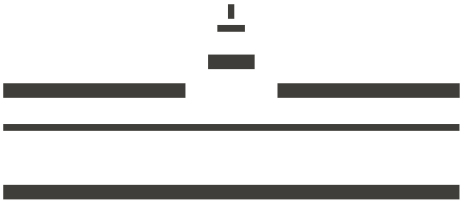


Westfälische
Wilhelms-Universität
Münster

Transformations-invariantes Sparse Coding und seine Nutzung für Objekt-Klassifikation in natürlichen Bildern

Transformation invariant sparse coding and its use for object classification in natural images

Masterarbeit

Eingereicht von:	Tobias Beckmann
Datum:	18.10.2019
Erstgutachter:	Prof. Dr. Benedikt Wirth
Zweitgutachter:	Dr. Frank Wübbeling

Inhaltsverzeichnis

1 Einleitung

Sparse Coding ist eine wichtige Technik der Datenrepräsentation, welche in verschiedensten Anwendungsgebieten wie Data Compressing, Image Denoising, Inpainting und Image Restoring zum Einsatz kommt. Vielversprechende Resultate liefert es auch als Hilfsmittel zur Bildklassifizierung und Objektdetektion, wo es zur Generierung von Features eingesetzt wird. Vor allem aufgrund letzterer Anwendungsgebiete wurden in der Vergangenheit verschiedene Erweiterungen des grundsätzlichen Sparse Codings vorgestellt, die durch transformationsinvariante Ansätze vielversprechende Resultate für Objekte in verschiedenen Ausrichtungen und Positionen versprechen.
Besonders beeindruckend sind die Resultate in *Fast Rotational Sparse Coding* von McCann, Unser und Depeursinge [6], die sowohl mit einem laufzeiteffizienten Ansatz des Sparse Codings aufwarten als auch beeindruckende Ergebnisse in der Klassifikation von Texturen erzielen. Daher soll in dieser Arbeit die folgende Forschungsfrage beantwortet werden: In wie fern können transformationsinvariante Algorithmen genutzt werden, um die Klassifikationsmethode erfolgreich auf die Objektdetektion zu übertragen?
Die Arbeit ist nach positivistischer Struktur aufgebaut und validiert die theoretisch motivierte Methodik. Die dafür genutzten Daten selbst haben jedoch auch eine grundlegende Bedeutung für die Performance. Daher werden diese direkt in Kapitel 2 beschrieben. Dies sind zum einen Texturdaten, die in dem Paper *Fast Rotational Sparse Coding* [6] ebenfalls verwendet werden, was eine direkte Vergleichbarkeit der verschiedenen Ansätze und Implementierungen ermöglicht. Zum anderen wird die Objektdetektion auf aus Mikroskopie-Videos entnommenen Zellbildern getestet, sodass die Algorithmen in Anbetracht eines aktuellen Forschungskontextes bewertet werden können.
In Kapitel 3 wird im Hinblick auf die spätere Objektdetektion besonderes Interesse darauf gelegt, wie man den Sparse Coding Algorithmus transformationsinvariant erweitern kann. Dazu wird sowohl ein rotationsinvarianter Ansatz als auch ein translationsinvarianter Ansatz inklusive jeweiligem Algorithmus vorgestellt und analysiert. Erfreulicherweise stellt sich heraus, dass der Grundgedanke des rotationsinvarianten Ansatzes, der auf *Fast Rotational Sparse Coding* [6] basiert, noch ausgebaut werden kann. Dies mündet in einem rotationsinvarianten Algorithmus basierend auf Polarkoordinaten.
In Kapitel 4 wird beschrieben, wie die durch das Sparse Coding gewonnenen Informationen als Features für die Objektdetektion verwendet werden können. Dazu wird zunächst das Verfahren der Autoren Unser, McCann und Depeursinge präsentiert, das die Klassifikation von Bildern löst [6]. Dieses wird anschließend auf das Detektionsproblem übertragen, indem die Detektion als Klassifikation von Pixeln aufgefasst wird.
Kapitel 5 dient der Validierung der vorgestellten Methoden. Zu Beginn werden die Repräsentationsfähigkeit und die Laufzeit der verschiedenen Algorithmen gegenübergestellt. Dabei werden, auch im Hinblick auf die folgenden Klassifikations- und Detektionspro-

bleme, die verschiedenen Einflussgrößen diskutiert, da diese eine entscheidende Rolle spielen. Anschließend werden die Klassifikationsergebnisse des grundlegenden Papers [6] verifiziert und darauf folgend die Detektionsergebnisse aller Algorithmen auf den verschiedenen Datensätzen präsentiert.
Schließlich resümiert Kapitel 6 das Forschungsvorgehen und die Durchführung zur Beantwortung der Forschungsfrage. Dabei werden kritisch reflektierend sowohl theoretische Aspekte, wie die Auswahl der Problemformulierungen des Sparse Codings, als auch praktische Einflüsse, wie die verschiedenen Datensätze, beleuchtet.
Die Resultate in dieser Arbeit beziehen sich komplett auf die Python-Programmierung, die auf der beiliegenden CD abgespeichert ist. Das explizite Programm ist in dieser Arbeit nicht niedergeschrieben, allerdings sind alle wichtigen Passagen als Pseudocode an entsprechenden Stellen dieser Arbeit zu finden.

2 Daten

In diesem Kapitel werden alle in dieser Arbeit verwendeteten Datensätze vorgestellt. Natürlich sind die Algorithmen keinesfalls auf diese Datensätze beschränkt, jedoch lässt sich anhand dieser Daten systematisch der Hintergedanke der verschiedenen Problemstellungen austesten.
Im Folgenden werden zwei Datensätze betrachtet. Zunächst wird dies ein Datensatz aus der Outex Texturen Datenbank [13] sein. Dieser wurde ausgewählt, da er in dem als Grundlage fungierenden Paper *Fast Rotational Sparse Coding* [6] ebenfalls verwendet wird. Damit ist eine direkte Vergleichbarkeit der Ergebnisse gewährleistet.
Anschließend wird ein Datensatz beleuchtet, der aus Miskropie-Videos von Endothelzellen entstanden ist. Anhand diesem wird die Übertragbarkeit der Algorithmen auf ein natürliches, sehr schwieriges Detektionsproblem getestet.

2.1 Texturen

Dieser Datensatz enthält natürliche Fotos von 24 verschiedenen Texturen (z.B. Teppich und Fliesen, siehe Abbildung 2.1). Die mit dem Datensatz verknüpfte Zielformulierung ist die Klassifikation der Bilder in die 24 verschiedenen Klassen an Texturen. Diese wurden mittels eines standardisierten Protokolls (für Details siehe [7]) erstellt. Die Qualität der Bilder ist somit sehr gut. Unter anderem wurden die Bilder auf einen Mittelwert von 128 mit Standardabweichung 20 normalisiert.
In der Outex Datenbank [13] sind verschiedene Datensätze zu finden. Konsistent zu dem Paper [6] wird in dieser Arbeit die Datenreihe Outex_TC_00010 verwendet. Diese Daten sind laut den Autoren die meistgenutzten Daten für Texturklassifikationsprobleme.

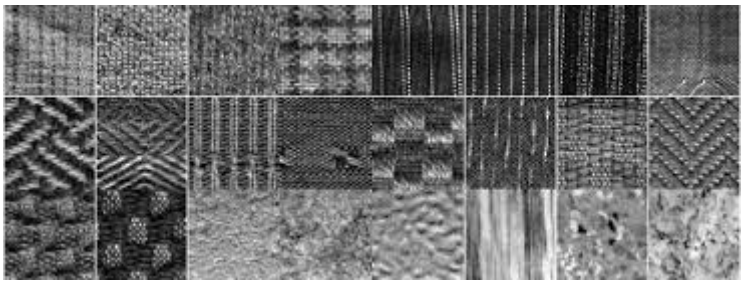


Abbildung 2.1: Beispiele aller Texturen

Die Daten sind bereits in Trainings- und Testbilder der Größe 128×128 unterteilt. Die Trainingsdaten enthalten jeweils 20 Bilder der 24 verschiedenen Texturen. Im Testdatensatz sind rotierte Versionen der Trainingsbilder zu finden. Die Bilder wurden um die

Winkel $5°, 10°, 15°, 30°, 45°, 60°, 75°$ und $90°$ gedreht, was insgesamt 3840 Testbilder ergibt.
Die hier unternommene Einteilung des Datensatzes in Trainings- und Testbilder sollte kritisch hinterfragt werden. Da die Testbilder rotierte Versionen der Trainingsbilder darstellen, sind Trainings- und Testdaten nicht unabhängig. Dies ist jedoch üblicherweise eine zwingende Voraussetzung an die Unterteilung. Da die Daten inklusive Einteilung in Trainings- und Testset durch die Wissenschaftler der Universität Oulu, Finnland, veröffentlicht (siehe [13]) und durch die Autoren McCann, Unser und Depeursinge übernommen (vgl. [6]) wurden, werden diese auch hier verwendet, um eine Vergleichbarkeit der Ergebnisse zu ermöglichen. Es sollte jedoch bedacht werden, dass diese Unterteilung die außergewöhnlich guten Resultate (vgl. Kapitel 5.3) begünstigt.

Modifizierter Texturdatensatz Im Laufe dieser Arbeit wird nicht nur die Klassifikation von Bildern, sondern auch die Detektion von Objekten diskutiert. Da die Algorithmen recht ähnlich sein werden, bietet es sich an, diese zunächst auf ähnlichen Daten zu testen. Dazu wird eine modifizierte Version des Texturdatensatzes erstellt: Es werden zufällig gewählte Ausschnitte der Größe 25×25 horizontal und vertikal aneinandergereiht (Gesamtgröße des entstehenden Bildes: 250×250).

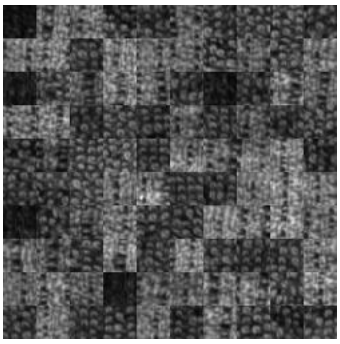


Abbildung 2.2: Trainingsbild der modifizierten Texturdaten

Die Unterteilung in Trainingsdaten und Testdaten bleibt unberührt, das heißt, dass für die Trainingsdaten nur Bilder des Trainingssets und für die Testdaten nur Bilder des Testsets verwendet werden. In Hinblick auf die Anwendung 'Objektdetektion' wird außerdem die Anzahl der Klassen auf zwei reduziert. In Abbildung 2.2 ist beispielhaft ein Trainingsbild zu sehen.

2.2 Biologische Zellen

Der zweite Datensatz repräsentiert Miskropie-Videos von Endothelzellen. Aus diesem werden entsprechend des Bedarfs mehrere Bilder der Größe 512×512 entnommen. Diese werden in dieser Arbeit für zwei verschiedene Aufgabenstellungen verwendet. Zunächst werden die Zellkerne detektiert. Die Zellkerne sind in den Bildern gut erkennbar, der

Algorithmus liefert entsprechend gute Resultate. Anschließend wird versucht die sogenannten JAIL zu detektieren. Diese sind sehr kleine, unregelmäßig und eher selten auftauchende Formationen am Rande der Zellwände. Um die Problemstellung und Schwierigkeit dieses Problems einschätzen zu können, wird dieses hier nun kurz erklärt. Die Erläuterung ist auf die beiden Paper [11] und [10] zurückzuführen. Die Videos stammen von Prof. Dr. Med. Hans-Joachim Schnittler und Dr. rer. nat. Jochen Seebach aus dem Institut für Anatomie und Vaskuläre Biologie der WWU Münster.
Interzellulare Verbindungen sind kritische Regulatoren für eine Vielzahl von physiologischen und pathologischen Prozessen und müssen daher schnell reagieren und hochdynamisch sein. Die Endothelzellen bilden diese Verbindung und somit eine regulierbare Barriere zwischen dem Blutgefäß im Inneren und dem Flüssigkeitsraum außerhalb. Endothelzellen kommen in allen Gefäßen des Herz-Kreislauf-Systems vor. Das vorliegende Video zeigt Endothelzellen der Venen einer Nabelschnur(„Human umbilical vein endothelial cell“ oder kurz „HUVEC“).
Die Bildung eines Endothels geschieht über Adhärenz-Verbindungen, wofür vaskuläre endotheliale Cadherine (VE-Cadherine) nötig sind. Die Formation neuer VE-Cadherine wird durch sogenannte ”junction associated intermittent lammellipodia”(kurz: JAIL) ausgelöst. Diese JAIL sind klein und nur kurz zu sehen. Man kann sie zu (scheinbar) beliebigen Zeipunkten an verschiedenen Stellen der Zellwände entdecken. Nach der Entstehung werden sie zunächst größer, dann kleiner und verschwinden schließlich wieder. In Abbildung 2.3 sind sie, durch die gelben Pfeile markiert, zu sehen. Sie sind vor allem im oberen grünen Kanal ersichtlich. Da die Entstehung und Entwicklung der JAIL ein sehr aktuelles Forschungsthema sind, ist die automatische Detektion in den Videos sehr gefragt.
Die Videos bestehen aus den beiden Kanälen rot und grün. Um die JAIL und VE-Cadherine sichtbar zu machen, wurden sie mit entsprechenden Substanzen angereichert. Dies führt dazu, dass die JAIL letztendlich im grünen Kanal zu sehen sind, die VE-Cadherine im roten (vgl. Abbildung 2.3). Die betrachtete Problemstellung in dieser Ar-

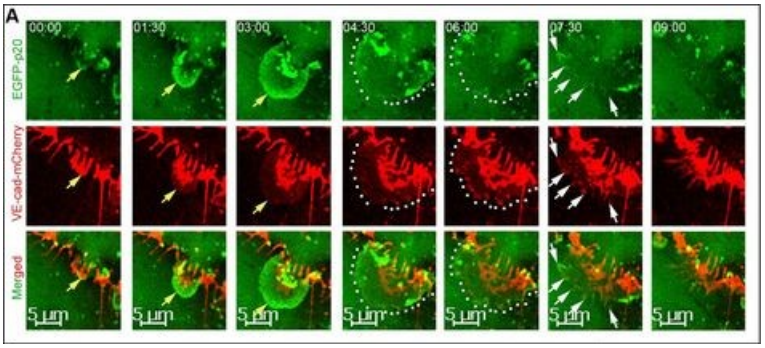


Abbildung 2.3: JAIL(Pfeile), die neue VE-Cadherine bilden(gepunktete Linie) (aus [11])

beit ist das Detektieren dieser JAILs. Da die JAIL im grünen Kanal deutlich besser zu identifizieren sind, werden die Daten in dieser Arbeit als einfarbig betrachtet. Der rote Kanal wird ignoriert. Ebenfalls ignoriert wird der zeitliche Zusammenhang der Bilder.

Diese Arbeit analysiert lediglich die Verarbeitung von Bildern, daher werden die Videos als einzelne Bilder betrachtet.
Zur Verfügung standen mehrere Aufnahmen. Alle hier veröffentlichten Ergebnisse beziehen sich auf die entrauschte Aufnahme "MD_Aufnahme_0002", welche die beste Qualität aufweist.
Abschließend sei auch hier erneut auf die Unterteilung der Daten in Test- und Trainingsdaten eingegangen. Da lediglich Daten desselben Videos verwendet werden, sind auch hier wieder Abhängigkeiten zwischen den Bildern vorhanden. Diesen wird insofern entgegengewirkt, dass nur wenige Einzelbilder in den Datensätzen zu finden sind. Zwischen diesen Einzelbildern sind in den meisten Fällen viele komplett unbenutzte Bilder zu finden, sodass durch die zeitliche Distanz eine gewisse Unabhängigkeit der Bilder entsteht. In Einzelfällen werden auch (fast) aufeinanderfolgende Bilder einbezogen, diese werden dann aber stets einheitlich dem Trainings- oder Testdatensatz zugeordnet.

3 Sparse Coding

Dieses Kapitel widmet sich dem Sparse Coding, auch Dictionary Learning genannt. Zunächst wird das Problem des Sparse Coding grundsätzlich vorgestellt und anschließend um Transformationsinvarianz erweitert. Dabei werden je nach Formulierung verschiedene Algorithmen präsentiert, die das jeweilige Problem lösen.

3.1 Problemstellung

Das Ziel des Sparse Coding ist einen vorgegebenen Datensatz x mithilfe von wenigen Repräsentanten D spärlich darzustellen: $x = Da$. Dabei sind die Zielkriterien sowohl eine möglichst genaue Approximation der Daten $\|x - Da\|_2$ als auch eine möglichst geringe Anzahl $\|a\|_0$ an verwendeten Repräsentanten.
Die Repräsentanten werden Atome genannt und in einem sogenannten Dictionary zusammengefasst. Diese Atome sind nicht gegeben, sondern müssen durch Lösen des Optimierungsproblems bestimmt, also erlernt, werden, daher stammt der synonym zu Sparse Coding verwendete Begriff Dictionary Learning.
Die Schwierigkeit dieses Problems besteht darin, dass neben dem Dictionary, also den Repräsentanten, auch die Repräsentation selbst, im Folgenden Code genannt, unbekannt ist.

$$\underset{D,\{a_p\}}{\operatorname{argmin}} \sum_{p=1}^{P} \|x_p - Da_p\|^2 \, mit \, \|a_p\|_0 \leq K \; \forall p \tag{3.1}$$

Dabei stehen $x_p \in \mathbb{R}^{N^2}$ hier für die zu approximierenden Bilder und Da_p stellt die Linearkombination des Dictionarys $D \in \mathbb{R}^{N^2 \times M}$ und des Codes $a_1, \ldots, a_P \subset \mathbb{R}$ dar. In dieser Arbeit werden stets P quadratische Bilder der Länge N als Spaltenvektoren $x_p \in \mathbb{R}^{N^2}$ betrachtet, die durch M Atome approximiert werden sollen.
An die Atome selbst werden somit keine Bedingungen geknüpft. Dies heißt insbesondere, dass sie weder orthogonal noch unabhängig sein müssen. Daher werden die Approximationen der Bilder schlussendlich nicht eindeutig sein. Außerdem sollte klar sein, dass dies kein konvexes Optimierungsproblem ist. Es wird also stets nur nach lokalen Optimierern gesucht, das globale Problem zu lösen erweist sich als zu komplex.

l_0-Norm Als $l_0 - Norm$ wird im Folgenden stets der Operator

$$\forall x \in \mathbb{R}^n : \|x\|_0 = \sum_{i=0}^{n} |x|^0$$

bezeichnet, wobei $0^0 = 0$ gilt. Die l_0-Norm zählt also die Einträge eines Vektors, die nicht Null sind. Die l_0-Norm ist keine Norm im mathematischen Sinne ist, da Sie nicht absolut homogen ist: $\|2 \cdot 1\|_0 = 1 \neq 2 = 2 \cdot 0 = \|1\|_0$. Da diese in der verwendeten Literatur dennoch als l_0-Norm bezeichnet wird, wird diese Bezeichnung hier übernommen. Die Norm-Eigenschaften werden ohnehin nicht benötigt.

Sparse Coding mit der l_1-Norm In der Literatur sind verschiedene Fassungen des Grundproblems zu finden. Eine häufig genutzte Alternative ersetzt die l_0-Norm durch die l_1-Norm und nutzt statt der Sparsity-Einschränkung einen Regularisierungsterm:

$$\underset{D,\{a_p\}}{\operatorname{argmin}} \sum_{p=1}^{P} \|x_p - Da_p\|^2 + \lambda \|a_p\|_1 \ , \lambda > 0$$

Dieses Problem ist ebenfalls kein konvexes Problem, sondern nur in beiden Variablen einzeln konvex. Der Vorteil dieser Formulierung ist es, andere Optimierungsverfahren verwenden zu können. In dieser Arbeit wird jedoch die Formulierung als $l_0 - Norm$-Problem gewählt. Diese ergibt eine Approximation der Bilder durch nur wenige Atome K, die dann besonders aussagekräftig sein sollen. Diese werden dann in einem späteren Schritt für die Klassifikation der Bilder genutzt.

3.2 Problemlösung

Es wurden bereits viele verschiedene Algorithmen getestet und teilweise zielgerichtet entwickelt, um das Optimierungsproblem 3.1 zu lösen. Wie bereits erwähnt handelt es sich um kein konvexes Optimierungsproblem, was eine globale Lösung kompliziert macht. Stattdessen befolgen alle bekannten Algorithmen den Ansatz, die beiden Unterprobleme 'Dictionary' und 'Code' alternierend (approximativ) zu lösen.
In dieser Arbeit wird das Code-Problem durch den (orthogonalen) Matching Pursuit und das Dictionary-Problem durch den K-SVD Algorithmus gelöst, wie dies auch in dem zugrunde liegenden Paper *Fast Rotational Sparse Coding* [6] der Fall ist und in der betrachteten Referenz *K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation* [2] vorgeschlagen wird. Die hier vorgestellten Algorithmen lösen das grundlegende Sparse Coding Problem 3.1 und es wird sich zeigen, dass die später vorgestellten transformationsinvarianten Formulierungen durch angepasste Versionen derselben Algorithmen lösbar sein werden.

3.2.1 K-SVD

Der K-SVD Algorithmus ist eine 2005 von den Autoren Michal Aharon Michael Elad Alfred Bruckstein vorgestellte Verallgemeinerung des K-Means Algorithmus [1]. Der Algorithmus und alle vorgestellten Erkenntnisse dieses Kapitels beziehen sich auf eine erweiterte Veröffentlichung derselben Autoren aus dem Jahr 2006 [2].
Die generelle Idee des K-SVDs ist, dass ein Atom mehrere Bilder beschreiben muss. Daher werden die Bilder als Vektoren spaltenweise zu einem Array zusammengefasst und

anschließend per Singulärwertzerlegung der erste Singulärvektor des Arrays bestimmt. Die Idee dahinter ist, dass dieser Vektor nun die beste Beschreibung für die Bilder ist, da er die Hauptkomponente widerspiegelt. Der endgültige Algorithmus ist jedoch etwas komplizierter, da es mehrere Atome gibt. Das heißt zum einen, dass nicht jedes Atom jedes Bild beschreiben muss und zum anderen, dass pro Bild K Atome summiert werden.

Beschreibung des K-SVD Der K-SVD Algorithmus (siehe Alg. 1) ist im Wesentlichen ein Algorithmus zum Updaten des Dictionarys. Das Initialisieren der Atome $D_m \in \mathbb{R}^{N^2}, m \leq M$, (Zeile 2) und das Updaten des Codes $a_p \in \mathbb{R}^M, p \leq P$, (Zeile 4) wird zwar der Vollständigkeit halber in dem Algorithmus mit aufgeführt, kann aber durch andere Initialisierungen beziehungsweise Code-Updates ausgetauscht werden.

Algorithm 1 K-SVD

Input: $\{x_p\}_{p \leq P}, M, K$
Output: a, D

```
1:  function K-SVD({x_p}_{p≤P}, M, K)
2:      D ← M randomly initialized atoms
3:      for for fixed number of iterations do
4:          {a_p}_{p≤P} ← OMP({x_p}_{p≤P}, D, K)
5:          for m = 1 to M do
6:              Y ← [ ]
7:              for p = 1 to P do
8:                  if a_p[m] ≠ 0 then
9:                      b ← a_p
10:                     b[m] ← 0
11:                     Y ← [Y (x_p − Db)]
12:                 end if
13:             end for
14:             D_m ← first singular vector of Y
15:         end for
16:     end for
17:     return a, D
18: end function
```

Der Hauptteil des Algorithmus (ab Zeile 5) bezieht sich also auf das Update des Dictionarys. Zur Reduzierung der Komplexität dieses Problems werden die Atome des Dictionarys iterativ einzeln aktualisiert (ab Zeile 5). Für den Fall, dass die Sparsity $K = 1$ ist, stellt dies keine Einschränkung dar. Die Atome können (und sollen) mehrere Bilder gleichzeitig repräsentieren, daher muss zur Aktualisierung des Atoms nun zunächst einmal bestimmt werden, zur Repräsentation welcher Bilder das Atom eingesetzt wird (Zeile 7 & 8). Dann wird der Einfluss der anderen Atome auf jene Bilder herausgerechnet. Aufgrund der Tatsache, dass die anderen Atome zum Aktualisieren fest gehalten werden, geschieht dies durch ein simples Bilden des Residuums zwischen dem jeweiligem

Bild und der jeweiligen Rekonstruktion des Bildes durch die anderen Atome (Zeile 9 bis 11). Nach Durchlauf aller Bilder kann, wie oben bereits angedeutet, das Atom optimiert werden, indem der erste Hauptvektor aus den Residuen berechnet wird (Zeile 14).
Dieses Verfahren wird wiederholt, bis ein Abbruchkriterium erreicht wird. Das Abbruchkriterium ist hier durch eine feste Anzahl an Durchläufen ersetzt worden. Dies vereinfacht den Algorithmus und hat sich für die weiteren Aufgaben als ausreichend erwiesen.

Implementierungsdetails Der K-SVD ist wie der K-Means Algorithmus anfällig dafür, in lokalen Minima zu enden, die teilweise nicht besonders gut sind. Ein simpler Trick ist es, Atome, die nur wenige Bilder repräsentieren, neu zu initialisieren. Dazu werden Bilder verwendet, die besonders schlecht approximiert und damit unterrepräsentiert sind. Die Umsetzung dieses Tipps zeigte, dass dies in den ersten Iterationen häufig vorkommt, in späteren Iterationen jedoch nur noch höchst selten auftritt.
Wie anfangs erwähnt, kann die Form der Initialisierung der Atome frei gewählt werden. Die eigene Implementierung hat gezeigt, dass eine Initialisierung der Atome mit zufälligen Werten dafür sorgt, dass viele Atome ignoriert werden. Deutlich besser ist es, für jedes Atom einige Bilder zufällig zu wählen und das jeweilige Atom auf den ersten Singulärvektor dieser Bilder zu setzen. Dies entspricht im Wesentlichen einem zufälligen Initialisieren des Codes mit anschließendem Dictionary-Update, wobei hier nicht darauf geachtet wird, dass jedes Bild entsprechend der Regel $\|a_p\|_0 \leq K \; \forall p$ gesetzt wird. Diese Initialisierung ist fast genauso schnell wie eine zufällige Initialisierung (da nur wenig Bilder ausgewählt werden), führt jedoch dazu, dass in der ersten Iteration des Code-Updates die Atome deutlich gleichmäßiger ausgewählt werden.

Beziehung zwischen K-SVD und K-Means Wie eingangs erwähnt ist der K-SVD Algorithmus eine Verallgemeinerung von Lloyd's K-Means Cluster Algorithmus. Setzt man die Sparsity $K = 1$ (hier muss eigentlich gelten $\|a_p\|_0 = K$, der Fall $\|a_p\|_0 = 0$ ist als Lösung aber ohnehin nur theoretisch für einige spezielle Konstruktionen möglich) und erlaubt als Koeffizienten für die Einträge in a_p nur den Wert 1, so entsprechen die beiden Algorithmen einander.
Was bedeutet dies nun für den K-SVD Algorithmus? Zum einen liegen Clusterzentren exakt in der Mitte ihrer zugeordneten Punkte (dies gilt für Lloyd's Algorithmus, es gibt Abwandlungen in denen die Clusterzentren zum Beispiel über den Median statt den Mittelwert berechnet werden. Die Idee dahinter ist allerdings immer gleich). Atome hingegen sollen nur Richtungen (mathematisch: die Hauptkomponente) der zugeordneten Bilder (im Fall $K = 1$) darstellen, daher sind im K-SVD beliebige Koeffizienten erlaubt. Zum anderen wird im K-Means jeder Punkt genau einem Clusterzentrum zugeordnet, im K-SVD dagegen können Bilder K Atomen zugeordnet werden. Damit spiegeln Atome nur Anteile (Residuen) der zugeordneten Bilder wieder.

3.2.2 (Orthogonaler) Matching Pursuit

Der Matching Pursuit ist ein Algorithmus, der mithilfe eines bekannten Dictionarys gegebene Daten unter Einhaltung der Sparsity K möglichst gut approximiert. Der Al-

gorithmus findet für den Fall $K > 1$ jedoch keine globale Lösung, da er das komplexe Problem in Teilprobleme aufteilt, indem er die Atome schichtweise auswählt. Pati, Rezaiifar und Krishnaprasad [8] haben 1993 den Algorithmus um das Orthogonalisieren erweitert, sodass die resultierende Approximation eine Projektion der Daten auf den Unterraum der verwendeten Atome darstellt.

Matching Pursuit Das globale Lösen des Code-Updates (entspricht Optimierungsproblem 3.1 mit festem Dictionary D) ist vor allem für große K ein zu komplexes Problem. Daher werden die Atome für jedes Bild iterativ einzeln ausgewählt. Im ersten Schritt wird per Skalarprodukt für jedes Bild das Atom bestimmt, welches am ähnlichsten (d.h. das größte absolute Skalarprodukt besitzt) ist. Dafür ist wichtig, dass die Atome normalisiert sind (denn die Koeffizienten, mithilfe derer die Bilder durch die Atome approximiert werden, sind beliebig). Dieses wird fortan zur Beschreibung des Bildes verwendet. Im zweiten Schritt wird versucht das Residuum mit dem Rest des Dictionarys zu beschreiben. Wieder wird das Atom mit dem größten absoluten Skalarprodukt (hier mit dem Residuum) ausgewählt. Dieses Verfahren wird solange wiederholt, bis die gewünschte Sparsity K erreicht ist.

Orthogonaler Matching Pursuit Dieser soeben beschriebene Matching Pursuit entspricht dem Algorithmus 2 ohne Zeile 7. Die Erweiterung des Algorithmus um diese Zeile, dann orthogonaler Matching Pursuit genannt, sorgt für eine bessere Konvergenz. Dazu muss das Optimierungsproblem

$$\underset{\hat{a}_p}{\operatorname{argmin}} \|x_p - \hat{D}\hat{a}_p\|$$

für alle Bilder x_p gelöst werden. Dabei steht $\hat{D}$ für die Spalten des Dictionarys (also Atome) die zu Einträgen ungleich Null des jeweiligen Codes a_p korrespondieren. $\hat{a}_p$ stellt entsprechend jene Einträge ungleich Null dar.
Die Lösung des Optimierungsproblems ist die orthogonale Projektion der verwendeten Atome auf das Bild. Damit ist dieser Unterraum des Bildes, der durch die verwendeten Atome aufgespannt wird, bereits exakt approximiert. Dies ist im ursprünglichen Matching Pursuit Algorithmus leider nicht der Fall, da die Koeffizienten unabhängig und nacheinander bestimmt werden.
Im Gegensatz zum ursprünglichen Code-Update Problem ist die Komplexität dieses Problems durch die Reduktion der Dimensionalität nun niedrig genug, sodass dieses lineare Optimierungsproblem durch klassische Verfahren, wie das QR-Verfahren, gelöst werden kann.
Das Orthogonalisieren bringt zwar eine bessere Konvergenz, geht jedoch aufgrund des zusätzlich eingebauten linearen Problems zulasten der Laufzeit. Für das Basisproblem des Sparse Codings ist die OMP Variante aufgrund der deutlich besseren Konvergenz jedoch meist zu bevorzugen. Dies wird unter anderem durch ein Beispiel in dem Paper [8] gestützt, die die die Konvergenz der beiden Algorithmen in abhängig von den Rechenkosten (in Flops) verglichen haben (siehe Abb. 3.1). Im Verlauf der Arbeit wird sich

Algorithm 2 Orthogonal Matching Pursuit

Input: $\{x_p\}_{p \leq P}, D, K$
Output: $\{a_p\}$

function OMP($\{x_p\}_{p \leq P}, D, K$)
  for $i = 1$ to P **do**
    $residual \leftarrow x_p$
    for $k = 1$ to K **do**
      $m^* \leftarrow \text{argmax}_m \, |\langle residual, D[m] \rangle|$
      $a_p += \langle residual, D[m^*] \rangle \cdot e_{m^*}$
      $a_p \leftarrow orthogonalize(a_p, D, x_p)$
      $residual \leftarrow x_p - Da_p$
    end for
  end for
  return $\{a_p\}$
end function

zeigen, dass im Convolutional Matching Pursuit das Orthogonalisieren das Lösen des (fast) gleichen linearen Optimierungsproblems darstellt, dieses jedoch aufgrund der hohen Anzahl der nichtnegativen Einträge einen deutlich höheren Einfluss auf die Laufzeit darstellt. Daher wird dieses nicht mehr in jeder Iteration durchgeführt. Davon wird erhofft, dass einerseits Teileffekte der guten Konvergenz beibehalten werden, andererseits die Laufzeit nicht allzu stark beeinträchtigt werden.

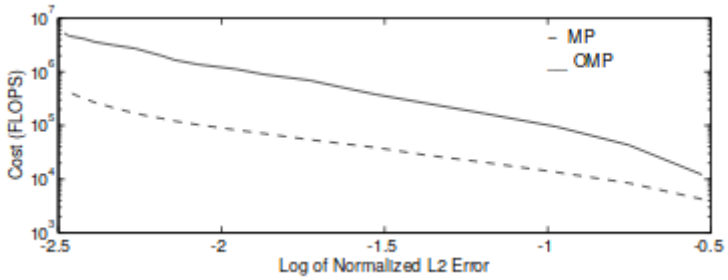


Abbildung 3.1: Beispiel der besseren Konvergenz des OMP gegenüber MP (aus [8])

Abschließend sei noch darauf hingewiesen, dass im ersten Durchlauf jeden Bildes das Orthogonalisieren keinen Effekt hat. Dies ist insbesondere in der Komplexitätsanalyse wichtig, da der Fall $K = 1$ durchaus Relevanz hat, wie sich später zeigen wird.

3.3 Invariante Formulierungen

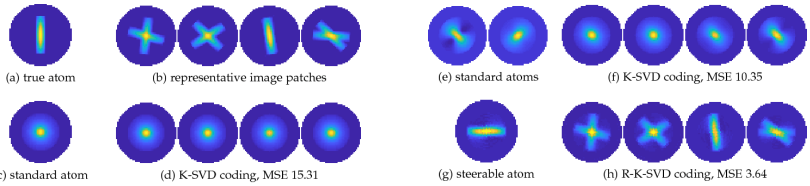
Eine sinnvolle Erweiterung des Grundproblems 3.1 ist Transformationsinvarianz. Eine transformationsinvariante Formulierung des Sparse Codings lautet:

$$\underset{D,\{a_{p,r}\}}{\operatorname{argmin}} \sum_{p=1}^{P} \left\| x_p - \sum_{r=0}^{R-1} T_r(D) a_{p,r} \right\| mit \sum_{r=0}^{R-1} \|a_{p,r}\|_0 \leq K \ \forall p \tag{3.2}$$

Dabei stellt T_r eine beliebige Folge von Transformationen dar. Zu beachten ist, dass nun die Atome nicht mehr zwingend dieselbe Größe wie die zu rekonstruierenden Bilder aufweisen müssen, sondern diese Bedingung nun für die transformierten Atome gilt: $T_r(D) \in \mathbb{R}^{N^2}$.
Die bildliche Idee dazu ist, dass ein Objekt nicht immer an genau derselben Stelle im Bild auftaucht, sondern sich gleiche Objekte in verschiedenen Bildern durch Ort und Ausrichtung unterscheiden können. Letztendlich handelt es sich jedoch um dieselben Objekte, das heißt gemäß der Idee des Sparse Codings sollte es möglich sein, sie durch dieselben Atome zu repräsentieren.
Bilder repräsentieren dreidimensionale Objekte zweidimensional. Das heißt es gibt viele Transformationen, die die dritte, nicht durch das Bild erfasste, Dimension beinhalten. Diese Transformationen zu betrachten führt zu einem sehr komplexen Problem, das hier nicht betrachtet wird. Betrachtet man die Rotationen die im Zweidimensionalen auftauchen, wird das Ganze jedoch weitaus simpler: Dort gibt es die Translation, die Rotation und Skalierung. In dieser Arbeit werden Translations- und Rotationsinvarianz betrachtet.

3.3.1 Rotationsinvariantes Sparse Coding

Rotationsinvarianz bedeutet, den Algorithmus robust gegen Drehungen im Bild zu machen. Dies ist wichtig, da Objekte im Bild nicht immer mit der gleichen Orientierung zu sehen sind. Dies kann zum Beispiel durch das Drehen der Kamera oder durch die Bewegung des entsprechenden Objekts geschehen.
Ist ein Objekt in zwei Bildern zu sehen, in einem davon jedoch zum Beispiel um 90 Grad gedreht, so sollen beide Objekte im Sinne des Sparse Codings durch dasselbe Atom repräsentiert werden. Auch bei Überlagerungen eines Objekts mit verschiedenen Rotationen sollte dies als mehrmalige Anwendung eines einzigen Atoms in verschiedenen Orientierungen repräsentiert werden.
Ein einfaches Beispiel dazu liefert das später diskutierte Paper *Fast Rotational Sparse Coding* [6] (siehe Abbildung 3.2). Dort sieht man sehr deutlich die Unfähigkeit des klassischen Sparse Codings Rotationen von Objekten darzustellen. Die Grafik zeigt in b) Bilder, die durch kombinierte Rotationen des Atoms aus a) erzeugt wurden. Man sieht, dass Sparse Coding weder mit einem [vgl. c)] noch mit zwei [vgl. e)] Atomen in der Lage ist, dieses originale Atom zu rekonstruieren. Dementsprechend schlecht ist die Qualität der rekonstruierten Bilder [d) bzw. f)]. Mit einem rotationsinvarianten Ansatz und sogenannten 'lenkbaren Atomen' [g)] lassen sich die originalen Bilder rekonstruieren



a) b) Originalatom
c) d) Standard-SC-Atom
e) f) 2 Standard-SC-Atome
g) h) lenkbares Atom

Abbildung 3.2: Atome (links) mit jeweiligen (re-)konstruierten Bildern (aus [6])

[siehe h)]. Dies bedeutet, dass das Atom in beliebiger Rotation eingesetzt werden darf. Es kann also gewissermaßen in die korrekte Richtung 'gelenkt' (im englischen Original [6] 'steerable') werden.
In der Praxis ist eine Invarianz für beliebige Rotationen zu komplex, stattdessen wird das Problem, wie in der Bildverarbeitung üblich, diskretisiert. Die Transformationen der transformationsinvarianten Formulierung 3.3 werden durch Rotationen spezifiert, sodass man das folgende Optimierungsproblem erhält:

$$\underset{D,\{a_{p,r}\}}{\operatorname{argmin}} \sum_{p=1}^{P} \left\| x_p - \sum_{r=0}^{R-1} R_r D a_{p,r} \right\| mit \sum_{r=0}^{R-1} \|a_{p,r}\|_0 \leq K \ \forall p \tag{3.3}$$

Hier ist R_r eine $N^2 \times N^2$ Matrix, die Drehungen um den Winkel $\frac{2\pi r}{R}$ ausführt. Diese Formulierung ist nun invariant in dem Sinne, dass man einzelne Originalbilder drehen kann (um abgebildete Winkel $\frac{2\pi r}{R}$), ohne den Rekonstruktionsfehler zu verändern (Rotationen r werden modulo R gerechnet):

$$\begin{aligned}
\left\| R_{r_0} x - \sum_{r=0}^{R-1} R_r D a_r \right\| &= \left\| R_{r_0} x - R_{r_0} \sum_{r=0}^{R-1} R_{r-r_0} D a_r \right\| \\
&= \left\| R_{r_0} \left(x - \sum_{r=0}^{R-1} R_{r-r_0} D a_r \right) \right\| \\
&= \left\| x - \sum_{r=0}^{R-1} R_{r-r_0} D a_r \right\| \\
&= \left\| x - \sum_{r=0}^{R-1} R_r D a_{r+r_0} \right\|
\end{aligned}$$

Voraussetzung für diese Gleichungskette, dass die Rotationsmatrix unitär ist, was in der dritten Gleichung genutzt wird. In der ersten, zweiten und letzten Gleichung wird lediglich die Linearität der Matrixmultiplikation benötigt. Die Gleichungskette zeigt, dass der Rotationsfehler invariant ist, da ein rotiertes Bild $R_{r_0} x$ mithilfe des gleichen Fehlers wie das Originalatom x dargestellt werden kann. Dazu wird statt des Atoms a_r das Atom

a_{r+r_0} verwendet. Dies entspricht der intuitiven Vorstellung, dass gedrehte Bilder durch entsprechend gedrehte Atome dargestellt werden müssen.
In der Literatur sind verschiedene Ansätze zu finden, die sich dem rotationsinvarianten Sparse Coding widmen. Folgt man dem Prinzip, das Lösen des Optimierungsproblems in die beiden Subprobleme Codeupdate und Dictionaryupdate zu unterteilen, so muss die Rotationsinvarianz in beiden Subproblemen genau einmal beachtet werden. Im Codeupate muss jede (erlaubte) rotierte Version aller Atome als mögliches zu verwendendes Atom betrachtet werden. Im Dictionaryupdate muss beachtet werden, dass, falls eine rotierte Version eines Atoms verwendet wurde, dies im Update-Schritt des Atoms entsprechend Berücksichtigung findet. Beide Herausforderungen sind relativ simpel, haben jedoch enormen Einfluss auf die Laufzeit des Algorithmus.

Rotationsinvariantes Sparse Coding mittels lenkbarer Basis

In diesem Unterkapitel wird der Ansatz von McCann, Unser und Depreursinge vorgestellt, den sie im Juni 2018 im Paper *Fast Rotational Sparse Coding* [6] veröffentlicht haben. Die Grafiken aus diesem Kapitel stammen ebenfalls aus dem Paper.
Grundsätzlich nutzen sie den K-SVD Algorithmus für das Sparse Coding. Um diesen rotationsinvariant zu gestalten, verändern sie diesen an zwei Stellen. Dies geschieht beides innerhalb des K-SVD Algorithmus, beeinflusst jedoch sowohl Code- als auch Dictionary-Update.
Zunächst wird das Dictionary vor dem Matching Pursuit um rotierte Versionen der Atome erweitert (siehe Alg. 3, Zeile 4). Somit erreicht man, dass im Matching Pursuit automatisch die rotierten Versionen aller Atome berücksichtigt werden. Es wird also eine gewisse Rotationsinvarianz des Code-Updates erreicht, jedoch eingeschränkt auf die zur Verfügung gestellten Rotationen.
Anschließend muss dann im Dictionary-Update beachtet werden, dass mehrere Elemente des zuvor erweiterten Dictionarys zum demselben Atom zugehörig sind. Es müssen daher für jedes Atom alle Verwendungen in beliebiger Rotation gefunden werden (Alg.3, Zeile 8 - 9). Für jeden Einsatz dieser Atome soll nun also ein Eintrag zu der Residuenliste hinzugefügt werden. Im Gegensatz zu dem ursprünglichen K-SVD Algorithmus kann ein Atom nun in verschiedenen Ausrichtungen mehrmals für dasselbe Bild zur Rekonstruktion eingesetzt werden. Herausgerechnet wird zum Update nur der Einsatz der jeweiligen Rotation (Alg.3, Zeile 11). Dies verhindert zwar erneut eine globale Lösung, folgt aber wie zuvor dem Ansatz, alle anderen Atome fest gesetzt zu lassen um das betrachtete Atom zu verbessern. Denn um das Atom schließlich zu updaten, ist die Ausrichtung des Atoms von elementarer Bedeutung. Dies wird beim Hinzufügen des Atoms zur Liste der Residuen beachtet, indem das Residuum dort in umgekehrter Richtung gedreht wird (siehe Alg. 3, Zeile 12).
Die Eingriffe, die in den K-SVD nötig sind, betreffen insgesamt also nur wenige Zeilen des Algorithmus. Jedoch wirken diese sich aufgrund gleich zweier Faktoren sehr negativ auf die Laufzeit aus. Zunächst wirkt sich die Aufblähung des Dictionarys negativ auf die Laufzeit des Matching Pursuit aus. Da dort zum Finden des besten Atoms stets alle Atome berücksichtigt werden, erhöht sich die Laufzeit des Matching Pursuits um

Algorithm 3 Rotational K-SVD

Input: $\{x_p\}_{p\leq P}, M, K$
Output: a, D

function R-K-SVD($\{x_p\}_{p\leq P}, M, K$)
  $D \leftarrow M$ randomly initialized atoms
  for for fixed number of iterations **do**
    $\hat{D} \leftarrow [R_0 D\ R_1 D \dots R_{R-1} D]$
    $\{a_{p,r}\}_{p\leq P, r\leq R} \leftarrow OMP(\{x_p\}_{p\leq P}, \hat{D}, K)$
    for $m = 1$ to M **do**
      $Y \leftarrow [\,]$
      for $p = 1$ to P **do**
        if $a_{p,r}[m] \neq 0$ for some $r = r_0$ **then**
          $b_r \leftarrow a_{p,r} \forall r$
          $b_{r_0}[m] \leftarrow 0$
          $Y \leftarrow [Y\ R_{-r_0}(x_p - \hat{D}b)]$
        end if
      end for
      $D \leftarrow$ first singular vector of Y
    end for
  end for
  return a, D
end function

den Faktor O(R). Glücklicherweise ist zumindest das laufzeitintensive Orthogonalisieren des Orthogonalen Matching Pursuits einzig von der Sparsity und nicht von der Größe des Dictionarys abhängig. Außerdem ist vielfaches Rotieren der Atome und Residuen nötig. Die Atome werden zur Erweiterung des Code-Updates rotiert, die Residuen im Dictionary-Update Schritt. Der wesentliche Beitrag des Papers [6] ist die Art und Weise, wie sie das Rotieren beschleunigen.

Diskrete lenkbare Basis Die generelle Idee ist, die Rotationen zu diagonalisieren. Dies geschieht mittels $R_r = \Phi S_r \Phi^*$, wobei S_r eine Diagonalmatrix und Φ ein Basiswechselmatrix ist. Die Matrix S_r wird 'lenkbare Matrix' (im Original: 'steering matrix') genannt. Von der Basiswechselmatrix Φ wird gefordert, dass sie unitär ist. Damit verschwindet diese aus dem Fehlerterm, da sie die letzte Gleichheit der folgenden Gleichungskette

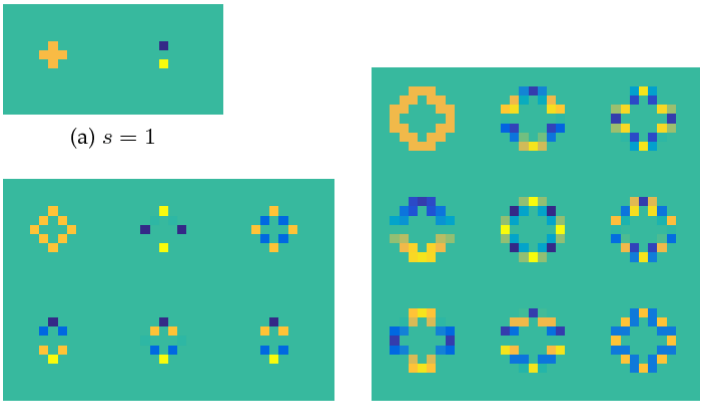
ermöglicht:

$$\begin{aligned}
\left\| x - \sum_{r=0}^{R-1} R_r D a_r \right\| &= \left\| x - \sum_{r=0}^{R-1} \Phi S_r \Phi^* D a_r \right\| \\
&= \left\| x - \Phi \sum_{r=0}^{R-1} S_r \Phi^* D a_r \right\| \\
&= \left\| \Phi \left(\Phi^* x - \Phi \sum_{r=0}^{R-1} S_r \Phi^* D a_r \right) \right\| \\
&= \left\| \hat{x} - \sum_{r=0}^{R-1} R_r \hat{D} a_r \right\|
\end{aligned}$$

$\hat{D} = \Phi^* D$ und $\hat{x} = \Phi^* x$ entsprechen dem Dictionary beziehungsweise den Bildern, jedoch dargestellt mithilfe einer anderen Basis. Die Matrix Φ wird im Folgenden beschrieben.

$$\phi_{s,t}(r,\theta) = \begin{cases} e^{jt\theta} & \text{if } \dfrac{N(s-1)}{2S} \leq r \leq \dfrac{Ns}{2S}, \\ 0 & \text{otherwise} \end{cases}$$

Wie der Bedingung der Funktionsdefinition leicht zu entnehmen ist, ist diese Funktion bis auf einen Ring überall null. Das Skalarprodukt zweier Funktionen kann demnach nur dann ungleich null sein, wenn diese auf demselben Ring liegen, also die erste Koordinate ('s') übereinstimmt. Auf dem Ring entsprechen sie dem Orthonormalsystem $e_t(\theta) = e^{jt\theta}$. Damit sind die Funktionen auch bezogen auf den Doppelindex s, t orthonormal.
Ein paar visualisierte Funktionswerte kann man Abbildung 3.3 entnehmen.



(a) $s = 1$

(b) $s = 2$

(c) $s = 3$

Abbildung 3.3: Diskrete lenkbare Basis mit $S = 3$ und $T_1 = 1$, $T_2 = 5$, $T_3 = 8$. Nur der Realteil mit nichtnegativen diskreten Frequenzen wird gezeigt. (aus [6])

Um diese Funktionen zu erstellen, werden diese auf einem Gitter G berechnet.

$$G = \left\{ -\frac{N-1}{2}, \ldots, \frac{N-1}{2} \right\} \times \left\{ -\frac{N-1}{2}, \ldots, \frac{N-1}{2} \right\}$$

Es gilt $\phi_{s,t}[k] = \phi_{s,t}(\|k\|, \angle k)$. Der Vektor $(\phi_{s,t}[k])_k$ wird normalisiert und spaltenweise gestapelt.

$$\Phi = [\phi_{1,-T_1} \ldots \phi_{1,T_1} \ \phi_{2,T_2} \ldots \phi_{S-1,T_{S-1}} \ \phi_{S,-T_S} \ldots \phi_{S,T_S}]$$

Dies ist die gewünschte Basiswechselmatrix Φ. Mithilfe dieser lassen sich Rotationen diagonalisieren. Aus der Funktionsdefinition folgt unmittelbar:

$$\phi_{s,t}(r, \theta - \theta_0) = e^{-jt\theta_0} \phi_{s,t}(r, \theta)$$

Die Rotation ist somit nur noch von der Spalte t und nicht mehr von der Zeile s abhängig. Damit erhält man die lenkbare Matrix S_r:

$$S_r = diag\left(\left[e^{-j(-T_1)\frac{2\pi r}{R}} \ldots e^{-jT_1\frac{2\pi r}{R}} \ e^{-j(-T_2)\frac{2\pi r}{R}} \ldots e^{-j(-T_S)\frac{2\pi r}{R}} \ldots e^{-j(-T_S)\frac{2\pi r}{R}} \right] \right)$$

Die drei Autoren weisen in ihrem Paper [6] zudem auf einige technische Hinweise hin. Zunächst muss das Maximum der diskreten Frequenzen $\{T_s\}$ niedrig genug gehalten werden um Aliasing zu vermeiden. Das vorgeschlagene Setzen von T_s als den halben Umkreis des s-ten Rings, gemessen in Pixeln, erweist sich als passend. Auch die Wahl von $S = \lfloor \frac{N}{2} \rfloor$ erweist sich als gut.
Ein letzter wichtiger Hinweis ist, dass die Spaltenanzahl von Φ kleiner als N^2 ist. Es wird nur ein Unterraum von $\mathbb{C}^{N^2}$ dargestellt, was bedeutet, dass es einen Approximationsfehler geben wird. Dies liegt einerseits daran, dass die kreisförmigen Basismatrizen (vgl. Abbildung 3.3) die Ecken der Bilder nicht in den neuen Raum übertragen und andererseits an dem Diskretisierungsfehler. Um dem Diskretisierungsfehler abzuschwächen, wird $\hat{\Phi} = (\Phi^*\Phi)^{-1}\Phi^*$ verwendet, was die Approximation etwas verbessert. Im Paper [6] werden Φ und $\hat{\Phi}$ 'steerable biorthogonal pair of bases' genannt.
Schlussendlich ergibt sich so folgender Algorithmus:

1.) Erzeuge die lenkbare Basis Φ

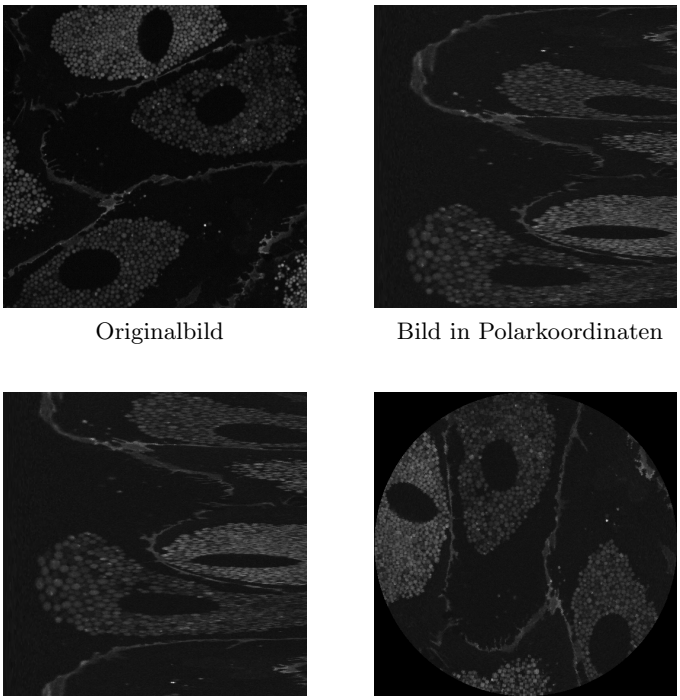
2.) Berechne die Koordinaten des Originalbilder bezüglich Φ, d.h. $\hat{x}_p = (\Phi^*\Phi)^{-1}\Phi^* x_p$

3.) Führe den R-K-SVD Algorithmus auf $\hat{x}_p$ aus. Beachte, dass die Rotationsmatrizen R_r durch die Diagonalmatrizen S_r ersetzt werden.

Der Algorithmus liefert die Repräsentation des Dictionarys in der lenkbaren Basis wieder. Will man die entstandenen Atome als Bild betrachten, sollten diese daher mittels $D = \Phi\hat{D}$ zurücktransformiert werden. Für die Klassifikations- und Detektionsaufgaben wird dies nicht nötig sein, da dort nur der Code von Interesse ist. Dieser bleibt von den Basiswechseln unberührt.

Rotationsinvariantes Sparse Coding durch Polartransformation

Den entscheidenden Ansatz, den die drei Autoren McCann, Unser und Depeursinge in ihrem Paper 3.3.1 beschreiben, ist im Wesentlichen die polare Fouriertransformation. Mit dem Wissen wird hier nun ein Ansatz präsentiert, der mittels der Fouriertransformation im Polarraum auch das Code-Update beschleunigt.



Originalbild | Bild in Polarkoordinaten

Rotiertes Bild in Polarkoordinaten | Rotiertes Bild

Abbildung 3.4: Rotieren eines Zellbildes um 70° mittels Polarkoordinaten

Statt in der lenkbaren Basis werden die Bilder nun in Polarkoordinaten (r, θ) dargestellt. Somit ist direkt ersichtlich, dass ein Rotieren des Bildes lediglich eine Verschiebung in der zweiten Koordinate ist. Damit ist der Dictionary-Update Schritt bereits sehr vereinfacht. Statt die Residuuen drehen zu müssen, reicht es nun bereits das Array entsprechend entlang der zweiten Koordinatenachse zu verschieben(vgl. Abbildung 3.4).
Dies schafft die Möglichkeit, die Rotationsinvarianz direkt im Codeupdate mit zu berücksichtigen, ohne das komplette Dictionary vervielfachen zu müssen (wie dies in Alg. 3, Zeile 4 geschieht). Im (orthogonalen) Matching Pursuit wird das Skalarprodukt des Residuums mit allen Atomen gebildet. Für den rotationsinvarianten Ansatz bedeutet dies, das Skalarprodukt mit jeder rotierten Version aller Atome zu berechnen. Da Rotieren einer Verschiebung in der zweiten Dimension entspricht, muss also für jedes Atom die Faltung entlang der zweiten Koordinate berechnet werden. Da Faltung einer Multiplikation im Fourierraum entspricht, kann das Bilden des Skalarprodukts für jede Rotation in diesem Fall also durch eine einfache Multiplikation im Fourierraum ersetzt werden.

Algorithm 4 Rotationsinvarianter orthogonaler Matching Pursuit

Input: $\{x_p\}_{p \leq P}, D, K$
Output: $\{a_p\}$

1: **function** R-OMP($\{x_p\}_{p \leq P}, D, K$)
2: **for** $i = 1$ to P **do**
3: $residual \leftarrow x_p$
4: **for** $k = 1$ to K **do**
5: $\{impacts\}_{r \leq R, m \leq M} \leftarrow \{|residual * D[m]|^T\}_{m \leq M}$
6: $r^*, m^* \leftarrow argmax_{r,m}\{impacts\}_{r \leq R, m \leq M}$
7: $a_p[r^*, m^*] += impacts[r^*, m^*]$
8: $\hat{D} \leftarrow add\_necessary\_rotations(D, a_p)$
9: $a_p \leftarrow orthogonalize(a_p, \hat{D}, x_p)$
10: $residual \leftarrow x_p - \hat{D}a_p$
11: **end for**
12: **end for**
13: **return** $\{a_p\}$
14: **end function**

Die rotierten Varianten des Dictionarys müssen nun nur noch für die tatsächlich verwendeten Atome berechnet werden (Alg. 4, Zeile 8). Dies geschieht, wie oben beschrieben, über das Verschieben des Bildes in der zweiten Koordinate.
Obgleich die Idee für den rotationsinvarianten orthogonalen Matching Pursuit (Alg. 4) simpel ist, ist birgt sie Herausforderungen in der Notation. Zunächst muss in Zeile 5 nicht mehr nur das beste Atom gefunden werden, sondern gleichzeitig zweidimensional die beste Kombination aus Ausrichtung r und Atom m. Da nun nur noch einige Rotationen des Dictionarys berechnet werden, muss dies sowohl beim Orthogonalisieren (Zeile 9) als auch beim Bilden des Residuums (Zeile 10) berücksichtigt werden. Für die Notation bietet sich dabei zum Beispiel an, das Dictionary mit Nullvektoren aufzufüllen. In der Praxis sollte man die Berechnungen durch Verwendung entsprechend verkleinerter Code-Matrizen anpassen, um unnötige Berechnungen zu vermeiden.
Der K-SVD Algorithmus für diese Variante entspricht fast genau dem Algorithmus 3. Jedoch ist zum einen zu beachten, dass die Erweiterung des Dictionarys aus Zeile 4 dort nicht nötig ist. Dies muss jedoch entweder später durch Übergabe des erweiterten Dictionarys aus dem rotationsinvarianten orthogonalen Matching Pursuits oder durch Neuberechnung der verwendeten Rotationen der Atome nachgeholt werden. Beides ist jedoch deutlich weniger laufzeitintensiv, da nur einige wenige Rotationen berechnet werden müssen. Zum anderen muss beachtet werden, dass die Notation des Codes etwas anders aussieht, was sich insbesondere ab Zeile 9 bemerkbar macht. Inhaltlich ändert sich dort jedoch nichts.

Anzahl Rotationen Abschließend sei noch darauf hingewiesen, dass mit diesem Ansatz die Anzahl der Rotationen automatisch durch die Bildgröße festgelegt ist. Wird beispiels-

weise ein Bild der Größe 15×15 betrachtet, so werden automatisch 15 Rotationen (also ein Winkel von 24°) betrachtet, da die Bilder stets um einen Pixel verschoben werden.
Andere Anzahlen von Rotationen sind natürlich ebenfalls möglich, erfordern jedoch zusätzliche Arbeit. Die Reduktion der Anzahl um einen Faktor ist recht einfach: Man ignoriert einfach die anderen Rotationen. Da dies jedoch keinen Laufzeitvorteil bringt, ist dies allerdings wohl selten von Interesse. Das Hinzufügen von weiteren Rotationen hingegen bedarf Interpolation. Am einfachsten ist es, zu den bereits bestehenden Rotationen einen konstanten Faktor an Rotationen hinzuzufügen: Eine Verdoppelung wird zum Beispiel erreicht, indem anfangs einmal das Bild durch Interpolation verschoben wird und daraufhin beide Bilder mit der obigen Methode betrachtet werden.
In dieser Arbeit wurden auf solche Herangehensweisen, die die Anzahl an Rotationen verändern, verzichtet, da kein Bedarf gesehen wurde. Es sei jedoch noch darauf hingewiesen, dass dies als möglicher Vorteil des 'lenkbaren' Verfahrens gesehen werden könnte, da dort bereits beliebige Rotationen abgedeckt sind.

Komplexitätsanalyse

Dieses Unterkapitel dient einer groben Einschätzung der Komplexität der vorgestellten Algorithmen. Es wird das asymptotische Verhalten in Abhängigkeit der Eingangsvariablen in den Landau-Symbolen angegeben. Dies lässt jedoch keine zuverlässigen Rückschlüsse über die Laufzeit der Algorithmen zu, da konstante Faktoren wie das mehrmalige Ausführen einer teuren Operation einen weitaus größeren Einfluss haben können als einzelne Eingangsvariablen. Zum Beispiel wird die Komplexität auch in Abhängigkeit der Sparsity angegeben. Diese wird aber letztendlich meist sehr niedrig sein. Häufig gilt für die Sparsity $K = 1$.
Ein weiterer wichtiger Faktor ist konkrete Implementierung. So schafft es zum Beispiel das python-Paket 'numpy' mittels Vektorisierung von Operationen, welche dann sogar noch durch individuell kompilierten C-Code optimiert werden, Laufzeitverbesserungen von zwischen Faktor 50 und Faktor 100 zu erreichen. Nichtsdestotrotz wird hier nun die Komplexität betrachtet. Dies dient insbesondere dazu, auf kritische, weil teure, Operationen in den Algorithmen hinzuweisen.
Zunächst wird die Komplexität des OMP (Alg. 2) betrachtet. Die teuersten Schritte sind in den FOR-Loops die Zeilen 5 und 7. Das Berechnen des Skalarprodukts in Zeile 5 kostet pro Atom N^2 Multiplikationen und Additionen. Zeile 7 berechnet das Optmierungsproblem $argmin_{a_p} \|x_p - Da_p\|$, welches per Pseudoinverse eine Komplexität von $O(N^2k^2)$ besitzt, wobei N^2 die Länge der Bildvektoren ist und k die Sparsity in der jeweiligen Iteration. Damit ergibt sich insgesamt für I Aufrufe eine Komplexität von $O(IPKN^2M)$ für den Matching Pursuit und von $O(IPKN^2(K^3 + M))$ für den OMP.
Der K-SVD (Alg. 1) hat in seiner ursprünglichen Version viele For-Loops. Im Inneren dieser hat Zeile 11 selbst eine Komplexität von $K \cdot N^2$ für die Addition aller Atome, die Kosten für die Listenerweiterung werden hier nicht betrachtet, da sie mathematisch nicht ins Gewicht fallen. Damit erhalten wir insgesamt für die Durchläufe der Zeile 11 $O(IMPKN^2)$. Die Singulärwertzerlegung einer $m \times n$ Matrix kostet $O(mn(m + n)$. In Zeile 14 wird die Singulärwertzerlegung einer Matrix mit maximaler Dimension $P \times N^2$

(für maximal P Bilder stehen dort Bildvektoren der Länge N^2) berechnet, dies führt zu einer Komplexität von $O(N^4P)$ (wobei das P durch eine eher hohe Konstante geteilt wird, da Atome im Schnitt nur in $\frac{K}{M}$ eingesetzt werden). Insgesamt ergibt sich daraus für das Dictionary-Update (ohne Betrachtung des Code-Updates) eine Komplexität von $O(IMPN^4)$, wobei nur in etwa IPN^4 Multiplikationen wirklich durchgeführt werden.
Es zeigt sich, dass für die realistischen Parameter $K^3 < M$ und $K \cdot M \approx N^2$ die Komplexität von MP, OMP und K-SVD übereinstimmen. Daher ist ein Laufzeitgewinn des Code- und des Dictionary-Updates gleichermaßen wichtig. Das Einbeziehen von Rotationen wirkt sich auf den Matching Pursuit, aufgrund der Erhöhung der Anzahl an Atomen, linear aus. Auf die Komplexität des Dictionary-Updates wirken sich die Rotationen zwar nicht direkt aus, da die Anzahl der Atome für die Singulärwertzerlegung gleich bleibt. Allerdings kostet die Drehung einer Matrix durch Matrixmultiplikation N^3 Operationen in Zeile 11. Da diese Zeile ebenfalls sehr teuer ist, wird man die Auswirkung also auch dort merken.
Der polare Ansatz verbessert vor allem die Laufzeit des (O)MPs. Das Dictionary muss zunächst nicht um den Faktor R vergrößert werden. Dafür werden in der Zeile 5 des R-OMP (Alg. 4) in der Faltung alle Rotationen berücksichtigt. Mittels Fouriertransformation ist hier immerhin eine Beschleunigung möglich, sodass die Laufzeit des gesamten Algorithmus schließlich um einen Faktor $\frac{R}{const}, const > 1$ steigt.
Der lenkbare Ansatz verbessert die Laufzeit des OMP zwar nicht, dafür werden aber die Vielzahl an Rotationen schneller durchgeführt. Dies wirkt sich nicht auf die O-Notation des Algorithmus aus. Da allerdings alle Rotationen diagonalisiert sind und eine Laufzeit von N^2 statt N^3 aufweisen, ist auch hier eine deutlich kürzere Laufzeit als eine Verlängerung um den Faktor R zu erwarten.
Alles in allem kann man also erwarten, dass sich das Einbeziehen von Rotationen in den Algorithmen erheblich in der Laufzeit des Algorithmus zeigen wird. Da sowohl der K-SVD (Alg. 1) als auch der (O)MP (Alg. 2) in der Komplexität linear von der Anzahl der Atome abhängen, ist also als Obergrenze eine Laufzeitverlängerung um den Faktor R für R berücksichtigte Rotationen zu erwarten. In dieser Komplexitätsanalyse wurde jedoch gezeigt, dass sowohl der polare wie auch der lenkbare Ansatz die Algorithmen an entscheidenden Stellen beschleunigt, sodass zu erwarten ist, dass sich die Laufzeit nur um den Faktor $\frac{R}{const}, const > 1$ erhöht. Eine Entkopplung von Laufzeit und berücksichtigten Rotationen scheint jedoch unmöglich zu sein.

3.3.2 Translationsinvarianes Sparse Coding

Einen grundlegend anderen transformationsinvarianten Ansatz stellt das translationsinvariante Sparse Coding dar. Dieser ermöglicht es, Repräsentationen von Objekten unabhängig davon zu erlernen, wo diese sich im Bild befinden. Damit ist also insbesondere das Zentrieren der Objekte im Bild nicht mehr nötig. Die Translationsinvarianz wird meist erreicht, indem die Atome nicht dieselbe Größe wie das zu rekonstruierende Bild aufweisen, sondern deutlich kleiner sind. Sie sollen schließlich nur einzelne Objekte oder Strukturen des Bildes darstellen.

Problemformulierungen Es gibt mehrere äquivalente Möglichkeiten dieses Problem mathematisch präzise zu formulieren. Eine sehr intuitive Möglichkeit ist, die Atome $d_m \in D$ des Dictionarys in der Größe zu reduzieren, sodass diese deutlich kleiner sind als das zu reproduzierende Bild x_p. Das Bild erhält man nun als Summe über Faltungen $x_p = \sum_{m=0}^{M} d_m * a_{p,m}$. Der Code $a_{p,m}$ enthält nun also zusätzlich zu den jeweiligen Gewichtungen des Atoms die Information, an welchen Position(en) des Bildes das Atom eingesetzt wird.

$$\underset{D,\{a_{p,m}\}}{\operatorname{argmin}} \sum_{p=1}^{P} \left\| x_p - \sum_{m=0}^{M} d_m * a_{p,m} \right\| mit \sum_{m=0}^{M} \|a_{p,m}\|_0 \leq K \ \forall p \tag{3.4}$$

Eine äquivalente Formulierung des Problems ist in dem Paper [9] angedeutet. Diese Formulierung basiert darauf, dass man die verkleinerten Atome zunächst durch Zero-Padding auf die ursprüngliche Größe vergrößert. Dabei wird das Dictionary aufgebläht, sodass jede mögliche Postion des Atoms durch ein neues Atom im aufgeblähten Dictionary abgebildet ist. Verschiebt man das Atom von oben links beginnend stellenweise durchs Bild und schreibt die entstehenden Matrizen als Vektoren in das Dictionary, entsteht so eine Block-Toeplitz-Matrix. Daher wird für ein Bild $x_p \in \mathbb{R}^{k_1 \times k_2}$ aus einem Atom d_m der Größe $j_1 \times j_2$ ein Atom $\hat{d_m}$ der Größe $j_1 j_2 \times (k_1 - j_1) \cdot (k_2 - j_2)$.

$$\underset{D,\{a_{p,m}\}}{\operatorname{argmin}} \sum_{p=1}^{P} \left\| x_p - \sum_{m=0}^{M} \hat{d}_m a_{p,m} \right\| mit \sum_{m=0}^{M} \|a_{p,m}\|_0 \leq K \ \forall p \tag{3.5}$$

Diese Formulierung entspricht in etwa der Formulierung 3.1. Hier wird jedoch statt der Summe über die verschiedenen Transformationen des Dictionarys die Summe über die über die transformierten Atome gebildet. Eine Formulierung wie in 3.1 ist freilich ebenfalls möglich. Darauf wird hier allerdings verzichtet, da die im Folgenden präsentierten Algorithmen sich maßgeblich an den beiden vorangehenden Formulierungen orientieren.

Convolutional K-SVD Die Translationsinvarianz hat recht geringe Auswirkungen auf den K-SVD Algorithmus (Alg. 1, Kap. 3.2.1). Im K-SVD müssen die Zeilen 7 bis 13 angepasst werden. Dort wird bestimmt, welche Residuen durch das jeweilige Atom dargestellt werden sollen. Im originalen Algorithmus hat ein Atom stets gleichermaßen zu dem ganzen rekonstruierten Bild beigetragen. Nun repräsentiert jedes Atom nur noch (möglicherweise mehrere) Teile der Bilder. Dementsprechend muss nun geprüft werden, an welcher Positionen des Bildes das jeweilige Atom zum Einsatz gekommen ist. Das Residuum wird nun über den jeweiligen Bereich des Bildes berechnet und zur Liste hinzugefügt (wobei immer noch der Einfluss des betrachteten Atoms rausgerechnet wird).

Convolutional Matching Pursuit Der orthogonale Matching Pursuit Algorithmus (Alg. 2) muss ungleich mehr geändert werden. Zunächst einmal wird Zeile 5 ersetzt, da nun nicht mehr die Atome gesucht sind, welche das Bild als Ganzes am besten repräsentieren, sondern die Atome stets Teile des Bildes repräsentieren. Da eine globale Lösung laufzeittechnisch viel zu teuer ist, wird wieder ein Greedy-Algorithmus angestrebt. Da nun nicht

allein nach dem besten Atom gefragt wird, sondern auch nach der optimalen Position, wird das Skalarprodukt des Atoms mit jeder möglichen Position im Bild gebildet. Wie im Optimierungsproblem 3.3.2 bereits ersichtlich, entspricht dies der zweidimensionalen Faltung $d_m * a_{p,m}$.
Es wird die Atom-Position-Kombination ausgewählt, welche den höchsten Übereinstimmungswert ausweist. Im OMP (Alg. 2) wird vor der nächsten Zuweisung das Residuum neu bestimmt (vgl. Zeile 8). Dies führt in dem hier vorliegenden Fall jedoch zu erheblichen Laufzeitproblemen, da eine immens höhere Anzahl an Atomen benötigt wird, um das Bild zu rekonstruieren. Für das ursprüngliche Problem des Sparse Coding 3.1 reichen meist bereits wenige Atome (auch ein einzelnes Atom kann ausreichend sein), um eine gute Rekonstruktion zu ermöglichen (vgl. Ergebnisse in Kapitel 5.1). Nun decken Atome jedoch nur einen Teil des Bildes ab. Typischerweise werden Atomgrößen von 9 bis 17 verwendet. Im Fall von Atomen der Größe 15 kann man sich leicht ausrechnen, dass mindestens 900 (30 in der Breite $\cdot$ 30 in der Länge) Atome benötigt werden, um ein Bild der Größe 450×450 komplett abzudecken. Diese Herausforderung kann man meistern, wenn man dem Ansatz der Autoren Plaut und Giryes folgt. Diese stellen in ihrem Paper *A Greedy Approach to $l_{0,\infty}$ Based Convolutional Sparse Coding* (ab Seite 4 in [9]) heraus, dass in dem translationsinvarianten Fall nicht die Anzahl der Atome entscheidend ist, sondern die Anzahl an Überlappungen. Dazu führen sie die $l_{0,\infty}$-Norm ein (welche wie die l_0-Norm nicht wirklich eine Norm ist, jedoch zur Vereinfachung so genannt wird). Diese zählt im Gegensatz zur l_0-Norm nicht die Anzahl an Einträgen ungleich Null, sondern die die Anzahl an Einträgen ungleich Null pro Gebiet und bildet das Maximum über diese. Mit anderen Worten wird also die maximale Anzahl an Einträgen gesucht, die zu einem beliebigen Pixel beitragen. Somit beschränkt die folgende Formulierung des translationsinvarianten Sparse Coding Problems, welche in dieser Arbeit maßgeblich ist, effektiv die Anzahl der Atome die pro Pixel des Bildes genommen werden.

$$\underset{D,\{a_{p,m}\}}{\operatorname{argmin}} \sum_{p=1}^{P} \left\| x_p - \sum_{m=0}^{M} d_m * a_{p,m} \right\| mit \sum_{m=0}^{M} \|a_{p,m}\|_{0,\infty} \leq K \ \forall p \tag{3.6}$$

Für den Algorithmus bedeutet dies, dass für die Sparsity K wie im OMP (Alg. 2) K Berechnungen der Skalarprodukte nötig sind. Die Größe der Dimensionen des Codes $a_{p,m} \in \mathbb{R}^{L_1 L_2}$ hängt nun davon ab, ob man erlaubt, dass Atome an den Rändern der Bilder in abgeschnittener Form eingesetzt werden dürfen, das heißt gewissermaßen über die Bilder hinaus ragen. Dies hat den Vorteil, dass Teile von Objekten, die am Rand von Bildern zu sehen sind, durch entsprechende Teile der Atome dargestellt werden dürfen. Nachteilig ist jedoch, dass, sobald das Dictionary geupdatet wird, keine vollständige Grundwahrheit für den Residuenvektor bereitsteht. Beide Einflüsse sind jedoch gerade für größere Bilder als nicht sehr gravierend einzuschäten, da sie nur das Verhalten des Algorithmus an den Rändern des Bildes beeinflussen und für den Großteil des Bildes (dem Bildinnern) keine Rolle spielt. In dieser Arbeit wird aufgrund der Tatsache, dass es keine Knappheit an Daten gibt, die Variante gewählt, wo Atome am Rand auch nur als Teilatome verwendet werden dürfen, jedoch mit der Einschränkung, dass stets mindestens die Hälfte des Atomes verwendet wird. Somit gilt, falls j_1, j_2 ungerade sind,

Algorithm 5 Group Convolutional Orthogonal Matching Pursuit

Input: $\{x_p\}_{p\leq P}, D, K$
Output: $\{a_p\}$

function GCOMP($\{x_p\}_{p\leq P}, D, K$)
  for $i = 1$ to P **do**
    $residual \leftarrow x_p$
    for $k = 1$ to K **do**
      $b \leftarrow \left[corr(d_1, residual)^T \dots corr(d_M, residual)^T\right]^T$
      while $\max_{x,y} \|b[x, y]\| > 0$ **do**
        $best\_x, best\_y \leftarrow \text{argmax}_{x,y} \|b[x, y]\|$
        $a_p[x, y] = b[best\_x, best\_y]$
        for $x, y \in \Omega_{best\_x, best\_y}$ **do**
          $b[x, y] \leftarrow 0$
        end for
      end while
      if $bool\_ortho(k)$ **then**
        $\alpha_p \leftarrow \text{argmin}_{\alpha_p} \|residual - D_p\alpha_p\|_2^2$
      end if
      $residual \leftarrow x - \sum_{j=1}^{P} D_j\alpha_j$
    end for
  end for
  return $\{a_p\}$
end function

für die Atomgröße $d_m \in \mathbb{R}^{j_1\times j_2}$ und Bildgröße $x_p \in \mathbb{R}^{k_1\times k_2}$, dass der Code die Länge und Breite $a_{p,m} \in \mathbb{R}^{(j_1+k_1-1)\times(j_2+k_2-1)}$ besitzt.

Orthogonalisieren Für das Orthogonalisieren des Codes erweist es sich als hilfreich, die alternative Problemformulierung 3.5 zu betrachten. Da einzig und allein die Werte der nichtnegativen Einträge optimiert werden sollen, ist es nicht nötig, das komplette Dictionary $D = [\hat{d_1} \dots \hat{d_M}]$ zu betrachten (und insb. zu speichern), sondern es müssen nur diejenigen Spalten des Dictionarys berücksichtigt werden, die mit den nichtnegativen Einträgen des Codes korrespondieren. Damit wird das entstehende Optimierungsproblem $\text{argmin}_\alpha \|residual - D\alpha\|$ mit üblichen Lösern wie dem in dem Paper [9] vorgeschlagenen QR-Verfahren lösbar, es ist allerdings immer noch recht komplex. Daher ist es durchaus eine Option, das Orthogonalisieren nicht in jeder Iteration, sondern nur in bestimmten Abständen durchzuführen.

Komplexität Eine vollständige Komplexitätsanalyse wird für diesen Algorithmus nicht durchgeführt, da er in dieser Arbeit ohnehin nicht mit ähnlichen Ansätzen verglichen wird. Jedoch zeigt sich in der Praxis, dass insbesondere die Laufzeit des GCOMP (Alg. 5) sehr hoch ist. Dies gilt in besonderem Maße für die Bilder des Bio-Datensatzes (Kap.

2.2), da diese in beiden doppelt so groß wie die Texturbilder (Kap. 2.1) sind.
Die hohe Laufzeit lässt sich leicht anhand des Pseudocodes (Alg. 5) erklären. Der Hauptteil besteht aus zwei geschachtelten For-Schleifen, in denen eine While-Schleife durchlaufen wird. Die While-Schleife wird durchlaufen, bis das komplette Bild abgedeckt ist. Dies sind maximal (bei perfekter Abdeckung) $(\frac{N}{A})^2$ Durchläufe, minimal (bei größtmöglichen Lücken zwischen den Atomen) etwa ein Viertel der Durchläufe (N = Bildlänge/-breite, A = Atomlänge/-breite). In jedem Durchlauf muss das maximale Argument über alle Einträge der Matrix b gefunden werden. Die Größe der Matrix und damit die Komplexität der Operation beträgt (ungefähr - Atome, die über das Bild hinaus gehen dürfen vergrößern die Matrix sogar um eine kleine Konstante) $N \cdot N \cdot M$ (siehe Zeile 5).
Alles in allem werden somit in etwa $PK\frac{N}{A}^2 N^2 M$ Einträge der Matrix b pro Aufruf des GCOMP kontrolliert. Im Gegensatz zu den vorherigen rotationsinvarianten muss hier beachtet werden, dass die Bilder nicht in Patches unterteilt werden, sondern beispielsweise im Falle des Bio-Datensatzes gilt $N = 512$. Aus diesem Grund werden für diesen Algorithmus später diese Bilder geviertelt. Dies sorgt zwar dafür, dass bei gleicher Anzahl an betrachteten ursprünglichen Bildern das P vervierfacht wird, ist aber offensichtlich aufgrund des N^4 Faktors für die Laufzeit vorteilhaft. Die Bilder sollten aber nicht beliebig klein in Patches unterteilt werden, da dies dem Grundgedanken des translationsinvarianten Ansatzes widerspricht.

Abschließende Bemerkung Unglücklicherweise lässt sich die hier vorgestellte Methode nicht mit den vorher vorgestellten Methoden des rotationsinvarianten Sparse Codings verbinden. Dies liegt an der Tatsache, dass alle vorgestellten Verfahren explizit die Darstellung des Bildes in der entsprechenden Basis verwenden. Da Rotation und Translation jedoch nicht kommutieren, gibt es keine Basis, in der durch entsprechende Faltung der Matching Pursuit rotations- und translationsinvariant durchgeführt werden kann.

4 Klassifikation

Sparse Coding ist eine Technik, die Bilder mithilfe möglichst weniger Informationen rekonstruiert. Dazu wird ein Dictionary so an die Daten angepasst, dass ein möglichst kleiner Rekonstruktionsfehler entsteht. Dieses Anpassen führt dazu, dass die Atome des Dictionarys typische Strukturen der Bilder darstellen. Die Verwendung der Atome zur Rekonstruktion der Bilder gibt somit Aufschlüsse darüber, welche Bilder ähnliche Strukturen aufweisen. Dieses Kapitel veranschaulicht, wie man diese Informationen auf verschiedene Weise nutzen kann, um Bilder oder sogar einzelne Pixel zu klassifizieren.

4.1 Methode zur Klassifikation von Bildern

Diese Methode stammt von den Autoren McCann, Unser und Depeursinge [6]. Die Methodik zur Klassifikation ist recht simpel gehalten, da der Wert der Arbeit in dem Testen des Sparse Codings zur Klassifikation liegt. Die Methode wird hier nun vorgestellt und in Kapitel 5.3 getestet.

Motivation Sparse Coding sorgt dafür, dass sich ein Dictionary an die grundlegenden Strukturen der Daten, auf denen es trainiert wird, anpasst. Kann man nun die grundlegenden Strukturen des Dictionarys bestimmen, so lassen sich darüber Rückschlüsse für die repräsentierten Daten ziehen. Genutzt wird dies, indem auf jeder der 24 Texturen ein Dictionary trainiert wird. Können nun neue Daten mit einem der Dictionarys besonders gut rekonstruiert werden, dann, so die Idee, zeigen die neuen Bilder dieselbe Textur, auf der das Dictionary trainiert wurde.
In der Praxis zeigt sich schnell, dass nicht jedes Patch durch exakt die Klasse bestmöglich rekonstruiert wird, zu der es gehört. Dies liegt daran, dass natürliche Objekte nie exakt gleich aussehen und diese Varianz dafür sorgen kann, dass die Objekte teilweise ähnlicher zu Objekten anderer Klassen sind. Wie man später sehen kann, ist es keinesfalls so, dass man den Atomen direkt ansehen kann, zu welcher Klasse diese gehören (selbst bei einer Sparsity $K = 1$). Um den Algorithmus robuster zu gestalten wird daher eine Vielzahl von Patches aus demselben Trainingsbild gebildet. Jedes Patch wird der Klasse zugeordnet, mit dessen Dictionary es bestmöglich rekonstruiert werden kann. Aus den Zugehörigkeiten dieser Patches wird dann ein Histogramm gebildet.
Die simpelste Methode zur Klassifikation ist es, die Trainingsbilder der Klasse zuzuordnen, die den größten Wert im Histogramm aufweist. Dies folgt dem Prinzip, dass die Mehrzahl der Patches eines Bildes bestmöglich durch das Dictionary der Klasse repräsentiert werden, zu der das Bild gehört. Die Autoren McCann, Unser und Depeursinge haben diese Methode nicht erwähnt (und somit nicht getestet). Die eigene Implementation zeigt jedoch, dass diese Methode sehr gut funktioniert.

Die Methode, die in dem Paper [6] verwendet wird, ist etwas fortgeschrittener. Klassen, die abweichend von der korrekten Klasse zum Rekonstruieren des Patches gewählt werden, sind optisch sehr ähnlich. Andere Klassen, die völlig verschieden aussehen, werden in den seltensten Fällen zur Rekonstruktion der Patches verwendet werden. Die Häufigkeit, welche Klassen wie oft die beste Rekonstruktion der Patches desselben Trainingsbildes liefern, unterliegt also für jede Texturklasse einer gewissen Systematik. Diese Systematik kann von einem Klassifikator erlernt werden und so zu einer besseren Klassifikation führen.

Umsetzung Zunächst werden die Bilder in Test- und Trainingsdaten unterteilt. Aus den Trainingsdaten soll nun ein Dictionary gelernt werden. Dazu werden die Trainingsbilder in kleine Patches unterteilt. Die Autoren wählen für jedes Bild 4000 dieser Patches auf zufällige Weise. Somit erhält man 80000 Patches pro Klasse. Mit diesen wird dann für jede Klasse ein Dictionary gelernt.
Anschließend werden für jedes dieser Dictionarys einzeln alle Patches kodiert. Es wird für jedes Patch bestimmt, mit welchem Dictionary der niedrigste Rekonstruktionsfehler erreicht wurde. Die Klasse, auf der das Dictionary trainiert wurde, wird nun für das Patch ausgewählt. So erhält man also für jedes Patch eine Klasse. Die Patches wiederum gehören Trainingsbildern an. Für jedes dieser Trainingsbilder werden Histogramme erstellt, die aussagen, wie oft Patches dieses Bildes den jeweiligen Klassen zugeordnet wurden. Diese Histogramme dienen als Trainingsfeatures. Die Zielvariable für diese Histogramme ist die Klasse des Trainingsbildes. Auf diesen kann der Klassifikator trainiert werden.
Die Testfeatures werden schließlich auf die gleiche Weise erstellt. Man sollte beachten, dass dieselben Dictionarys verwendet werden, denn das Testen der Methode soll den Eingang neuer Daten simulieren, bei denen man die Klasse nicht kennt. Somit können die Dictionarys auch nicht auf den neuen Daten gleicher Klasse trainiert werden. Man erhält so pro Testbild ein Histogramm an Klassen. Anhand dieses kann der Klassifikator das Testbild einer Klasse zuordnen.
Zu den Parametern dieser Methode gehören neben der Anzahl die Patches auch die Größe des Dictionarys M, die Größe der Atome A, die Sparsity K, die Anzahl an Iterationen I und, für den rotationsinvarianten Algorithmus mit lenkbarer Basis, die Anzahl an betrachteten Rotationen.

4.2 Methode zur Detektion von Objekten

Die Methode zur Klassifikation von Bildern wird nun auf die Detektion von Objekten im Bild übertragen. Dies wird realisiert, indem das Bild pixelweise klassifiziert wird. Das Verfahren aus Kapitel 5.3 lässt sich dann mit den im Folgenden vorgestellten zwei Änderungen zur Detektion von Objekten anwenden. Hier wird das Detektionsproblem stets für zwei Klassen betrachtet, sodass ein Objekt in den Bildern detektiert werden kann. Die vorgestellte Methode ist jedoch uneingeschränkt auf das Multidetektionsproblem mehrerer Objekte übertragbar.

Histogrammbildung Die erste Herausforderung, die gemeistert werden muss, wenn man die vorherige Methode zur Klassifikation von Pixeln nutzen will, ist, dass nun keine Bildung von Histogrammen über das ganze Bild mehr geschehen sollte. Damit würde jede räumliche Information verloren gehen und eine sinnvolle Klassifikation einzelner Pixel wäre nicht mehr möglich. Es gibt natürlich die Möglichkeit, keine Histogramme zu bilden und lediglich die Information einzelner Pixel zu nutzen. Dieses Verfahren ist jedoch nicht sehr robust. Stattdessen ist es sinnvoll, die Information von benachbarten Pixeln miteinzubeziehen. Dies kann ungewichtet oder gewichtet geschehen.

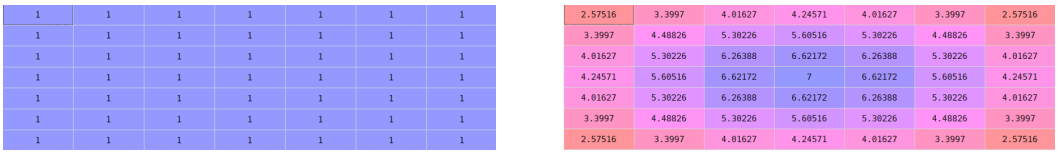


Abbildung 4.1: Uniforme (links) und Gaußsche Matrix der Größe 7

In den Implementierungen wird dies durch die von Faltung mit einer 'uniformen' beziehungsweise einer 'gaußschen Faltungsmatrix' umgesetzt. Als 'uniforme Matrix' dient dabei die Matrix, die nur aus Einsen besteht. Die 'gaußsche Matrix' basiert auf der zweidimensionalen Normalverteilung $h(x,y) = \frac{1}{2\pi} e^{-\frac{x^2+y^2}{2}}$, deren Ursprung das Zentrum der Matrix darstellt. Die Werte wurden um einen Faktor auf meist einstellige Zahlen erhöht, da dies zum einen robuster gegenüber Rundungsfehlern und anschaulicher ist, andererseits aufgrund der skalierungsinvarianten Klassifikatoren ohnehin keine Rolle spielt. Die Matrix entspricht somit einer skalierten Version des bekannten Gauß-Filters[12].
Der Vorteil der 'gaußschen Matrix' ist, dass Nachbarn gewichtet werden. Dies unterstützt die Intention, dass weiter entfernte Nachbarn weniger Einfluss auf die Klasse eines Pixels haben. Der Nachteil ist, dass die Werte verändert werden, also nicht mehr klar ist, mit welcher Intensität die Atome verwendet werden. Die Auswahl der Matrix ist somit Teil der Hyperparameteroptimierung. Ebenso Teil der Hyperparameter ist die Größe der Matrix, die bestimmt, wie viele Nachbarpixel miteinbezogen werden.

Nutzung des Codes als Feature Die zweite Herausforderung ist die Nutzung des Approximationsfehlers: Für das Verfahren mit dem translationsinvarianten Sparse Coding ist die Nutzung der Information, mit welchem Dictionary der kleinste Approximationsfehler des Bildes erreicht wird, problematisch. Da das Bild nicht in einzelne Patches unterteilt, sondern als Ganzes kodiert wird, ist es nicht möglich, für jedes Patch einzeln alle Dictionaryklassen zu trainieren und am Ende zu schauen, welches Dictionary den geringsten Approximationsfehler produziert hat. Andererseits ist eine pixelweise Auswertung der Approximationsfehler auch nicht besonders sinnvoll, da dies zu einem sehr starken Rauschen führen würde. Eine Methode wäre sicherlich, das Ganze für kleine Nachbarschaften regional auszuwerten. Ein bleibendes Problem ist jedoch, dass an Klassengrenzen die fremden Klassen gut mitklassifiziert werden müssen, da ansonsten ein sehr hoher Rekonstruktionsfehler droht.

Daher wird hier eine andere Methodik angewandt: Statt die Bilder für alle Dictionarys einzeln zu trainieren, werden die Patches/Bilder auf dem vereinigten Dictionary aller Klassen trainiert. Die Idee ist, dass der Matching Pursuit dann die Atome der Klasse auswählt zu dem die jeweiligen Patches/Teile des Bildes gehören. Dies entspricht exakt der Intuition, dass die verschiedenen Dictionarys die verschiedenen Klassen darstellen und der Matching Pursuit stets das Atom der Klasse zur Rekonstruktion wählt, die es jeweils rekonstruieren will.
Beide Ansätze sind für den (meistgenutzten) Fall Sparsity $K = 1$ ohnehin äquivalent. Dies liegt daran, dass der Matching Pursuit (der im Fall $K = 1$ dem orthogonalen Matching Pursuit entspricht) die Energieerhaltungsgleichung erfüllt (Beweis Seite 4 in [4]):

$$\begin{aligned}\|x\|^2 &= |\langle x, atom\rangle|^2 + \|residual\|^2 \\ &= |\langle x, atom\rangle|^2 + \|x - \langle x, atom\rangle atom\|^2\end{aligned}$$

Da die linke Seite $\|x\|^2$ der obigen Gleichung fest ist, entspricht das Auswählen des Atoms mit minimalem Residuum $\text{argmin}_{atom} \|residual\|^2 = \text{argmin}_{atom} \|x - \langle x, atom\rangle atom\|^2$ dem Auswählen des Atoms mit maximalem Skalarprodukt $argmax_{atom}|\langle x, atom\rangle|^2$, wie es im Matching Pursuit (vgl. Alg. 2) geschieht. Somit entspricht die Klasse des ausgewählten Atoms der Klasse, die den geringsten Rekonstruktionsfehler erbringt.

Weitere Anpassungen In der Klassifikationsmethode werden stets alle Informationen klassenweise gebündelt. Statt zu schauen, mit welchen Atomen die beste Rekonstruktion möglich ist, wird diese Information über alle Atome einer Klasse aggregiert. Dies ist keinesfalls nötig. Da nun nur noch zwei Klassen betrachtet werden, ist die Komprimierung der Information sehr stark und nicht zwingend vorteilhaft. Daher wird der Parameter $flatten\_to\_classes$ eingeführt. Dieser bestimmt, ob die Information per Summe über alle Atome einer Klasse gebündelt wird, oder ob jedes Atom ein einzelnes Feature darstellt.
Die Detektionsmethodik eröffnet im Gegensatz zur Klassifikationsmethodik außerdem eine neue Möglichkeit, da der Code als Feature für die Detektion verwendet wird. Nach obiger Beschreibung heißt dies, dass mithilfe der Information, welche Atome verwendet wurden, klassifiziert wird. Es kann aber natürlich auch die Information, wie stark die Atome verwendet werden (d.h. der tatsächliche Wert des Codes), einbezogen werden. Dies wird durch den Parameter $take\_actual\_values$ abgebildet.

Abschließende Bemerkung Es wird gefordert, dass die Dictionarys anfangs auf allen Klassen einzeln trainiert werden. Dies ist keinesfalls eine zwingend nötige Vorgabe, da im nächsten Schritt das Bild mit dem vereinigten Dictionary kodiert wird und die Klasseneinteilung letztendlich durch einen Klassifikator geschieht. Dieser ist nicht darauf angewiesen zu schauen, auf welchen Daten die jeweiligen Atome trainiert wurden, sondern kann Muster und Zusammenhänge erkennen und anhand derer die Klasseneinteilung vornehmen.
Die Aufteilung des Dictionarytrainings hat dennoch einige kleine Vorteile. Einerseits wird so dem Algorithmus als Starthilfe vorgegeben, in welchen Bildern ähnliche Strukturen

vorliegen (da alle Bilder zur derselben Klasse gehören). Dies ermöglicht ein schnelleres und besseres Anpassen des Dictionarys an die Bilder. Andererseits hat dies auch zur Auswertung kleine Vorteile. Es kann manuell betrachtet werden, mit Atomen welcher Klasse eingehende Bilder kodiert wurden. Dies hilft Klassifikationsentscheidungen zu verstehen und ermöglicht einfache Klassifikationsverfahren wie: Ein Bild/Patch wird der Klasse zugeordnet, dessen Atome am häufigsten zur Rekonstruktion verwendet wurden.
Im Gegensatz zur Bildklassifikation muss hier beachtet werden, dass Bilder nicht mehr nur zu einer Klasse gehören, sondern beide zu detektierende Klassen beinhalten können. Für das patch-basierte Verfahren bedeutet dies, dass für jedes Patch einzeln überprüft wird, zu welcher Klasse dies gehört (anhand des mittleren Pixels des Patches). Das Dictionary wird für jede Klasse auf allen Patches trainiert, die auf diese Weise der jeweiligen Klasse zugeordnet wurden.
Für den translationsinvarianten Algorithmus muss man hingegen einen Schritt weiter gehen und den Matching Pursuit selbst abändern: Mithilfe einer 'Aktivierungsmatrix' werden die Dictionarys so trainiert, dass Atome nur für jeweils eine Klasse verwendet werden. Das heißt, alle Positionen, an denen ein Atom nicht stehen darf, werden ausgeschlossen, indem der jeweilige Wert in der Matrix b (vgl. Alg. 5, Zeile 5) durch Multiplikation mit der Aktivierungsmatrix (enthält nur die Werte 0 und 1) auf Null gesetzt wird.

4.3 Verwendete Klassifikationsalgorithmen

Für die Klassifikation wird auf bewährte Algorithmen gesetzt. Zum einen wird der Nearest Neighbors Klassifikator verwendet, da dieser bereits im *Fast Rotational Sparse Coding* Paper [6] zum Einsatz kommt und dort sehr gute Ergebnisse liefert. Zum anderen wird der Random Forest Klassifikator verwendet. Dieser ist deutlich weniger laufzeit- und speicherintensiv. Dies ist insbesondere für das Detektionsproblem wichtig, wo sich die Anzahl der Klassifikationen der Anzahl aller Pixel entspricht. Für die Klassifikation der Texturen hat sich gezeigt, dass die Klassifikatoren ohnehin Resultate gleicher Güte erzeugen.
Im Folgenden werden beide Klassifikatoren prinzipiell vorgestellt. Es besteht hier kein Anspruch darauf, die Funktionsweise vollständig zu erklären. Die Beschreibung dient hauptsächlich der Darlegung der Vor- und Nachteile der Klassifikatoren, da diese direkten Einfluss auf die Endresultate haben. Die Darstellungen beruhen maßgeblich auf dem erworbenen Wissen aus der Vorlesung Mustererkennung (Sommersemester 2018, gehalten von Prof. Dr. Xiaoyi Jiang an der WWU Münster).
Vorab sei bereits kurz darauf eingegangen, dass hier zwei nichtlineare Klassifikatoren ausgewählt sind. Da die bereitgestellten Bilder jedoch bereits vorverarbeitet wurden und in Folge dessen deren Helligkeit vereinheitlich wurde, kann davon ausgegangen werden, dass sämtliche Klassifikationsprobleme nicht linear sind.

4.3.1 Nearest Neighbor Klassifikator

Der Nächste-Nachbarn Klassifikator ist einer der ältesten und vor allem simpelsten Klassifikatoren. Er ist nichtparametrisch und ordnet Instanzen einzig und allein aufgrund eines Distanzmaßes und einer bekannten Trainingsmenge den vorhandenen Klassen zu. Dies geschieht, indem für jede zu klassifizierende Instanz der Datenpunkt der Trainingsmenge gefunden wird, der die kürzeste Entfernung zu der Instanz aufweist. Die zu klassifizierende Instanz wird dann der Klasse des gefundenen Trainingsdatenpunkts zugeordnet. Formal bedeutet dies für eine Menge an Trainingspunkten $\{Z_1, \ldots, Z_n\}$, verschiedenen Klassen $\{C_1, \ldots, C_m\}$, Distanzmaß D und Testinstanz x:

$$Z_j = \underset{Z_j}{\operatorname{argmin}}\{d(x, Z_i) | 1 \leq i \leq M\} \wedge Z_j \in C_k \Rightarrow x \in C_k$$

Als Distanzmaß wird wie in dem Paper [6] die χ^2 Distanz ($d(a,b) = \sum_{i=1}^{24}(a[i] - b[i])^2/(a[i] + b[i])$) verwendet.
Der Vorteil dieses Klassifikators ist, dass er dem intuitiven Prinzip folgt, die Klassifikation dem ähnlichsten bekannten Beispiel anzupassen. In dieser Arbeit sollen Bilder anhand ihrer Kodierungen klassifiziert werden. Es wird davon ausgegangen, dass für gleiche Objekte eine gleiche/ähnliche Kombination an Atomen verwendet wird. Für dieses Problem scheint der Klassifikator optimal geeignet.
Ein weiterer Vorteil ist, dass der Klassifikator nichtparametrisch ist. Damit ist keine langwierige Hyperparameteroptimierung nötig, die die Resultate entscheidend beeinflussen kann.
Der Nächste-Nachbarn Klassifikator hat auch einige Nachteile. Einer ist dessen Robustheit. Ein falsch klassifiziertes oder auch ein ungewöhnliches Trainingsobjekt sowie das Vorhandensein von Rauschen führt schnell zu Missklassifikationen. Oft wird auf dies durch Verwendung des 'K-Nächste-Nachbarn' Klassifikators reagiert. Dieser unternimmt die Klasseneinteilung einer neuen Instanz durch eine (möglicherweise Distanz-gewichtete) Mehrheitsentscheidung der K nächsten Nachbarn. Für den Fall K=1 stimmen beide Klassifikatoren trivialerweise überein. Einige eingestreute Tests im Laufe dieser Arbeit zeigten jedoch, dass der Nearest-Neighbor Klassifikator auf allen verwendeten Daten robust genug ist und es keinen Vorteil durch höhere Werte von K gibt. Daher wird in dieser Arbeit durchgängig der Nearest-Neighbor Klassifkator genutzt.
Ein sehr viel größerer Nachteil sind die hohe Laufzeit und die hohe Speichernutzung. Der Nächste Nachbarn Klassifikator vergleicht die Testinstanz stets mit allen Trainingsinstanzen (es existieren Optimierungen, sodass nicht zwingend mit allen verglichen werden muss, die Menge bleibt jedoch in jedem Fall sehr groß). Im Gegensatz zu den parametrischen Klassifikatoren müssen also nicht nur einige Parameter, sondern alle Trainingsdaten im Speicher gehalten werden. Außerdem ist im Gegensatz zu anderen Klassifikatoren keine (oder bestenfalls wenig) Vorverarbeitung möglich und die komplette Berechnung erfolgt erst mit der Klassifikation. Daher ist die Klassifikation selbst sehr langsam.

4.3.2 Random Forest Klassifikator

Der Random Forest Klassifikator ist ein absoluter Klassiker unter den Klassifikationsalgorithmen. Er basiert seine Klassifikationsentscheidung auf den Entscheidungen vieler einzelner möglichst unabhängiger Entscheidungsbäume. Die Idee folgt dem Prinzip der "Weisheit der Vielen": Die Kumulation von Entscheidung zu gemeinsamen Gruppenentscheidung ist oft besser als die Entscheidungen einzelner Bäume, da schlechte Entscheidungen einzelner Bäume durch die Masse aufgefangen werden.
Ein Entscheidungsbaum liefert regelbasierte Entscheidungen. Die genaue Findung dieser Regeln würde hier zu weit führen. Wichtig ist nur, dass in jeder Ebene dieser Bäume ein Feature (d.h. eine Variable) anhand eines kritischen Wertes in zwei Teile gespalten wird. Das Feature und der Wert sind so gewählt, dass die Teile möglichst in die zu klassifizierenden Klassen unterteilt werden. Damit ist schon klar, dass dies kein linearer Klassifikator ist.
Die Entscheidungsbäume sind stets nur mit einem Teil der vorhanden Informationen (Features) erstellt. So soll die Unabhängigkeit der Bäume erreicht werden. Zusammengefasst werden die Entscheidungen mithilfe einer Wahl, meist eine Mehrheitswahl aller Bäume. Dies sichert, dass einzelne schlechte Entscheidungen nicht zu schlechten Endresultaten führen.
Der große Vorteil ist, dass der Random Forest ein rein parametrischer Klassifikator ist. Zur Klassifikation braucht er lediglich alle Entscheidungsbäume. Die Trainingsdaten werden nicht mehr benötigt. Dies sorgt einerseits dafür, dass der Speicherverbrauch konstant ist und nicht von der Größe der Trainingsdaten abhängt. Andererseits wird auch die Laufzeit, die zur Klassifikation eines Datenpunktes benötigt wird, nicht von der Größe der Trainingsdaten beeinflusst und ist bezogen auf diese Variable ebenfalls konstant. Aus diesem Grund wird der Klassifikator in dieser Arbeit häufig eingesetzt.

5 Ergebnisse

In diesem Kapitel werden alle Ergebnisse systematisch zusammengefasst. Um die einzelnen Algorithmen besser vergleichen zu können, werden die Ergebnisse nach Problemstellung strukturiert.
Insgesamt wurden in dieser Arbeit drei verschiedene Aufgabenstellungen diskutiert. Zunächst geht es um das Sparse Coding als Repräsentation von Bildern. Daher wird zuerst aufgezeigt, wie gut die Bilder rekonstruiert werden können. Anschließend wird kurz auf die Laufzeiten der verschiedenen Algorithmen eingegangen. Dies war insbesondere für die rotationsinvarianten Algorithmen ein wesentliches Unterscheidungskriterium. Schlussendlich wird die Aussagequalität der Algorithmen in Bezug auf die Klassifikation von Bildern und die Detektion von Objekten präsentiert.

5.1 Rekonstruktionsqualität des Sparse Coding

Die Rekonstruktionsfähigkeit des Sparse Coding hängt von vielen Faktoren ab. Von grundlegender Bedeutung sind die zu wählenden Parameter. Dieses Unterkapitel erfüllt daher zwei Funktionen: Zum einen soll es zeigen, wie gut sich die Bilder durch die Atome repräsentieren lassen, zum anderen soll auch ein Gefühl für die Eingabeparameter entwickelt werden.
Die Einflussgrößen lassen sich direkt aus der grundlegenden Problemformulierung ablesen:

$$\underset{D,\{a_p\}}{\operatorname{argmin}} \sum_{p=1}^{P} \|x_p - Da_p\|^2 \, mit \, \|a_p\|_0 \leq K \; \forall p \tag{5.1}$$

Gewählt werden die Sparsity K, die Größe des Dictionarys M und das Abbruchkriterium, im Fall des K-SVD (Alg. 1) die Iterationsanzahl. Weiterhin ist auch die Anzahl und Auswahl der Bilder $\{x_p\}$ von Bedeutung.
Als Vergleichskriterium wird im folgenden häufig der 'normierte quadratische Fehler' herangezogen. Dieser ergibt sich, indem zunächst für alle Bilder die $l_2 - Norm$ des Residuums durch die $l_2 - Norm$ des jeweiligen Bildes (Wurzel der Summe aller quadrierten Pixel) genommen und dann gegebenfalls über alle Bilder gemittelt wird. In den Abbildungen wird dieser mit 'norm_mse' abgekürzt.

Baseline Zunächst wird ein grundlegendes Resultat gezeigt, dass auf einigen frei gesetzten Parametern besteht. Diese wurden bewusst eher so gewählt, dass die Laufzeit kurz ist. Die Parameter lauten $P = 400$ (Anzahl Bilder), $K = 1, M = 10$ (Anzahl Atome), $I = 10$ (Anzahl Iterationen).

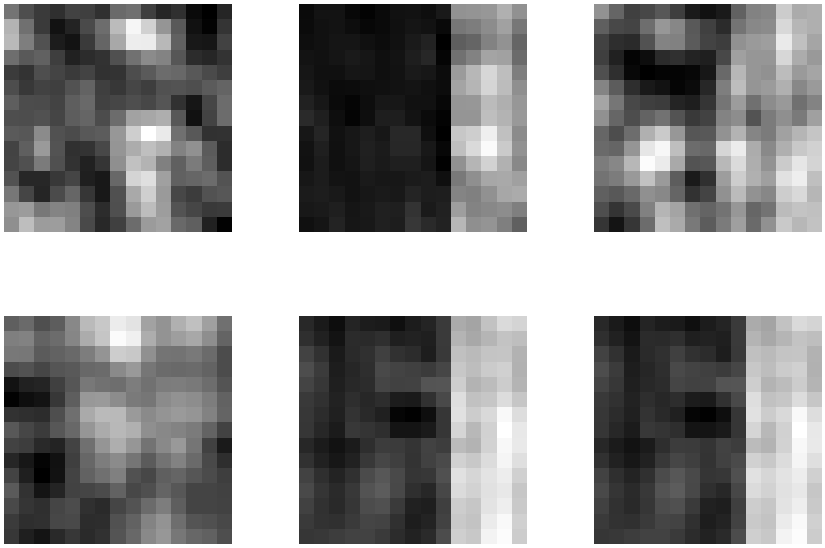


Abbildung 5.1: Einige Texturbilder (oben) und deren Rekonstruktionen. Bild 2 und 3 werden durch dasselbe Atom repräsentiert.

In Abbildung 5.1 ist gut zu sehen, dass, aufgrund der niedrigen Anzahl an Atomen, die Atome die Bilder noch nicht sonderlich gut repräsentieren. Gleiches gilt für die Zellbilder in Abbildung 5.2. In Abbildung 5.3 kann man die Rekonstruktionsfehler im Histogramm sehen.

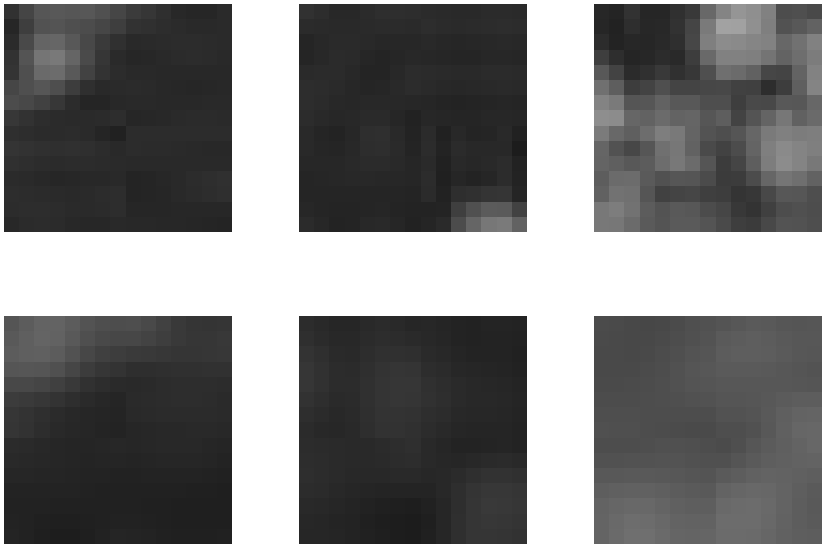


Abbildung 5.2: Einige Zellbilder (oben) und deren Rekonstruktionen.

Diversität der Bilder Einen massiven Einfluss auf den Rekonstruktionsfehler hat die Auswahl der Bilder. Als Experiment wird in Abbildung 5.3 die Verteilung der Residuen des Baseline-Tests für den modifizierten Datensatz und den Zelldatensatz gezeigt.

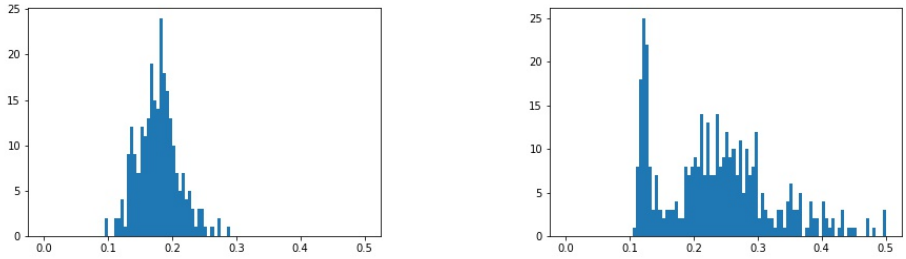
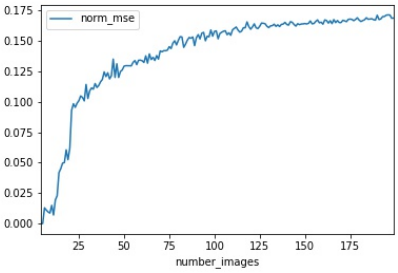


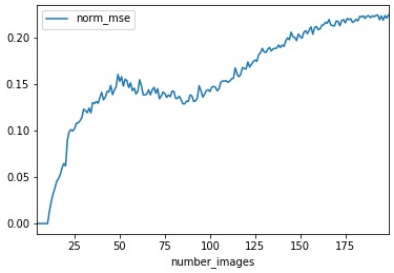
Abbildung 5.3: Histogramme der Residuen der modifizierten Texturbilder (links) und Zellbilder.

Es ist deutlich zu sehen, dass zwar einige Bilder gut getroffen wurden, jedoch der Anteil der schlecht rekonstruierten Bilder im Zelldatensatz deutlich höher ist. Bei der Betrachtung einiger Beispielbilder (Abbildung 5.2) fällt auf, dass die Bilder stärker verrauscht sind.

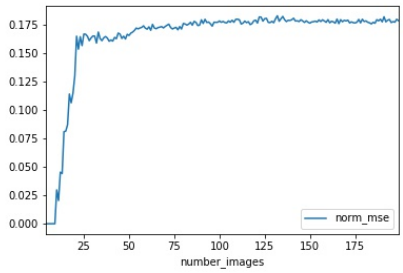
Anzahl der Bilder Obwohl im Folgenden stets der Rekonstruktionsfehler pro Bild betrachtet wird, hat die Anzahl der Bilder dennoch einen Einfluss auf die Rekonstruktionsfehler.
Bleiben alle anderen Parameter unverändert, die Anzahl der Bilder wird jedoch erhöht, hat dies einen negativen Effekt auf die Fehlerrate. Dies liegt daran, dass die gleichen Anzahl an Atomen nun mehr Bilder repräsentieren müssen. Der Effekt, dass die Fehlerrate in Abhängigkeit der Bilder steigt, ist umso größer, desto verschiedener die Strukturen in den Bildern sind. Dies führt dazu, dass einzelne Atome gegebenenfalls völlig verschiedene Strukturen in den einzelnen Bildern repräsentieren müssen. Dazu sind sie jedoch nicht in der Lage, sodass der Fehler in dem Fall sehr hoch sein wird.
In den vorliegenden Grafiken 5.4 sieht man, dass die durchschnittliche Fehlerrate allgemein für eine höhere Anzahl an Bildern steigt. Einzelne Rückgänge sind einerseits zufallsbedingt (es werden keine globalen Lösungen gefunden), andererseits können sie dadurch erklärt werden, dass durch Zunahme ähnlicher Bilder die Anzahl der gut repräsentierten Bilder steigt und damit die durchschnittliche Fehlerrate abnehmen kann.
In der Abbildung 5.4 ist durch den Vergleich der Fehlerraten auch die Wichtigkeit der Diversität zu sehen. Sind die Bilder ähnlicher (wie in Abbildung 5.4 oben, in der die Bilder aus dem Gesamtbild durch Stripe 1 entstanden sind), steigt der Fehler langsamer und ist allgemein geringer.
Generell ist die Tatsache, dass ein Atom in mehreren Bildern verwendet wird gewünscht. Als Gegenbeispiel bedenke man zum Beispiel den Fall $P \leq M$. In diesem Fall ist eine



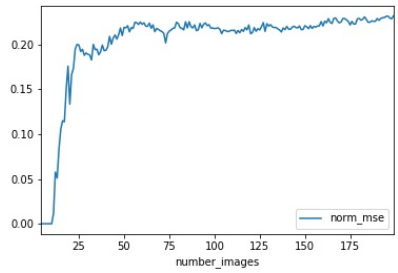
Texturpatches mit Überschneidungen



Zellpatches mit Überschneidungen



Texturpatches ohne Überschneidungen



Zellpatches ohne Überschneidungen

Abbildung 5.4: Fehlerraten in Abhängigkeit der Anzahl an Bildern

perfekte Rekonstruktion möglich, auch wenn man Abbildung 5.4 links oben sieht, dass der Algorithmus diese nicht immer findet. Im Falle dieser perfekten 1 zu 1 Rekonstruktion sind die Atome jedoch keine generalisierten Strukturen der Bilder und das Sparse Coding ist wertlos.

Sparsity K Eine größere Sparsity führt theoretisch immer zu einer besseren Approximation des Bildes. Dies liegt daran, dass im Matching Pursuit in jeder Iteration das verbliebene Residuum betrachtet und approximiert wird. Da immer die bestmögliche Approximation für das Residuum verwendet wird, ist eine Verschlechterung der Gesamtapproximation nicht möglich (in dem seltenen Fall, dass das Bild bereits vollständig rekonstruiert wurde, ist beispielsweise der Code 'null' für das neue Atom zugelassen). In der Abbildung 5.5 sieht man jedoch, dass der Fehler bei einer Sparsity von 8 ansteigt. Dies liegt erneut an der Tatsache, dass keine globalen Lösungen gefunden werden.
Man sieht also, dass die Sparsity nicht beliebig hoch sein muss, um gute Approximationsresultate zu erzielen. Ohnehin sollte die Sparsity mit bedacht gewählt und nicht zwingend auf hohe Werte gesetzt werden. Obgleich eine hohe Sparsity stets einen positiven Effekt auf den Rekonstruktionsfehler hat, ist dies für gewisse Probleme nicht förderlich. In dieser Arbeit geht es beispielsweise um die Nutzung des Codes zur Klassifikation der Bilder. Daher will man möglichst aussagekräftige Atome, was bedeutet, dass möglichst wenige Atome verwendet werden sollten. Glücklicherweise ist es so, dass das erstverwendete Atom stets den größten Effekt auf die Minderung des Residuums hat (im Algorithmus

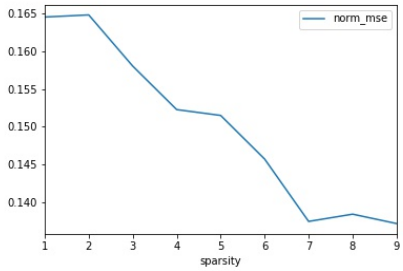


Abbildung 5.5: Fehlerrate in Abhängigkeit der Sparsity

werden in jeder Iteration stets alle Atome als Approximatoren zugelassen, die Lösung für jeden Iterationsschritt ist somit die optimale Lösung). Daher wird sich später zeigen, dass eine Sparsity von eins in vielen Fällen für das beste Klassifikationsergebnis sorgt.

Anzahl Atome M Mehr Atome bedeuten ebenfalls, dass der Rekonstruktionsfehler der globalen Lösung stets besser wird. Dies liegt an der einfachen Tatsache, dass die Menge an zugelassenen Variablen, über die optimiert wird, größer wird. Dabei ist es freilich nicht notwendig alle Atome zu nutzen, weshalb eine Erhöhung der absoluten Anzahl niemals zu einer Verschlechterung führen kann.

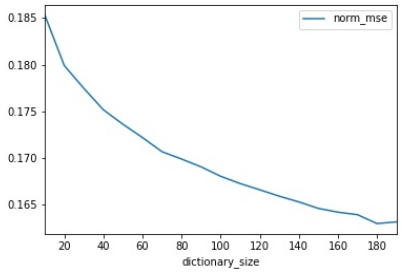


Abbildung 5.6: Fehlerrate in Abhängigkeit der Anzahl an Atomen

Wie bei der Sparsity ist einer Erhöhung der Anzahl nicht nur negativ für die Laufzeit, sondern auch nicht förderlich für das Klassifikationsproblem. Dies liegt abermals an der Tatsache, dass die Atome nicht mehr aussagekräftig genug sind. In diesem Fall liegt dies daran, dass sie, falls es zu viele gibt, nicht mehr genug verschiedene Bilder approximieren müssen und daher nicht mehr genug generalisieren. Man hat es also mit dem im Machine Learning klassischen Problem des Overfittings zu tun.

Anzahl Iterationen Eine Erhöhung dieses Parameters hat ebenfalls stets positive Auswirkungen auf den Rekonstruktionsfehler. Dies gilt für diesen Parameter sogar in alle Problemstellungen, insbesondere also auch für das Klassifikationsproblem. Der Grund, warum er dennoch nicht beliebig hoch gewählt werden sollte, ist offensichtlich: die An-

zahl der Iterationen hat direkte Auswirkungen auf die Laufzeit.
Im vorliegenden Beispiel zeigt sich, dass eine Iterationsanzahl von 10 bereits gute Ergebnisse liefert. Zur Erstellung der Grafik wurden verschiedene Läufe mit der jeweiligen Anzahl an Iterationen durchgeführt, welche das Rauschen erklären. Daher wird in dieser Arbeit häufig eine Iterationsanzahl zwischen 10 und 15 gewählt.

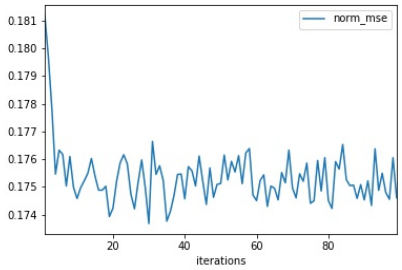
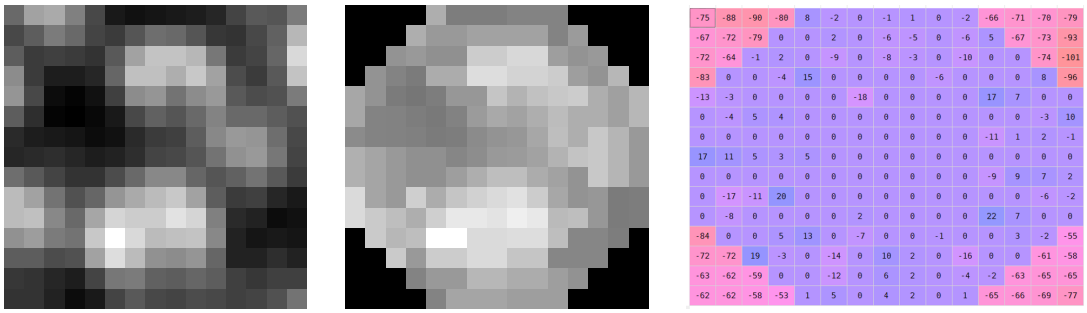


Abbildung 5.7: Fehlerrate in Abhängigkeit der Anzahl an Iterationen

Rotationsinvarianz Die Möglichkeit, dass Atome rotiert zur Rekonstruktion eingesetzt werden dürfen, verbessert die Rekonstruktionsqualität. Der Effekt ist ähnlich zu dem der Erhöhung der Anzahl der Atome. Jedoch ist auch klar, dass fünf verschiedene Atome ein mindestens ebenso gutes Ergebnis liefern, wie ein Atom in fünf Rotationen, da die Lösung des zweiten Problems eine gültige Lösung für das Erste ist, andersherum jedoch nicht.



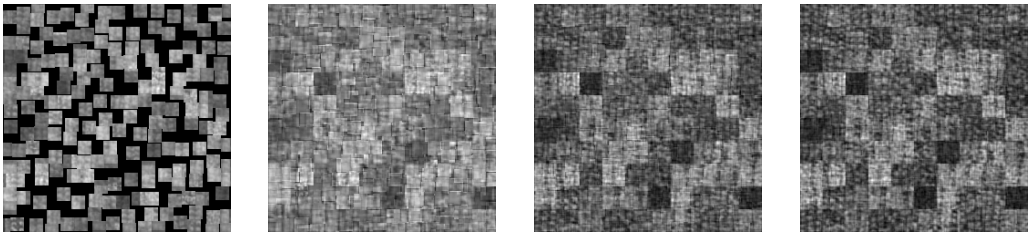
Originalpatch | Patch nach Transformation und Rücktransformation | Transformationsfehler

Abbildung 5.8: Transformationsfehler der 'lenkbaren Methode'

Problematisch ist allerdings, dass für die Rotationen auf approximative Lösungen zurückgegriffen wird. In Abbildung 5.8 sieht man links ein originales Patch des Texturdatensatzes und in der Mitte das gleiche Patch, nachdem es zunächst in Polarkoordinaten dargestellt wurde und anschließend zurück transformiert wird. Rechts ist die Matrix der Abweichungen dargestellt. Es ist direkt ersichtlich, dass die Ecken in der Transformation nicht berücksichtigt werden. Auch in der Mitte ist die Approximation indes nicht per-

fekt. Für das Verfahren mit lenkbarem Ansatz ist der Effekt der gleiche (da der Ansatz sehr ähnlich ist).
Dennoch ist die Rotationsinvarianz sehr wünschenswert. Das liegt daran, dass dies zum häufigeren Einsatz der einzelnen Atome führt und so Overfitting entgegenwirkt. Man bekommt somit eine bessere Aussagefähigkeit der Atome. Außerdem ist es für die Klassifikation von Objekten egal, in welcher Rotation sie im Bild vorkommen. Der rotationsinvariante Algorithmus ist in der Lage, eine Verbindung zwischen den Objekten verschiedener Rotation herzustellen, da er dasselbe Atom verwenden kann. Ist der Algorithmus nicht in der Lage, rotierte Atome zu verwenden, kann er keine Verbindung zwischen den Objekten herstellen (außer er verwendet zufällig dennoch dasselbe Atom, zum Beispiel für den Fall, dass das Objekt rund ist). Selbst in dem Fall, dass mittels einer erhöhten Anzahl an Atomen dieselbe Rekonstruktionsqualität erreicht wird, ist also eine schlechtere Performance für das Klassifikationsproblem möglich.
In der Praxis zeigt sich, dass die rotationsinvarianten Algorithmen für eine etwas schlechtere Approximation sorgen. Der Effekt ist jedoch marginal und in Bildern wie Abbildung 5.10 nicht ersichtlich. Der Standard-Algorithmus hat für diese Abbildung einen normierten quadratischen Fehler von 0,0246, der 'lenkbare Ansatz' 0,0272 und der 'polare Ansatz' 0,0356 geliefert. Dies ist nicht repräsentativ, jedoch haben andere eingestreute Tests ähnliche Ergebnisse erzeugt. Die Bilder nebeneinander zu legen bringt aufgrund des kleinen Performance-Unterschieds keinen Erkenntnisgewinn und wird hier daher auch nicht gemacht. Die einzige Beobachtung, die erwähnt werden kann, ist, dass in den Ecken bei den rotationsinvarianten Ansätzen einige Pixel gar nicht rekonstruiert werden. In der Mitte passiert dies nicht, da dort bei einem Stripe von 1 stets Patches zu einem Pixel beitragen, für das der jeweilige Pixel nicht in einer Ecke des Patches liegt.
Letztendlich sollte auch nicht vergessen werden, dass für das spätere Klassifikationsproblem die Bilder zwar in den neuen Koordinatenraum transformiert werden müssen, eine Rücktransformation jedoch nicht stattfindet. Es ist dort nur der Code von Interesse. Das bedeutet, dass der Approximationsfehler der Rücktransformation für das Klassifikationsproblem irrelevant ist.

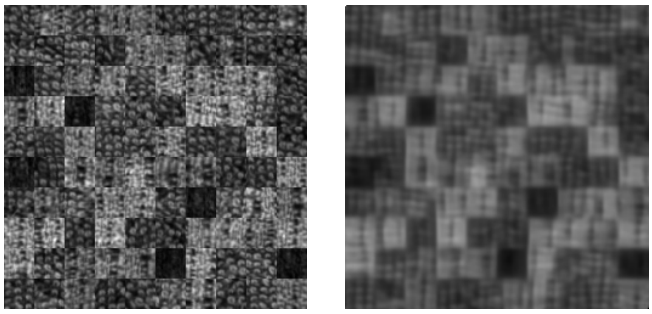
Translationsinvarianz Translationsinvarianz sorgt generell für eine eher bessere Approximation des Bildes. Jedoch hat der gewählte Algorithmus auch große Nachteile, die anhand von Beispielen hier einmal gezeigt werden.
Der vorgestellte translationsinvariante Algorithmus ist, wie alle in realistischer Laufzeit ausführbaren Algorithmen, kein optimaler Algorithmus. Die Atome werden schichtweise ausgewählt, was zu einer Zerstückelung des Bildes führt. Eine typische Rekonstruktion mit Sparsity $K = 1$ ist in Abbildung 5.9 zu sehen. Dort sieht man, dass das Bild viele schwarze, nicht rekonstruierte Stellen aufweist.
Daher sollte die Sparsity K in dem translationsinvarianten Algorithmus generell um einiges höher gewählt werden. Eine hohe Sparsity schränkt, wie oben erwähnt, die Aussagekräftigkeit der einzelnen Atome stark ein. Eine Sparsity unter $K = 3$ erzeugt jedoch meist nur ungenügende Ergebnisse und wird deshalb selten zum Erfolg führen.



$K = 1$ $K = 5$ $K = 10$ $K = 15$

Abbildung 5.9: Translationsinvariante Rekonstruktion des Bildes aus Abb. 5.10 mit verschienenden Sparsitys K

Genauigkeit in der Praxis Rekonstruktionen der Bilder können beliebig genau werden. Dafür kann man einerseits mit genügend vielen Atomen oder andererseits mit sehr hoher Sparsity K sorgen (vgl. Abb. 5.9). Hier wird nun auch einmal gezeigt, wie genau eine Repräsentation in einer Anwendung ist.



Originalbild

Rekonstruiertes Bild mit der Sparsity $K = 1$

Abbildung 5.10: Rekonstruktion mit Parametern des finalen patch-basierten Laufs

Abbildung 5.10 zeigt die Rekonstruktion, die mit den Parametern der finalen patch-basierten Texturdetektion erreicht wird. Die genauen Parameter sind in Kapitel 5.4 zu finden, aber hier auch nicht allzu relevant. Einzig die niedrige Sparsity von $K = 1$ sei erwähnt, da diese sicher nicht die beste Wahl für das Rekonstruktionsproblem ist. Das Bild wurde erstellt, indem die durch das Sparse Coding rekonstruierten Patches mit Stripe 1 übereinander gelegt und gemittelt wurden.

5.2 Laufzeit der Algorithmen

Generell lässt sich feststellen, dass der patch-basierte Ansatz deutlich schneller ist, als die Verwendung von translationsinvariantem Sparse Coding. Für den patch-basierten Ansatz zeigt sich in allen Durchläufen, dass der Algorithmus ohne Rotationen wie erwartet stets mit Abstand am schnellsten ist. Weiterhin hat sich gezeigt, dass bei ähnlicher Anzahl an Rotationen, die polare Variante des rotationsinvarianten Ansatzes etwas schneller ist.

Beispielhaft aufgeführt sind hier die Laufzeiten des finalen Laufs zum modifizierten Texturdatensatz. Dies ist ein kompletter Durchlauf inklusive Dictionarytraining, Featuregenerierung und Klassifikation mit 100 Atomen, Sparsity 1 und 12 Iterationen Training der Dictionarys. Es gibt jeweils vier Trainings- und Testbilder der Größe 250×250. Diese werden mittels Stripe 1 in Patches unterteilt. Die Anzahl der Patches wird für das Dictionary-Training nicht reduziert, das heißt die Dictionarys werden dort für jede Klasse auf etwa 110000 Patches trainiert. Es wurden 15 Rotationen (Winkel von 24°) gewählt. Dies resultiert in 50 min Laufzeit für den Algorithmus ohne Rotation, 7,5 Stunden für den polaren Ansatz und 9,25 Stunden für den Ansatz mit der lenkbaren Basis. Es wurde keine Parallelisierung der Algorithmen vorgenommen. Der translationsinvariante Algorithmus braucht mit 66 Stunden deutlich länger, was unter anderem auf die Dictionarygröße $M = 200$ und die Sparsity $K = 9$ zurückgeführt werden kann.
Die Klassifikation der Zellkerne dauert im Vergleich deutlich länger. Dies liegt an den dort größeren Bildern. Der Standard-Algorithmus benötigt dafür ca. 4 Stunden, der polare 24,75 Stunden (11 Rotationen) und der lenkbare 33,75 (15 Rotationen). Der translationsinvariante Algorithmus braucht trotz verkleinerter Bildgröße je nach Parameterauswahl einen bis mehrere Tage.
Abschließend sei noch darauf hingewiesen, dass die Laufzeit deutlich von der konkreten Implementierung abhängt. In Python funktioniert es beispielsweise deutlich schneller, die Vektoren der Residuenliste (vgl. Alg. 1, Zeile 11) zunächst als Vektoren in eine Liste zu schreiben und dann mittels 'np.concatenate' zu einer Matrix zusammenzufassen, als wenn man in jedem Durchgang die Residuenmatrix um eine Spalte erweitert.

5.3 Klassifikation von Bildern

In diesem Absatz werden die Ergebnisse der Methode aus dem Kapitel 4.1 präsentiert. Es werden wieder die Texturbilder aus Kapitel 2.1 betrachtet. Dies entspricht exakt dem Vorgehen aus dem *Fast Rotational Sparse Coding* Paper [6]. Dort wurde zunächst mittels Cross Validation die beste Parameterkombination bestimmt. Die besten Parameter liefern schlussendlich folgende Ergebnisse:

Methode	M	N	K	Anzahl Testbilder	Patches	Genauigkeit
Standard	100	11	3	3840	4000	0,4432
Rotation	100	15	1	3840	4000	1,0000

Die Autoren des Papers [6] haben ihren Code jedoch nicht veröffentlicht. Daher wurde der Algorithmus anhand des beschriebenen Ansatzes und des teils vorhandenen Pseudocodes selbst implementiert. Es gibt allerdings einige Unklarheiten. Beispielsweise ist unklar, wie der orthogonale Matching Pursuit umgesetzt wurde. Der einzige Hinweis ist der Verweis auf das Paper [8] von Pati, Rezaiifar und Krishnaprasad. Technische Details wie das Runden lassen Raum für Ungenauigkeiten. Ein weiterer Unterschied kann möglicherweise durch die Wahl der Programmiersprache bedingt sein. In dem Paper wird angedeutet, dass Matlab benutzt wurde. Die Resultate dieser Arbeit beruhen auf Python-Code. Auch bezüglich der vorgestellten Methode gibt es Unklarheiten, was algorithmische Details angeht. Dies gilt beispielsweise für den verwendeten Winkel und das

Bestimmen von T_s der diskreten lenkbaren Basis (vgl. Kapitel 3.3.1).
Die simple Methode aus Kapitel 4.2, die Bilder der Klasse zuordnet, die zur Rekonstruktion am häufigsten gewählt wurde, liefert folgendes Ergebnis:

Methode	M	N	K	Winkel	Testbilder	Patches	Genauigkeit
Standard	100	11	3	-	3840	800	0,4638
Rotation	100	15	1	5°	1280	4000	0,9898
Polare Rotation	100	15	1	24°	384	4000	0,9688

Die Hinzunahme eines Klassifikators führt zu folgenden Resultaten

Methode	M	N	K	Winkel	Testbilder	Patches	Genauigkeit
Standard	100	11	3	-	3840	800	0,5354
Rotation	100	15	1	5°	1280	4000	0,9797
Polare Rotation	100	15	1	24°	384	4000	0,9479

Damit ist gezeigt, dass auch in der eigenen Implementation die rotationsinvarianten Algorithmen deutliche bessere Ergebnisse liefern als der standardmäßige Algorithmus. Ebenso wird deutlich, dass die Methode der polaren Rotation wie erwartet ein ähnliches Resultat wie die Variante mittels lenkbarer Basis liefert. Negativ hat sich für die polare Variante sicherlich ausgewirkt, dass weniger Rotationen berücksichtigt wurden. Ein exakt gleiches Ergebnis ist aber ohnehin nie zu erwarten, da durch die Diskretisierung in beiden Varianten Rundungsfehler entstehen und diese zu unterschiedlichen Ergebnissen führen. Außerdem unterscheiden sich in diesem Beispiel die Größe des Testsets und die Anzahl der Iterationen.
Interessanterweise sind für die rotationsinvarianten Ansätze die Ergebnisse ohne Klassifikator besser, wenn auch nur in sehr geringem Maße. Dafür ist für den Standard-Algorithmus, der insgesamt deutlich schlechtere Resultate hervorbringt, die Hinzunahme eines Klassifikators sehr wichtig.
Ein Ärgernis ist, dass die Erfolgsrate des Papers [6] nicht ganz getroffen wurde. Dort wurden keine Testbilder falsch klassifiziert. Die Rate von 99% zeigt jedoch, dass der Algorithmus richtig umgesetzt wurde, kleine Implementierungsdetails jedoch ein perfektes Ergebnis zunichte machen können. Die oben angesprochenen Unklarheiten machen eine exakte Rekonstruktion des Algorithmus leider unmöglich. Man sieht hingegen beispielsweise an den Ergebnissen der Standardmethode, dass der Algorithmus keinesfalls zwingend schlechter umgesetzt wurde. Für exakt die gleichen Parameter wurde ein deutlich besseres Ergebnis erzielt, obwohl auf weniger Patches trainiert wurde. Die Patches wurden, wie bereits beschrieben, zufällig gewählt. Weniger Patches zu benutzen hat also niemals (bis auf Rauschen) einen positiven Effekt. Falls zu wenig Patches benutzt werden, hat dies hingegen negative Effekte.
Hier wird ausdrücklich erneut darauf hingewiesen, dass die Testdaten rotierte Versionen der Trainingsdaten sind (siehe Kap. 2.1). Dies ist der Grund, warum so ungewöhnlich gute Klassifikationsergebnisse von (nahezu) 100 % möglich sind. Ein Algorithmus, der die Testbilder mit allen rotierten Versionen der Trainingsbilder vergleichen kann, sollte somit eine Genauigkeit von 100 % erreichen. Die Einteilung ist insofern kritisch, da sie in realen

Anwendungen für gewöhnlich so nicht vorliegt. In der Praxis werden Klassifikationsalgorithmen genutzt, um neue Bilder zu klassifizieren. Dort liegen keine Klasseneinteilungen in rotierten Versionen der jeweiligen Bilder vor.

5.4 Detektion von Objekten

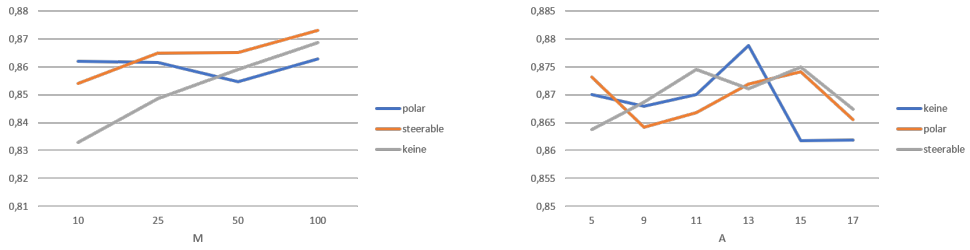
Bevor die translationsinvarianten Algorithmen auf den Zelldatensatz aus Kapitel 2.2 angewendet werden, werden diese als Zwischenschritt auf dem modifizierten Datensatz der Texturdaten (vgl. Kapitel 2.1) ausgetestet. Die Tests folgen dem Vorgehen, welches in Kapitel 4 beschrieben ist. Anschließend werden Detektionen auf den biologischen Daten vorgenommen.

5.4.1 Modifizierter Texturdatensatz

Im Gegensatz zum vorherigen Kapitel, wo bereits eine gute (laut dem Paper [6] die beste) Parameterkombination bekannt war, gilt es nun zunächst die beste Parameterkombnination herauszufinden.
Eine globale Optimierung aller Parameter ist aufgrund der hohen Anzahl an Parametern und laufzeitintensiven Algorithmen nicht möglich. Stattdessen werden strategisch Parameterkombinationen getestet, wobei die Parameter größtenteils nacheinander optimiert werden. Dies führt nicht zwingend zu der besten Parameterkombination, da möglicherweise Abhängigkeiten zwischen den Parametern bestehen.
Da die Patch-basierten Algorithmen eine deutlich geringere Laufzeit aufweisen, wird nun zunächst eine gute Parameterkombination für diese gesucht. Die Erkenntnisse werden später für den translationsinvarianten Ansatz genutzt, wo dann wesentlich weniger Parameter getestet werden.

Detektion durch Unterteilung in Patches Hier werden nun die Resultate auf Basis der Patch-basierten Methode aus Kapitel 4.2 vorgestellt. Das Durchlaufen des kompletten Detektionszyklus von Dictionary-Training bis Detektion der Testbilder nimmt je nach Parametern mehrere Minuten bis wenige Stunden in Anspruch. Es gibt jedoch Parameter, die ohne Neutrainieren und Neucodieren der Bilder optimiert werden können. Diese werden daher zusammen optimiert, da dies laufzeittechnisch vertretbar ist. Alle anderen Parameter, im Folgenden 'laufzeitintensiv' genannt, werden einzeln betrachtet.
Zunächst wird mithilfe der Erkenntnisse des vorherigen Kapitels eine Voreinstellung für die laufzeitintensiven Parameter getroffen, wobei teils bewusst Werte genommen werden, die nicht zwingend optimal sind, dafür jedoch eine etwas kürzere Laufzeit versprechen. Dies sind Dictionarygröße $M = 25$, Atomgröße $A = 11$, Sparsity $K = 1$, Iterationen $I = 10$. Die Anzahl der Patches wird zunächst auf 800 beschränkt, um die Laufzeit zu verringern, und im Laufe der Tests sukzessive erhöht. Für die anderen Parameter werden nun simultan folgende Werte getestet: $flatten\_to\_classes \in \{True, False\}$, $take\_actual\_values \in \{True, False\}$, $Filter \in \{gaussian, uniform\}$, $depth \in \{5, None\}$, $filter\_size \in \{5, 11\}$. Für alle Algorithmen stellt sich eindeutig heraus, dass $depth = None$, $flatten\_to\_classes = False$, $take\_actual\_values = True$ und

der uniforme Filter die beste Wahl sind.
Für den Random Forest wurde exemplarisch versucht den Tiefeparameter *depth* zu optimieren, der die maximal zugelassene Tiefe der einzelnen Bäume reguliert. Die Idee des Random Forest ist jedoch mithilfe von vielen unabhängigen vollständig ausgewachsenen Parametern eine optimale Vorhersage zu treffen. Dementsprechend zeigt es sich hier, dass es sinnvoll ist, die Tiefe der Bäume nicht zu beschränken (d.h. $depth = None$).
Weiterhin zeigt es sich, dass im Gegensatz zum Klassifikationskapitel 5.3 die Information der einzelnen Atome behalten und nicht zu Klassen zusammengefasst werden sollte. Da hier nur noch zwei statt 24 Klassen verfügbar sind, ergibt dies durchaus Sinn, da eine Bündelung der Information in zwei Variablen (eine pro Klasse) einen immensen Informationsverlust bedeutet und deutlich weniger Information erhält, als dies bei 24 Klassen der Fall war.
Eine kleine Überraschung ist, dass der uniforme Filter bessere Resultate geliefert hat. Dieser nimmt keine Gewichtung der Nachbarn nach Entfernung vor, dafür verfälscht er die Werte des Codes aber auch nicht. Diese Parameter werden aber am Ende ohnehin noch einmal getestet, da diese schnell zu testen sind und so mögliche Abhängigkeiten zwischen den Parametern zumindest zum Teil berücksichtigt werden.



Effekt der Dictionarygröße M Effekt der Atomgröße A

Abbildung 5.11: Fehlerraten der drei Algorithmen

Mit den Parametern werden nun die laufzeitintensiven Parameter nacheinander getestet. Zunächst werden für die Dictionarygröße M die Parameter $\{10, 25, 50, 100\}$ getestet. Es zeigt sich, dass für alle Algorithmen die größte getestete Größe $M = 100$ das beste Resultat aufweist. Der Unterschied zwischen den einzelnen Größen beeinflusst das Ergebnis stark (siehe Abb. 5.11). Die Atomgröße $A \in \{5, 9, 11, 13, 15, 17\}$ hat dagegen kaum Auswirkungen (Abb. 5.11). Dort sind für die rotationsinvarianten Algorithmen $A = 15$ und für den anderen $A = 13$ am besten.
Die Sparsity K hingegen zeigt wieder mehr Auswirkungen. Eine hohe Sparsity K führt definitiv zu schlechteren Ergebnissen, eine Sparsity von $K = 1$ scheint insgesamt am besten zu sein. Zu beachten ist, dass in der Abbildung 5.12 für den polaren Algorithmus der normale statt des orthogonalen Matching Pursuits gewählt wurde. Dies führt zu signifikant schlechteren Ergebnissen. Man sieht, dass selbst der orthogonale Matching Pursuit nicht dafür sorgt, dass eine höhere Sparsity als $K = 1$ gewählt werden sollte.
Schlussendlich zeigen letzte Tests, dass eine Erhöhung der Anzahl an Iterationen des Dictionary-Trainings auf $I = 15$ keine Verbesserung bringt. Weiterhin werden die nicht

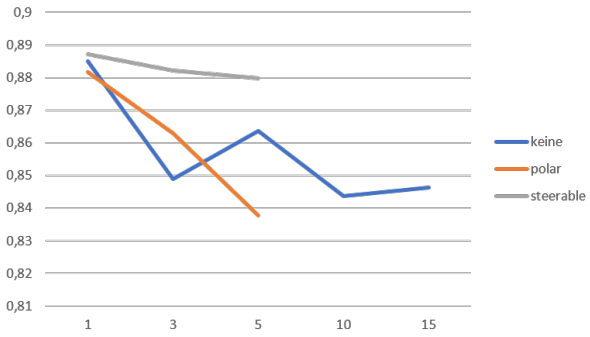
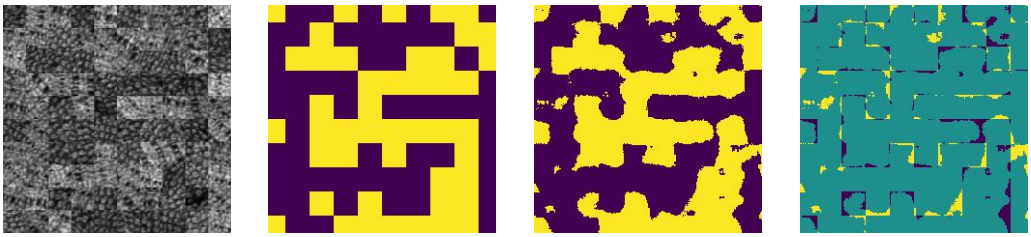


Abbildung 5.12: Einfluss der Sparsity K auf den Klassifikationsfehler

laufzeitintensiven Parameter, die anfangs festgelegt wurden, nochmal überprüft. Dies führt zu einer Änderung der Filtergröße für den Standard-Algorithmus auf 15 und für die rotationellen Algorithmen auf 13.



Bild

Zielmatrix: Schematische Darstellung der Klassen

Klassifikationsmatrix: Schematische Darstellung der vorhergesagten Klassen

Abweichungsmatrix: Darstellung der falsch klassifizierten Pixel

Abbildung 5.13: Resultat des finalen patch-basierten Laufs

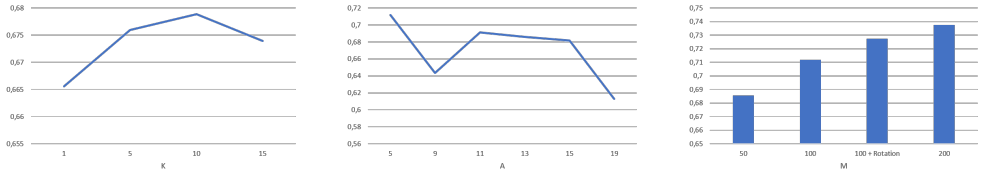
Mit diesen Parametern werden die Algorithmen nun auf neuen, bisher ungesehenen, Testdaten getestet. Im Gegensatz zum Trainings- und Validierungsdatensatz beinhaltet dieser nun rotierte Patches (vgl. Kapitel 2.1). Dies ist der Grund, warum der nicht rotationsinvariante Algorithmus dort wesentlich schlechtere Ergebnisse liefert.

Methode	Winkel	Genauigkeit
Standard	-	0,7979
lenkbare Rotation	5°	0,9136
lenkbare Rotation	24°	0,9080
lenkbare Rotation	30°	0,9065
Polare Rotation	24°	0.9104

In Abbildung 5.13 kann man das Klassifikationsergebnis visuell betrachten. Dargestellt ist eins der vier Testbilder, 91,5 % der Pixel wurden in diesem korrekt klassifiziert. Man sieht, dass sich wie erwartet die problematischen Pixel allesamt an den Klassengrenzen befinden.

Durch Nachbearbeitung lassen sich bessere Ergebnisse erzielen. Insbesondere ermöglicht die Information, dass das Bild aus aneinandergereihten quadratischen Patches gleicher Größe besteht, eine perfekte Klassifikation des Bildes, da stets ein Großteil der Pixel jedes Klassenpatches richtig klassifiziert wurde.

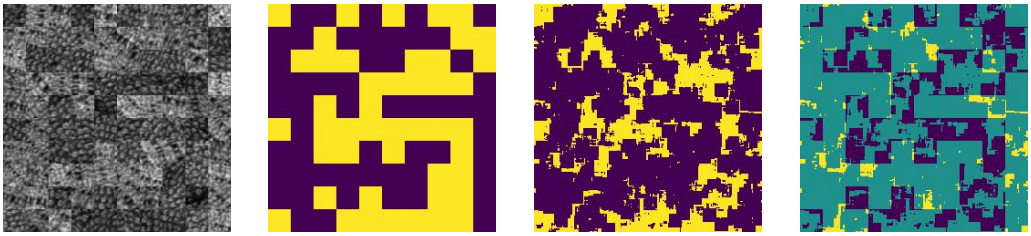
Translationsinvarianter Ansatz Auf Basis der vorherigen Erfahrungswerte werden nun die Parameter des translationsinvarianten Algorithmus nacheinander getestet. Zunächst die Sparsity K, da diese, wie bereits erwähnt, für diesen Ansatz deutlich höher gewählt werden sollte. Dies wird durch die Tests, zu sehen in Abbildung 5.14, bestätigt. Mit der Sparsity $K = 9$ werden die laufzeitarmen Parameter getestet, die wieder das Resultat $flatten\_to\_classes = False$, $take\_actual\_values = True$ und den uniformen Filter liefern. Bei der Atomgröße gewinnt der Wert 5 mit zusammengehöriger Filtergröße 15.



Sparsity K | Atomgröße A | Dictionarygröße

Abbildung 5.14: Einflüsse der Parameter auf den translationsinvarianten Ansatz

Anschließend wurden verschiedene Dictionarygrößen getestet. Als Alternative der Erhöhung der Dictionarygröße von 100 auf 200 wurde die Hinzunahme der um 90° rotierten Atome getestet (siehe Abb. 5.14). Dies ergab keine Verbesserung. Die Verwendung von 200 Atomen inklusive Rotation wurde wegen der hohen Laufzeit ausgeschlossen, ebenso wie noch größere Dictionarys.



Bild

Zielmatrix: Schematische Darstellung der Klassen

Klassifikationsmatrix: Schematische Darstellung der vorhergesagten Klassen

Abweichungsmatrix: Darstellung der falsch klassifizierten Pixel

Abbildung 5.15: Resultat des finalen translationsinvarianten Laufs

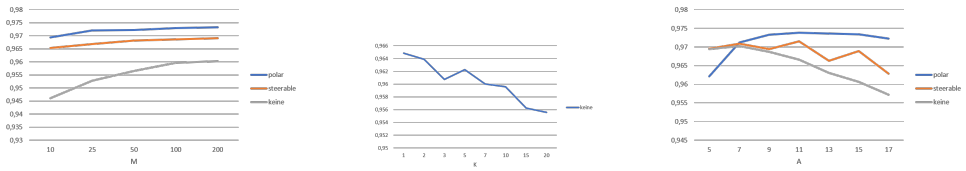
Alles in allem muss man feststellen, dass die Klassifikationsresultate deutlich hinter denen des patch-basierten Ansatzes zurückbleiben. Dies galt für alle erzielten Zwischenergebnisse während der Parameterfindungsphase und insbesondere auch für das finale

Ergebnis (siehe Abbildung 5.15). Dort wurden, über alle Bilder gemittelt, 66,51% Prozent der Pixel korrekt klassifiziert.
Die Gründe, warum der translationsinvariante Ansatz deutlich schlechtere Resultate fabriziert hat, sind hier leider nicht klar ersichtlich. Ein mögliche Ursache könnte die hier stark verkürzte Parametersuche sein. Allerdings zeigt sich im Vergleich, dass der Patch-basierte Ansatz auch schon zu Beginn der Parametersuche deutlich bessere Ergebnisse versprochen hat. Der Patch-basierte Ansatz ist also mindestens robuster, wahrscheinlich aber sogar insgesamt deutlich besser.
Eine zweite mögliche Ursache ist die Sparsity K. Für den Patch-basierten Ansatz hat sich gezeigt, dass eine kleine Sparsity von $K = 1$ die besten Detektionsresultate erbracht hat. Wie im Rekonstruktionskapitel 5.1 bereits dargelegt, repräsentiert der translationsinvariante Algorithmus das Bild für den Fall $K = 1$ nur sehr unzureichend, da viele Lücken auftauchen(vgl. Abb. 5.9). Dies könnte der Grund sein, dass auch für das Detektionsproblem, dass aufgrund der Features auf dem Rekonstruktionsproblem aufbaut, eine Sparsity von $K = 1$ ungeeignet ist. Allerdings führt eine höhere Sparsity dazu, dass die Informationen der Atome nicht mehr so aussagekräftig sind. Diese Störinformationen verhindern anscheinend ein gutes Detektionsresultat. Dies ist insofern belegt, dass auch für dieses Problem keine beliebig hohe Sparsity gewählt werden sollte (siehe Abb. 5.15).

5.4.2 Detektion der Zellkerne

Die Optimierung der Hyperparameter für die Detektion der Zellkerne geschieht analog zur vorherigen Vorgehensweise. Allerdings werden basierend auf den vorherigen Resultaten einige Tests angepasst

Detektion durch Unterteilung in Patches Als Ausgangsparameter fungieren Dictionarygröße $M = 100$, Atomgröße $A = 13$ ($A = 15$ für den rotationsinvarianten Fall), Sparsity $K = 1$ und Iterationen $I = 10$. In allen Fällen stellt sich wieder eindeutig heraus, dass der uniforme Filter, $flatten\_to\_classes = False$ und $take\_actual\_values = True$ die besten Klassifikationsresultate liefern.



Dictionarygröße Sparsity Atomgröße

Abbildung 5.16: Klassifikationsfehler

Wie Abbildung 5.16 zu entnehmen ist, liefert die Wahl der größten Dictionarygröße (hier $M = 200$) wieder das beste Resultat. Für die Sparsity wurde anhand des Standardalgorithmus der Wert $K = 1$ bestätigt. Die Atomgrößen wurden simultan mit der Filtergröße getestet. Die Abbildung 5.16 zeigt die Klassifikationsraten in Abhängigkeit der Atomgrößen, wobei jeweils das Maximum über alle verfügbaren Filtergrößen (alle ungeraden

Zahlen von 1 bis 19) abgebildet wird. Als beste Kombinationen ergeben sich für den Standardalgorithmus $A = 7$ mit $F = 19$, für den lenkbaren Ansatz $A = 11$ mit $F = 17$ und für den polaren Ansatz $A = 11$ mit $F = 17$.
Schlussendlich wurden alle Algorithmen auf neuen Daten getestet. Für den lenkbaren Algorithmus ergab die Wahl einer Winkelgröße von 5° einen Memory-Error, daher wurden dort analog zum vorherigen Kapitel wieder 15 Rotationen gewählt. Trotz dieser größeren Anzahl an Rotationen gegenüber dem polaren Ansatz (11 Rotationen, da 11 Pixel), fällt das Ergebnis mit 97,01% gegenüber 97,05% minimal schlechter aus. Der Unterschied ist jedoch nicht signifikant. Der Algorithmus ohne Rotationen liefert 96,58% korrekt klassifizierte Pixel. Das Resultat des polaren Ansatzes wird in Abbildung 5.17 gezeigt.

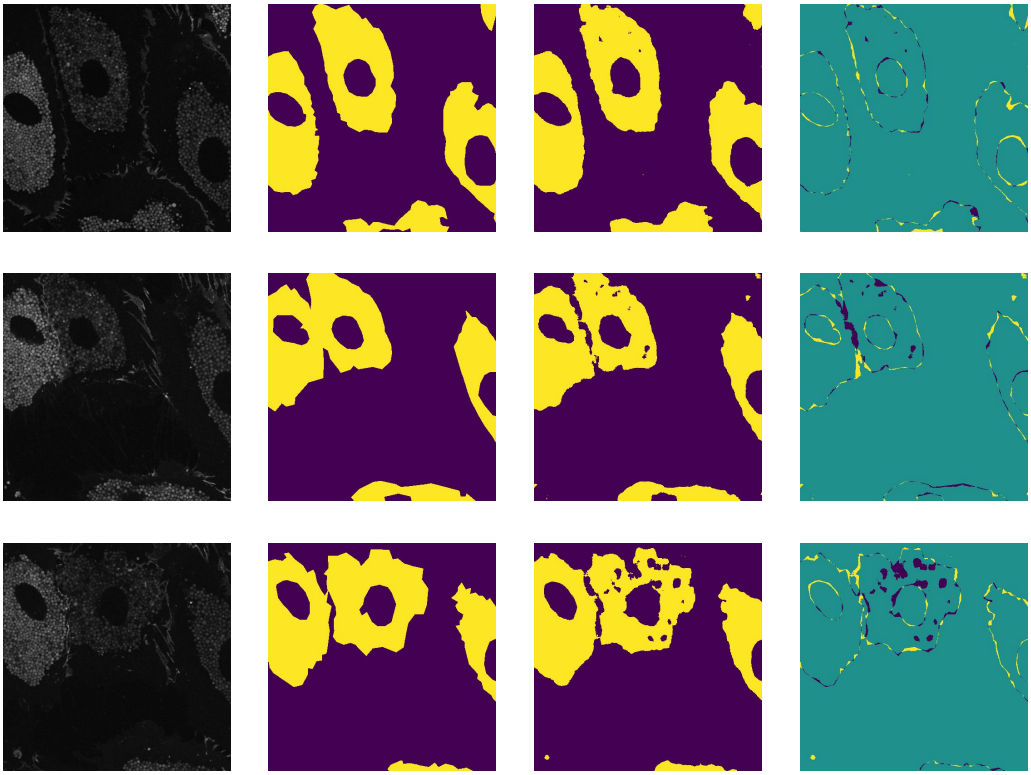
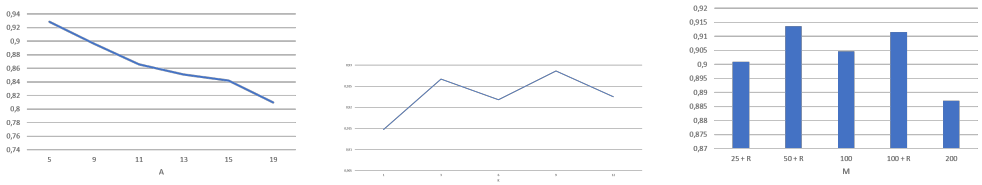


Bild | Zielmatrix: Schematische Darstellung der Klassen | Klassifikationsmatrix: Schematische Darstellung der vorhergesagten Klassen | Abweichungsmatrix: Darstellung der falsch klassifizierten Pixel

Abbildung 5.17: Resultate des finalen patch-basierten Laufs

Detektion mittels dem translationsinvarianten Ansatz Auch hier werden wieder die Resultate der vorherigen Testläufe für die Initialisation genutzt. Wie bereits in Kapitel 3.3.2 angekündigt, werden die Bilder geviertelt, um die Laufzeit zu verringern. Basierend auf den Parametern des finalen translationsinvarianten Durchlaufs der Texturdaten er-

geben sich als beste Parameter in aufeinanderfolgenden Tests eine Atomgröße von $A = 5$, eine Sparsity von $K = 9$ und die Dictionarygröße $M = 100$ (vgl. Abbildung 5.18).



Atomgröße A | Sparsity K | Dictionarygröße

Abbildung 5.18: Einflüsse der Parameter auf den translationsinvarianten Ansatz

Einige finale Ergebnisse sind in Abbildung 5.19 dargestellt.. Insgesamt sieht man, dass auch hier die Detektionsergebnisse mit 92,5 % besser als für den Texturdatensatz sind. Das Klassifikationsproblem scheint also einfacher zu sein. Jedoch bleiben die Ergebnisse des translationsinvarianten Ansatzes erneut hinter den Resultaten der Patch-basierten Verfahren zurück.

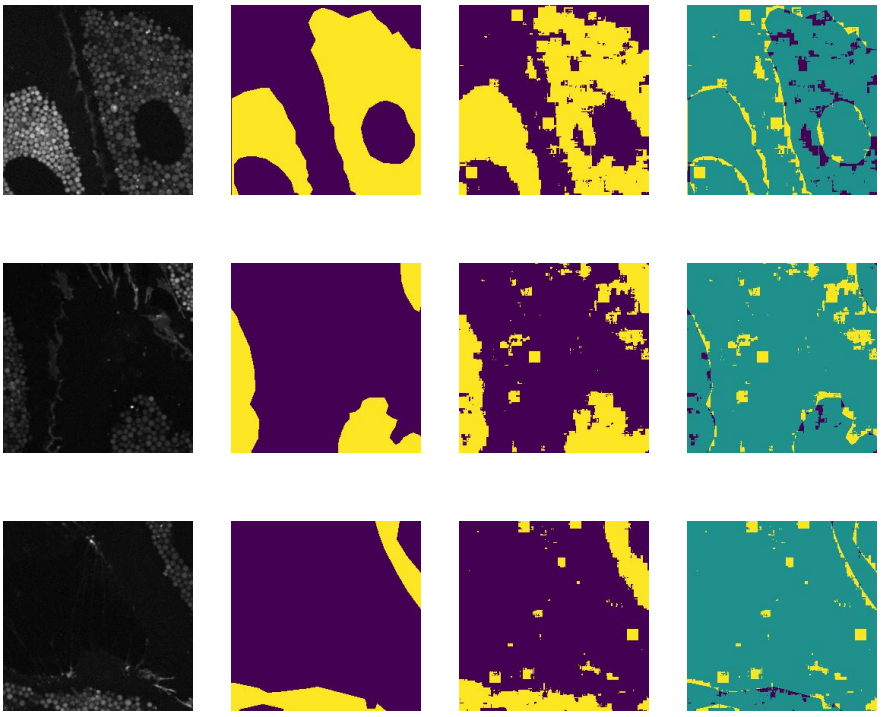


Bild | Zielmatrix: Schematische Darstellung der Klassen | Klassifikationsmatrix: Schematische Darstellung der vorhergesagten Klassen | Abweichungsmatrix: Darstellung der falsch klassifizierten Pixel

Abbildung 5.19: Klassifikationsergebnis des translationsinvarianten Algorithmus

Hier würde also sicherlich eine Nachbearbeitung der Klassifikationsergebnisse helfen, denn selbst große zusammenhängende Strukturen werden nicht komplett korrekt zugeordnet. Dies könnte geschehen, indem nach der Klassifikation für alle Pixel eine erneute Klassifikation, basierend auf den Klassen der Nachbarpixel, geschieht. Es würde also gewissermaßen eine Glättung des Klassifikationsergebnisses vorgenommen. In dieser Arbeit wird darauf jedoch verzichtet, da die Ergebnisse dennoch schlechter als die Ergebnisse der Patch-basierten Verfahren wären.

5.4.3 Detektion der JAIL

Die Detektion der JAIL stellt sich als ungleich schwieriger heraus. Dies liegt einerseits an der Struktur der JAIL. Diese sind klein und veränderlich, wie in Kapitel 2.2 bereits geschildert wurde. Andererseits sind die JAIL auch nur an sehr wenigen Pixeln des Bildes zu finden. Damit liegt also ein extrem unbalanciertes Klassifikationsproblem vor.
Für die Auswertung der Performance muss die Unbalanciertheit beachtet werden. Es sind in etwa auf 1 % aller Pixel JAIL zu finden. Man sollte daher nicht nur den Gesamtklassifikationsfehler betrachten, da beispielsweise ein Klassifikator, der überhaupt keine JAIL findet, eine Genauigkeit von 99 % erzielt. Darum fließen neben der Gesamtrate auch andere Messgrößen wie Precision und Recall in die Auswertung ein.
Auch mit dieser Einschränkung sind die Ergebnisse der Algorithmen schlechter als bei den vorherigen Datenproblemen. Die Vergleichbarkeit leidet unter den wenigen klassifizierten JAIL. Daher werden hier nur die Ergebnisse des polaren Algorithmus präsentiert. Dies ist vor allem auch insofern vertretbar, dass das Patch-basierte Verfahren in den vorherigen Kapiteln stets deutlich bessere Ergebnisse hervorgebracht hat als das translationsinvariante Verfahren. Die Unterschiede der Algorithmen für das Patch-basierte Verfahren sind hingegen gering. Als bester Algorithmus hat sich stets der polare Algorithmus hervorgetan.
Die Tests folgen wieder demselben Schema, wie in den vorherigen Subkapiteln. Einziger Zusatz ist das Testen verschiedener Werte für den 'class_weight' Paramter des Random Forests. Dieser kann durch die Gewichtung der einzelnen Datensätze die Unbalanciertheit des Gesamtproblems ausgleichen. Die Ergebnisse, die dieses Verfahren hervorgebracht hat, waren jedoch sehr schlecht. Aus diesem Grund werden im Folgenden nur die Tests präsentiert, in denen dieser Parameter nicht verändert wurde.
Die vergangenen Tests haben gezeigt, dass große Dictionarygrößen für die Patch-basierten Verfahren stets förderlich waren. Daher werden auch für dieses Problem große Dictionarys verwendet. Allerdings wird bei den Dictionarygrößen die stark unterschiedlichen Anzahl an Datensätzen berücksichtigt. Es werden die Verhältnisse 50:250 und 25:250 getestet, wobei erstere Zahl die Anzahl der Atome für die Klasse JAIL und letztere die Anzahl der Atome für die andere Klasse darstellt.
Die Atomgrößen haben sich nicht als irrelevant herausgestellt, Werte wie $A = 11$ und $A = 15$ haben aber stets gute Performance geliefert. Insbesondere sind die Werte $A = 11$ und $A = 15$ die Gewinner der anderen beiden Detektionsaufgaben. Daher werden nun nur diese beiden Atomgrößen getestet. Als Filtergrößen sind alle ungeraden Zahlen zwischen 9 und 19 zugelassen.

Für die weiteren Parameter werden die bereits gesammelten Erkenntnisse genutzt. Für beide vorherigen Datensätze waren stets $flatten\_to\_classes = False$, $take\_actual\_values = True$, $Filter = uniform$ und $depth = None$ signifikant besser als die anderen getesteten Werte, weswegen diese zur Vereinfachung ungetestet übernommen werden.
So erhält man für die Parameteroptimierung auf dem Validierungsset folgende Ergebnisse:

M(JAIL)	A	FilterSize	Korrekt(rel.)	TNR	Precision	Recall
50	15	13	0,9930	0,9934	0,7442	0,1726
50	15	17	0,9929	0,9932	0,8156	0,1418
50	15	11	0,9929	0,9934	0,7259	0,1689
50	15	15	0,9929	0,9932	0,7696	0,1433
25	11	13	0,9928	0,9931	0,7345	0,1144
25	11	13	0,9928	0,9931	0,7345	0,1144
50	15	19	0,9927	0,9928	0,8798	0,0980
50	11	17	0,9927	0,9928	0,9064	0,0771
25	11	15	0,9927	0,9929	0,7664	0,0973
25	11	15	0,9927	0,9929	0,7664	0,0973
50	11	13	0,9927	0,9930	0,7361	0,1036
25	11	11	0,9927	0,9931	0,6908	0,1181
25	11	11	0,9927	0,9931	0,6908	0,1181
25	11	17	0,9927	0,9928	0,8327	0,0742
25	11	17	0,9927	0,9928	0,8327	0,0742
25	11	9	0,9926	0,9931	0,6449	0,1158
25	11	9	0,9926	0,9931	0,6449	0,1158
50	15	9	0,9926	0,9932	0,6383	0,1499
50	11	11	0,9926	0,9930	0,6601	0,1027
50	11	19	0,9926	0,9926	0,9897	0,0486
50	11	15	0,9925	0,9927	0,7333	0,0615
25	11	19	0,9925	0,9926	0,8212	0,0498
25	11	19	0,9925	0,9926	0,8212	0,0498
50	11	9	0,9925	0,9930	0,5976	0,1012

Es stellt sich heraus, dass einzig die Filtergrößen einen großen Einfluss auf die Detektionsgenauigkeit haben. In Abbildung 5.20 sieht man die Entwicklung von Precision und Recall in Abhängigkeit der Filtergröße für die JAIL-Dictionarygröße 50. Je größer die Filter gewählt werden, desto weniger JAIL werden detektiert. Dies sorgt einerseits für eine bessere Precision, andererseits aber für einen schlechteren Recall. Dies ergibt durchaus Sinn, da die JAIL sehr kleine Gebilde sind. Je mehr Nachbarinformationen miteinbezogen werden, desto mehr 'Nicht-JAIL' Pixel werden miteinbezogen. Dadurch werden weniger Pixel überhaupt als JAIL klassifiziert.
Die Wahl des Parameters ist also abhängig von der Problemstellung. Ist gewünscht, dass

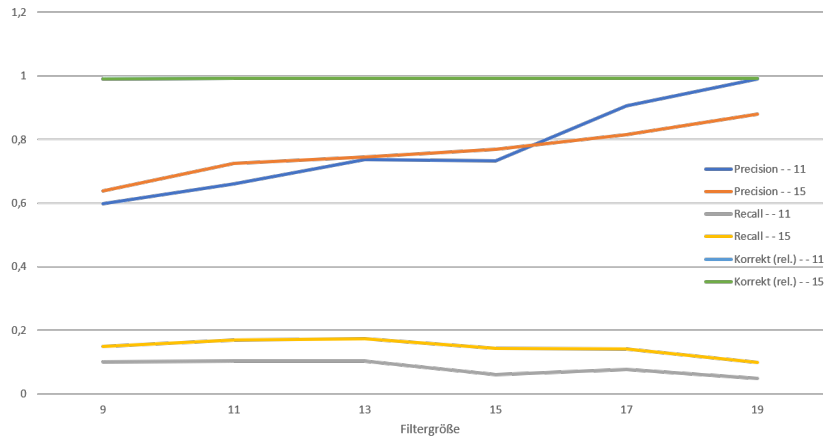
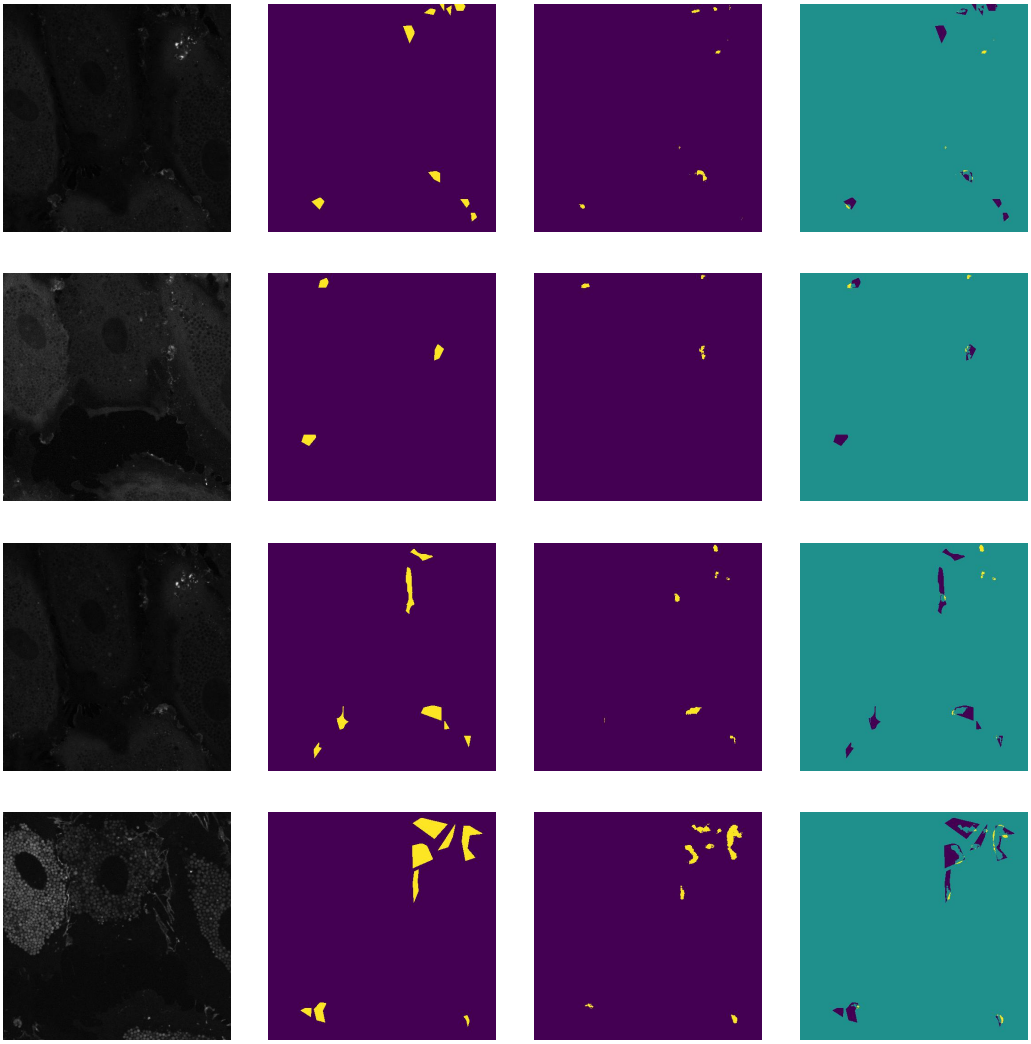


Abbildung 5.20: Gesamtfehler, Precision und Recall für die Atomgrößen 11 und 15.

kaum falsche JAIL detektiert werden, so ist eine große Filtergröße zu wählen. Es werden dann allerdings nur einzelne Pixel der JAILs detektiert. Sollen die JAIL als Ganzes erkannt werden, müssen mehr Falschdetektionen hingenommen werden und eine kleinere Filtergröße kann gewählt werden. Hier werden die Parameter genommen, die den höchsten Recall und gleichzeitig besten Gesamtfehler erzeugen. Das sind $M = 50 : 250$, $A = 15$ und Filtergröße $= 13$.
Schlussendlich liefert der die Objektdetektion mit diesem Vorgehen eine Genauigkeit von 98,76% korrekt klassifizierter Pixel über die vier Testbilder. Dies setzt sich aus einer True Negative Rate (entspricht den korrekt klassifizierten 'Nicht-JAIL' Pixeln) von 98,85%, einer Precision von 78,27% und einem Recall von 23,14% zusammen. Diese verschiedenen Messgrößen lassen sich in den Bildern der Abbildung 5.21 auch visuell betrachten. Es werden sehr wenige Pixel fälschlicherweise als JAIL klassifiziert, was die hohe True Negative Rate ausdrückt. Auch die Zuordnung zu der Klasse JAIL geschieht mit einer recht hohen Sicherheit, was durch die Precision ausgedrückt wird. Allerdings werden nur 4368 JAIL überhaupt als JAIL detektiert, obwohl 14772 Pixel JAIL zeigen. Dies erklärt den mit 3419 von 14472 korrekt klassifizierten JAIL Pixeln kleinen Recall.
Alles in allem kann man also feststellen, dass sich eine pixelgenaue Detektion der JAIL als zu schwierig herausstellt. In den vorherigen Detektionsproblemen war zu sehen, dass die Algorithmen insbesondere an den Rändern der Klassen große Probleme haben, die Pixel korrekt zu klassifizieren. Jedoch ist es so, dass die JAIL sehr klein sind, also kaum innere Pixel der Klasse JAIL vorliegen. Bei genauer Betrachtung der Bilder fällt aber auf, dass für den Großteil der JAIL zumindest einige Pixel als JAIL klassifiziert werden. Dies ist dennoch ein gutes Ergebnis, denn es detektiert die JAIL. Eine pixelgenaue Umrandung wird so nicht erreicht, könnte allerdings durch Nachbearbeitung auf Basis von Helligkeitswerten oder anderen Verfahren möglicherweise sogar erreicht werden. Es ist allerdings unklar, ob diese pixelgenaue Beschreibung überhaupt gewünscht wird, oder eine positionsgetreue Detektion der JAIL ausreicht. Dies kann durch Clusterung der detektierten Pixel leicht erreicht werden.



Bild

Zielmatrix: Schematische Darstellung der Klassen

Klassifikationsmatrix: Schematische Darstellung der vorhergesagten Klassen

Abweichungsmatrix: Darstellung der falsch klassifizierten Pixel

Abbildung 5.21: Resultate des finalen patch-basierten Laufs

6 Fazit

In dieser Arbeit wurden zunächst sowohl für das rotationsinvariante als auch für das translationsinvariante Sparse Coding Optimierungsproblem Lösungsalgorithmen vorgestellt und analysiert (Kap. 3). Für das rotationsinvariante Sparse Coding wurde darüber hinaus ein Algorithmus geschaffen, der bei gleicher Rekonstruktionsqualität die Laufzeit des *Fast Rotational Sparse Coding* Algorithmus unterbietet.
Anschließend wurde gezeigt, wie mithilfe der transformationsinvarianten Algorithmen das Klassifikationsproblem erfolgreich auf das Detektionsproblem übertragen werden kann, was im Wesentlichen durch eine pixelweise Klassifikation erreicht wird (Kap.4). Nach dieser erfolgreichen theoretischen Übertragung des Problems, wurde diese praktisch validiert. Da eine laufzeiteffiziente Kombination auf rotationsinvariantem und translationsinvariantem Ansatz nicht möglich ist, wurden die beiden Verfahren gesondert betrachtet (Kap. 5).

Für den rotationsinvarianten Ansatz beruht die Detektion auf der Unterteilung des Bildes in Patches. Die überragenden Klassifikationsergebnisse von (nahezu) 100% konnten so auf dem modifizierten Texturdatensatz mit einer Genauigkeit von über 90 % übertragen werden. Falsch klassifizierte Pixel sind dabei ausschließlich an den Klassengrenzen zu finden. Auch die Detektion der Zellkerne gelingt mit ungefähr 97 % korrekt zugeordneter Pixel sehr gut. Für das ungleich schwierigere Detektionsproblem der JAIL wurden jedoch schlechtere Resultate erzielt. Dort konnten zwar viele JAIL korrekt entdeckt werden, eine pixelgenaue Detektion scheint jedoch nicht möglich zu sein.
Eine mögliche Fortführung dieser Variante wäre die Nachbearbeitung der Resultate. Die Klassifikation eines Pixels könnte beispielsweise in einem nachgelagerten Schritt anhand der Klassen der Nachbarpixel angepasst werden, was etwa durch eine Glättung der Klassifikationsmatrix geschehen kann. Auch komplexere Verfahren, die sowohl die Klassen der Nachbarpixel als auch die Sparse Coding Information des jeweiligen Pixels selbst in Betracht ziehen, sind als Zusatzschritt denkbar.
Eine andere Möglichkeit, die Methodik auszubauen, ist der Einsatz anderer Klassifikatoren. Der Random Forest hat zwar gute Ergebnisse geliefert, was jedoch nicht heißt, dass andere Klassifikatoren für dieses Problem schlechter geeignet sind. Der Random Forest wurde auch aufgrund der Überlegung ausgewählt, einen Klassifikator zu nehmen, der mit einer angemessenen Komplexität die durchs Sparse Coding gewonnen Features erfolgreich nutzt. So konnte belegt werden, dass durch das Sparse Coding wertvolle Informationen generiert werden. Da aufgrund der pixelweisen Betrachtung aber eine sehr hohe Anzahl an zu klassifizierenden Datenpunkten vorliegt, kann mithilfe eines komplexeren Klassifikators möglicherweise eine noch bessere Detektion der Objekte erlangt werden.

Die Ergebnisse des translationsinvarianten Algorithmus sind hingegen enttäuschend. Die Detektion ist zwar besser als eine zufällig generierte Detektion, für praktische Einsätze sind die Resultate jedoch nicht zu gebrauchen. Als möglicher Grund wurde die nötige hohe Sparsity K genannt, die auf die Beschaffenheit des Algorithmus zurückzuführen ist. Auch die lange Laufzeit des Algorithmus war problematisch, denn dies führt zu einer Einschränkung der Datenmenge und der Trainingszeit.
In dieser Arbeit wurden für das Sparse Coding ausschließlich Problemformulierungen mit der $l_0 - Norm$ verwendet. Dies stellt eine Einschränkung dar, da in der Literatur die Formulierungen des Optimierungsproblems sowohl mit der $l_0 - Norm$ als auch mit der $l_1 - Norm$ zu finden sind. Insbesondere für das translationsinvariante Sparse Coding sollte also ein Algorithmus auf Basis der $l_1 - Norm$ getestet werden. Hier kommt beispielsweise der Algorithmus aus dem Paper *Fast Convolutional Sparse Coding* von Bristow, Eriksson und Lucey [3] infrage, der eine Lösung des Laufzeitproblems verspricht.
Eine weitere Einschränkung stellen die verwendeten Daten dar. Für das Objektdetektionsproblem wurden zwei Datensätze mit insgesamt drei Fragestellungen getestet. Aufgrund der verschiedenen Schwierigkeitsgrade ist zwar eine erste gute Einschätzung der Güte des Verfahrens möglich. Außerdem konnte gezeigt werden, wie gut die Methode auf Daten, die direkte Verbindung zu aktuellen biologisch-medizinischen Forschungsfragen haben, funktioniert. Für eine endgültige Bewertung der Algorithmen sollten diese aber auf noch anderen Daten angewendet werden.
Letztlich sind auch die betrachteten Transformationen eine Einschränkung. Insbesondere die Skalierung, die zu den zweidimensionalen Transformationen gehört, wurde nicht betrachtet. Des Weiteren wurden in dem translationsinvarianten Algorithmus zwar bis zu vier Rotationen berücksichtigt, was aufgrund der Laufzeit nicht weiter erhöht werden konnte. Dies entspricht aber keiner vollkommenen Kombination von rotationsinvariantem und translationsinvariantem Sparse Coding. Falls eine erfolgreiche translationsinvariante Methodik zur Objektdetektion gefunden wird, sollte also sicherlich darüber nachgedacht werden, wie dort Rotationsinvarianz eingebracht werden kann.

Alles in allem kann dank der Patch-basierten Methodik ein positives Fazit gezogen werden. Das Potenzial der auf den Codes des Sparse Codings basierenden Objektdetektion konnte erfolgreich aufgezeigt werden. Zudem wurde gezeigt, dass Rotationsinvarianz die Detektion verbessert. Das Ausmaß der Verbesserung hängt dabei stark von den Daten ab. Translationsinvariantes Sparse Coding konnte hingegen nicht zufriedenstellend für die Objektdetektion genutzt werden. Dort sollten in der Zukunft andere Algorithmen eingesetzt werden, um die translationsinvarianten Codes zu bestimmen.

Literaturverzeichnis

[1] Aharon, M.; Elad, M. und Bruckstein, A.M.: *K-SVD: An algorithm for Designing of Overcomplete Dictionaries for Sparse Representation* Technion—Israel Inst. of Technology, Tech. Ref. (2005)

[2] Aharon, M.; Elad, M. und Bruckstein, A.M.: *K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation* IEEE Transactions on signal processing, vol. 54, no. 11, (November 2006)

[3] Bristow, H.; Eriksson, A. und Lucey, S.: *Fast Convolutional Sparse Coding* IEEE Conference on Computer Vision and Pattern Recognition June 2013

[4] Davis, G.; Mallat, S. und Zhang, Z.: *Adaptive Time-Frequency Decompositions with Matching Pursuits* Proceedings of the IEEE-SP International Symposium on Time-Frequency and Time-Scale Analysis (1992)

[5] Mallat, S. G. und Zhang, Z.: *Matching Pursuits with Time-Frequency Dictionaries.* IEEE Transactions on Signal Processing. 1993 (12): 3397–3415. Bibcode:1993ITSP...41.3397M. doi:10.1109/78.258082 (1993)

[6] McCann, M.T.; Unser M. und Depeursinge A.: *Fast Rotational Sparse Coding.* arXiv:1806.04374v1 (12. Juni 2018)

[7] Ojala, T.; Mäenpää, T.; Pietikäinen, M.; Viertola, J.; Kyllönen, J. und Huovinen, S.: *Outex - New Framework for Empirical Evaluation of Texture Analysis Algorithms http : //www.ee.oulu.fi/research/imag/mvg/files/pdf/pdf*$_3$*14.pdf* (2002)

[8] Pati, Y. C.; Rezaiifar, R. und Krishnaprasad, P. S. : *Orthogonal Matching Pursuit: Recursive Function Approximation with Applications to Wavelet Decomposition.* Proceedings of 27th Asilomar Conference on Signals, Systems and Computers (1. November 1993)

[9] Plaud, E. und Giryes, R.: *A Greedy Approach to* $l_{0,\infty}$ *Based Convolutional Sparse Coding.* arXiv:1812.10538v1 [eess.IV] (26. Dezember 2018)

[10] Seebach, J.; Cao, J.; Schnittler, H.-J.: *Quantitative dynamics of VE-cadherin at endothelial cell junctions at a glance: basic requirements and current concepts* discoveriesjournals.org (2016)

[11] Taha, A.A.; Taha, M; Seebach; J. und Schnittler, H.-J.: *ARP2/3-mediated junction-associated lamellipodia control VE-cadherin–based cell junction dynamics and maintain monolayer integrity.* Mol Biol Cell 25:245-256 (2014)

[12] University of Auckland, New Zealand: *Gaussian Filtering* https://www.cs.auckland.ac.nz/courses/compsci373s1c/PatricesLectures/Gaussian%20Filtering_1up.pdf (abgerufen 30.09.2019)

[13] University of Oulu Finland: Center for Machine Vision Research: *Outex Texture Database* (2002)

Plagiatserklärung der / des Studierenden

Hiermit versichere ich, dass die vorliegende Arbeit über _____ _____ selbstständig verfasst worden ist, dass keine anderen Quellen und Hilfsmittel als die angegebenen benutzt worden sind und dass die Stellen der Arbeit, die anderen Werken – auch elektronischen Medien – dem Wortlaut oder Sinn nach entnommenen wurden, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht worden sind.

_____
(Datum, Unterschrift)

Ich erkläre mich mit einem Abgleich der Arbeit mit anderen Texten zwecks Auffindung von Übereinstimmungen sowie mit einer zu diesem Zweck vorzunehmenden Speicherung der Arbeit in eine Datenbank einverstanden.

_____
(Datum, Unterschrift)