktorenanalyse als relevant qualifizierten zählten (Richtzahl, Fraktale Dimension, Konvexität, Pixelgröße, Abweichungsfaktor, Eulerzahl und Streckung). Bzgl. der Klassifikationsgüte zeigte sich hier ein überraschend gutes Ergebnis

(vgl. Abbildungen 6.6 bis 6.9). Wie lässt sich dieses Ergebnis erklären? Die Faktorenanalyse berechnete für das Modell NotAllFeatures2 drei wichtige Klassifikationsparameter: Konvexität des Umfangs, Richtzahl und Abweichungsfaktor. Zusammen vereinigen sie ca. 70% der Gesamtvarianz in sich. Diese Parameter hatten auch schon im Defaultmodell hohe Ladungen. Als wichtige Parameter wurden dort jedoch nur solche mit einer Ladung >0,85 ausgewählt - ein Schwellwert, der mehr oder weniger willkürlich gewählt war. Lässt man nun die Parameter mit einer starken Aussagekraft weg, so zeigt sich, dass auch andere Parameter einen sehr hohen Informationsgehalt haben und ein gutes Klassifikationsergebnis erzielen. Diese Untersuchungen lassen vermuten, dass das Ergebnis der Faktorenanalyse weiter minimiert werden kann. Zwar sind die dort ermittelten Faktoren von hoher Relevanz - eventuell reicht jedoch schon eine kleine Teilmenge von Klassifikationsparametern aus, sofern schon das Training nur mit dieser Teilmenge arbeitet. Auch der genannte Schwellwert ist offenbar von Bedeutung - 85% war hier vermutlich ein Wert, der einige wichtige Faktoren eliminiert. Nicht zuletzt könnte die *Anzahl* der gewählten Hauptfaktoren von entscheidender Bedeutung sein. Interessant wäre außerdem die Frage, ob einzelne Klassifikationsparameter oder bestimmte Kombinationen von Parametern die Klassifikation eher schwieriger machen, indem sie die lineare Separabilität der Daten negativ beeinflussen. Hier besteht noch ein breites Feld für mögliche weitere Untersuchungen.

6.5 Klassifikation von heterogenen Bildern

Bei der Klassifikation der heterogenen Bilder stand der visuelle Vergleich mit den eingefärbten Bildern des Vergleichsklassifikators im Mittelpunkt. Wie schon in Absatz 6.2 dargestellt, ist die Vergleichbarkeit beider Klassifikatoren nur eingeschränkt gegeben. Als Beispiel zeigt der Anhang die Ergebnisse für drei Testbilder (sw_GJL_01.png, sw_GJV_01.png, sw_GJS_01.png), die die einzelnen Teilchentypen in jeweils unterschiedlicher Konzentration beinhalten, klassifiziert mit *POCA* (vgl. Abbildung 7.1, 7.3 und 7.5) und mit dem Vergleichsklassifikator (vgl. Abbildung 7.2, 7.4 und 7.6). Analog zu den in 6.4 beschriebenen Testläufen wurden auch für heterogene Bilder die Performanz der Modelle Linear und NotAllFeatures sowie die Auswirkung von weggelassenen relevanten Parametern (Rundheit und Kompaktheit) untersucht. Erwartungsgemäß decken sich die Ergebnisse mit denen aus Absatz 6.4, was an dieser Stelle nur die Bilder im Anhang (Abbildung 7.7 bis 7.13) illustrieren sollen.

Teilchenorientierte Analyse

Typ I

	Default	Linear	NotAllFeatures	SVM-Light	Default ohne Rundheit	Default ohne Kompaktheit	Default ohne Fraktale Dimension	NotAllFeatures2
precision	0,990121	0,986755	0,965005302	0,99226519	0,993597951	0,991452991	0,968944099	0,991247265
recall	0,948475	0,940063	0,956887487	0,94426919	0,815983176	0,487907466	0,984227129	0,952681388
F1	0,968851	0,962843	0,96092925	0,96767241	0,896073903	0,653981677	0,976525822	0,971581769

TypIII

	Default	Linear	NotAllFeatures	SVM-Light	Default ohne Rundheit	Default ohne Kompaktheit	Default ohne Fraktale Dimension	NotAllFeatures2
precision	0,939262	0,930811	0,94488189	0,93607801	0,826756497	0,599587912	0,975778547	0,942265795
recall	0,990847	0,985126	0,961098398	0,98855835	0,982837529	0,998855835	0,967963387	0,989702517
F1	0,964365	0,957198	0,952921157	0,96160267	0,898065865	0,749356223	0,971855256	0,965401786

Typ IV/V

	Default	Linear	NotAllFeatures	SVM-Light	Default ohne Rundheit	Default ohne Kompaktheit	Default ohne Fraktale Dimension	NotAllFeatures2
precision	0,921977	0,914839	0,91383812	0,92238034	0,491525424	0,625	0,93986014	0,933862434
recall	0,890704	0,890704	0,879396985	0,89572864	0,983668342	0,873115578	0,844221106	0,886934673
F1	0,90607	0,90261	0,896286812	0,90885915	0,655504395	0,72851153	0,889477167	0,909793814

Typ VI

	Default	Linear	NotAllFeatures	SVM-Light	Default ohne Rundheit	Default ohne Kompaktheit	Default ohne Fraktale Dimension	NotAllFeatures2
precision	0,910429	0,909988	0,901085645	0,91801242	0	0,997375328	0,873417722	0,899879373
recall	0,926342	0,921348	0,93258427	0,92259675	0	0,474406991	0,947565543	0,93133583
F1	0,918317	0,915633	0,916564417	0,92029888	0	0,642978003	0,908982036	0,915337423

Flächenorientierte Analyse

Typ I

	Default	Linear	NotAllFeatures	SVM-Light	Default ohne Rundheit	Default ohne Kompaktheit	Default ohne Fraktale Dimension	NotAllFeatures2
precision	0,990293	0,990601	0,93598821	0,99038462	0,998718989	0,995955056	0,90826818	0,990441864
recall	0,941967	0,888833	0,973833333	0,93386667	0,779633333	0,2216	0,98086	0,95264

TypIII

	Default	Linear	NotAllFeatures	SVM-Light	Default ohne Rundheit	Default ohne Kompaktheit	Default ohne Fraktale Dimension	NotAllFeatures2
precision	0,967598	0,943788	0,982092941	0,96454784	0,88165311	0,686561527	0,985600171	0,973217167
recall	0,9954	0,9948	0,966166667	0,99441667	0,847283333	0,999816667	0,950476667	0,99528

Typ IV/V

	Default	Linear	NotAllFeatures	SVM-Light	Default ohne Rundheit	Default ohne Kompaktheit	Default ohne Fraktale Dimension	NotAllFeatures2
precision	0,953127	0,946923	0,948695742	0,95195989	0,452610796	0,654528877	0,957729461	0,954309262
recall	0,91342	0,912	0,92014	0,9151	0,99492	0,91888	0,867134	0,906882

Typ VI

	Default	Linear	NotAllFeatures	SVM-Light	Default ohne Rundheit	Default ohne Kompaktheit	Default ohne Fraktale Dimension	NotAllFeatures2
precision	0,934975	0,933029	0,939876259	0,93979006	0	0,978064657	0,901734292	0,928687514
recall	0,963367	0,9592	0,959566667	0,96096667	0	0,596	0,968915	0,963816667

Abbildung 6.11: Klassifikationsgüte für alle Typen in allen Testläufen

	LIBSVM	SVM-Light
RBF	18 Sekunden	45 Sekunden
Linear	184 Sekunden	154 Sekunden

Abbildung 6.12: Laufzeiten von RBF- und Linearer Kernfunktion

6.6 Vergleich der SVM-Bibliotheken und Kernfunktionen

Wie schon in Absatz 6.4 dargestellt, unterscheiden sich weder die beiden angewandten Kernfunktionen (Linear und RBF) noch die Softwarebibliotheken LIBSVM und SVM-Light bzgl. der Klassifikationsgüte wesentlich voneinander. Beim Training der SVM erwies sich jedoch die RBF-Kernfunktion als deutlich performanter als die lineare Kernfunktion, sowohl bei LIBSVM als auch bei SVM-Light. Tabelle 6.12 zeigt die Laufzeiten für beide Kernfunktionen und Bibliotheken jeweils bzgl. der gleichen Trainingsdaten (4180 Teilchen) in Sekunden.

Kapitel 7

Zusammenfassung und weiterer Ausblick

Im Zentrum dieser Arbeit stand die automatische Klassifikation der verschiedenen Typen von Gusseisen nach DIN EN 945. Im Gegensatz zur vorangehenden Arbeit von Roberts [21], bei der die *Anordnung* der Teilchen betrachtet wurde, sollte dabei *teilchenbasiert* verfahren werden, d.h. die einzelnen Graphitteilchen sollten aufgrund ihrer Form klassifiziert werden. Dreizehn Formparameter wurden aus den Grundparametern Fläche, Umfang, konvexe Hülle und Feretsche Durchmesser berechnet und zusammen mit Richtzahl und Pixelgröße als Klassifikationsparameter verwendet. Die Arbeit konnte zeigen, dass eine Klassifikation über SVM (Support Vector Machine) auch für diesen Anwendungsfall ein gutes Ergebnis erzielt. Für die vorliegenden Testdaten ließ sich eine hohe Klassifikationsgüte (Präzision, Ausbeute und F1 größer als 90%) erreichen. Der entstandene Prototyp der Anwendung, *POCA*, ist zudem so gestaltet, dass damit einfach experimentiert werden kann. So können Klassifikationsparameter bei Training und/oder Klassifikation weggelassen oder die SVM-Einstellungen verändert werden. Die Erweiterung des Systems um Graphitklassen und Parameter, wurde bewusst einfach gehalten, so dass sich auch für die zukünftige Forschung Ansatzpunkte ergeben dürften. Die Faktorenanalyse in Form einer Hauptkomponentenanalyse mit anschließender orthogonaler Rotation der Faktoren erwies sich als hilfreiches Verfahren, um die Relevanz der einzelnen Parameter einzuschätzen. Dies konnte experimentell bestätigt werden.

Für die Zukunft bieten sich zahlreiche Möglichkeiten der Erweiterung oder Verbesserung. Ein wichtiger Schritt ist die Erstellung heterogener Testbilder in einer gesichert guten Qualität, die einen exakten Vergleich mit den Ergebnissen von *POCA* ermöglichen.

So könnte die Abschätzung der Klassifikationsgüte auch auf heterogene Bilder ausgeweitet werden. Erweiterungen sind auch bzgl. der Komponente der Faktorenanalyse möglich. Wie die Testläufe in Kapitel 6 zeigten, schätzte die Faktorenanalyse zwar die Relevanz der einzelnen Parameter gut ab. Wenn es jedoch, wie im vorliegenden Fall, viele sehr aussagekräftige Parameter gibt, dann lässt sich die Anzahl der Parameter weiter minimieren. Hier sollten geeignete Verfahren gefunden und implementiert werden. Zudem scheint eine Erweiterung der Anzahl der ausgewählten Hauptfaktoren wünschenswert, insbesondere für anders strukturierte Testdaten. Momentan ist die Auswahl der Faktoren immer auf zwei begrenzt, da dies die orthogonale Rotation ungleich einfacher macht. Wenn hier eine beliebige Anzahl an Faktoren ausgewählt werden kann, muss neben einem sinnvollen Selektionskriterium für die Faktoren auch ein Iterationsverfahren bei der orthogonalen Rotation implementiert werden (vgl. Kapitel 4.5). Auch im Bereich der Bildverarbeitung gibt es Möglichkeiten der Verbesserung und Erweiterung. Hierbei sei insbesondere an die Performanz bei der Erkennung der inneren Struktur der Teilchen gedacht (vgl. Kapitel 2.2.3). Daneben wäre es sinnvoll, die Struktur der zu verarbeitenden Bilder so zu scannen, dass Bilder, die nicht für eine teilchenbasierte Analyse geeignet sind, im Vorfeld erkannt und ausgeschlossen werden. Im Kontext der Klassifikation wäre die Anbindung der Multi-Class-SVM auch für SVM-Light wünschenswert. Außerdem stellt sich die Frage, inwieweit die Lage eines Teilchens im Klassifikationsprozess eine Rolle spielen kann und die Typen der Nachbarteilchen mit berücksichtigt werden können; ein Teilchen das nur von Teilchen des Typs X umgeben ist, kann z.B. eine höhere Wahrscheinlichkeit haben, auch der Klasse X anzugehören. Denkbar wäre dann auch die anschließende Segmentierung des Bildes in Bereiche, die von einem bestimmten Teilchentyp dominiert werden. Das Webinterface könnte um eine Benutzerverwaltung erweitert werden, sodass die Zugriffsrechte feiner granuliert werden könnten. Aus Performanzgründen scheint es zudem sinnvoll, bei der Klassifikation mehrere Alternativen für die Darstellung des Ergebnisses anzubieten, da die Übertragung des eingefärbten Bildes je nach Bandbreite der Verbindung ungeduldige Benutzer verärgern könnte. Nicht zuletzt bietet sich die Erstellung automatisierter Testauswertungen an. Eine einfache Implementierung (Berechnung der Daten und Export nach Excel) zur Berechnung der Klassifikationsgüte liegt der Software bei. Es gibt aber in diesem Zusammenhang bestimmt noch weitere Möglichkeiten der Automatisierung.

Anhang A

Bedeutung der Farben bei *POCA*

- rosa: Typ I,
- blau: Typ III,
- gelb: Typ IV/V,
- orange: Typ VI,
- schwarz: nicht klassifizierbare Teilchen.

Bedeutung der Farben bei dem verwendeten Vergleichsklassifikator

- rosa: Typ I,
- grün: Typ III,
- gelb: Typ IV/V,
- orange: Typ VI,
- schwarz: nicht klassifizierbare Teilchen,
- blau: Teilchen, die manuell ausgeschlossen wurden.

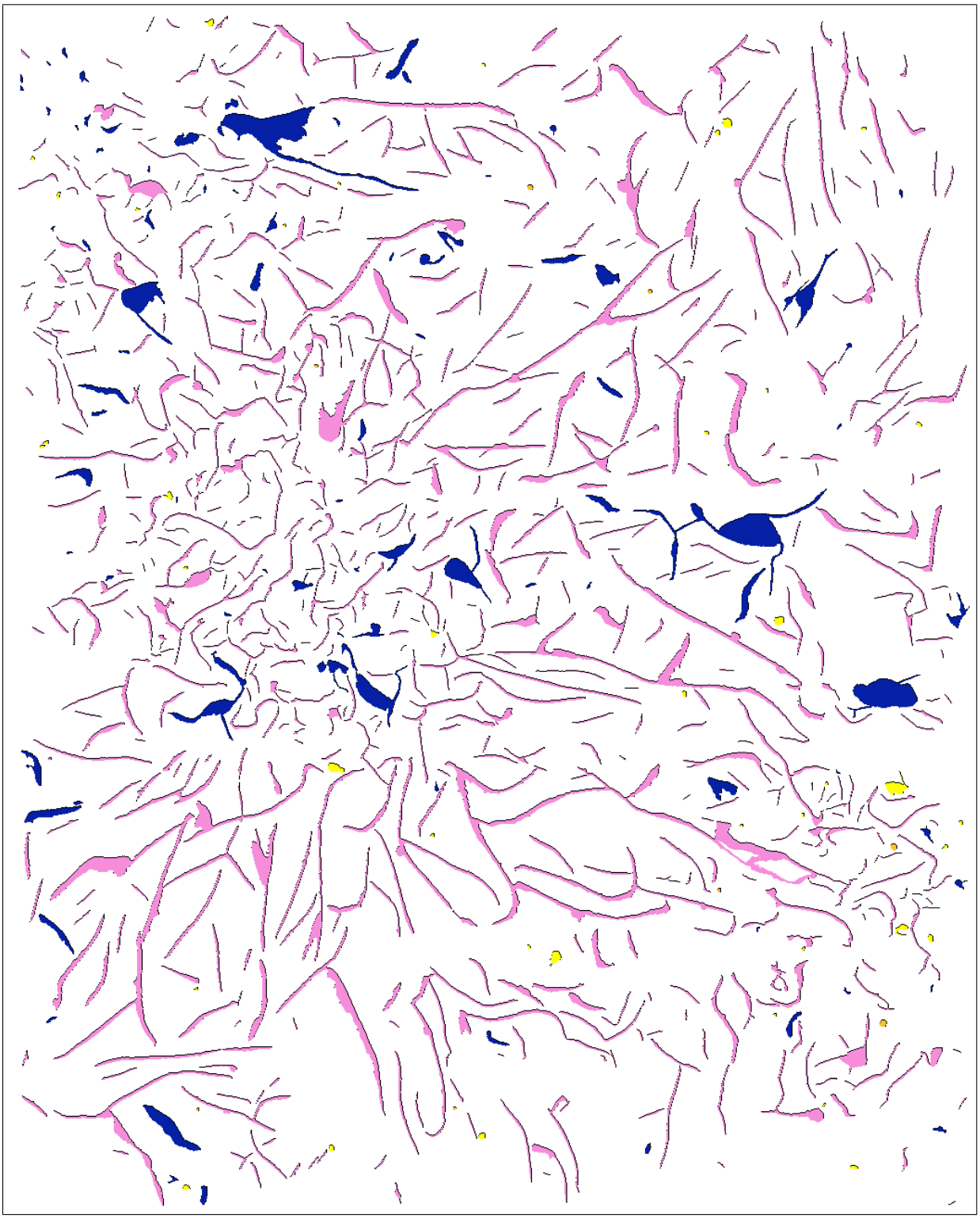

Abbildung 7.1: Klassifikation von sw_GJL_01.png mit POCA

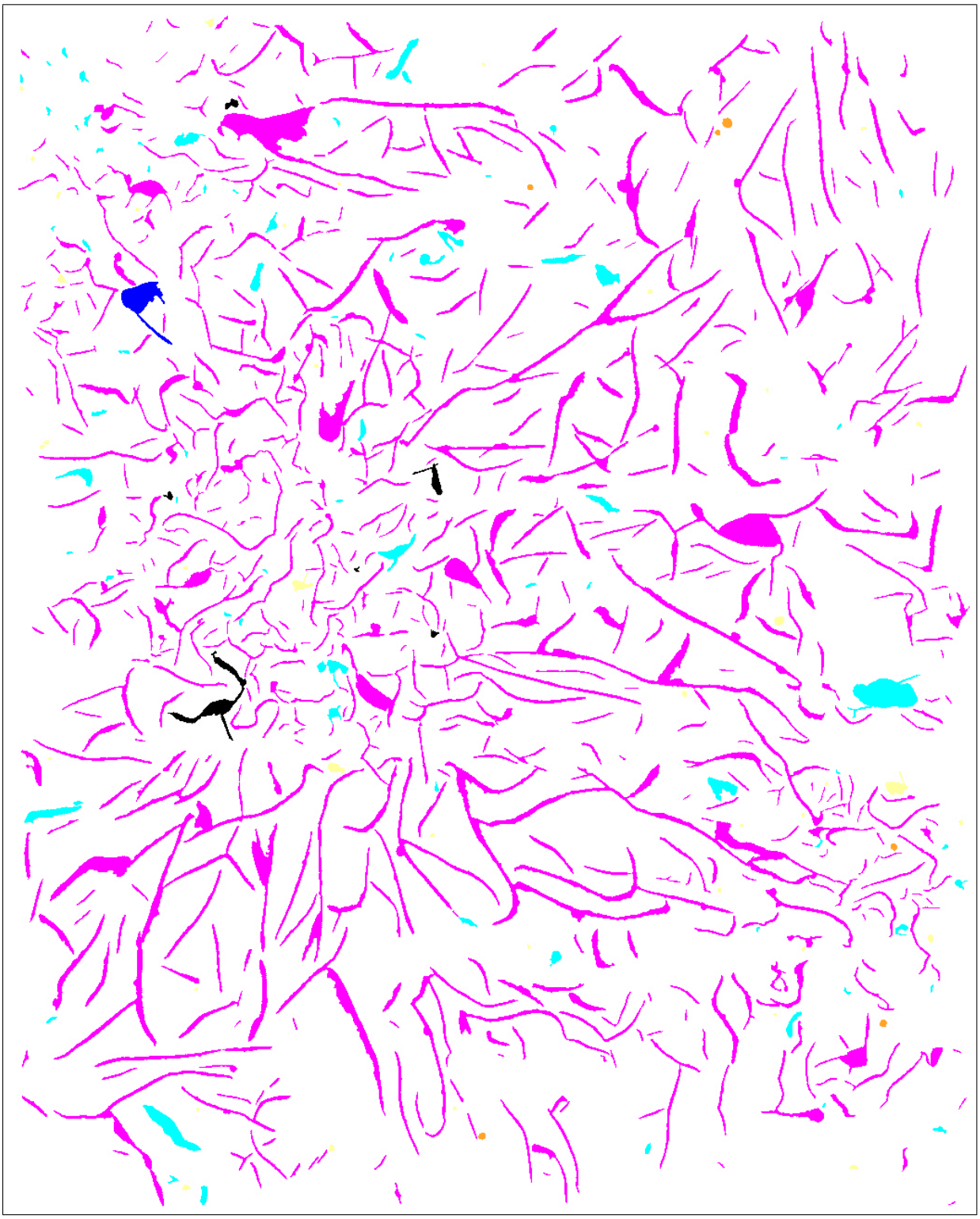

Abbildung 7.2: Klassifikation von sw_GJL_01.png mit dem Vergleichsklassifikator

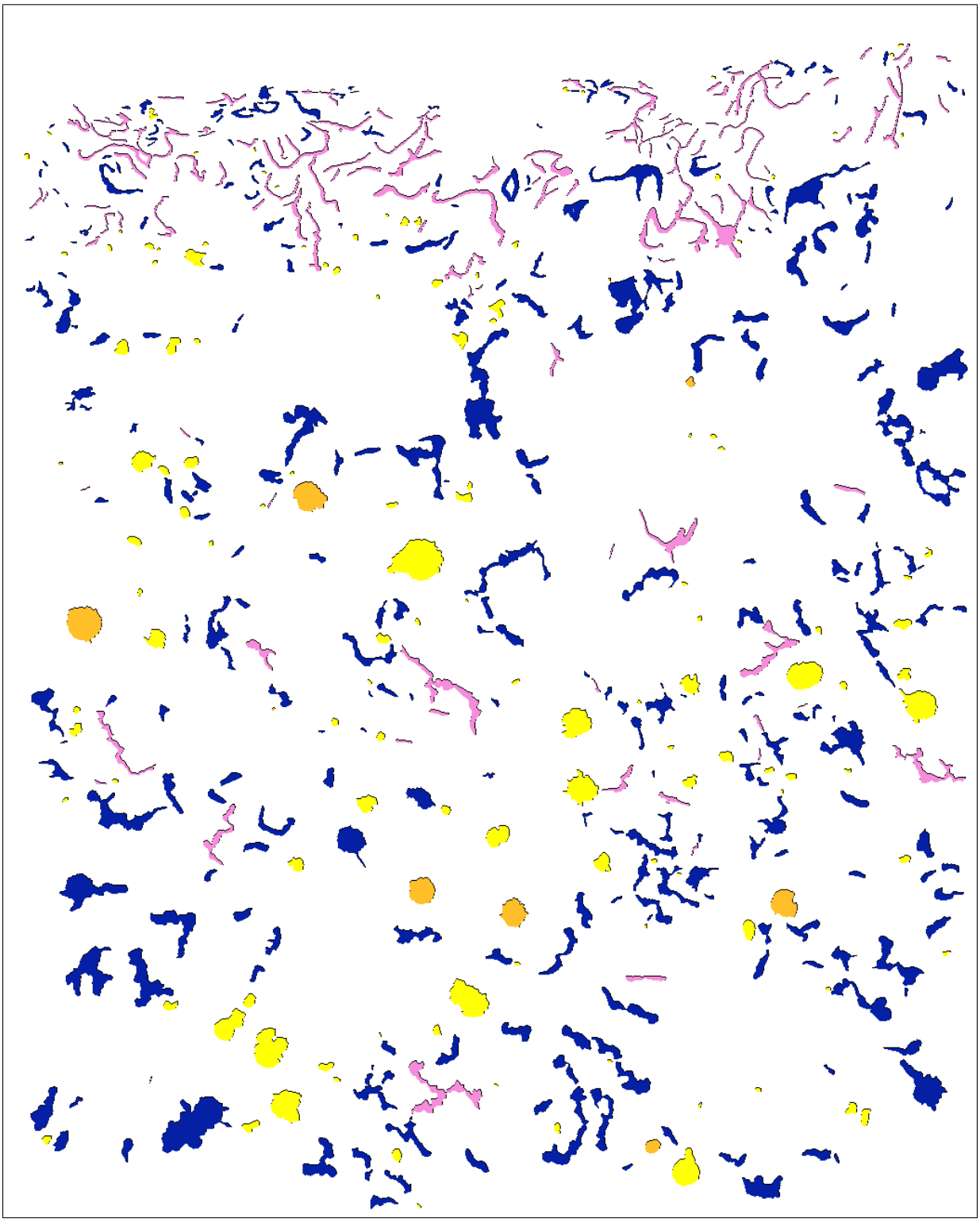

Abbildung 7.3: Klassifikation von sw_GJV_01.png mit POCA

Abbildung 7.4: Klassifikation von sw_GJV_01.png mit dem Vergleichsklassifikator

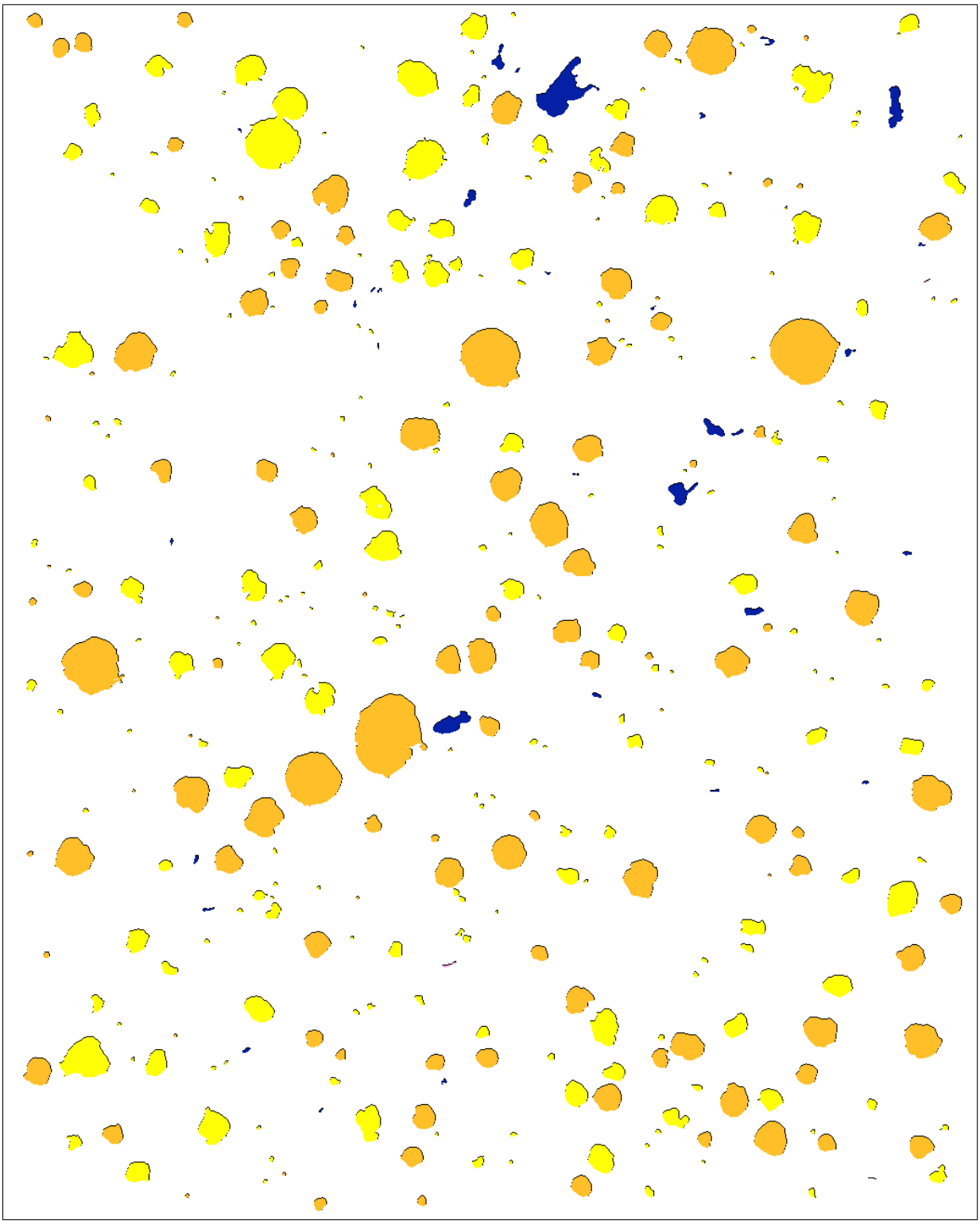

Abbildung 7.5: Klassifikation von sw_GJS_01.png mit POCA

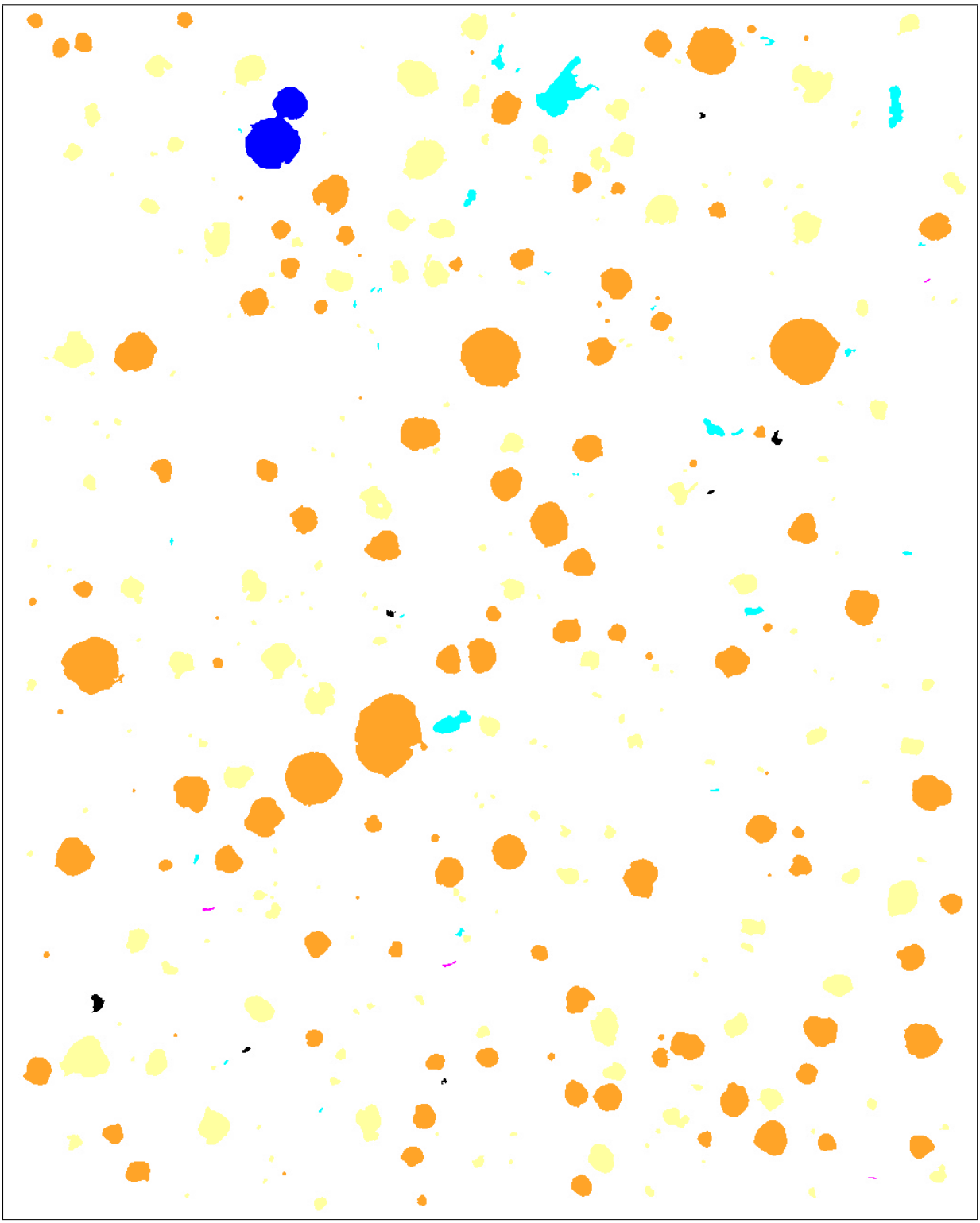

Abbildung 7.6: Klassifikation von sw_GJS_01.png mit dem Vergleichsklassifikator

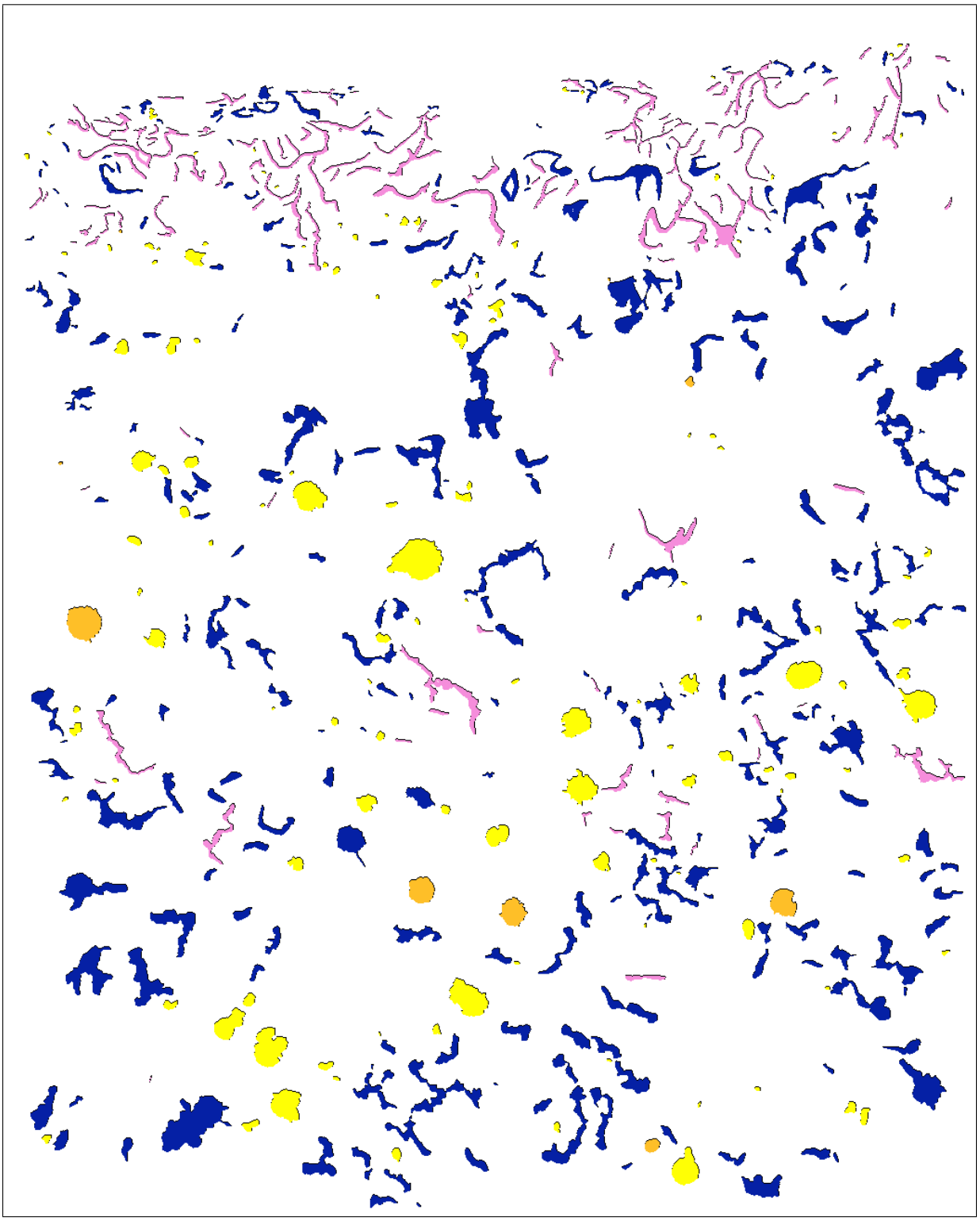

Abbildung 7.7: Klassifikation von sw_GJV_01.png mit Linearem Kernel

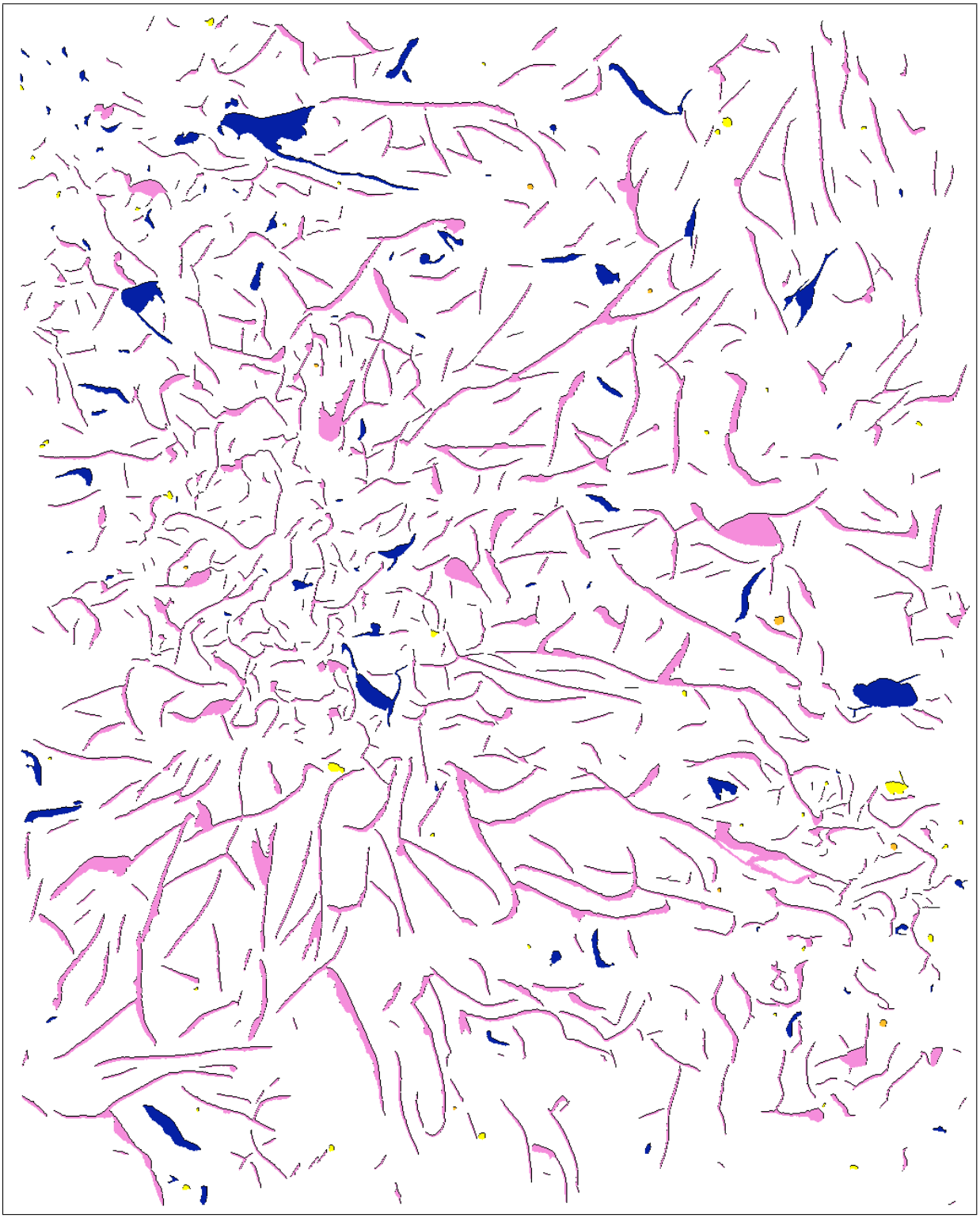

Abbildung 7.8: Klassifikation von sw_GJL_01.png anhand des Modells *NotAllFeatures* (vgl. *Default*-Modell Abb. 7.1)

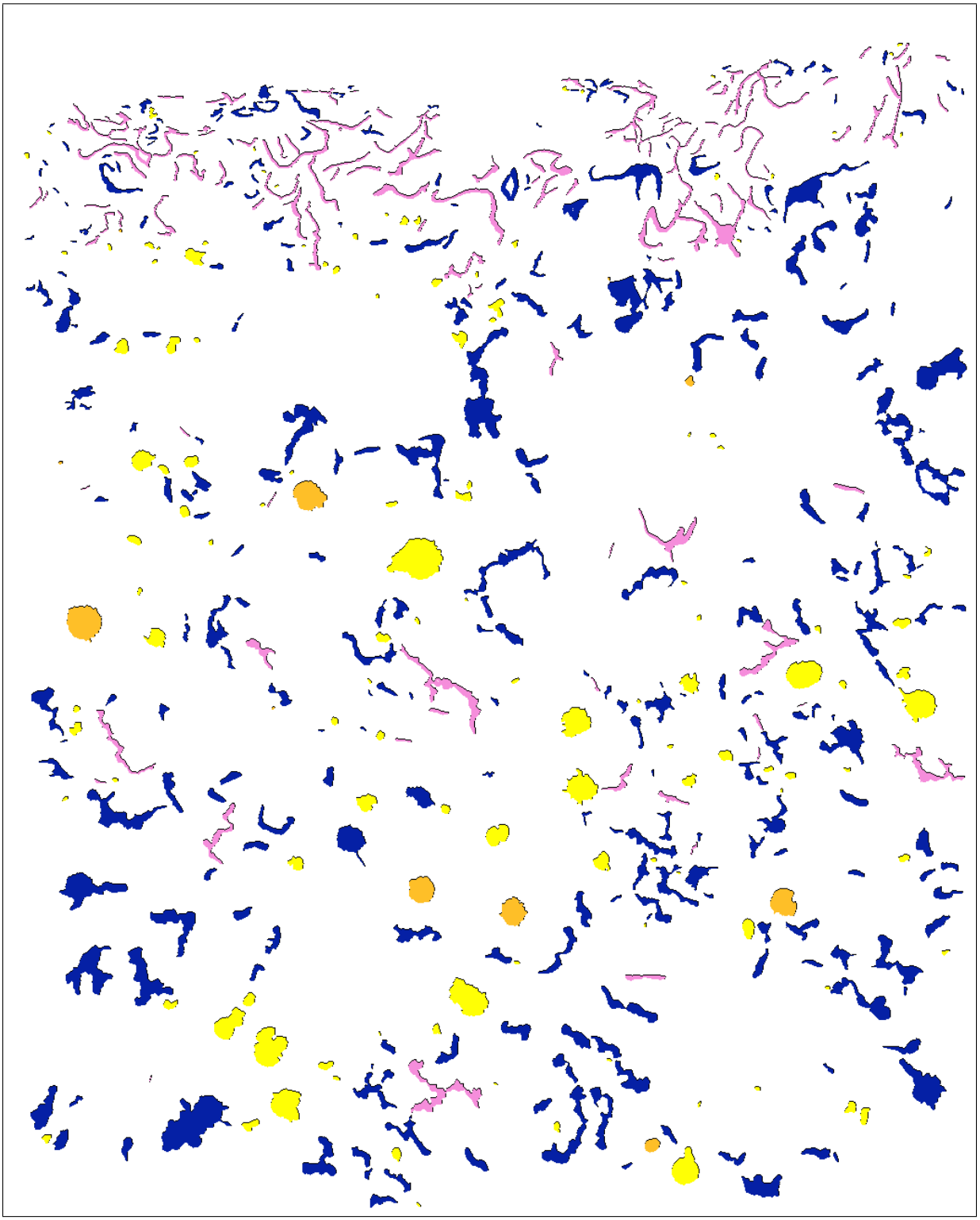

Abbildung 7.9: Klassifikation von sw_GJV_01.png anhand des Modells *NotAllFeatures* (vgl. *Default*-Modell Abb. 7.3)

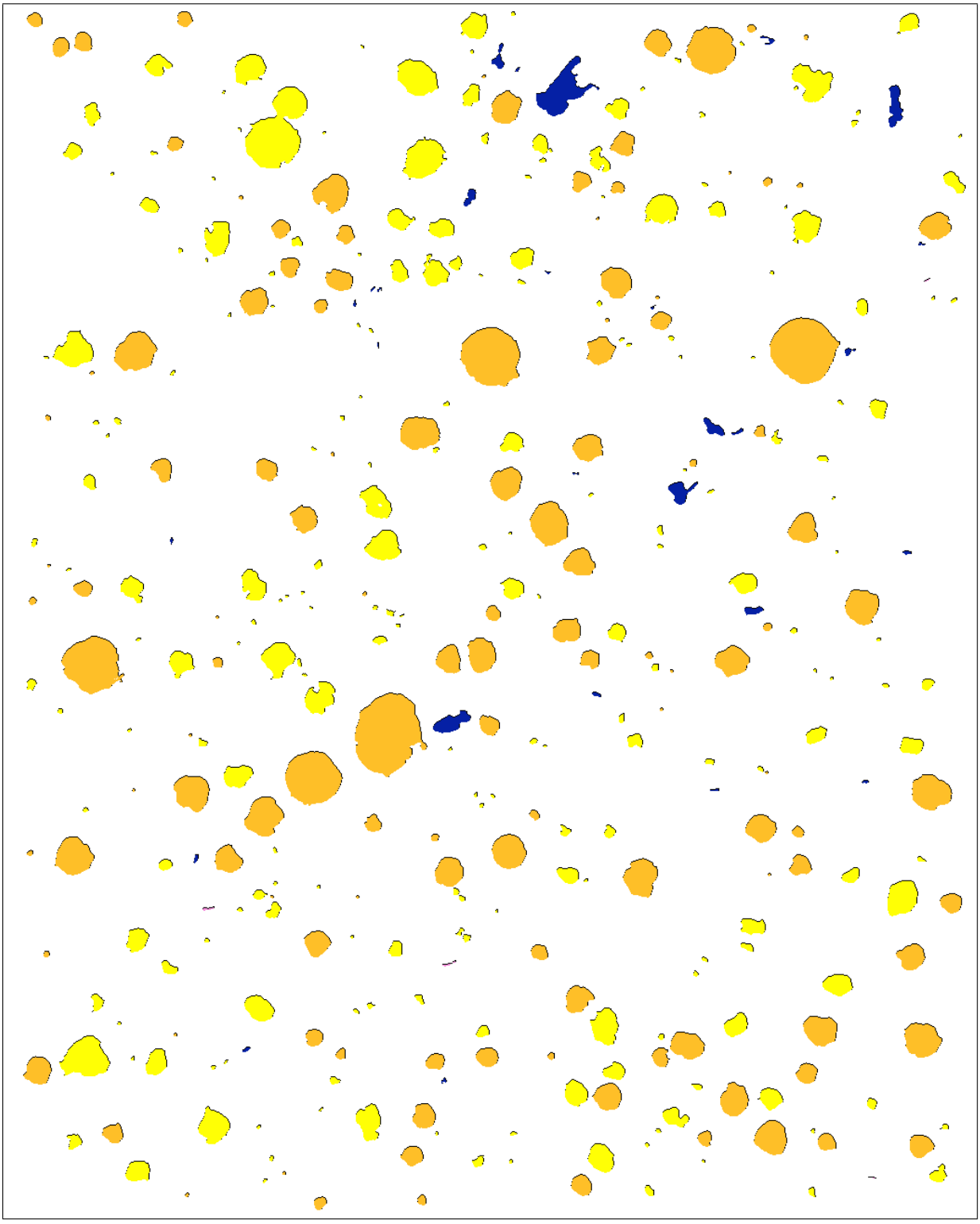

Abbildung 7.10: Klassifikation von sw_GJS_01.png anhand des Modells *NotAllFeatures* (vgl. *Default*-Modell Abb. 7.5)

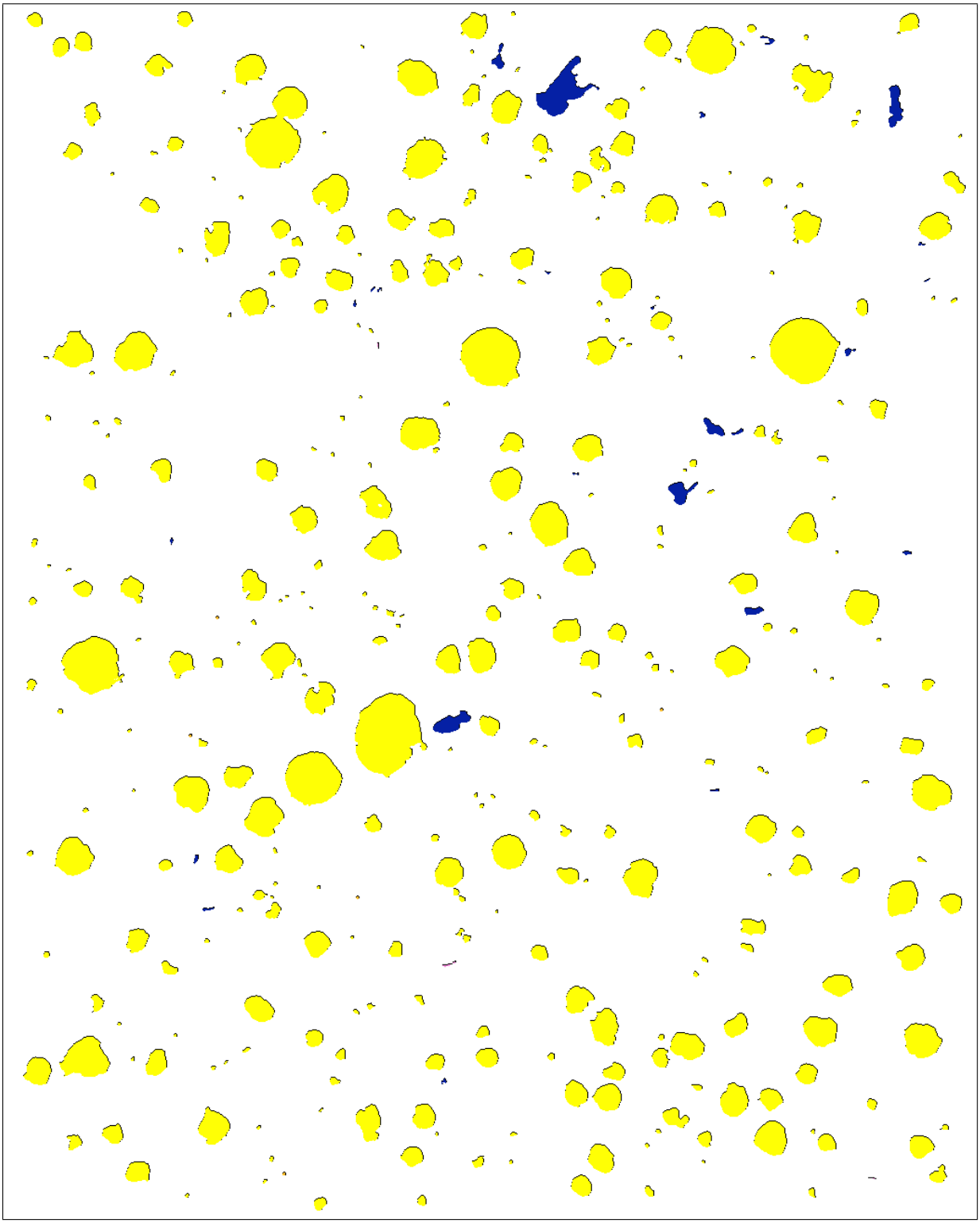

Abbildung 7.11: Klassifikation von sw_GJS_01.png ohne den Parameter Rundheit (vgl. *Default*-Modell Abb. 7.5)

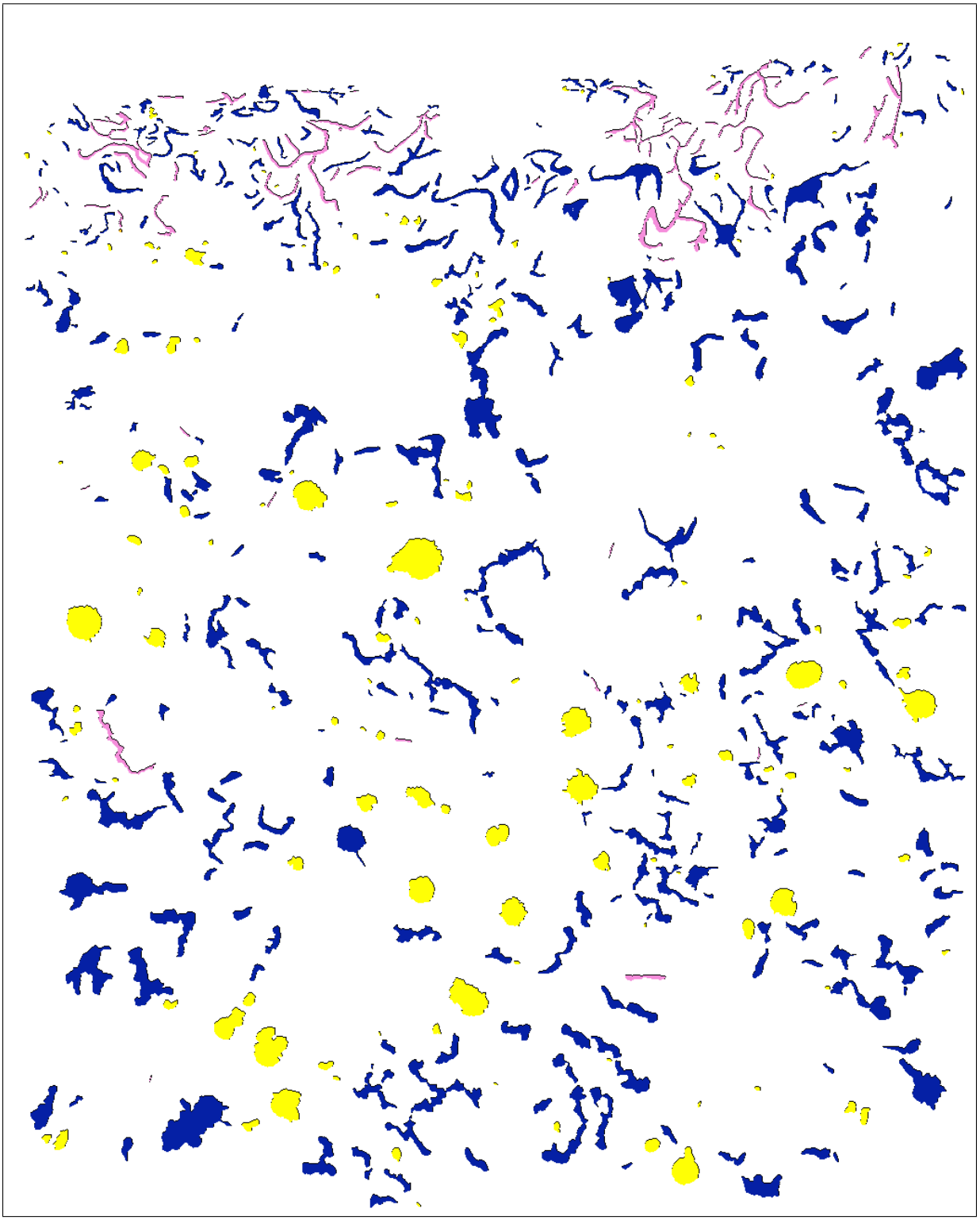

Abbildung 7.12: Klassifikation von sw_GJV_01.png ohne den Parameter Rundheit (vgl. *Default*-Modell Abb. 7.3)

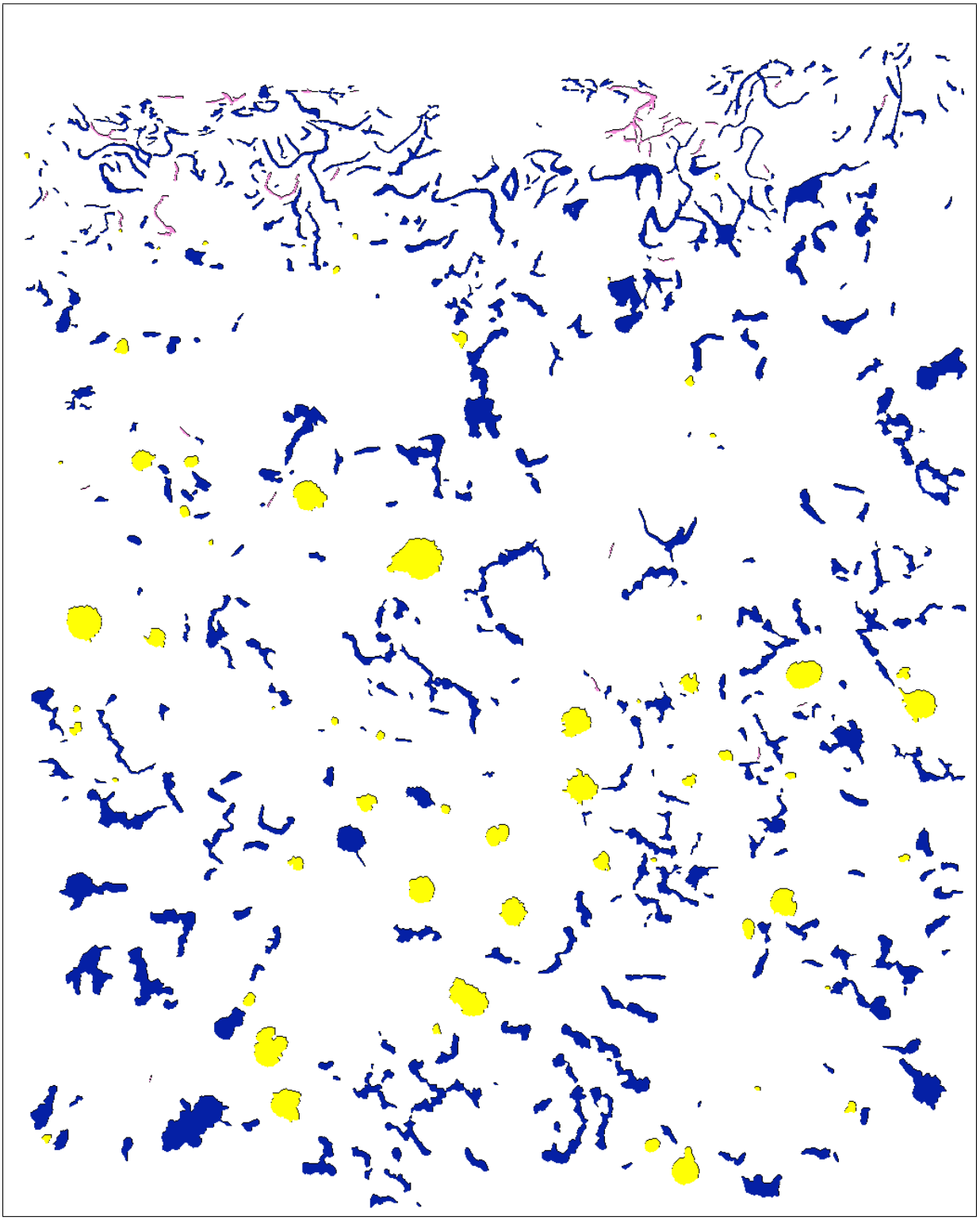

Abbildung 7.13: Klassifikation sw_GJV_01.png ohne die Parameter Kompaktheit/Formfaktor (vgl. *Default*-Modell Abb. 7.3)

Literaturverzeichnis

[1] A.M.Vossepoel and A.W.M.Smeulders, *Vector Code Probability and Metrication Error in the Representation of Straight Lines of Finite Length*, Computer Graphics and Image Processing **20** (1982), 347–364.

[2] Isabelle M. Guyon und Vladimir N. Vapnik. Bernhard E. Boser, *A Traininig Algorithm for Optimal Margin Classifers.*, D. Haussler (Hg.), Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory, 1992, pp. 144–152.

[3] U.Wendt B.I.Imasogie, *Characterization Of Graphite Particle Shape in Speroidal Graphite Iron Using A Computer-Based Image Analyzer*, Journal of Minerals & Materials Characterization & Engineering **3** (2004), 1–12.

[4] Christopher J. C. Burges, *A Tutorial on Support Vector Machines for Pattern Recognition*, Data Mining and Knowledge Discovery **2** (1998), no. 2, 121–167.

[5] C.A.Groen and P.W.Verbeek, *Freeman-code probabilities of object boundary quantized contours*, Computer Graphics and Image Processing **7** (1978), 391–402.

[6] L. Dorst and A. W. M. Smeulders, *Discrete Straight Line Segments: Parameters, Primitives and Properties*, Vision Geometry, series Contemporary Mathematics (P.and Rosenfeld A.Melter, R.and Bhattacharya, ed.), vol. 119, American Mathematical Society, 1991, pp. 45–62.

[7] D.Proffitt and D.Rosen, *Metrication Errors and coding efficiency of chain-encoding schemes for the representation of lines and edges*, Computer Graphics and Image Processing **10** (1979), 318–332.

[8] IMTRONIC GmbH, *ImageC Anwenderdokumentation*, fifth ed., IMTRONIC GmbH, Berlin, 2001.

[9] Freeman H. and Davis L., *A corner-finding algorithm for chain-coded curves*, vol. C-26, IEEE Transactions on Computers, 1977, pp. 297–303.

[10] Dr.Christoph Heckenkamp, Tech. report, Fachhochschule Darmstadt, 2004.

[11] H.F.Kaiser, *The Varimay Criterion for Analytic Rotation in Factor Analysis*, Psychometrika **23** (1958), 187–200.

[12] August Hitzbeck, *Temperguss*, Tech. report.

[13] Man On Lai Jianming Li, Li Lu, *Quantitative analysis of the irregularity of graphite nodules in cast iron*, Materials Characterization **45** (2000), 83–88.

[14] N.Llorca-Isern J.M.Guilemany, *Die bildanalytische Identifizierung der Graphitmorphologie in Gußeisen*, Practical Metallography **27** (1990), 189–194.

[15] W.Stets und W.Gerber J.Ohser, K.Sandau, *Bildanalytische Charakterisierung von Graphit im Grauguss und Klassifikation der Lamellenanordnung*, Praktische Metallographie.

[16] Gerhard Tutz (Hrsg.) Ludwig Fahrmeir, Alfred Hamerle, *Multivariate Statistische Verfahren*, second ed.

[17] Joachim Ohser and Frank Mücklich, *Statistical Analysis of Microstructures in Materials Science*, John Wiley & Sons, Inc., Chichester, 2000.

[18] Trevor Parc, *A Note on Terse Coding of Kaiser's Varimax Rotation Using Complex Number Representation*, Tech. report.

[19] Theo Pavlidis, *Algorithmen zur Grafik und Bildverarbeitung*, Heise, Hannover, 1990.

[20] Richard A. Johnson and Dean W. Wichern, *Applied Multivariate Statistical Analysis*, fifth ed., Prentice Hall, Pearson International Edition, Upper Saddle River, 2002.

[21] Kathrin Roberts, Master's thesis.

[22] R.Ruxanda and D.M.Stefanescu, *Graphite shape characterisation in cast iron - from visual estimation to fractal dimension*, Int. J. Cast Metals Res. **14** (2002), 207–216.

[23] Leszek Wojnar, *Image Analysis*, Springer, 1998.

Erklärung

Hiermit erkläre ich an Eides Statt, dass ich die vorliegende Arbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe.

Saarbrücken, den 31. März 2005

Corinna Richter