Eine vergleichende Leistungsanalyse zur Skalierbarkeit von Anomaliedetektions-Methoden

T. Wolf, F. Waßelewsky, M. Pape, A. Limberg, D. M. Güler lf und E. Uhlmann, IPK, Berlin

Abstract

Diese Arbeit präsentiert eine vergleichende Analyse der Skalierbarkeit verschiedener Anomaliedetektionsmethoden im Kontext industrieller Bildverarbeitung. Durch eine systematische Evaluation werden die Leistungsparameter dieser Methoden hinsichtlich ihres Speicherbedarfs und ihrer Laufzeit analysiert, wobei besonderes Augenmerk auf ihre Skalierungseigenschaften bei steigender Bildauflösung gelegt wird.

Die Studie zeigt signifikante Unterschiede in der Skalierbarkeit zwischen den untersuchten Methoden. Durch detaillierte Messungen wurden die Modellanteile am Ressourcenverbrauch ermittelt, was eine präzise Identifikation von Optimierungspotentialen ermöglicht. Die Extrapolation der Messwerte erlaubt zudem Vorhersagen über das Skalierungsverhalten bei sehr großen Auflösungen, was für praxisnahe Anwendungen in industriellen Umgebungen von besonderer relevant ist.

Die Ergebnisse liefern Einblicke für die Auswahl geeigneter Anomaliedetektionsmethoden in Abhängigkeit von spezifischen Systemanforderungen und verfügbaren Ressourcen.

1 Einleitung

Im industriellen Kontext hat die automatische Fehlererkennung einen signifikanten Einfluss auf die gleichbleibende Qualität von Produktionsgütern. Dabei sind neben Verfahren der Defektklassifikation gegenwärtig vor allem Anomaliedetektions(AD)-Methoden im Fokus der Forschung. Im Gegensatz zu konventionellen Bildverarbeitungsmethoden oder Ansätzen des überwachten Lernens, der Defektklassifikation, werden hierbei vorrangig defektfreie Bilddaten für das Training verwendet. Auf diese Weise wird die aufwändige Datenerhebung spezifischer Defektbilder minimiert. Die Methode lernt Abweichungen vom Soll-Zustand zu erkennen, wodurch sie auch unbekannte Defekte detektieren kann.

Aktuelle wissenschaftliche Arbeiten zur Anomaliedetektion beschäftigen sich dabei vorrangig mit spezifischen Optimierungen einzelner Methoden und dem Vergleich von Metriken zur Bewertung der generellen Diskriminierungsfähigkeit sowie der Lokalisierung von Defekten. Die Skalierbarkeit von AD-Methoden mit zunehmender Bildauflösung, als kritischer Faktor für den Einsatz in praxisnahen und industriellen Anwendungen, wird jedoch zumeist vernachlässigt. Speziell bei der Detektion von Mikrodefekten bzw. Fehlern, welche in Relation zum Gesamtbild klein sind, nimmt die Performance von AD-Methoden hinsichtlich der Fehlererkennung signifikant ab. Eine Ursache hierfür ist die nötige Skalierung der Auflösung.

Gleichzeitig bieten höhere Bildauflösung zwar detailliertere Bildinformationen, stellen jedoch auch Herausforderungen an Recheneffizienz, Robustheit gegen komplexe Szenen und die Fähigkeit zur Extraktion relevanter Merkmale. Industrielle Anwendungen sind aufgrund hoher Taktraten und kurzer Prozesslaufzeiten darauf angewiesen, dass Defekte schnell und zuverlässig detektiert werden.

Diese Arbeit setzt sich zum Ziel diese Lücke zu schließen und untersucht inwiefern die Laufzeit und der Speicherbedarf aktueller AD-Methoden mit der Bildauflösung skaliert. Konkret werden drei

unterschiedliche AD-Ansätze für diese Untersuchungen herangezogen, welche sich in ihrer Methodik grundlegend unterscheiden. Es handelt sich dabei um PatchCore (Speicherbank-basiert), EfficientAD (Knowledge Distillation) und DDAD (Rekonstruktions-basiert) [1 bis 3]. Auf Grundlage dieser Analyse wird in einem ersten Schritt deduktiv ermittelt, welche Modelltypen am vielversprechendsten mit steigender Bildauflösung skalieren. Zusätzlich wird im gleichen Kontext untersucht, wo die Systemgrenzen aktuell verfügbarer Hardware liegen.

2 Methode

Im Rahmen dieser Arbeit werden drei AD-Methoden in Hinblick auf Laufzeit und Speicherbedarf bei steigender Bildauflösung untersucht. Dabei handelt es sich um PatchCore, EfficientAD und DDAD. Alle drei Methoden unterscheiden sich in ihrer Methodik. Auf diese Weise kann analysiert werden, inwiefern die Performance unterschiedlicher Modelltypen durch die Bildgröße beeinflusst wird. Nachfolgend werden die einzelnen Methoden kurz und bündig erläutert.

PatchCore

PatchCore von Roth et al. [1] ist eine AD-Methode, welche den distanzbasierten AD-Methoden zuzuordnen ist. Ein vortrainierter Merkmalsextraktor isoliert relevante Bildinformationen, welche anschließend in einer sogenannten Merkmalsdatenbank abgespeichert werden. Da im Training ausschließlich defektfreie Bilder verwendet werden, sind auch nur typische Merkmale aus fehlerfreien Bildern in besagter Speicherbank enthalten. Wird nun ein Bild während der Inferenz auf Anomalien untersucht werden, werden zunächst Merkmale aus dem Bild extrahiert und im Anschluss mit der Merkmalsdatenbank verglichen. Als Anomaliebewertung dient der euklidische Abstand im Merkmalsraum.

Grundlegend unterscheidet sich *PatchCore* insofern von den vorherigen Speicherbank-basierten Methoden, als dass zur Optimierung der Laufzeit und des Speicherbedarfs auf das sogenannte Coreset Subsampling zurückgegriffen wird. Anhand dieses Verfahrens werden redundante Informationen in der Merkmalsdatenbank eliminiert. Die Autoren von *PatchCore* beobachten, dass bei einer Reduktion der Merkmalsdatenbank um den Faktor 100 weiterhin vergleichbar zufriedenstellende Ergebnisse in Bezug auf die Diskriminierungsfähigkeit des Modells erzielt werden.

EfficientAD

Die AD-Methode *EfficientAD* von Batzner et al. [2] verwendet den Ansatz der Knowledge Distillation, bei dem das Wissen eines Lehrermodells an ein kleineres Schülermodell weitergegeben wird. Ziel hierbei ist, die Effizienz eines kleinen Modells mit der Leistung eines großen Modells zu kombinieren. Während des Trainings lernt das Schülermodell, Merkmale aus defektfreien Bildern zu extrahieren und diese mit den vom Lehrermodell extrahierten Merkmalen zu vergleichen. Anomalien entstehen, wenn die Vorhersage des Schülermodells stark von der des Lehrermodells abweicht.

Beide Modelle, Lehrer und Schüler, verwenden die gleiche Modellarchitektur, das Patch Descriptor Network (PDN), welches speziell für effizientes Merkmalsextrahieren aus Bildern variabler Größe

entwickelt wurde. Um folglich sicherzustellen, dass das Schülermodell nicht einfach das Lehrermodell nachahmt, verwenden Batzner et al. u. a. architektonische Asymmetrie.

Ein Autoencoder wird zusätzlich trainiert, um auch semantische Anomalien wie unzulässige Positionen und Orientierungen zu erkennen. *EfficientAD* erzielt sehr niedrige Latenzzeiten von bis zu 2 ms und kann bis zu 600 Bilder pro Sekunde verarbeiten.

DDAD

Denoising Diffusion Anomaly Detection (DDAD) von Mousakhan et al. [3] ist eine rekonstruktionsbasierte AD-Methode. Während des Trainings werden die Trainingsbilder im sogenannte *Diffusion Process* mit Rauschen überlagert. Ein *U-Net* lernt anschließend den Rauschanteil vorherzusagen. Während des Trainings lernt das Modell auf diese Weise verrauschte defektfreie Bilder zu rekonstruieren.

Das Vorgehen beruht auf der Annahme, dass während des *Denoising*, sollte das Eingangsbild einen Defekt aufweisen, der Defekt bzw. die Anomalie nicht rekonstruiert wird. Anhand eines einfachen pixelweisen Vergleichs der beiden Bilder kann die Anomalie schließlich detektiert und lokalisiert werden.

Um Defekte, die durch einen einfachen Pixelvergleich nicht messbar sind, ebenfalls zu detektieren, wird außerdem auf einen vortrainierten Merkmalsextraktor zurückgegriffen.

Dieser wird während des Trainings anhand der Bilder auf die Merkmaldomäne feinabgestimmt. Zusätzlich zu dem Vergleich auf Pixelebene werden auch die Merkmale beider Bilder extrahiert und anschließend verglichen. Eine allgemeine Übersicht über das beschriebene Verfahren kann Abbildung 1 entnommen werden.

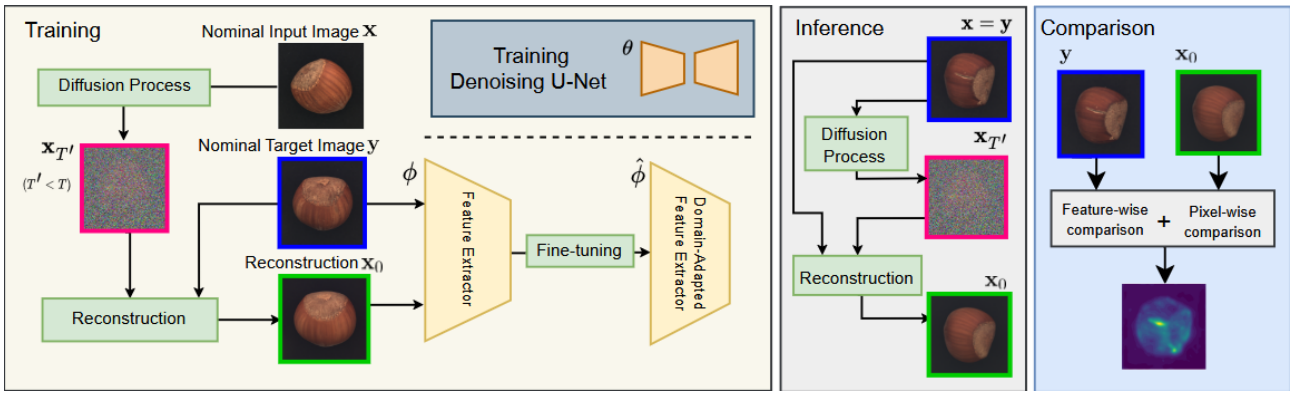

Abbildung 1: DDAD Prinzip [3]

Speicherbedarf

Der Speicherbedarf in der Inferenz des jeweiligen Modells setzt sich aus der Anzahl der Modellparameter, sowie den zwischengespeicherten Daten innerhalb der Schichten einer Architektur zusammen. Da während der Inferenz keine Gradienten berechnet werden, müssen diese bei der Berechnung nicht berücksichtigt werden. Sowohl die Modellparameter als auch die intermediären Daten liegen als 32 Bit Gleitkommazahlen im Speicher. Es wird angenommen, dass ein einzelnes Bild von der betrachteten Architektur ausgewertet wird (Batchgröße von 1). Damit ergibt sich der gesamte Speicherbedarf M eines Netzwerks L zu

$$\boldsymbol{M = \sum_{l}^{L} \mathcal{O}\left(p(l) \cdot d_p + a(l) \cdot d_a\right)} \quad (1)$$

mit die Anzahl der Parameter des Modells, d_p dem Datentyp der Modellparameter, $a(l)$ als Aktivierung der Schichten und d_a der Datentyp der Aktivierungen.

Der Speicherbedarf der Parameter einer Faltungsschicht (engl. *Convolutional Layer*) ergibt sich für einen Eingabetensor mit der Dimension $H_{in} \times W_{in} \times C_{in}$, mit C_{in} der Anzahl der Inputkanäle, C_{out} der Anzahl der Ausgabekanäle und K als Filtergröße eines $K \times K$ Filters, aus

$$\boldsymbol{\mathcal{O}(C_{in} \cdot C_{out} \cdot K^2 + C_{out})} \quad (2)$$

Für einen Ausgabetensor mit einer Höhe H_{out} und Breite W_{out} ergibt sich die Speicherauslastung für eine Aktivierung aus

$$\boldsymbol{\mathcal{O}(H_{out} \cdot W_{out} \cdot C_{out})} \quad (3)$$

(s. [4 bis 6]).

Um über die Systemgrenzen hinaus den Speicherbedarf für höhere Auflösungen zu bestimmen, werden Messungen bis zu einer Auflösung von 1024x1024 px durchgeführt und die aufgenommenen Messpunkte durch ein Polynom zweiten Grades interpoliert. Als Modellgleichung wird das folgende Polynom 2. Grades angenommen.

$$\boldsymbol{y = \beta_0 + \beta_1 x + \beta_2 x^2 + \varepsilon} \quad (3)$$

Die Modellparameter werden durch Minimierung des mittleren quadratischen Fehlers (Gleichung 3) bestimmt *(Least-Squares-Methode)*.

$$\boldsymbol{E = \sum_{j=0}^{k} \left| p(x_j) - y_j \right|^2} \quad (4)$$

Anhand der Modellgleichung aus (2) wird anschließend der Speicherbedarf für höhere Auflösungen extrapoliert. Eine quadratische Funktion kann nach [4 bis 6], sowie (2,3) angenommen werden, da die betrachteten Methoden in wesentlichen Teilen aus Schichten mit einer quadratischen Speicherkomplexität von bestehen [7].

Laufzeit

Für die Laufzeitmessung wird *Nvidia-Nsight Systems* [8] in Kombination mit der Softwarebibliothek *NVTX* [9] verwendet. Dies minimiert den Einfluss der CPU-Leistung, sowie der Datenübertragung auf das Messergebnis.

Darüber hinaus wird vor der eigentlichen Messung eine Aufwärmphase mit 100 Bildern durchgeführt, um zu verhindern, dass Initialisierungseffekte die Messung beeinflussen. Anschließend erfolgt die Erfassung der Laufzeiten. Hierbei wird die Laufzeit über eine Stichprobe von 1000 Bildern gemittelt, um den Einfluss kurzfristiger Schwankungen in der Systemleistung zu minimieren. Laufzeiten für Auflösungen, die außerhalb der Systemgrenzen liegen, werden ebenfalls anhand der Modellgleichung aus (2) extrapoliert.

Die Messumgebung bzw. verwendete Systemkonfiguration kann Tabelle 1 entnommen werden.

CPU	GPU	CUDA-Version	Driver-Version
AMD Ryzen 9 7950X	NVIDIA GeForce RTX 4090	12.4	550.144.03

Tabelle 1**: **Systemkonfiguration

3 Ergebnisse

Die Ergebnisse der im Methodenkapitel beschriebenen Untersuchungen werden nachfolgend für *PatchCore* [1], *EfficientAD* [2] und *DDAD* [3] ausgeführt.

Für die Ermittlung des Speicherbedarfs im Inferenzschritt wird eine Batchgröße von 1 verwendet. Für alle Algorithmen werden die durch die Autor*innen empfohlenen Konfigurationen verwendet.

EfficientAD

Die Ergebnisse für die Messungen der Speicherauslastung und der Laufzeit für *EfficientAD-M* (*Medium*) sind in Abbildung 2 und Abbildung 3 dargestellt. Das Suffix kennzeichnet hierbei die Modellgröße des verwendeten Lehrermodells.

Während die Speicherbelegung durch die Modellparameter konstant bleibt, steigt der Anteil des durch die Aktivierungen belegten Speichers mit quadratischer Tendenz.

Der *Autoencoder* skaliert deutlich schwächer mit der Auflösung, da die Architektur *des U-Nets* in der ersten und letzten Schicht stark herunterskalierte Merkmalskarten erzeugt. Die intermediären Schichten zeigen eine schwach auflösungsabhängige Charakteristik zeigen.

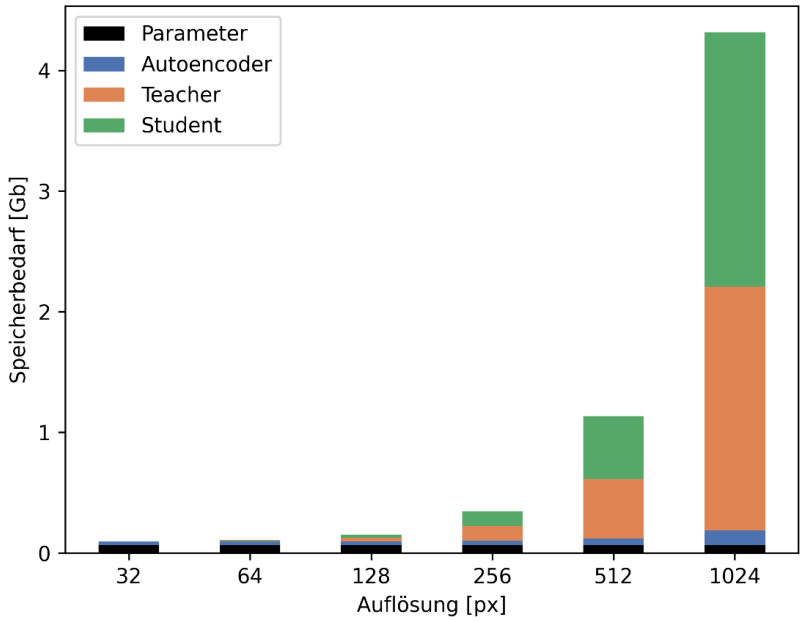

Abbildung 2**: **Anteiliger Speicherbedarf – EfficientAD

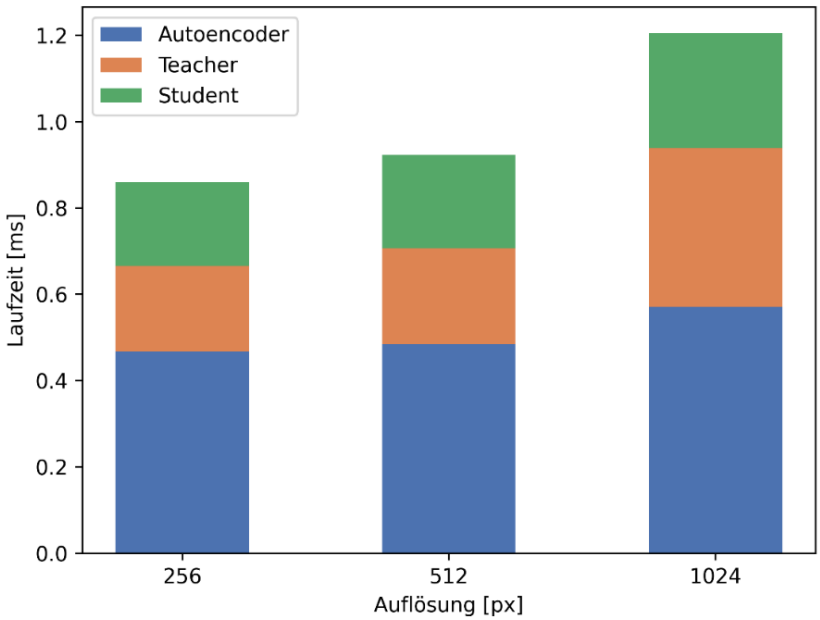

Abbildung 3**: **Anteilige Laufzeit – EfficientAD

Bei Betrachtung der Laufzeit zeigt sich eine andere Charakteristik. In Abbildung 3 sind die Ergebnisse der Laufzeitmessung für die Auflösungen {256x256; 512x512; 1024x1024} px dargestellt. Die Laufzeit der einzelnen Teilmodelle wurde gemessen und zur Gesamtlaufzeit addiert. Im Gegensatz zum Speicherbedarf braucht das effizienteste Modell, der *Autoencoder*, die längste Laufzeit. Das Verhalten erklärt sich durch die Verarbeitungseffizienz der verwendeten Hardware. Während der *Autoencoder* eine sehr tiefe Struktur, mit insgesamt 39 Schichten, inklusive Aktivierungsschicht, *Dropout*- und *Upsample*-Schichten besitzt, ist der *Student* bzw. *Teacher* aus lediglich 9 Schichten aufgebaut. Die Anzahl der Filter ist bei der *Student-Teacher* Architektur dabei deutlich höher (4.267.392 Parameter) als bei der *Autoencoder*-Architektur (1.096.320 Parameter). Die flachere Architektur nutzt das Parallelisierungspotential der GPU besser aus, da Schichten nur sequenziell berechenbar sind und erreicht so Laufzeitvorteile bei gleichzeitig deutlich höherem Speicherbedarf.

PatchCore

Der Speicherbedarf von *PatchCore-10* wird von der *WideResNet101* Architektur [10] dominiert, die als *Feature Extractor* genutzt wird. Nach [11] steigt der Speicherbedarf polynomiell und ist somit charakteristisch für diesen Ansatz. Die Größe der Memorybank hängt von der Anzahl der Patches ab, die wiederum in quadratischer Beziehung zur Auflösung steht. Somit steigt der Speicherbedarf der *Memorybank* ebenfalls quadratisch, wird aber durch das *Coreset Subsampling* [12] auf 10% der Features reduziert.

Die Laufzeit wird durch die *Feature Extraction* dominiert. Diese steigt, bedingt durch die patchweise Nachbearbeitung der extrahierten *Features*, polynomiell. Mit maximal linearer Laufzeit geht die NN-Suche in die Laufzeitbetrachtung ein. Douze et.al. geben an, dass mit *IndexL2Flat* Indizierung der Datenbank maximal $\mathcal{O}(n)$ beschränkte Komplexität zu erwarten ist [13]. Diese Indizierung wird in der Referenzimplementierung verwendet.

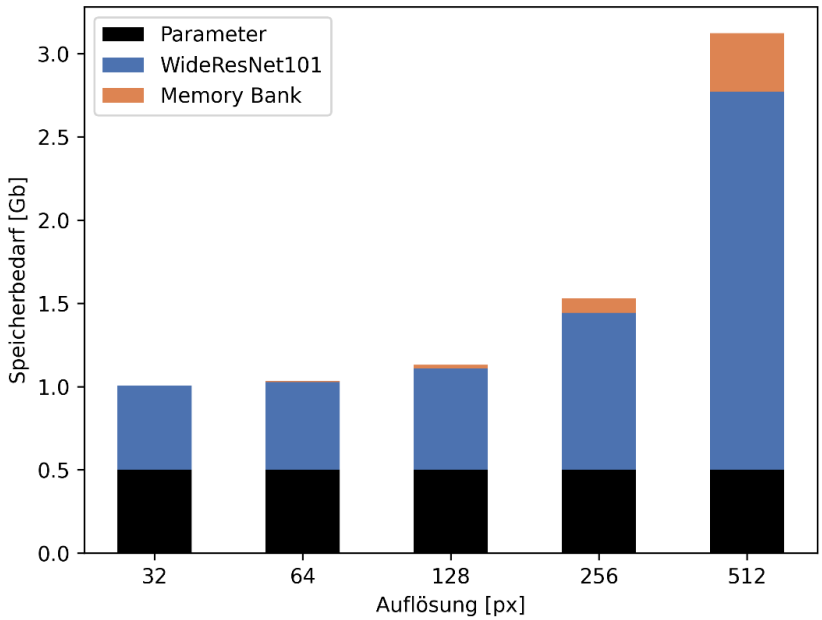

Abbildung 4: Anteiliger Speicherbedarf – PatchCore

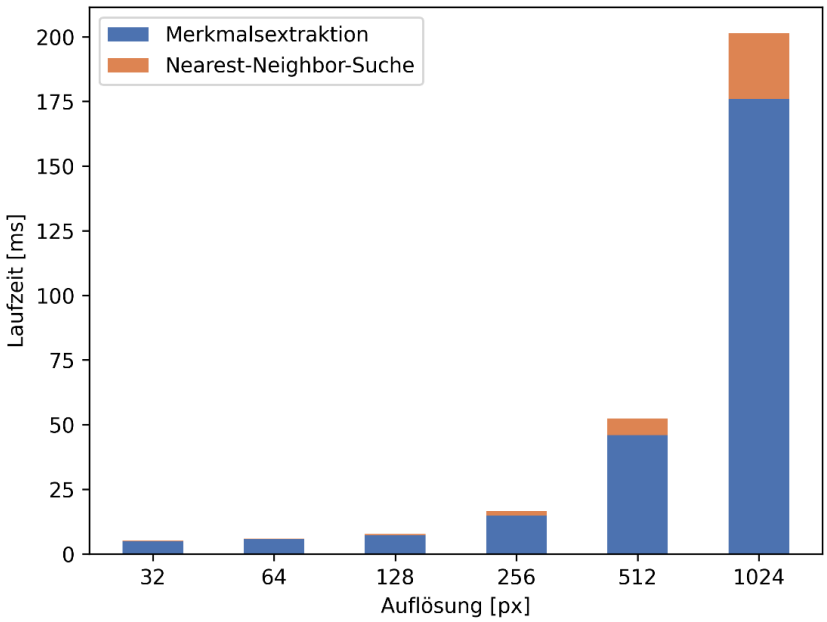

Abbildung 5: Anteilige Laufzeit – PatchCore

DDAD

Abbildung 6 zeigt den Speicherbedarf der Methode *DDAD-10*. Dabei wird der Bedarf des *Denoising U-Nets* einfach gezählt, da die zehn Iterationen sequenziell durchgeführt werden. Das U-Net erzeugt in der internen Repräsentation 3,69-mal mehr Daten als das domainadaptierte *WideResNet10*1. Dieses Verhalten ergibt sich aus der *Encoder-Decoder* Architektur, bei der beide Teile des Netzes aus *Convolutions* bestehen [3].

Der Rekonstruktionsschritt ist für 94% der Laufzeitkosten verantwortlich. Die verbleibenden 6% werden für die *Feature Extraction* benötigt. Da keine auflösungsabhängigen oder sequenziellen

Algorithmen verwendet werden, bleibt die Laufzeit dieses Algorithmus konstant, solange die GPU in der Lage ist, diese parallel zu verarbeiten.

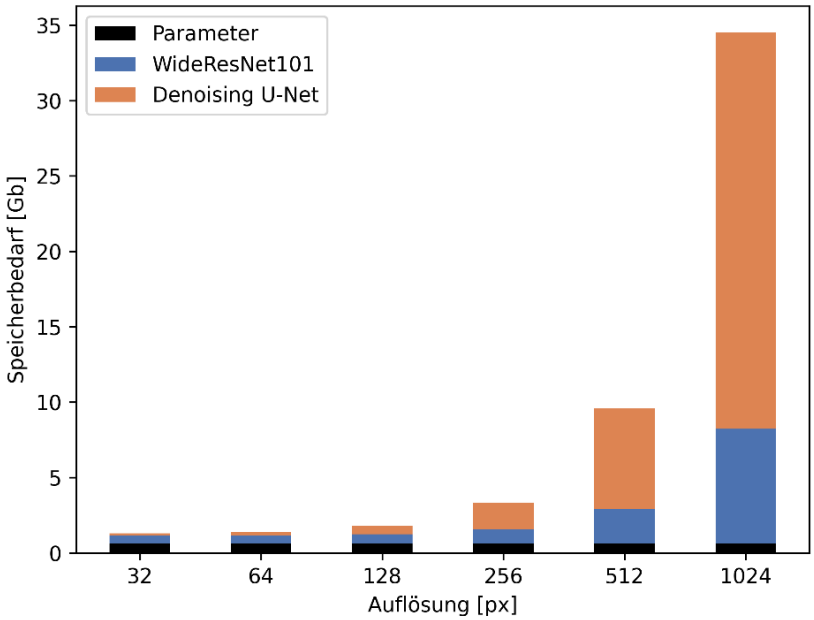

Abbildung 6: Anteiliger Speicherbedarf – DDAD

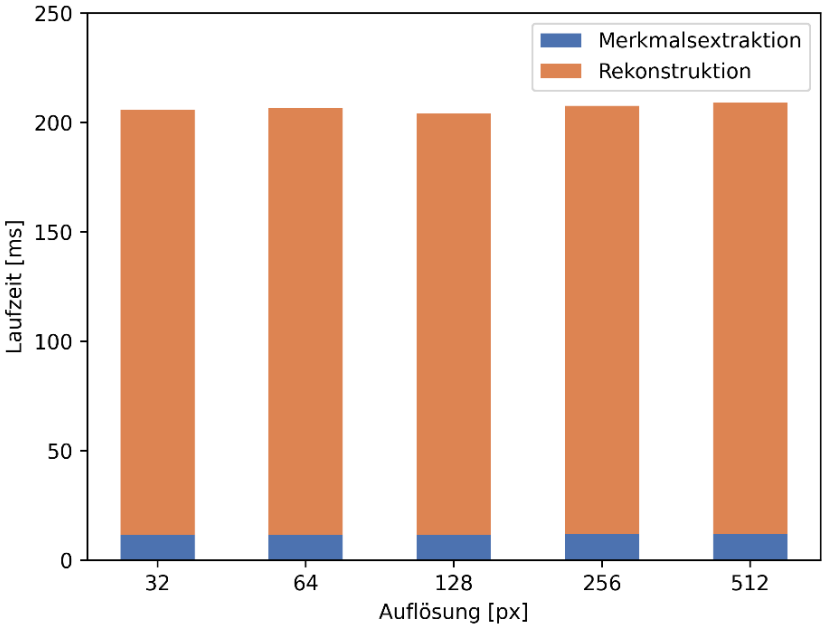

Abbildung 7: Anteilige Laufzeit – DDAD

Vergleichende Analyse

Auf Grund der Limitierung der eingesetzten Hardware (insb. *VRAM*) konnten die Messungen nur bis zu einer maximalen Auflösung von 1024x1024px durchgeführt werden. Wie oben beschrieben, wird auf Grundlage des polynomiellen Charakters, der den Modellen der betrachteten Methoden zugrunde liegt, ein Polynom zweiten Grade genutzt um den Speicherbedarf, sowie die Laufzeit der Algorithmen in höheren Auflösungsbereichen abzuschätzen.

Abbildung 8 zeigt den approximierten Verlauf des Gesamtspeicherbedarfs der Methoden bis zu einer Auflösung von 3000x3000 px. Zusätzlich wurde für ausgewählte GPU-Modelle der verfügbare *VRAM* eingetragen, um die maximale Auflösung zu schätzen.

Vertikale Linien sind eingezeichnet um die Auflösung in MP, welche für Kamerasensoren gängig sind, anzuzeigen. Dabei wird von einem 1:1 Format ausgegangen, da der Großteil der KI-Architekturen dieses Format erwartet.

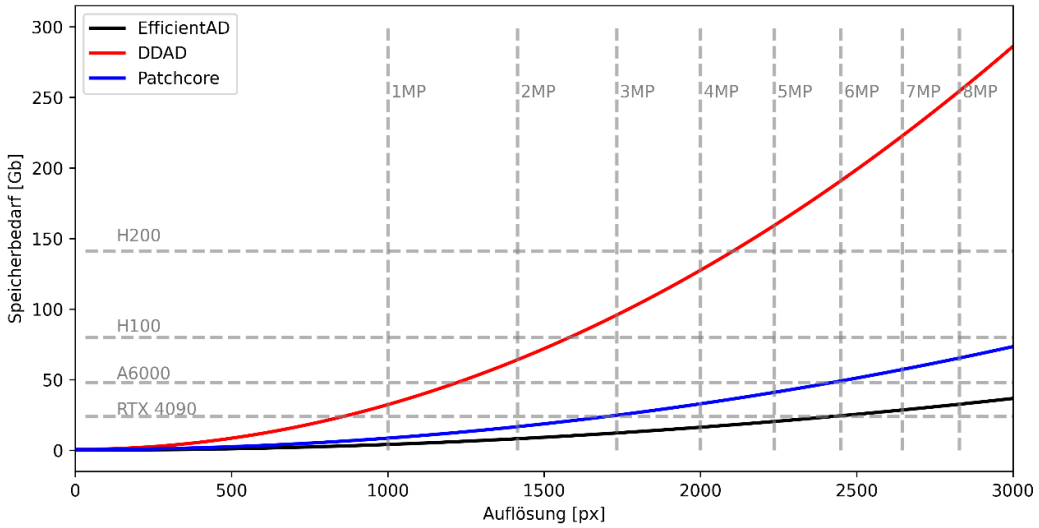

Abbildung 8**: **Speicherbedarf im Vergleich (Extrapoliert)

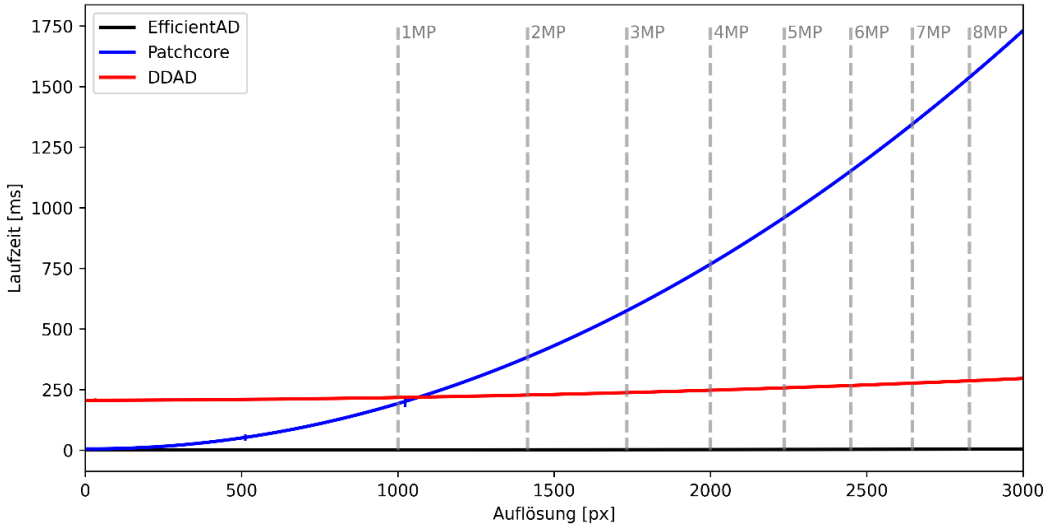

Abbildung 9**: **Laufzeit im Vergleich (Extrapoliert)

Auf Grund der Kombination von *WideResNet101* und *U-Net-Architektur* steigt der Speicherbedarf des *DDAD-10* Modells am schnellsten. Selbst auf in der Industrie gängigen GPUs sind Kamerasensoren mit 5MP nichtmehr voll nutzbar, sofern auf Techniken zur Speicherreduktion verzichtet wird.

PatchCore-10 und *EfficientAD-M* skalieren schwächer mit der Auflösung, so dass 5MP Sensoren ohne Einschränkungen mit diesen Methoden genutzt werden können. Die speziell entworfene Architektur von *EfficientAD* erlaubt die höchsten Auflösungen hinsichtlich des Speicherbedarfs. *PatchCore* hingegen arbeitet ohne ein destilliertes Modell als *Feature Extractor* und benötigt daher mehr Ressourcen trotz geringerer Anzahl an Komponenten.

Hinsichtlich der Laufzeit zeigt *PatchCore* das am stärksten auflösungsabhängige Verhalten, während *EfficientAD* und *DDAD* nahezu konstante Laufzeiten erreichen (vgl. Abbildung 9).

4 Schlussfolgerung

Es wurden drei Methoden mit unterschiedlichen Funktionsprinzipien ausgewählt und vergleichend hinsichtlich ihrer Laufzeit und ihres Speicherbedarfs analysiert, um zu beurteilen welche Methode in industriellen Szenarien einsetzbar sind. *PatchCore* und *EfficientAD* zeigen deutliche Speichervorteile gegenüber *DDAD*, was allerdings nicht ausreicht, um die Auflösungen moderne Kamerasensoren auszunutzen. Hinsichtlich der Laufzeit steigt *PatchCore* mit der Auflösung, während die alternativen Methoden konstante Laufzeiten im Rahmen der GPU Kapazität haben.

Im Rahmen dieser Arbeit wurde auf eine Optimierung der Methoden, wie Quantisierung [14] oder *Model Pruning* [15], verzichtet. Bei *EfficientAD* sind Optimierungsmaßnahmen bereits im Entwurf berücksichtigt. Daher kann eine weitere vergleichende Betrachtung der optimierten Methoden durchgeführt werden, um die hier gezeigten Tendenzen zu sichern.

Genauigkeitsmessungen gehen neben Laufzeit und Speicherbedarf als Auslegungskriterien in den Entwurf von bildverarbeitenden Inspektionssystemen ein. Diese enthalten anwendungsfall-spezifische Aspekte, weshalb eine Analyse gesondert durchgeführt werden muss.

5 Literaturverzeichnis

[1] Towards Total Recall in Industrial Anomaly Detection, Roth, K., Pemula, L., Zepeda, J., Schölkopf, B., Brox, T. u. Gehler, P., 2021

[2] Batzner, K., Heckler, L. u. König, R.: EfficientAD: Accurate Visual Anomaly Detection at Millisecond-Level Latencies. Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)

[3] Anomaly Detection with Conditioned Denoising Diffusion Models, Mousakhan, A., Brox, T. u. Tayyub, J., 2023

[4] MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications, Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M. u. Adam, H., 2017

[5] Tao Wei, Yonghong Tian u. C. Chen: Rethinking Convolution: Towards an Optimal Efficiency (2021)

[6] Xception: Deep Learning with Depthwise Separable Convolutions, Chollet, F., 2016

[7] Ahmad Farhan AlShammari: Implementation of Polynomial Regression using Least Squares and Gradient Descent in Python. International Journal of Computer Applications (2024)

[8] NVIDIA Developer: NVIDIA Nsight Systems, 2025. https://developer.nvidia.com/nsight-systems, abgerufen am: 11.03.2025

[9] NVIDIA Tools Extension Library (NVTX) :: NVIDIA Nsight VSE Documentation, 2023. https://docs.nvidia.com/nsight-visual-studio-edition/2020.1/nvtx/index.html, abgerufen am: 11.03.2025

[10] Wide Residual Networks, Zagoruyko, S. u. Komodakis, N., 2016

[11] Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaría, J., Fadhel, M. A., Al-Amidie, M. u. Farhan, L.: Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. Journal of big data 8 (2021) 1, S. 53

[12] Pankaj K Agarwal, Sariel Har, Peled Kasturi u. R Varadarajan: Geometric approximation via coresets. Combinatorical and Computational Geometry, Bd. 52. Cambridge University Press 2004

[13] The Faiss library, Douze, M., Guzhva, A., Deng, C., Johnson, J., Szilvasy, G., Mazaré, P.-E., Lomeli, M., Hosseini, L. u. Jégou, H., 2024

[14] A Survey of Quantization Methods for Efficient Neural Network Inference, Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M. W. u. Keutzer, K., 2021

[15] Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks, Hoefler, T., Alistarh, D., Ben-Nun, T., Dryden, N. u. Peste, A., 2021