Example title

Masterarbeit

von

Johannes Baßler, B.Sc.

am

Institut für Industrielle Informationstechnik

Zeitraum:	01. 01. 2010 – 30. 06. 2010
Hauptreferent:	Prof. Dr.-Ing. Michael Heizmann
Betreuer:	M.Sc. Max Mustermann
	Dr.-Ing. Hans Maier[1]
	Dipl.-Ing. John Doe[1]

[1] XYZ AG - Karlsruhe

Erklärung

Ich versichere hiermit, dass ich meine Masterarbeit selbstständig und unter Beachtung der Regeln zur Sicherung guter wissenschaftlicher Praxis im Karlsruher Institut für Technologie (KIT) in der aktuellen Fassung angefertigt habe.
Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und wörtlich oder inhaltlich übernommene Stellen als solche kenntlich gemacht.

Karlsruhe, den 30. Juni 2010

Kurzfassung

Hier könnte eine deutsche Kurzfassung kommen.

Abstract

Here comes an english abstract. (This is optional: If not needed, please delete this environment)

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Motivation

1.1 Anomaliedetektion in der Industrie

Hier wird die Relevanz von Anomaliedetektion in der Industrie erläutert.

1.2 Unüberwachte Anomaliedetektion

Hier wird ausgeführt warum überwachte Methoden an ihre Grenzen stoßen und warum unüberwachte Methoden sinnvoll sind.

1.3 Laufzeitoptimierung

Dann wird ausgeführt, warum eine Laufzeitoptimierung sinnvoll ist und warum es auch heute noch sinnvoll sein kann, auf teure GPUs zu verzichten.

Kapitel 2

Grundlagen

Hier wird kurz aufgezählt was erläutert wird. Mehr nicht. Es sollen hier nur Elemente erläutert werden, die für mindestens zwei der Methoden relevant sind. k-center greedy (PatchCore), Autoencoder (efficientad) oder Backpropagation (simplenet) also zB nicht.

2.1 Datensatz: MVTecAD[2]

Das „MVTec Anomaly Detection Dataset“ (MVTec AD) ist ein am 6. Januar 2021 veröffentlichter, umfangreicher Datensatz für die Anomalieerkennung in Bildern. Dieser bildet die Evaluationsgrundlage aller hier in dieser Arbeit vorkommendene Entwicklungen und Methoden. Treibende Kraft hinter der Entwicklung des Datensatzes ist die MVTec Software GmbH, ein deutsches Unternehmen, das sich auf industrielle Bildverarbeitung spezialisiert hat. Dieser Datensatz ist entwickelt worden, um einen internationalen Benchmark zu schaffen, der die Entwicklung von Algorithmen für die Unüberwachte Anomalieerkennung in Bildern vorantreibt und Methoden quantitativ vergleichbar macht. Betrachtet man die Anzahl an Veröffentlichungen, beispielsweise auf https://paperswithcode.com/sota/anomaly-detection-on-mvtec-ad, fällt auf, dass seit der Veröffentlichung des Datensatzes immer mehr Methoden auf diesem Datensatz evaluiert werden. Waren es 2020 22 Veröffentlichungen stieg die Anzahl streng monoton bis auf bereits 80 im laufenden Jahr 2023 (Stand 11.10.2023).
Der Datensatz besteht aus insgesamt 5354 Bildern, die 15 verschiedene Klassen von Objekten enthalten. Die folgende Tabelle gibt einen Überblick über die Klassen und die Anzahl an Bildern pro Klasse. Diese 15 Klassen lassen sich in zehn Objektklassen und fünf Texturklassen unterteilen. Die fünf ersten Klassen der Tabelle (Carpet, Grid, Leather, Tile, Wood) sind Texturen bzw. Strukturen, die weiteren zehn Klassen (Bottle, Cable, Capsule, Hazelnut, Metal nut, Pill, Screw, Toothbrush, Transistor, Zipper) sind Objekte.
Wie bereits erwähnt, handelt es sich um einen Datensatz für Unüberwachte Anomaliedetekion, was sich daran erkennen lässt, dass in den Trainingsdaten ausschließlich Bilder ohne Defekte (nominal) enthalten sind. Die Testdaten sind in zwei Klassen unterteilt: „Good“ und „Defective“. Zwar liegen in den meisten Fällen mehrere mögliche Defektklassen vor, die auch eindeutig gelabelt werden, allerdings handelt es sich um einen Binärklassifikationsdatensatz, was bedeutet, dass das Erkennen der Art des Defektes keine Zielstellung ist. Durch diese Vielzahl an Defekten kann aber eine gewisse Generalisierungsfähigkeit getestet werden.

Tabelle 2.1: Übersicht über Anzahl an Bildern, Auflösung und Defektgruppen des Datensatzes

Kategorie	#Training	#Test (nominal)	#Test (anomal)	#Defekt Gruppen	#Defekt Regionen	Seitenlänge
Teppich	280	28	89	5	97	1024
Gitter	264	21	57	5	170	1024
Leder	245	32	92	5	99	1024
Fließen	230	33	84	5	86	840
Holz	247	19	60	5	168	1024
Flasche	209	20	63	3	68	900
Kabel	224	58	92	8	151	1024
Kapsel	219	23	109	5	114	1000
Haßelnuss	391	40	70	4	136	1024
Metallmutter	220	22	93	4	132	700
Pille	267	26	141	7	245	800
Schraube	320	41	119	5	135	1024
Zahnbürste	60	12	30	1	66	1024
Transistor	213	60	40	4	44	1024
Reißverschluss	240	32	119	7	177	1024
Total	3629	467	1258	73	1888	-

Alle anomalen Testbilder habem eine pixelweise Annotation, die die Defekte markiert. Diese Annotationen sind in Form von Binärbildern gegeben, wobei die Pixel der Defekte mit 1 und die Pixel der nominalen Regionen mit 0 markiert sind. In dieser Arbeit liegt zwar der Fokus auf der Instanzklassifizierungsgenauigkeit und nicht auf der Segmentierung. Dennoch ist diese qualitativ hochwertige Annotation ein wohl wesentlicher Grund für die Beliebtheit des Datensatzes. Nicht nur lässt sich damit schlicht die Segmentierungsgenauigkeit testen, sondern der Vergleich einer von einer Methode zu einem Testbild festgestellte Anomaliekarte mit der Annotation ist ein wertvolles Werkzeug, um die Funktionsweise einer Methode zu verstehen und etwaige Schwachstellen zu identifizieren.
Anzumerken ist, dass es sich bei den vorliegenden Defekten um keine logischen Defekte handelt, sonder um lokale, strukturelle Abweichungen von der Norm. Die grundsätzliche Gestalt ist auch im Falle eines anomalen Bildes erhalten. Möchte man zum Beispiel ein Anomaliedetekionsverfahren entwickeln, das zu lang geratene oder stark gekrümmte Schrauben erkennt, so ist der Datensatz nicht optimal geeignet. Hierzu sei auf den neueren Datensatz aus dem Hause MVTec, MvTec LOCO AD verwiesen. [8] Beisple für nominale und anomale Bilder aus dem Datensatz finden sich in Abbildung 2.1.

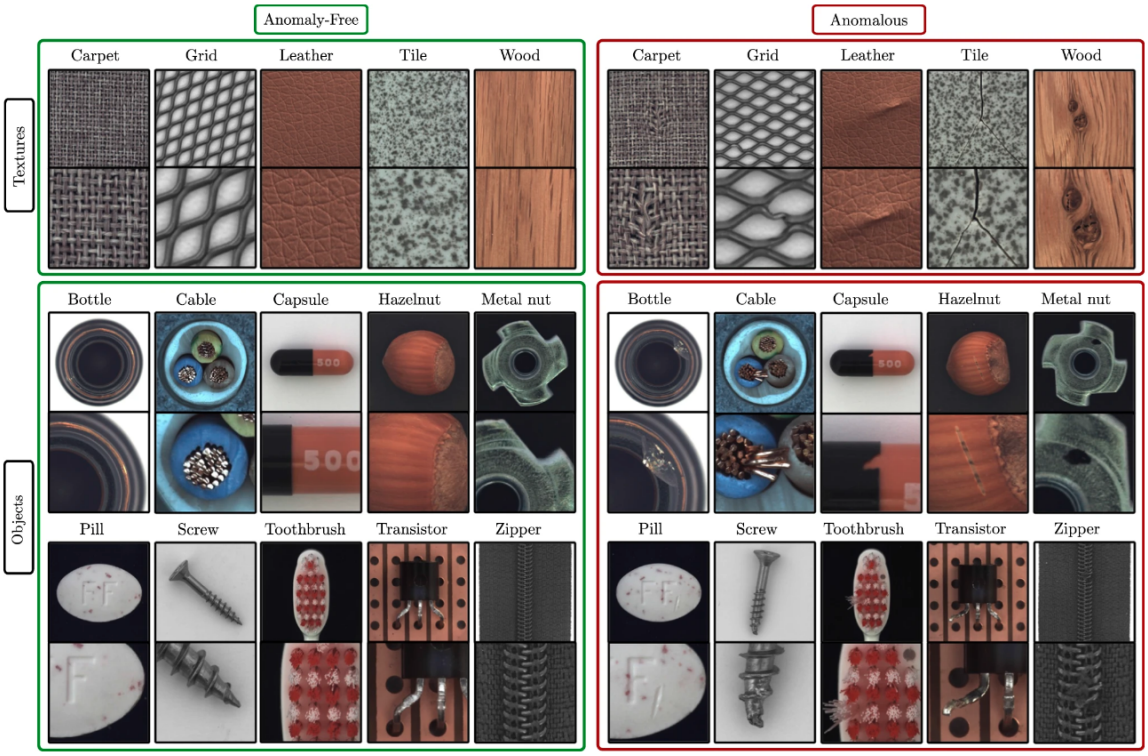


Abbildung 2.1: Beispiele für nominale und anomale Bilder aus dem MVTec AD Datensatz

2.2 Eigener Datensatz (Granulat)

Details und Beispiele zu eigenem Datensatz.

2.3 AUROC[5]

In dieser Arbeit, genauso wie in beinahe allen anderen Arbeiten im Bereich der binären Anomalieerkennung, wird die „Area Under the Receiver Operating Characteristic Curve“ (AUROC) als Leistungsmaß verwendet. Die AUROC ist ein Maß für die Fähigkeit eines binären Klassifikators, zwischen zwei Klassen zu unterscheiden. Nachfolgend wird schrittweise zum Begriff des AUROC hingeführt.

2.3.1 Logistische Regression

Die logistische Regression ist eine Art verallgemeinertes lineares Modell, das üblicherweise für binäre Klassifizierungsprobleme verwendet wird. Bei der logistischen Regression besteht das Ziel darin, die Wahrscheinlichkeit eines binären Ergebnisses (z. B. nominal oder anomal) auf der

Grundlage einer Reihe von Eingangsmerkmalen vorherzusagen. Das logistische Regressionsmodell verwendet eine logistische Funktion („Sigmoidfunktion“), um die Eingabemerkmale auf die vorhergesagte Wahrscheinlichkeit abzubilden.

Die logistische Funktion ist definiert als:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

wobei z eine lineare Kombination aus den Eingangsmerkmalen und ihren zugehörigen Gewichten ist:

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p$$

Dabei ist β_0 der Bias-Term und $\beta_1, \beta_2, \ldots, \beta_p$ sind die Koeffizienten für die Eingangsmerkmale $x_1, x_2, \ldots bzw. x_p$.

Das logistische Regressionsmodell wird trainiert, indem eine Verlustfunktion minimiert wird, die die Differenz zwischen den vorhergesagten Wahrscheinlichkeiten und den wahren binären Kennzeichnungen misst. Eine gängige Verlustfunktion für logistische Regression ist der binäre Kreuzentropie (‚Cross-Entropy“), die wie folgt definiert ist:

$$\mathcal{L}(\beta) = -\frac{1}{n} \sum_{i=1}^{n} y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)$$

wobei β die Modellparameter (d.h. den Achsenabschnitt und die Koeffizienten) darstellt, n die Anzahl der Trainingsbeispiele, y_i das wahre binäre Label für das i-te Beispiel und $\hat{y}_i$ die vorhergesagte Wahrscheinlichkeit für das i-te Beispiel ist.

Das logistische Regressionsmodell kann mithilfe des Gradientenabstiegs trainiert werden, bei dem die Modellparameter iterativ in Richtung des negativen Gradienten der Verlustfunktion aktualisiert werden. Der Gradient der Verlustfunktion in Bezug auf die Modellparameter kann mithilfe der Kettenregel der Infinitesimalrechnung berechnet werden. [3] (Kapitel 4) [11]

2.3.2 Konfusionsmatrix

Die Konfusionsmatrix ist eine Tabelle, die die Leistung eines binären Klassifikationsmodells zusammenfasst. Sie besteht aus vier Einträgen: wahr-positive (TP), falsch-positive (FP), wahr-negative (TN) und falsch-negative (FN). TP und TN stehen für die Anzahl der richtig klassifizierten positiven bzw. negativen Beispiele, während FP und FN für die Anzahl der falsch klassifizierten positiven bzw. negativen Beispiele stehen.

Hier stehen die Zeilen für die vorhergesagten Kennzeichnungen und die Spalten für die wahren Kennzeichnungen. Die Einträge in der Diagonale stehen für die richtigen Vorhersagen, während

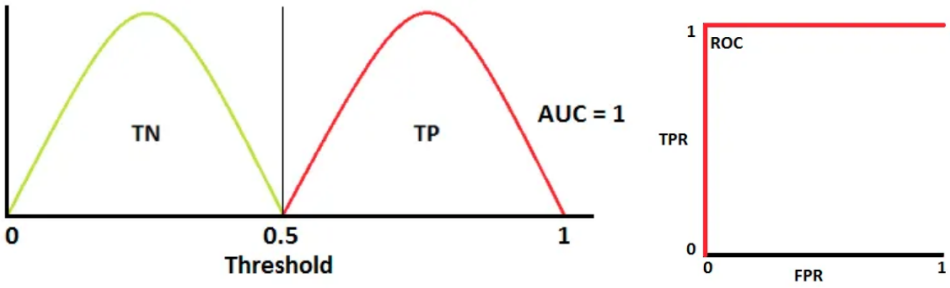
Tabelle 2.2: Konfusionsmatrix

	Tatsächlich Positive	**Tatsächlich Negative**
Prädizierte Positive	Wahre Positive (TP)	Falsche Positive (FP)
Prädizierte Negative	Falsche Negative (FN)	Wahre Negative (TN)

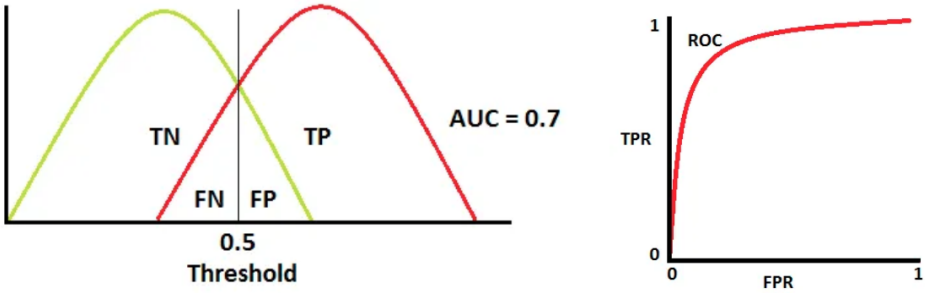
die Einträge außerhalb der Diagonale die falschen Vorhersagen darstellen. Die „True Positive Rate“(TPR), die auch als Sensitivität oder Recall bezeichnet wird, ist definiert als TP / (TP + FN), d. h. der Anteil der positiven Beispiele, die richtig klassifiziert wurden. Die „False Positive Rate“ (FPR) ist definiert als FP / (FP + TN), d. h. der Anteil der negativen Beispiele, die falsch klassifiziert werden. Die Konfusionsmatrix bietet eine Möglichkeit, die Leistung eines binären Klassifikationsmodells im Hinblick auf seine Fähigkeit, positive und negative Beispiele richtig zu klassifizieren, zu bewerten. Sie kann zur Berechnung verschiedener Leistungsmetriken verwendet werden, wie z. B. Genauigkeit, Präzision, Wiedererkennung, F1-Score und die Fläche unter der „Receiver Operating Characteristic“ (ROC)-Kurve (AUC), die im nächsten Abschnitt erläutert wird.

2.3.3 ROC-Kurve

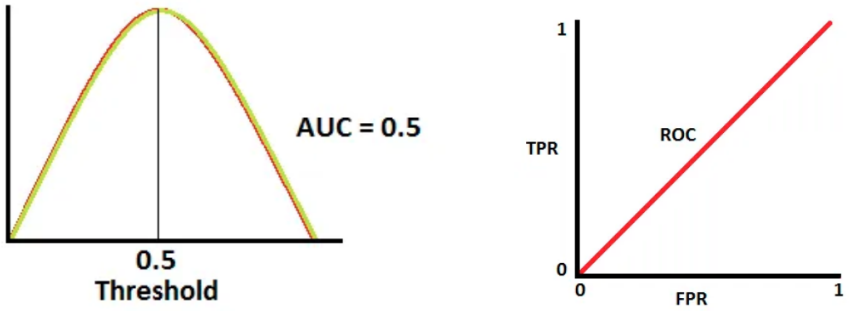
Die Receiver-Operating-Characteristic-Kurve (ROC-Kurve) ist eine grafische Darstellung der Leistung eines binären Klassifikationsmodells, wenn der Schwellenwert variiert wird. In der ROC-Kurve wird die Rate der richtig positiven Beispiele (TPR) gegen die Rate der falsch positiven Beispiele (FPR) für verschiedene Schwellenwerte aufgetragen. Die TPR ist der Anteil der positiven Beispiele, die richtig klassifiziert werden, während die FPR der Anteil der negativen Beispiele ist, die falsch klassifiziert werden.
Berechnet man die Fläche unter der ROC-Kurve erhält man das **„Area Under the Curve“ (AUC)** Maß. Es ist eines der aussagekräftigsten Maße, die es für eine binäre Klassifikation wie die Anomalieerkennung gibt. Die AUC ist ein Wert zwischen 0 und 1, wobei 1 für eine perfekte Klassifikation steht, $0{,}5$ für eine zufällige Klassifikation und 0 für eine gänzlich falsche Klassifikation steht. Nachfolgend sind mögliche Ausbabeverteilungen einer logistischen Regression, markiert mit der tatsächlichen Klassenzugehörigkeit, skizziert. Diese Veranschaulichen den Zusammenhang zwischen der Ausgabe einer logistischen Regression, der ROC-Kurve und dem AUROC-Maß.



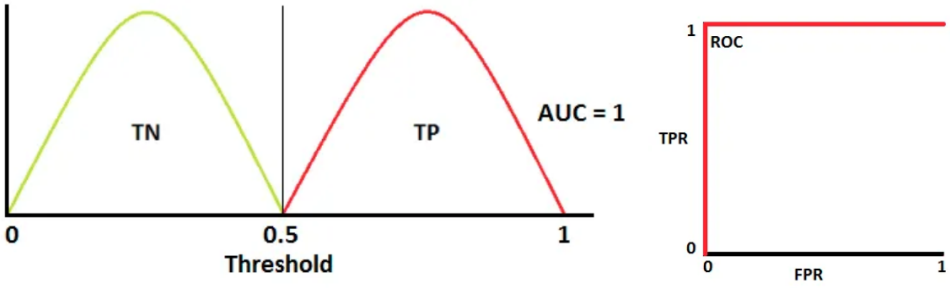
(a) $\mathbf{AUC} = 100\%$: Die beiden Verteilungen der Klassen sind vollständig getrennt. Eine perfekte Klassifikation ist möglich.



(b) $\mathbf{AUC} = 70\%$: Eine Überlappung der Verteilungen lässt mit keinem Schwellwert eine fehlerfreie Klassifikation zu. Ein Großteil der Beispiele kann aber richtig klassifiziert werden.



(c) $\mathbf{AUC} = 50\%$: Die Verteilungen der Klassen überlappen sich vollständig. Eine sinnvolle Klassifikation ist somit nicht möglich. Eine Zuordnung würde zufällig geschehen.



(d) $\mathbf{AUC} = 0\%$: Dieser Spezialfall lässt keine korrekte Klassifikation zu, obwohl die Klassen eindeutig getrennt sind. Dies liegt daran, dass die Verteilung der positiven Klasse vollständig links von der Verteilung der negativen Klasse liegt. Durch eine geeignete Abbildung kann dieses Problem gelöst werden.

Abbildung 2.2: Veranschaulichung verschiedener Verteilungen positiver und negativer Klassen und den dazugehörenden ROC-Kurven und AUC-Werten.

2.4 Residuale Netzwerke

In diesem Abschnitt werden die Grundlagen von Residual Networks (textbf„ResNets“) erläutert. Diese spielen für die Merkmalsextraktoren („Feature Extractors“) in den im Haupteil der Arbeit (TODO –> Link) eine wichtige Rolle. Der Schwerpunkt hierbei liegt auf der grundlegenden Idee, der Rolle, die ResNets historisch in der Entwicklung von Deep Learning gespielt haben und vor allem den Aspekten, die für die Anwendung in dieser Arbeit relevant sind. Für detailierte Informationen zu ResNets wird auf das Paper von Kaiming He et al. [9] und die zahlreichen Erläuterungen in der Literatur verwiesen.

2.4.1 Hintergrund & Idee hinter „ResNets“

Tiefe neuronale Netze (Deep Neural Networks, DNNs) eignen sich hervorragend für das Lernen hierarchischer Darstellungen aus Daten, aber sie stehen vor Herausforderungen, wenn sie „tiefer“ werden.„Tiefe“ bezeichnet in diesem Zusammenhang die Anzahl an Schichten, die sequentiell durchlaufen werden, um das Endergebnis bzw. die Ausgabe zu erhalten. Tiefere Netze können komplexere Merkmale in Daten erfassen, was für Aufgaben wie die Bildklassifizierung, bei der Objekte und Muster komplizierte Details aufweisen können, entscheidend ist. Einer der großen Herausforderungen bei tiefen Netzen ist das Problem der „verschwindenden Gradienten“ (*engl.* **Vanishing Gradients**). Diese Gradienten sind entscheidend für das Training eines Neuronalen Netzes, insofern, als dass das Optimierungsverfahren des Gradientenabstiegs die Grundlage auch moderner Optimierer wie „Adam“ (TODO –> Ref) ist. Dieses Problem lässt sich anschaulich dadurch erklären, dass frühe Gradienten, was sich leicht mithilfe der Kettenregel zeigen lässt, eine Multiplikation von allen nachfolgenden (im Sinne der Infernenzrichtung - „forward pass“) Gradienten darstellt. Betrachtet man nun ein sehr tiefes Netz, das heißt, viele Gradienten, die miteinander multipliziert werden und den wahrscheinliche Fall von Gradienten, die kleiner als 1 sind, so wird schnell klar, dass frühe Schichten einen sehr kleinen Gradienten haben können. Dies macht es schwierig, die Gewichte der frühen Schichten zu aktualisieren, was den Lernprozess behindert. Dabei ist es vor allem die Tiefe, die Neuronalen Netzen das Generalisieren von komplexen Zusammenhängen ermöglicht („Deep Learning “)
Residuale Netze, allgemein bekannt als ResNets und im Folgenden auch so bezeichnet, wurden von Kaiming He et al. (TODO –> Ref) in ihrem Paper von 2015 vorgestellt und bieten eine einfache und dennoch effektive Methode an, wie dieses Problem angegangen werden kann. Die Grundidee besteht darin, Verknüpfungen zwischen den Schichten einzuführen, die es dem Netz ermöglichen, eine oder mehrere Schichten zu „überspringen“. Anstatt die gewünschte Abbildung also direkt zu lernen, lernen ResNets die residuale Abbildung, was der Differenz zwischen Ein- und Ausgabe entspricht. Die Eingabe wird über sogenannte „Shortcut (Skip) Connections“, also einfach die Identitätsabbildung, vom Eingang zur residualen Ausgabe weitergeleitet, um dann durch Summation wieder miteinander verknüpft zu werden. Das Problem der verschwindenden Gradienten wird dadurch entschärft, dass die Gradienten direkt in frühere Schichten zurückfließen können. Anschaulich lässt sich das durch die Tatsache erklären, dass die Identitätsabbildung

einen konstanten Gradienten von 1 besitzt. Weil sich die Summenbildung am Ausgang eines nachfolgend noch im Detail besprochenen Residual-Blocks auch bei der Gradientenbildung als Summation widerspiegelt, ist der Gradient einer jeden Schicht nicht mehr das Produkt, sondern vielmehr die Summe aller nachfolgenden Gradienten. (Optional TODO: Formel) Das zugrundeliegende Paper ist eines der meistzitierten Paper im Bereich des Deep Learning und hat die Entwicklung von Deep Learning maßgeblich beeinflusst.

2.4.2 Residual Block & Architektur

Der Grundbaustein eines ResNet ist der Residualblock. Er besteht aus zwei Hauptpfaden: dem Identitätspfad (der Abkürzungsverbindung) und dem Residualpfad (dem Hauptfaltungspfad). Mathematisch wird die Ausgabe eines Residualblocks wie folgt berechnet:

$$\text{Output} = F(\text{Input}) + \text{Input}$$

wo F die Residualabbildung ist, die durch die Hauptfaltungsschichten gelernt wird. Nachfolgende Abbildung zeigt eine vereinfachte Darstellung eines „Building Blocks“ bzw. Residual Block.

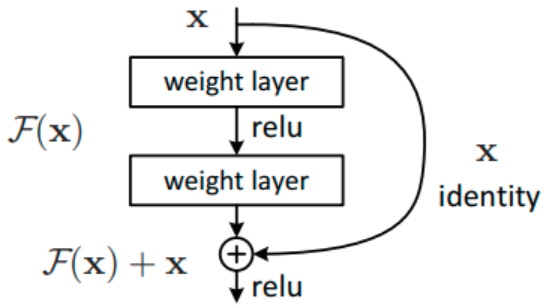


Abbildung 2.3: Residual Block (TODO –> Ref)

Ein ResNet besteht aus mehreren Residualblöcken, die sequentiell durchlaufen werden. Es ist also eine „Feed-Forward“-Architektur, weil die Daten zur Inferenz nur in eine Richtung durch das Netz fließen. Durch das „Stapeln “ können dann sehr tiefe Netze erstellt werden, die sich dennoch aufgrund der oben beschriebenen Eigenschaften gut optimieren bzw. trainieren lassen. Es existieren zahlreiche verschiedene Varianten von ResNets, die sich in der Art und Anzahl ihrer Residual Blöcke unterscheiden. Nachfolgende Tabelle gibt einen Überblick über die drei, in dieser Arbeit vor allem verwendeten Varianten: ResNet18, ResNet34 und WideResNet50

Zu erkennen ist eindeutig, dass die Anzahl der Parameter und die Inferenzzeit mit der Tiefe des Netzes steigt. Gleichzeitig lässt sich aber auch eine Verbesserung der Top-1 Fehlerrate erkennen, je tiefer bzw. mächtiger das Netz ist.

[1] in ILSVRC (TODO –> Ref)

[2] Raspberry Pi 4B 8GB. Betrachtet wurde die Laufzeit von einem Bild der Auflösung 224x224 und 3 (Farb-)Kanälen

Tabelle 2.3: Vergleich verschiedener ResNet Varianten (TODO –> Ref)

Netzwerk	Tiefe	# Parameter	Top-1 Fehlerrate[1]	Inferenz auf RBP4[2]
ResNet18	18	$11{,}7 \cdot 10^6$	30,24%	0, 82s
ResNet34	34	$21{,}8 \cdot 10^6$	26,70%	1, 45s
WideResNet50	50	$68{,}9 \cdot 10^6$	22,53%	3, 00s

Einer der Hauptvorteile von ResNets ist die Fähigkeit, Merkmale pyramidenförmig durch das Netz zu verbreiten. Das bedeutet, dass das Netz Merkmale auf verschiedenen Abstraktionsebenen erfassen kann, von Low-Level-Merkmalen wie Kanten und Ecken bis zu High-Level-Merkmalen wie Objektteilen und semantischen Konzepten. Mit zunehmender Tiefe wird also die Auflösung der „Feature Maps“ (TODO –> Ref) reduziert, während die Anzahl der Kanäle und die Komplexität der zugrundeliegenden Merkmale zunimmt. Dargestellt ist das in folgender Abbildung:

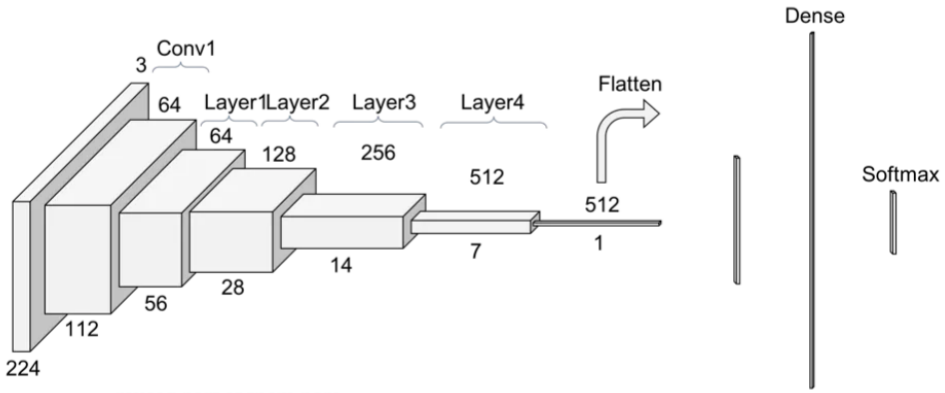


Abbildung 2.4: Pyramidale Merkmalsverteilung in ResNets (TODO –> Ref)

Die Quader in oben stehendem Netz stehen symbolhaft für die Auflösung der Feature Maps. Während die oben stehenden Zahlen die Anzahl der Kanäle repräsentieren, stehen die unten aufgeführten Zahlen für die räumliche Auflösung. Letztere hängt proportional von der Auflösung des Eingangsbildes ab und gilt für alle drei Modelle. Die Anzahl an Kanälen ist konstant für alle Auflösungen und für ResNet18 und ResNet34. Für das WideResNet50 gilt im Wensentlichen der gleiche Aufbau, allerdings sind die Kanäle „geweit“ gegenüber den anderen beiden Architekturen, was sich an der Vervierfachung der Anzahl an Kanäle durch Verwendung eines komplexeren Residual Blocks („Bottleneck“[18])zeigt. ResNet18 und ResNet34 unterscheiden sich ausschließlich in der Anzahl der Residual Blöcke bzw. der Tiefe des Netzes.
Alle der drei hier vorgestellten Architekturen lassen sich in fünf Faltungsschichten („Conv1“, „Layer1“, „Layer2“, „Layer3“, „Layer4“) unterteilen. Dem schließt sich eine „Average Pooling“-Schicht an, die die Auflösung der Feature Maps auf 1 reduizert („Flatten“). Dieser 1D-Vektor wird schließlich durch eine „Fully Connected“-Schicht („Dense“) auf die Anzahl der Klassen (1000) abgebildet. Schließlich wird durch eine „Softmax“-Aktivierungsfunktion Pseodo-

Wahrscheinlichkeiten erzeugt, die den Axiomen von Kolmogorov entsprechen und somit als Auftrittswahrscheinlichkeiten interpretiert werden können.
In dieser Arbeit werden die ResNets als Feature-Extraktoren verwendet, weshalb die letzten beiden Schichten, also „Flatten“ und „Dense“ nicht verwendet werden. Die Begriffe „Layer1 - Layer4“ werden im Folgenden in dem hier beschriebenen Kontext verwendet.

2.4.3 ResNets as Feature Extractor für Unüberwachte Lernverfahren

Zusätzlich zu ihrem Erfolg bei der überwachten Bildklassifizierung haben ResNets Anwendungen als leistungsstarke Merkmalsextraktoren bei Unüberwachten Klassifizierungsaufgaben wie der Anomalieerkennung gefunden. In unüberwachten Szenarien wie der Anomalieerkennung stehen oft keine markierten (gelabelten) Daten zur Verfügung, wie bereits in 1.2 beschrieben, um ein Modell explizit bzw. überwacht zu trainieren. Stattdessen verlässt man sich auf das Lernen von Darstellungen normaler Daten und identifiziert dann Abweichungen als Anomalien. ResNets können mit ihrer Fähigkeit, umfangreiche und hierarchische Merkmale zu erfassen, dazu verwendet werden, sinnvolle Merkmale aus den Daten zu extrahieren. Es kann mithilfe dieser Netze eine kompaktere und aussagekräftigere Darstellung der Daten erzeugt werden, die die Grundlage sind, um auch komplexere Anomalien zu erkennen. Allerdings ist festzuhalten, dass die extrahierten Feature in keiner Weise optimiert sind, um Anomalieklassifikationen zu ermöglichen, sondern um auf dem ImageNet-Datensatz Bilder richtig einzuordnen. Vor allem in späteren Schichten ist also zu erwarten, dass die Merkmale einene „Bias “ hin zu ImageNet aufweisen, der sich negativ auf die Anomalieklassifikation auswirken kann.[14] Hierzu werden die Gewichte offizieller Implementierungen von ResNets verwendet, womit das eigentliche Vortraining entfällt und Reproduzierbarkeit, Konsistenz und Vergleichbarkeit der Ergebnisse gewährleistet wird.
Alle hier vorgestellten Methoden verwenden ResNets als Feature-Extraktoren, wenn auch im Ansatz„EffientAD“ (4) nicht während der Inferenz. Gezeigt, dass eine solche Feature Extraktion im Kontext der Unüberwachten Anomalieerkennung sinnvoll ist, wurde erstmals in der Veröffentlichung„Deep Nearest Neighbor Anomly Detection“ von Bergman et al.[1] mit der Methode „DN2“ im Februar 2020. Im nachfolgenden Abschnitt wird die darauf aufbauende Methode „SPADE“, vom gleichen Team entwickelt, vorgestellt.

2.5 SPADE

Die Methode **SPADE** ist ein wichtiger Meilenstein in der Unüberwachten Anomalieerkennung. Zahlreiche erfolgreiche Methoden bauen auf SPADE auf und übernehmen wichtige Elemente und Konzepte. Das Paper wurde am 5. Mai 2020 von Niv Cohen und Yedid Hoshen von der Hebräischen Universität von Jerusalem veröffentlicht. Im Folgenden wird SPADE vorgestellt.

2.5.1 Funktionsweise

Feature Extraktion

Wie bereits in 2.4.3 beschrieben, werden ResNets erfolgreich als Feature-Extraktoren verwendet. Der Name „SPADE“ steht hierbei für „Semantic Pyramid Anomaly Detection“. Betrachtet man 2.4 wird klar warum von einer „Pyramide“ gesprochen werden kann: Es werden die Feature Maps der einzelnen Schichten, konkret der Schichten „Layer1, Layer2 und Layer3“ als Feature unverändert verwendet. Dabei wird auf einen Einbettungsprozess, wie bei anderen Methoden üblich, verzichtet. Durch die mit der Tiefe des Netzes abnehmende Auflösung der Feature Maps, entsteht so eine „Pyramidale“ Struktur, die eben die Grundlage für die Namensgebung ist. Zusätzlich wird der 1D-Vektor, der durch die „Flatten“-Schicht erzeugt wird, als Feature verwendet, um instanzweise, also für jedes Bild als Ganzes, zu entscheiden, ob eine Anomalie vorliegt, wie im nächsten Abschnitt beschrieben. Bezeichnen wir das ResNet als Feature-Extraktor mit F, so lassen sich die Feature f_i eines gegebenen Bildes x_i wie folgt beschreiben:

$$f_i = F(x_i)$$

In der Trainings- bzw. Initialisierungsphase werden die Feature f_i aller Trainingsbilder x_i, welche alle nominal sind, extrahiert und gespeichert. Diese Menge wird im Folgenden mit F_{train} bezeichnet.

Instanzklassifizierung

Die Instanzklassifizierung ist der zweite Schritt von SPADE. Es wird dabei für jedes Bild y_i aus der Menge an Testbilder Y entschieden, ob es sich um eine Anomalie handelt oder nicht. Dies geschieht ausschließlich anhand des 1D-Vektors, der durch die „Flatten“-Schicht erzeugt wird. Hierzu wird für ein gegebenes Testbild y_i, die K nächsten Nachbarn in F_{train} gesucht. Diese Menge an K Vektoren wird fortan als $N_k(f_y)$ bezeichent. Es wird eine Distanz auf die folgende Art und Weise bestimmt:

$$d(y_i) = \frac{1}{K} \sum_{f \in N_k(f_{y_i})} \left|\left| f - f_{y_i} \right|\right|^2$$

Die Distanz $d(y_i)$ wird dann mit einem Schwellenwert τ verglichen. Ist die Distanz kleiner als der Schwellenwert, so wird das Bild als nominal klassifiziert, ansonsten als Anomalie.

Segmentierung

Nachdem eine Instanzklassifiziergung ergeben hat, dass es sich um ein anomales Bild handelt, besteht die nächste Aufgabe darin, die Anomalie oder mehrere Anomalien auf dem Bild zu lokalisieren. Dieser Schritt wird übersprungen, wenn das Bild als nominal klassifiziert wurde.
Die naive Methode, die Anomalie zu lokalisieren, wäre es, die Differenz zwischen dem anomalen

Bild und dem ausgerichtete Bild des nächsten Nachbarn zu berechnen. Dort, wo die Differenz am größten ist, wird die Anomalie vermutet. Allerdings hat diese Methode einige Schwachstellen. Zum einen kann die Ausrichtung des nächsten Nachbarn so, dass das zu untersuchende Objekt im Bild an der gleichen Stelle ist, fehlschlagen. Insbesondere. Auch kann insbesondere bei eienem kleinen Datensatz und Objekten, die komplexe Variationen aufweisen möglicherweise kein passender Nachbar gefunden werden, was zu falsch positiven (anomalen) Klassifizierungen führen kann. Außerdem könnte die Berechnung der Differenz sehr empfindlich auf die verwendete Berechnungsmethode sein.(TODO... Vllt einfach weg lassen?)
Um diese Probleme zu überwinden, wird eine Übereinstimmungsmethode präsentiert, die mit vielen „Bildern“ arbeitet („multi-image correspondence method“). Diese Bilder entsprechen den Feature Maps der Schichten Layer1, Layer2 und Layer3, also gewissermaßen den Quader in 2.4. Nun gilt es zu jeder Pixelposition $p \in P$ die korresponierenden Feature $F(x_i, p|p \in P)$ zu bestimmen. Auf jedes Bild aus dem Trainingsdatensatz x_i angewandt und ergibt sich dann die „Gallerie“ zu $G = \{F(x_1, p|p \in P) \cup F(x_2, p|p \in P).. \cup F(x_K, p|p \in P)\}$.
Der Anomaliegrad eines Pixels p ist dann gegeben durch die durchschnittliche Distanz zwischen dem Feature $F(y_i, p)$ des anomalen Bildes y_i und den κ nächsten Featuren aus der Gallerie G. Es ergibt sich also für einen Pixel p im Testbild y_i folgende Formel:

$$d(y_i, p) = \frac{1}{\kappa} \sum_{f \in N_\kappa(F(y_i,p))} ||f - F(y_i, p)||^2$$

Für einen gegebenen Schwellwert θ wird ein Pixel p als anomales Pixel klassifiziert, wenn $d(y_i, p) > \theta$ gilt. Dies ist dann der Fall, wenn wir kein nominales Feature in G finden können, welches dem Feature $F(y_i, p)$ ausreichend ähnlich ist.
Dadurch, dass die Gallerie G die Positionsinformation p marginalisiert, wird die Segmentierung robust gegenüber der Ausrichtung. Anschaulich gesprochen, wird also in jedem Bild an allen Positionen nach ähnlichen Features gesucht.

2.5.2 Ergebnisse und Diskussion

Wie bereits erwähnt, markiert diese Arbeit einen wichtigen Entwicklungsschritt in der Forschung zur Unüberwachten Anomalieerkennung. Folgende Aspekte sollten in diesem Zusammenhang hervorgehoben werden:

- SPADE ist die erste ganzheitliche Methode, die auf ImageNet vortrainierte ResNets als Feature-Extraktoren verwendet. Dieser Ansatz hat sich als sehr erfolgreich erwiesen und wird von zahlreichen Methoden übernommen.
- Das Zusammenfassen aller extrahierten Features aus den Testbildern zu einer Menge G löst das Problem der Ausrichtung und ermöglicht eine robuste Segmentierung. Auch diese Idee wird uns bei PatchCore (3) wiederbegegnen.
- Das Bestimmen des Anomaliegrades mithilfe einer kNN-Suche ist ein einfacher und effektiver Ansatz, der sich auch in modifizierter Form in vielen anderen Methoden wiederfindet.

Insbesondere in der pixelweisen Klassifikation von Anomalien hat sich SPADE als geeignet erwiesen. Auch in der Instanzklassifiziergung erreicht diese Methode auf dem damals noch jungen Datensatz MVTecAD gute, zum Zeitpunkt der Veröffentlichung der Methode, sogar das beste Ergebnisse.
Für das Ziel dieser Arbeit, welche den Fokus auf die Instanzklassifiziergung und die Laufzeitoptimierung legt, ist SPADE allerdings eher ungeeignet. Zum Einen sind 85,5% erreichte Bildklassifizierungsgenauigkeit auf MVTecAD für die Instanzklassifizierung nicht mit neueren, zum Teil aber auf SPADE aufbauenden Methoden, konkurrenzfähig. Zum Anderen ist die notwendige kNN Suche, die für jedes Testbild durchgeführt werden muss, sehr rechenintensiv. Dieser Rechenaufwand skaliert linear mit der Anzahl an Bildern im Trainingsdatensatz und mit der Anzahl der Pixel pro Bild. Kann man auf eine Segmentierung verzichten, verkleinert sich der Rechenaufwand zwar deutlich, es muss aber dennoch eine kNN-Suche über alle Trainingsbilder für jedes Testbild durchgeführt werden, wodurch die Laufzeit wiederum von der Größe des Trainingsdatensatz abhängt, auch wenn die Auflösung der Bilder keinen Einfluss mehr hat. Den Anstoß und die Impulse, die durch diese Veröffentlichung gesetzt wurden, sind jedoch nicht zu unterschätzen.

2.6 PaDiM

Die Methode **PaDiM** wurde am 17. November 2020 von Defard et al. (Universität Paris-Saclay) veröffentlicht. Es werden einige Aspekte von SPADE übernommen, vor allem aber Schwächen der Methode erfolgreich addressiert. Einer der wesentlichen Unterschiede zwischen PaDiM und SPADE ist, die Weise auf die der Anomaliescore bestimmt wird. PaDiM markierte damit die beste Instanzklassifizierungsgenauigkeit auf dem Datensatz MVTecAD zum Zeitpunkt der Veröffentlichung und hielt dies bis zum Erscheinen der Methode „PatchCore“. Nachfolgend wird der Ansatz von PaDiM vorgestellt und diskutiert.

2.6.1 Funktionsweise

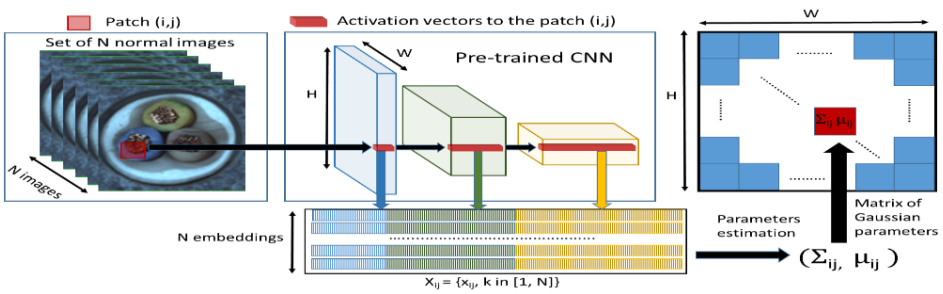


Abbildung 2.5: PaDiM: Funktionsweise (TODO –> Ref)

PaDiM kann in drei Schritte unterteilt werden:

1. Feature Extraktion und Einbettungsprozess mithilfe eines vortrainierten ResNets
2. Bestimmen der Gaußverteilungen durch Schätzen der Parameter μ und Σ für jede Position im Bild
3. Bestimmen des Anomaliegrades durch die Mahalanobis-Distanz

2.5 veranschaulicht den Prozess exklusive der Inferenz.

Feature Extraktion und Einbettungsprozess

Der Prozess des Erzeugens von Featuren ist bei PaDiM sehr ähnlich zu dem von SPADE: Auch hier werden Feature Maps unterschiedlicher Auflösung und Abstraktionsebene zu einem Featurevektor zusammengefasst, der mit einer Position bzw. einem Pixel des Eingangsbildes korresponiert. Die Feature Map, welche die höchste Auflösung besitzt, also die Feature Map der ersten zur Feature Extraktion ausgewählten Schicht, definiert die Auflösung der Anomaliesegmentierung. Nehmen wir an diese Feature Map habe eine Auflösung von $W \times H$, so korresponiert also zu jeder Position $(i, j) \in [1, W] \times [1, H]$ ein Featurevektor x_{ij} („Patch Embedding Vektor“).

Bestimmen der Gaußverteilungen

PaDiM modelliert für jede Position eine multivariate Normalverteilung, die durch die Featurevektoren aller Trainingsbilder an dieser Position definiert wird.
Zunächst wird hierzu die Menge an Patch Embedding Vektoren für eine Position (i, j) aus dem gesamten Trainingsdatensatz gebildet. Diese Menge $X_{ij} = \{x_{ij}^k | k \in [1, N]\}$ aus den N nominalen Bildern im Trainingsdatensatz wird dann benutzt, um die Normalverteilung für die Position (i, j) zu bestimmen. Es wird also angenommen, X_{ij} würde von der multivariaten Gaußverteilungen $N(\mu_{ij}, \Sigma_{ij})$ erzeugt werden. Die Parameter μ_{ij} und Σ_{ij} werden dann wie folgt geschätzt:

$$\mu_{ij} = \frac{1}{N} \sum_{k=1}^{N} x_{ij}^k$$

$$\Sigma_{ij} = \frac{1}{N-1} \sum_{k=1}^{N} (x_{ij}^k - \mu_{ij})(x_{ij}^k - \mu_{ij})^T + \epsilon I$$

wobei der Regularisierungsterm ϵI hinzugefügt wird, um die Invertierbarkeit bzw. den vollen Rang der Kovarianzmatrix Σ_{ij} zu gewährleisten. Schließlich wird die multivariate Gaußverteilung $N(\mu_{ij}, \Sigma_{ij})$ für jede Position (i, j) im Bild definiert.

Bestimmen des Anomaliegrades

Um den Anomaliegrad zu bestimmen, wird die Mahalanobis-Distanz herangezogen.Die Mahalanobis-Distanz ist eine Verallgemeinerung der euklidischen Distanz, die die Korrelation zwischen den Dimensionen der Daten berücksichtigt. In diesem Fall lässt sich die Mahalanobis-Distanz für die Position (i, j) und einem aus einem Testbild extrahierten Patch Embedding Vektor y_{ij} wie folgt berechnen:

$$M(y_{ij}) = \sqrt{(y_{ij} - \mu_{ij})^T \Sigma_{ij}^{-1} (y_{ij} - \mu_{ij})}$$

Damit kann dann eine Anomaliekarte M für ein Testbild y erzeugt werden:

$$M = (M(y_{ij}))_{1<=i<=W,1<=j<=H}$$

Hierbei deuten hohe Werte auf eine Anomalie hin, während niedringe Werte auf einen nominalen Bildbereich hinweisen. Durch Maximalwerbildung über die Anomaliekarte M kann dann ein Anomaliegrad s für ein Testbild y bestimmt werden:

$$s(y) = \max_{1<=i<=W,1<=j<=H} M(y_{ij})$$

2.6.2 Ergebnisse und Diskussion

Wie bereits zu Beginn dieses Kapitels erwähnt, erreicht PaDiM zum Zeitpunkt der Veröffentlichung die beste Instanzklassifizierungsgenauigkeit auf dem Datensatz MVTecAD mit 97,5%. Auch die Segmentierungsergebnisse sind mit 97,9% zum Veröffentlichungszeitpunkt State-of-the-Art.
Ein spannender Aspekt, der in diesem Veröffentlichung untersucht wird, ist das Reduzieren der Anzahl an Kanälen bzw. die Reduzierung der Dimensionalität der Featurevektoren:
Es wird gezeigt, dass im Falle eines ResNet18 als Backbone die Dimensionalität von ursprünglich 448 auf 200 oder sogar 100 reduziert werden kann, ohne dass die Klassifizierungsgenauigkeit signifikant sinkt. Dabei wurden die zu entfernenden Dimensionen zufällig ausgewählt und die Ergenisse über mehrere Durchläufe gemittelt. Im Falle aller 448 Dimensionen wird eine Genauigkeit von 97,1% erreicht, während bei 200 Dimensionen eine Genauigkeit von 97,0% und bei 100 Dimensionen eine Genauigkeit von 96,7% erreicht wird.
Dies ist vor allem deshalb eine sehr relevante Erkenntnis, da die Reduzierung der Dimensionalität einen ganz erhebliche Einfluss auf die Berechnungsgeschwindigkeit der Mahalanobis-Distanz hat. Die Komplexität in der Landau-Notation für die Berechnung der Mahalanobis-Distanz zwischen zwei Vektoren der Länge d ist $\mathcal{O}(d^3)$[3], wobei d die Dimensionalität der Featurevektoren ist, in diesem Beispiel also 448, 200 bzw. 100. Dass der Einbruch der Genauigkeit bei einer Reduzierung der Dimensionalität so gering ist, ist auf die Mahalanobis-Distanz zurückzuführen, die die Korrelation zwischen den Dimensionen der Daten berücksichtigt. Weil die einzelnen Dimensionen teilweise stark korreliert sind, kann die Dimensionalität reduziert werden, ohne dass die Genauigkeit signifikant sinkt. Dies ist ein Vorteil gegenüber der euklidischen Distanz,

die die Dimensionen unabhängig voneinander betrachtet, aber auch deutlich weniger komplex und damit laufzeitkritisch ist, als die Mahalanobis-Distanz.
Ein Verbesserung gegenüber SPADE ist, dass die Instanzklassifiziergung auf Grundlage der Anomaliekarte M erfolgt, die durch die Maximalwerbildung über alle Positionen im Bild erzeugt wird. Diese Anomaliekarte wird mithilfe der Feature aus den ersten drei von vier Schichten des ResNets erzeugt, die, wie in 2.4.3 beschrieben, nur einen eher geringen „Bias“ hin zu ImageNet aufweisen. SPADE wiederum nutzt den 1D-Vektor, der durch die „Flatten“-Schicht erzeugt wird, um die Instanzklassifizierung durchzuführen, was zwei Nachteile mit sich bringt: Es ist ein Bias zu erwarten und durch die durch Mittelwertbildung erreichte Dimensionsreduktion können wichtige, zu einer lokalen Anomalie gehörende Detailinformationen verloren gehen. Bei PaDiM können selbst lokale Anomalien, die nur wenige Pixel groß sind, erkannt werden, was eine enorme Verbesserung darstellt und in allen im Haupteil dieser Arbeit vorgestellten Methoden übernommen wird.

2.7 Raspberry Pi 4B

2.7.1 Allgemeines

Der Raspberry Pi 4, der im Juni 2019 von der Raspberry Pi Foundation veröffentlicht wurde, ist ein kleiner, erschwinglicher und vielseitiger Einplatinencomputer.

Die zentrale Recheneinheit (CPU) des Raspberry Pi 4 ist eine 64-bit Quad-Core-ARM-Cortex-A72-CPU, die mit 1,8 GHz (ältere Versionen mit 1,5 GHz) taktet. Er ist in drei Speicherkonfigurationen erhältlich: 1 GB, 2 GB, 4 GB und 8 GB LPDDR4 RAM mit 3200 MHz. Die Integration eines Broadcom VideoCore VI-Grafikprozessors verbessert die Multimedia-Fähigkeiten und ermöglicht eine flüssige Videowiedergabe und 3D-Grafik-Rendering.

Ein bemerkenswertes Merkmal des Raspberry Pi 4 sind die Anschlussmöglichkeiten. Er verfügt über zwei USB 3.0-Ports und zwei USB 2.0-Ports, die den Anschluss verschiedener Peripheriegeräte ermöglichen. Dualband-Wi-Fi (2,4GHz und 5GHz) und Gigabit-Ethernet sorgen für eine zuverlässige Netzwerkverbindung. HDMI- und Audioausgänge unterstützen hochauflösende Bildschirme und Audiogeräte. So können beispielsweise zwei 4K-Displays angeschlossen werden.

Das Gerät ist mit mehreren Betriebssystemen kompatibel, darunter Raspberry Pi OS (früher Raspbian), Linux-Distributionen und sogar Windows 10, je nach Vorlieben und Anforderungen des Benutzers. In dieser Arbeit wurde Raspberry Pi OS in der 64-bit Version verwendet. Das auf Debian basierende Betriebssystem ist für die Hardware des Raspberry Pi optimiert und ist kompatibel mit allen notwendigen Softwarepaketen.

Die 40 GPIO-Pins des Raspberry Pi 4 sind eine vielseitige Hardwareschnittstelle, die zahlreiche Anwendungen in verschiedenen Bereichen ermöglicht. Die Anwendungen des Raspberry Pi

4 reichen von Bildungs- und Hobbyprojekten bis hin zu professionellen Unternehmungen. Er kann für Aufgaben wie Heimautomatisierung, Robotik, Webserver und Softwareentwicklung verwendet werden. Dank seines geringen Stromverbrauchs von etwa 2,7 W (Idle) bis maximal 6,4 W (Volllast, keine Peripherie)[7] eignet er sich für eingebettete Systeme und Internet-of-Things-Anwendungen (IoT).[6][16]

2.7.2 Ressourcenbeschränktheit

Die Rechenkapazität des Raspberry Pi 4 ist für viele Anwendungen ausreichend. Dennoch muss die Leistungsfähigkeit realisitsch eingeordnet werden: Während der Paspberry Pi 4 eine Rechenleistung von 13,5 GLFOPS (Gleitkommaoperationen pro Sekunde) erreicht, ist ein auf der gleichen Architektur (ARM) beruhender Apple M1 Prozessor aus dem Jahr 2020, der für einfache Consumer Tätigkeiten konzipiert ist, mit 154 GFLOPS mehr als 11 mal so schnell. [12] Auch muss erwähnt werden, dass auf die Verwendung von modernen GPUs für die Inferenz in dieser Arbeit verzichtet wird, wie in Kapitel 1.3 bereits beschrieben. Setzt man die Leistung des Raspberry Pi 4 in Relation zu modernen GPUs, die viele Entwicklungen im Bereich *Deep Learning* überhaupt erst ermöglichten, so wird schnell klar, warum bei der Verwendung eines Raspberry Pi 4 von „Ressourcenbeschränktheit“ gesprochen werden kann. Auch wenn Rechenkapazität in FLOPS gemessen keine eindeutigen Schlüsse auf die Laufzeit eines konkreten Algorithmus zulässt, so kann trotzdem festgehalten werden, dass eine moderne GPU eine enorm höhere Rechenleistung besitzt. So erreichen moderne GPUs, wie Nvidia's Geforce RTX4090 Ti 82 600 GFLOPS.[13]

Um die Ressourcenbeschränktheit weiter zu verdeutlichen, sind in 2.4 die Laufzeiten von in 2.4 vorgestellten Modellen auf dem Raspberry Pi 4 B mit der Laufzeit auf einem AMD Ryzen R5 5600X und einer Nvidia GeForce RTX3060Ti verglichen. Die Laufzeiten wurden für ein Bild der Auflösung 224x224 und 3 (Farb-)Kanälen gemessen. Es handelt sich hierbei um Hardware der gehobenen Mittelklasse aus dem Jahr 2020, die auch für dieser Arbeit verwendet wurde, womit die Ergebnisse leicht selbst erzeugt werden konnten.
Es ist deutlich zu erkennen, dass die Laufzeit auf dem Raspberry Pi 4 B im Vergleich zu den anderen Plattformen um mehrere Größenordnungen höher ist. Die Laufzeit auf dem Raspberry Pi 4 B ist im Mittel und Vergleich zu der Laufzeit auf dem AMD Ryzen R5 5600X um den Faktor 61 und im Vergleich zur Nvidia GeForce RTX3060Ti um den Faktor 602 höher, wie in letzter Zeile der Tabelle 2.4 zu sehen. Anzufügen ist, dass dies nicht auf alle Szenarien zutrifft. Bei dem hier angebrachten Beispiel, der Laufzeit eines ResNets, ist eine GPU aufgrund der Parallelisierbarkeit von Faltungsschichten und der damit verbundenen hohen Anzahl an FLOPS deutlich im Vorteil. Wie im weiteren Verlauf dieser Arbeit zu sehen, ist dies aber ein durchaus representatives Beispiel, da die alle Modelle in dieser Arbeit zumindest teilweise auf eben jene Faltungsschichten (CNNs) zurückgreifen.

[3] Raspberry Pi 4B

Tabelle 2.4: Laufzeiten der Modelle auf verschiedenen Plattformen

Netzwerk	**Raspberry Pi 4B**	**AMD Ryzen R5 5600X**	**Nvidia GeForce RTX3060Ti**
ResNet18	820,0 ms	11,68 ms	1,46 ms
ResNet34	1450,0 ms	20,00 ms	2,58 ms
WideResNet50	3000,0 ms	74,83 ms	4,40 ms
Faktor zu RPB4[3]	1 (Referenz)	61	602

Kapitel 3

PatchCore [14]

3.1 Einleitung

Die Methode **PatchCore** wurde erstmals am 15. Juni 2021 in Zusammenarbeit der Universität Tübingen und Amazon AWS im Paper „Towards Total Recall in Industrial Anomaly Detection“ veröffentlicht. In seiner zweiten Fassung vom 5. Mai 2022 wurde das Paper bei der Konferenz CVPR 2022 (Computer Vision and Pattern Recognition) akzeptiert und mit über 290 Zitierungen eines der populärsten Paper im Bereich der Unüberwachten Anomaliedetektion.
Die Grundlage dieses Ansatzes sind wiederum „Einbettungen“ (Embeddings) von Merkmalen (Features), die aus den Eingabebildern mithilfe eines auf „ImageNet“ vortrainiertem „Convolutional Neural Network (CNN)“ erzeugt werden. Damit ähnelt sich die Methode PatchCore sowohl SPADE2.5, also auch PaDiM2.6 und greift die in 2.4.3 beschriebene Vorgehensweise auf. Wie wir später sehen werden, unterscheidet sich der Einbettungsprozess jedoch recht deutlich von den bisherigen Methoden. Weiter wird die eigentliche Anomaliedetektion, wie bereits bei der Methode SPADE mithilfe einer Nächsten Nachbar Suche (Nearest Neighbor Search; NN) in einer „Memory-Bank “ durchgeführt. Die wesentliche Weitereentwicklung gegenüber SPADE liegt vor allem in der Methode, wie die Memory-Bank aufgebaut wird. Durch die Auswahl möglichst representativer Elemente in der Memory Bank, kann die Anzahl der Elemente in der deutlich reduziert werden, was einer Reduzierung der Laufzeit bedeutet.
Auch gut 2 Jahre nach Veröffentlichung ist die PatchCore Methode insbesondere auf dem MVTecAD-Datensatz2.1 mit einer Genauigkeit (Auccuracy) von maximal 99,6% („PatchCore Ensemble“) absolut konkurrenzfähig und wird in vielen Veröffentlichungen als „State-of-the-Art“ Methode verwendet.
Im Laufe dieses Kapitels soll zunächst die Funktionsweise der Methode PatchCore erläutert werden. Anschließend evaluieren wir die Originalmethode im Hinblick auf Laufzeit und Genauigkeit. Im sich dann anschließenden Teil werden zahlreiche Modifikationen besprochen, die versuchen, die Laufzeit auf zu Reduzieren und dabei möglichst viel der Genauigkeit zu erhalten.

3.2 Funktionsweise

unächst kann zwischen zwei Phasen unterschieden werden: Der Trainingsphase und der Testphase. In der Trainingsphase werden die „(locally aware) **Patch Features**“ aus den Trainingsbildern

(„Nominal Samples“) extrahiert. Hierzu wird ein „Pretrained Encoder“ verwendet, analog zu 2.4.3. Anschließend findet eine Unterabtastung bzw. eine Auswahl der Patch Features statt, die in der „Memory Bank“ gespeichert werden. Dieser Vorgang wird als „Coreset Subsampling“ bezeichnet.Ist diese Memory Bank erzeugt, ist die Methode initialisiert und das Training abgeschlossen. In der Testphase werden die Patch Features auf die gleiche Weise aus den „Test Samples“ extrahiert, wie in der Trainingsphase. Jedes dieser Patch Features wird nun mit den Patch Features in der Memory Bank verglichen. Dies geschieht mit einer „Nearest Neighbor Search“ (NN). Aus den Distanzen zum Nächsten Nachbarn kann dann, wie in 2.6 eine räumlich aufgelöste anomaliekarte erzeugt werden. Auf Grundlage dieser Anomaliekarte geschieht dann die Instanzklassifizierung als nominal oder anomal. Nachfolgende Abbildung 3.1, die aus der Veröffentlichung übernommen wurde, zeigt die Funktionsweise der Methode PatchCore.

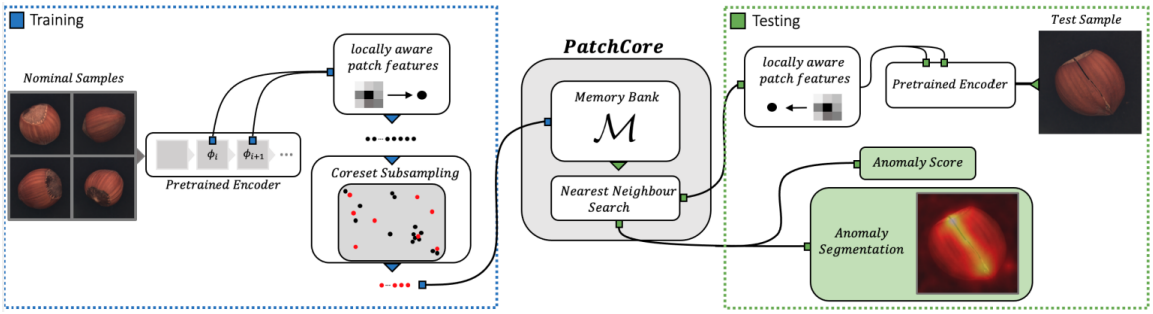


Abbildung 3.1: PatchCore

3.2.1 Erzeugen der Patch Features

Zunächst werden einige Notationen definiert, die im Folgenden verwendet werden. Es wird sich dabei auf die Notationen aus der Veröffentlichung bezogen. So wird die Menge aller Trainingsbilder als $\mathcal{X}_{train}$ bezeichnet. Die Menge aller Testbilder als $\mathcal{X}_{test}$. Für den Trainingsdatensatz gilt im Sinne der Unüberwachtheit, dass es sich um ausschließlich nominale Samples handelt. Im Testdatensatz können sowohl nominale als auch anomale Samples enthalten sein. Bezeichnen wir die wahre Klassenzugehörigkeit eines Bildes x als y_x, so kann diese entweder 0 (nominal) oder 1 (anomal) sein. Für den Trainingsdatensatz gilt dann: $\forall x \in \mathcal{X}_{train} : y_x = 0$ und für den Testdatensatz $\forall x \in \mathcal{X}_{test} : y_x \in \{0, 1\}$.
Den bereits in 2.5 und 2.6 angetroffenen „Pretrained Encoder“ wird als ϕ bezeichnet. Es wird dabei, wie bereits gesehen, nicht der Ausgang dieses Netzwerkes benutzt, sondern die Feature Maps aus einer bestimmten Schicht j des Netzwerkes. Im Falle von ResNets, die auch in dieser Veröffentlichung hauptsächlich verwendet werden, ist $j \in \{1, 2, 3, 4\}$. j wird folgend auch als „Hierarchielevel“ bezeichnet und spielt eine wichige Rolle. $\phi_{i,j} = \phi_j(x_i)$ bezeichnet die Feature Map des Bildes $x_i \in \mathcal{X}$ aus dem Hierarchielevel j. Wie bereits in 2.5.2 diskutiert, ist eine sinnvolle Auswahl der Hierarchielevel eine wichtige Voraussetzung für gute Ergebnisse. Auch die Autoren von PatchCore weisen auf diese Problemstellung hin. Man könne, wie bei SPADE, die letzte Ebene in der Merkmalshierarchie des Netzes verwenden. Dies bringe aber die

folgenden zwei Probleme mit sich. Erstens gehe dabei mehr lokalisierte nominale Informationen verloren. Das sei während der Trainingsphase kritisch, weil die Arten von Anomalien, die zum Testzeitpunkt auftreten, nicht im Voraus bekannt seien und die möglichst vollständige Erfassung des Normals notwendig sei. Zweitens seien die sehr tiefen und abstrakten Merkmale in den vortrainierten ImageNet-Netzwerken auf die Aufgabe der Klassifizierung natürlicher Bilder ausgerichtet, welche nur wenig direkte Überschneidungen mit der hier vorliegenden Aufgabe der industriellen aufweise. Es wird deshalb vorgeschlagen, Merkmale aus den mittleren Hierarchieleveles zu verwenden. Das entspricht bei ResNets $j \in \{2, 3\}$. Wie in 2.4 zu erkennen, handelt es sich bei $\phi_{i,j}$ um einen dreidimensionalen Tensor: $\phi_{i,j} \in \mathbb{R}^{c^* \times h^* \times w^*}$ mit c^* als Tiefe der Feature Maps, h^* als Höhe und w^* als Breite. $\phi_{i,j}(h, w) \in \mathbb{R}^{c^*}$ bezeichnet dann den zur Position $h \in \{1, ..., h^*\}$ und $w \in \{1, ..., w^*\}$ gehörenden Vektor der Länge c^*. Unter der Annahmen, dass die Größe des Feldes im Originalbild x_i, das Einfluss auf ein $\phi_{i,j}(h, w)$ nimmt („Receptive Field"), ausreichend groß ist, um einen ausreichenden räumlichen Kontext zu erfassen, eignet sich dieser Vektor als „Patch Feature" für eine gegenüber räumlichen Variationen robuste Anomaliedetektion.
Um diese wünschenswerte Annahme zu erfüllen, wird eine Aggregation der lokal umliegenden Regionen („local Neighborhood Aggregation") durchgeführt, das nachfolgend vorgestellt wird und die Größe des Receptive Fields steuert.
Dafür wird die oben eingeführte Notation für $\phi_{i,j}(h, w)$ um eine ungerade Feldgröße („patchsize") p erweitert, die die benachbarten Feauture Vektoren mit einbezieht. Zunächst wird diese Nachbarschaft wie folgt definiert:

$$\mathcal{N}_p^{(h,w)} = \left\{(a, b)|a \in \left[h - \left\lfloor \frac{p}{2} \right\rfloor, ..., h + \left\lfloor \frac{p}{2} \right\rfloor\right], b \in \left[w - \left\lfloor \frac{p}{2} \right\rfloor, ..., w + \left\lfloor \frac{p}{2} \right\rfloor\right]\right\}$$

Damit ergeben sich schließlich ein „Patch Feature" zu

$$\phi_{i,j}\left(\mathcal{N}_p^{(h,w)}\right) = f_{agg}\left(\{\phi_{i,j}(a, b)|(a, b) \in \mathcal{N}_p^{(h,w)}\}\right),$$

wobei f_{agg} eine Aggregationsfunktion ist. Die Aggregationsfunktionsfunktion, die in der PatchCore Methode verwendet wird, ist ein adptives „Average Pooling", in einer Dimension, die unabhängig von der Länge der Eingangsfeature, immer eine feste Länge d ausgibt.
Da diese Operation für alle Paare von (h, w) mit $h \in \{1, ..., h^*\}$ und $w \in \{1, ..., w^*\}$ durchgeführt wird, wird die Auflösung der Feature Map erhalten. Für einen gesamten Feature Map Tensor ergibt sich dementsprechend:

$$\mathcal{P}_{s,p}\left(\phi_{i,j}\right) = \left\{\phi_{i,j}\left(\mathcal{N}_p^{(h,w)}\right)|h \in \{1, ..., h^*\}, w \in \{1, ..., w^*\}\right\}$$

Wie bereits erwähnt, geschieht diese Operation für verschiedene Hierarchielevel j. Weil die Auflösung der Feature Maps mit steigendem Hierarchielevel abnimmt, wird $\mathcal{P}_{s,p}\left(\phi_{i,j+1}\right)$ berechnet und anschließend auf die Auflösung von $\mathcal{P}_{s,p}\left(\phi_{i,j}\right)$ bilinear interpoliert. Jedes Element wird dann mit dem korrespondieren Element, also dem Element an der gleichen Stelle, aggregiert. Würde auf eine Auswahl der Patch Feature, wie im folgenden Abschnitt erläutert, verzichtet, würde sich folgende Memory-Bank ergeben:

$$\mathcal{M} = \bigcup_{x_i \in \mathcal{X}_{train}} \mathcal{P}_{s,p}\left(\phi_{i,j}\right)$$

3.2.2 Coreset Subsampling

Insbesondere, wenn $\mathcal{X}_{train}$ eine große Kardinalität hat, also viele Bilder hat, wird die Memory-Bank $\mathcal{M}$ sehr groß. Wie bereits in 2.5 festgestellt ist diese Kardinalität besonders laufzeitkritisch, weil die Nächste Nachbar Suche in der Memory-Bank mit einer Komplexität von $\mathcal{O}(n)$ berechnet wird. Wie bereits in 2.6 zu sehen, ist ein „patchbasierter“ Vergleich zwischen allen Elementen in $\mathcal{M}$ und allen Elemnten in $\mathcal{P}_{s,p}\left(\phi_{i,j}\right)$ ein notwediger Schritt, nicht nur um die Anomaliekarte zu erstellen, die in dieser Arbeit ohnehin von keinem großen Interesse ist, sondern auch um eine robuste und präzise Instanzklassifizierung durchzuführen. Für die Laufzeitoptimierung ist es also wünschenswert, die Kardinalität der Memory-Bank zu reduzieren.
Auf der anderen Seite müssen die Elemente in $\mathcal{M}$ möglichst gut nominale Eigenschaften abbilden, um eine gute Anomaliedetektion zu ermöglichen. Wie in der Veröffentlichung gezeigt wird, (§4.4.2 - Importance of Coreset Subsampling) führt der naive Ansatz, zufällig Elemente aus $\mathcal{M}$ auszuwählen, nicht zu zufriedenstellenden Ergebnissen.
Das von PatchCore zugrundeliegenden Konzept setzt genau hier an. Es soll eine Teilmenge $\mathcal{S} \subset \mathcal{A}$ gefunden werden, bei der die Problemlösung über $\mathcal{A}$ am ehesten und vor allem schneller durch die über $\mathcal{S}$ berechnete Lösung approximiert werden kann. Dabei ist die Methode, die zu einer solchen Teilmenge führt, problemspezifisch. Im Falle von PatchCore wird eine Berechnung von Nächsten Nachbarn durchgeführt, weswegen gemäß [15] ein „MiniMax-Funktion“ sich anbietet, um eine annähernd ähnliche Abdeckung der $\mathcal{M}$ mit $\mathcal{M}_\mathcal{C}$ zu erreichen. Dies kann wie folgt gelöst werden:

$$\mathcal{M}_\mathcal{C}^* = \underset{\mathcal{M}_C \subset \mathcal{M}}{\arg\min} \max_{m \in \mathcal{M}} \min_{n \in \mathcal{M}_\mathcal{C}} \|m - n\|_2$$

Die exakte Berechnung von $\mathcal{M}_\mathcal{C}^*$ ist NP-schwer, also nicht in polynomieller Zeit berechenbar. Es handelt sich zwar um einen Prozess, der nicht während der Inferenz durchgeführt werden muss, sondern einmalig während der Trainingsphase, aber es muss dennoch mit iterativen, approximierenden Verfahren gearbeitet werden.
Aus [15] wird ein „Greedy Algorithmus“ übernommen, der iterativ Elemente aus $\mathcal{M}$ auswählt, die die größte Distanz zu allen bereits ausgewählten Elementen haben. Um die Laufzeit des Subsamplings weiter zu reduzieren, wird das „Johnson-Lindenstrauss Lemma“[4] verwendet, um die Dimensionalität der Elemente $m \in \mathcal{M}$ durch zufällige lineare Projektion $\psi : \mathbb{R}^d \to \mathbb{R}^{d^*}$ mit $d^* < d$ zu reduzieren. Anschaulich kann dies durch eine Punktwolke erklärt werden, die aus zufälligen Blickwinkeln betrachtet wird, wodurch die 3-D-Struktur als 2-D Struktur angenähert wird. (TODO –> Algorithmus?)

3.2.3 Bestimmen des Anomaliegrades

Wie bereits in der Einleitung zu diesem Katpitel erwähnt, ist die Grundlage der Bestimmung des Anomaliegrades die Distanz zu den Nächsten Nachbarn in der Memory-Bank. Das gilt sowohl für die Anomaliekarte bzw. die Segmentierung, als auch für die Instanzklassifizierung. Zunächst

muss während der Inferenz aus einem Testbild $x_i \in \mathcal{X}_{test}$ die Patch Features extrahiert werden. Dies geschieht auf die gleiche Weise, wie in der Trainingsphase:

$$\mathcal{P}(x_i) = \mathcal{P}_p\left(\phi_j(x_i)\right)$$

Diese Menge $\mathcal{P}(x_i)$ enthält nun die Patch Features m^{test} des Testbildes x_i. Nun gilt es zu jedem Patch Feature in $\mathcal{P}(x_i)$ den Nächsten Nachbarn in der Memory Bank $\mathcal{M}$ zu finden:

$$m^* = \arg\min_{m \in \mathcal{M}} \left\| m - m^{test} \right\|_2, \forall m^{test} \in \mathcal{P}(x_i)$$

Es gilt zu beachten, dass jedes m^{test} zu einer Position (h, w) im Bild x_i gehört. So kann eine räumlich aufgelöste Anomaliekarte M erzeugt werden, die die Distanz zu den Nächsten Nachbarn in der Memory-Bank enthält:

$$M = \left(\left\| m^*_{h,w} - m^{test}_{h,w} \right\|_2 \right)_{h,w}, \forall (h, w) \in \{1, ..., h^*\} \times \{1, ..., w^*\}$$

Diese Anomaliekarte M kann schließlich mittels bilinearer Interpolation und einer anschließenden Glättung auf die Auflösung des Originalbildes x_i gebracht werden, wodurch eine pixelweise Klassifikationskarte möglich wird. Weil dies in dieser Arbeit nicht von Interesse ist, wird darauf nicht weiter eingegangen.
Die Instanzklassifizierung könnte nun analog zu PaDiM (2.6.1) durch Maximalwerbildung durchgeführt werden. Die Autoren von PatchCore gehen ähnlich vor, fügen jedoch noch einen Gewichtungsschritt hinzu. Zunächst wird ganz analog vorgegangen und die maximale Distanz zu den Nächsten Nachbarn herangezogen, was nichts anderes als der Maximalwert der Anomaliekarte M ist.

$$m^{test,*}, m^\star = \arg\max_{m^{test} \in \mathcal{P}(x_i)} \arg\min_{m \in \mathcal{M}} \left\| m - m^{test} \right\|_2$$

$$s^* = \left\| m^{test,*} - m^\star \right\|_2 = \max\{M\}$$

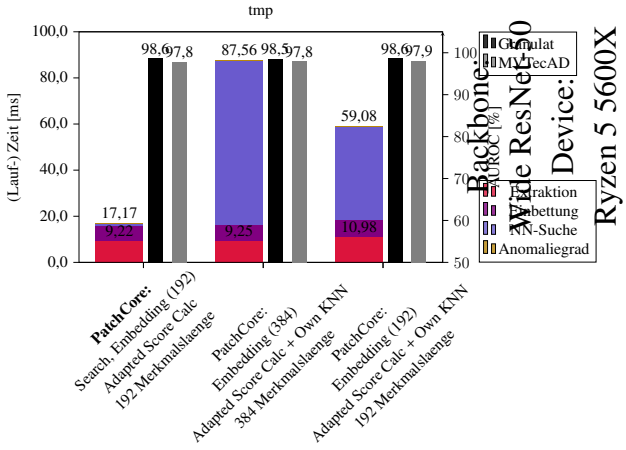
Die Gewichtung schließt die nächsten b Patch Features in M zu $m^\star$ mit ein. Diese Menge notieren wir als $\mathcal{N}_b(m^\star)$. Der Gedanke hinter dieser Gewichtung ist, den Anomaliegrad s dann zu erhöhen, wenn die Feature Patches in $\mathcal{N}_b(m^\star)$ selbst weit entfernt vom Anomaliekandidaten Patch Feature $m^\star$ sind und es sich somit ohnehin um seltene nominale Patch Features handelt. Der finale, für die Instanzklassifiziergung entscheidende Anomaliegrad s ergibt sich dann zu:

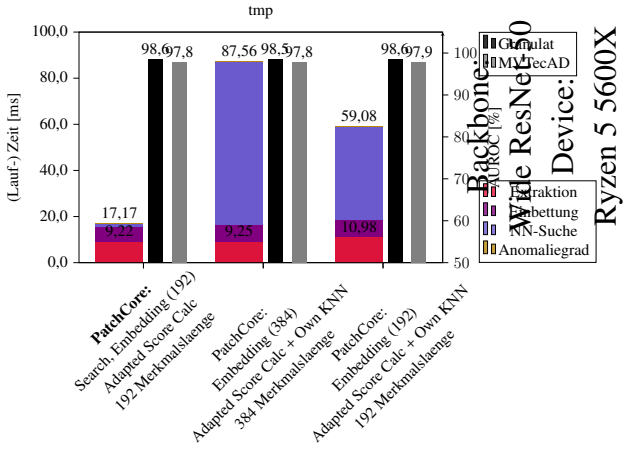
$$s = \left(1 - \frac{e^{\left\| m^\star - m^{test,*} \right\|_2}}{\sum_{m \in \mathcal{N}_b(m^\star)} e^{\left\| m - m^{test,*} \right\|_2}} \right) \cdot s^*$$

Durch diese Gewichtung wird die Instanzklassifizierung robuster und erhöht die Instanzklassifiziergungsgenauigkeit, wodurch es eine weitere Weitereentwicklung gegenüber PaDiM darstellt.

3.3 Ergebnisse und Diskussion der Originalmethode

In diesem Abschnitt werden die Ergebnisse der Originalmethode PatchCore vorgestellt und diskutiert. Dies soll vor allem im Hinblick auf die Eignung für eine Implementierung auf einem ressourcenbeschränkten Gerät, nämlich einem RaspberryPi 4B (2.7) geschehen. Es wird sich dabei auf den hier verwendeten Datensatz MVTecAD (2.1) bezogen. Weitere Datensätze, die in der Veröffentlichung verwendet wurden, werden mit dem Verweis auf die Veröffentlichung nicht weiter betrachtet. Die zur Evaluation herangezogene Metrik ist AUROC (2.3).
Wie bereits in der Einleitung erwähnt, ist die Methode PatchCore grundsätzlich in der Lage, sehr hohe Instanzklassifiziergungsgenauigkeiten zu erreichen, die auch zwei Jahre nach Veröffentlichung noch zu den besten gehören. Erreicht wird dies teilweise durch ein Ensemble von verschiendenen Feature Extraktoren bzw. Backbones und höhere Auflösungen. So wird ein „DenseNet 201“[10], ein „ResNext 101“[17] und ein bereits bekanntes Wide ResNet mit 101 Schichten[18] verwendet. Steht eine GPU zur Verfügung, welche, wie in 2.4 zu sehen, die Laufzeit von CNNs deutlich reduziert, können selbst mit einer solchen Konfiguration, bestehend aus vielen Backbones, einigermaßen schnelle Inferenzzeiten erreicht werden. Da die Zielsetzung dieser Arbeit jedoch die Implementierung auf einem RaspberryPi 4B ist und bereits gezeigt wurde, dass das Ausführen von CNNs auf einem solchen Gerät äußerst laufzeitkritisch ist, ist eine solche Konfiguration im Kontext dieser Arbeit nicht sinnvoll. Beschränken wir uns auf Konfigurationen, mit nur einem Backbone und auf die Auflösung von 256×256 Pixeln, so erreicht PatchCore mit einem Wide ResNet-50 Backbone eine Instanzklassifiziergungsgenauigkeit (AUROC) von 99,1 mit einer Reduktion der Memory Bank auf 25%. Wird die Größe der Memory Bank auf 1% reduziert, verschlechtert sich der AUROC nur leicht auf 99,0%. Wie im Laufe dieses Kapitels noch zu sehen sein wird, ist die Reduzierung der Kardinalität der Memory Bank ein wichtiger Schritt, um die Laufzeit zu reduzieren. Die hier angewandte Methode des Coreset Subsamplings ist also ein wichtiger Bestandteil und eine Errungenschaft der Methode PatchCore.
Neben dem Erzeugen der Patch Feature ist die Nächste Nachbar Suche in der Memory Bank besonder laufzeitkritisch. In der Veröffentlichung lediglich eine kleine Randnotiz im Anhang, ist diese Suche mit „FAISS“ (TODO –> ref und cite) implementiert. FAISS ist eine Bibliothek, entwickelt von Facebook's AI Research Department, die die Nächste Nachbar Suche enorm beschleunigt und im Laufe dieses Kapitels nochmal genauer erläuert und analysiert wird.
Insgesamt ist die Methode PatchCore zwar in ihrer Originalform eine ausgezeichnete Basis, aber für die Implementierung auf einem RaspberryPi 4B nur eingeschränkt geeignet. Es werden im Folgenden zahlreiche Adaptionen vorgestellt und evaluiert, die das Ziel haben, die Laufzeit zu reduzieren und dabei möglichst viel der Genauigkeit zu erhalten.


tmp
(Lauf-) Zeit [ms]
AUROC [%]
Granulat
MVTecAD
Extraktion
Einbettung
NN-Suche
Anomaliegrad
17,17
9,22
98,6
97,8
87,56
9,25
98,5
97,8
59,08
10,98
98,6
97,9
PatchCore:
Search, Embedding (192)
Adapted Score Calc
192 Merkmalslaenge
PatchCore:
Embedding (384)
Adapted Score Calc + Own KNN
384 Merkmalslaenge
PatchCore:
Embedding (192)
Adapted Score Calc + Own KNN
192 Merkmalslaenge
Backbone: Wide ResNet50
Device: Ryzen 5 5600X


tmp
(Lauf-) Zeit [ms]
AUROC [%]
Granulat
MVTecAD
Extraktion
Einbettung
NN-Suche
Anomaliegrad
17,17
9,22
98,6
97,8
87,56
9,25
98,5
97,8
59,08
10,98
98,6
97,9
PatchCore:
Search, Embedding (192)
Adapted Score Calc
192 Merkmalslaenge
PatchCore:
Embedding (384)
Adapted Score Calc + Own KNN
384 Merkmalslaenge
PatchCore:
Embedding (192)
Adapted Score Calc + Own KNN
192 Merkmalslaenge
Backbone: Wide ResNet50
Device: Ryzen 5 5600X

Kapitel 4

EfficientAD

4.1 Einleitung

In diesem Kapitel wird die Methode EfficientAD vorgestellt.

Kapitel 5

SimpleNet

Hier dann SimpleNet....

Anhang A

ASCII-Tabelle

…

Literaturverzeichnis

[1] Bergman, Liron und Hoshen, Yedid: *Classification-Based Anomaly Detection for General Data*, 2020.

[2] Bergmann, Paul, Batzner, Kilian, Fauser, Michael, Sattlegger, David und Steger, Carsten: *The MVTec Anomaly Detection Dataset: A Comprehensive Real-World Dataset for Unsupervised Anomaly Detection*. International Journal of Computer Vision, 129, 04 2021.

[3] Bishop, Christopher M: *Pattern Recognition and Machine Learning*. Springer, 2006.

[4] Dasgupta, Sanjoy und Gupta, Anupam: *An elementary proof of a theorem of Johnson and Lindenstrauss*. Random Structures & Algorithms, 22(1):60–65, 2003.

[5] Fawcett, Tom: *Introduction to ROC analysis*. Pattern Recognition Letters, 27:861–874, 06 2006.

[6] Foundation, The Raspberry Pi: *Specifications for the Raspberry Pi 4 Model B*. https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2711/rpi_DATA_2711_1p0_preliminary.pdf, 2023. Accessed: 2023-10-09.

[7] Geerling, Jeff: *Power Consumption Benchmarks | Raspberry Pi Dramble*. https://www.pidramble.com/wiki/benchmarks/power-consumption. Accessed: 2023-10-09.

[8] GmbH, MvTec: *MVTec LOCO: The MVTec logical constraints anomaly detection dataset*. https://www.mvtec.com/company/research/datasets/mvtec-loco, 2023.

[9] He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing und Sun, Jian: *Deep Residual Learning for Image Recognition*, 2015.

[10] Huang, Gao, Liu, Zhuang, Maaten, Laurens van der und Weinberger, Kilian Q.: *Densely Connected Convolutional Networks*, 2018.

[11] James, Gareth, Witten, Daniela, Hastie, Trevor und Tibshirani, Robert: *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated, 2014.

[12] Maine, The Weaver Computer Engineering Research Group of University of: *FLOPS/Watt of Various CPUs*. https://web.eece.maine.edu/~vweaver/group/green_machines.html, 2023. Accessed: 2023-10-09.

[13] PC Games Hardware, Maximilian Holm von: *RTX 4090: Mit starker Übertaktung mehr als 100 TFLOPS Rechenleistung erreicht*. https://www.pcgameshardware.de/Grafikkarten-Grafikkarte-97980/News/RTX-4090-Mit-starker-Uebertaktung-mehr-als-100-TFLOPS-Rechenleistung-erreicht-1405138/, 2022. Accessed: 2023-10-09.

[14] Roth, Karsten, Pemula, Latha, Zepeda, Joaquin, Schölkopf, Bernhard, Brox, Thomas und Gehler, Peter: *Towards Total Recall in Industrial Anomaly Detection*, 2022.

[15] Sener, Ozan und Savarese, Silvio: *Active Learning for Convolutional Neural Networks: A Core-Set Approach*, 2018.

[16] Wikipedia contributors: *Raspberry Pi 4 — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Raspberry_Pi_4&oldid=1178956001, 2023. [Online; accessed 9-October-2023].

[17] Xie, Saining, Girshick, Ross, Dollár, Piotr, Tu, Zhuowen und He, Kaiming: *Aggregated Residual Transformations for Deep Neural Networks*, 2017.

[18] Zagoruyko, Sergey und Komodakis, Nikos: *Wide Residual Networks*, 2017.