Ein evolutionärer Ansatz für die automatische Ermittlung der Topologie neuronaler Netze

An evolutionary approach to automatically determine the topology of a neural network

Norbert Mitschke[1], Michael Heizmann[1], Klaus-Henning Noffz[2] und Ralf Wittmann[2]

[1] Karlsruher Institut für Technologie,
Institut für Industrielle Informationstechnik,
Hertzstraße 16, 76187 Karlsruhe
[2] Silicon Software GmbH
Konrad-Zuse-Ring 28, 68163 Mannheim

Zusammenfassung Infolge von sinkenden Hardware-Preisen und der zunehmenden Automatisierung wird maschinelles Lernen für industrielle Anwendungen wie klassischen Sichtprüfungsaufgaben immer attraktiver. In diesem Artikel wird ein metaheuristischer Ansatz für die automatische Ermittlung der Topologie eines neuronalen Netzes präsentiert, der auf differentieller Evolution basiert. Dieser ist in der Lage, anhand eines gegebenen Datensatzes und ohne zusätzliches Vorwissen einen geeigneten Klassifikator zu entwerfen. Gleichzeitig wird durch die Wahl einer geeigneten Fitnessfunktion der Ressourcenbedarf der Inferenz des neuronalen Netzes begrenzt bzw. minimiert. Für typische industrielle Datensätze kann mit dem Ansatz eine Topologie gefunden werden, die eine Genauigkeit von im Mittel über 98 % erreicht, während die Rechendauer relativ kurz bleibt.

Schlagwörter Neuronale Netze, Differentielle Evolution, Neuroevolution, Netzwerktopologie, Maschinelles Lernen.

Abstract As a result of falling hardware prices and increasing automation, machine learning is becoming increasingly attractive for industrial applications such as automatic visual inspection. This article presents a metaheuristic approach to automatically determine the topology of a neural network based on differential evolution for various applications on the basis of a given data set and without additional previous knowledge. Another design goal is to limit or minimize the resource requirements of the inference of the neural network. For typical industrial data sets, a topology can be found with this approach that achieves an accuracy of more than 98 % on average, while the processing time remains relatively short.

Keywords Neural networks, differential evolution, neuroevolution, structure learning, deep learning.

1 Einleitung

Neuronale Faltungsnetze (engl. *Convolutional Neural Network*, kurz: CNN) stellen ein heutzutage viel genutztes Verfahren zur Klassifikation von Bilddaten dar. Seit dem Durchbruch von AlexNet [1] beim ImageNet Wettbewerb 2012 wurden erhebliche Verbesserungen durch immer tiefere Netze in der Klassifikation von komplexen Szenen erreicht.

Die Suche nach einer geeigneten Netztopologie stellt dabei eine Herausforderung dar. Einerseits wird eine hohe Testgenauigkeit verlangt und andererseits sind kleine CNNs vorteilhaft für die Implementierung. In den 1990ern und 2000ern wurden relativ langsame Verfahren entworfen, die nach den optimalen Verbindungen und Gewichten zwischen Neuronen mittels Heuristiken suchen [2–4]. Moderne Verfahren setzen auf das aufwändigere Reinforcement-Learning [5], um tiefe CNNs für komplexere Anwendungen zu finden. Jedoch werden bei diesem Ansatz enorm hohe Rechendauern benötigt, da sehr viele CNNs hierfür mehrere Epochen trainiert werden.

Der Einsatz von CNNs in industriellen Anwendungen spielt bisher eine untergeordnete Rolle. Es werden immer noch meist klassische Methoden verwendet, obwohl der Einsatz von CNNs in ähnlichen Anwendungen bereits erfolgreich untersucht wurde.

Für die Oberflächeninspektion reichen kleine CNNs bereits aus. So wird in [6] ein Verfahren beschrieben, bei dem ein vortrainiertes CNN

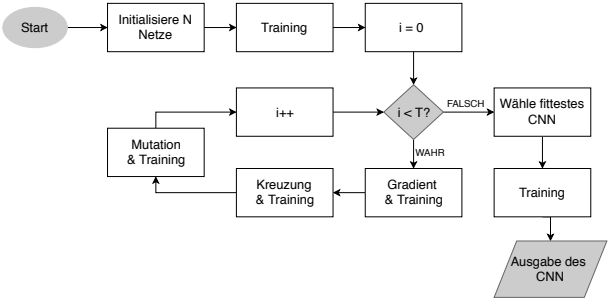

Abbildung 1: Ablauf der Metaheuristik.

mit fünf Schichten für die Inspection von Holz, Lötstellen und Stahl verwendet wird. In [7] werden Defekte wie Verschmutzung, Kratzer und Grate ebenfalls mit CNNs inspiziert. Dabei wird die Topologie durch manuelles Variieren von Neuronen und Schichten bestimmt. In [8] wird die Topologie eines CNN durch die beiden Parameter *Tiefe* und *Breite* beschrieben. Anschließend wird der Einfluss der beiden Hyperparameter auf die Genauigkeit bei der Defektdetektion auf einem künstlichen Datensatz untersucht.

Jedoch fehlt in diesen Beiträgen ein Automatismus für das Auffinden einer geeigneten Netztopologie. In diesem Beitrag wird ein metaheuristisches Verfahren vorgestellt, das aus einem gegebenen Datensatz ein neuronales Netz automatisch entwirft und trainiert. Der Beitrag ist folgendermaßen aufgebaut: In Kapitel 2 wird der Ansatz des Verfahrens vorgestellt. Kapitel 3 thematisiert den Testaufbau und die verwendeten Datensätze. Anschließend werden die Ergebnisse in Kapitel 4 und das Fazit in Kapitel 5 behandelt.

2 Methodik

Der Ansatz basiert auf differentieller Evolution [9], welcher nach [10] im Allgemeinen mindestens genauso gute Ergebnisse liefert wie eine Partikelschwarmoptimierung oder andere genetische Evolutionsstrategien. Ein CNN wird grundsätzlich durch die Anzahl und Art seiner Schichten, die zugehörigen Parameter und die Gewichte sowie Biases

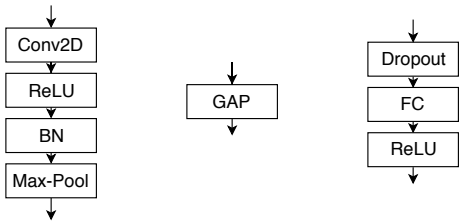

Abbildung 2: Die Abbildung zeigt die drei Bausteine des hier verwendeten CNN. Links sind der Block mit einer Faltungsschicht, in der Mitte die GAP-Schicht und rechts der Block um die FC-Schicht dargestellt. Die letzte FC-Schicht besitzt anstelle einer ReLU eine Softmax-Aktivierung.

beschrieben. Da die Anzahl und Art der Schichten variiert, variiert auch die Anzahl der Parameter, die das CNN beschreiben. Daher wird ein zweistufiger Ansatz gewählt, der die Schichten und seine Parameter getrennt innerhalb eines Iterationsschrittes bezüglich einer Fitnessfunktion optimiert. Der Algorithmus besteht aus eine Initialisierung, T Iterationsschritten und im Training des ausgewählten CNN am Ende. Der Ablauf ist in Abb. 1 illustriert.

In diesem Kapitel wird zunächst die allgemeine Topologie sowie die Initialisierung des vorgestellten Algorithmus dargestellt. Anschließend werden die Fitnessfunktion und dann die Rechenschritte während der Iteration vorgestellt.

2.1 Allgemeine Topologie und Initialisierung

Um die Konvergenz des Algorithmus zu beschleunigen, muss der Suchraum klein gehalten werden. Dies führt zu Einschränkungen in der Topologie des Netzes: Es werden lediglich drei verschiedene Blöcke für die Topologie verwendet. Im Kern eines Blocks steht eine vollständig verbundene Schichten (engl. *fully connected layer*, kurz: FC), eine 2D-Faltungsschichten (engl. *convolutional layer*, kurz: Conv2D) oder eine globale Mittelungsschicht (engl. *global averaging pooling layer*, kurz: GAP). Faltungsschichten werden um eine Bündel-Normalisierungsschicht (engl. *batch normalization layer*, kurz: BN) [11], eine Maximum-Pooling-Schicht (kurz; Max-Pool) und um eine ReLU-

Aktivierungsfunktion ergänzt. Der FC-Baustein besteht aus jeweils einer Ausfallschicht (engl. *dropout layer*), einer FC-Schicht und einer ReLU. Diese drei Blöcke sind in Abb. 2 abgebildet. Beim Aufbau werden zunächst die Faltungsblöcke, deren Anzahl auf maximal 10 beschränkt ist, hintereinander geschaltet, bevor die GAP-Schicht den Ausgabetensor in einen Vektor formt. Anschließend werden maximal vier FC-Blöcke hinzugefügt.

Für die Optimierung können in FC-Schichten die Anzahl der Neuronen und in den Faltungsbausteinen die Anzahl der Kernel, die Schrittweite der Filter und das Maximum-Pooling verändert werden. Die Filterkernelgröße (3×3) und andere Parameter sind hingegen fest. Ferner wird eine L_2-Norm der Gewichte als zusätzliche Regularisierung verwendet.

Bei der Initialisierung werden zufällig $\frac{N}{2}$ Topologien aus einer Liste mit Initialtopologien ausgewählt. Anschließend werden die CNNs mit dem Datensatz b_0 Epochen trainiert, die Gewichte gespeichert und die CNNs dupliziert, sodass die Population aus N Individuen besteht.

2.2 Fitnessfunktion

Das Ziel der Metaheuristik ist es, die Fitnessfunktion zu optimieren. Wie zuvor beschrieben bildet die Fitnessfunktion die Validierungsgenauigkeit und den Ressourcenbedarf einer Topologie ab. Dieser setzt sich aus dem benötigten Speicher s für Gewichte, Biases und Aktivierungen und den MAC-Operationen m der Inferenz zusammen.

Empirisch wurde als Fitnessfunktion die Funktion $F(a, \eta, m, s)$ mit

$$F(a, \eta, m, s) = \frac{1}{1 - a + \epsilon} + C \cdot a + \frac{1}{\eta + \epsilon} - \frac{m}{M} \quad \text{für } s \leq s_{\text{max}} \qquad (1)$$

ermittelt. Hierbei stellt a die Klassifikationsgenauigkeit des Validierungsdatensatzes, η die Kreuzentropie, M sowie C Skalierungsfaktoren und s_{max} den maximal zur Verfügung stehenden Speicherplatz dar. ϵ dient zur numerischen Stabilität. Die Fitness von CNNs mit $s > s_{\text{max}}$ wird formal zu $F(s > s_{\text{max}}) = \infty$ definiert, was durch den begrenzten Speicherplatz auf mobilen Geräten oder FPGAs motiviert wird.

Die einzelnen Terme der Fitnessfunktion haben die Aufgabe zu verhindern, dass die gesamte Fitness nur durch Verkleinerung des Netzes,

d. h. durch Reduktion von s und m, erhöht wird, während die Validierungsgenauigkeit bei niedrigen Werten verharrt. Mit der inversen Validierungsgenauigkeit wird erreicht, dass bei hohen absoluten Validierungsgenauigkeiten noch ein Anstieg in der Fitness verzeichnet werden kann. Bei geringer Genauigkeit sorgt hierfür der lineare Term $C \cdot a$. Falls die Klassen stark unterschiedlich viele Objekte beinhalten, dauert es u. U. mehrere Trainingsdurchläufe, bis sich die Genauigkeit erhöht. Daher wird ebenfalls die inverse Kreuzentropie η verwendet.

2.3 Iteration

Der Übergang einer Population $\mathfrak{P}_t$ in die nächste Generation $\mathfrak{P}_{t+1}$ besteht drei Schritten: Im ersten Schritt wird innerhalb der Population ein Gradient der Fitness bezüglich der Schichten gebildet. Danach folgt eine Kreuzung innerhalb der CNNs mit gleichen Schichten. Der dritte Schritt ist eine zufällige Mutation, um die Diversität der Population zu erhöhen. Nach jedem der drei Schritte wird das sich neu ergebende CNN für b_t Epochen trainiert, um die Gewichte an die Veränderung anzupassen und die Klassifizierungsgenauigkeit zu erhöhen. Die Gewichte werden während des Prozesses für jede Topologie gespeichert und vor dem Training in das veränderte CNNs geladen, damit der erreichte Fortschritt erhalten bleibt.

Die Gewichte neuer Schichten werden zufällig initialisiert, während die Gewichte entfernter Schichten ebenfalls gelöscht werden. Falls sich die Anzahl an Neuronen bzw. an Filterkerneln ändert, werden aus der Gewichtsmatrix zufällig Zeilen entfernt bzw. Zeilen mit zufälligen Einträgen hinzugefügt. Darüber hinaus ist es nötig, in der darauf folgenden Schicht sowohl die Anzahl der Spalten der Gewichtsmatrix als auch den Bias-Vektor entsprechend anzupassen. Die Anzahl der Neuronen in der letzten FC-Schicht kann nicht verändert werden.

Individuen mit der gleichen Anzahl an Faltungsschichten c und FC-Schichten d innerhalb der Population $\mathfrak{P}_t$ werden zur Gattung $\mathfrak{G}_t^{(c,d)}$ zusammengefasst.

Im ersten Schritt wird für jede Gattung $\mathfrak{G}_t^{(c,d)}$ ein Individuum i ausgewählt und der Gradient $\vec{g}_i$ seiner Fitness $F(i)$ bezüglich der Anzahl der Schichten gebildet. Als Referenz wird eine zufällig ausgewählte Gruppe $\tilde{\mathfrak{P}}_t$ von $\tilde{N}$ Individuen aus $\mathfrak{P}_t$ herangezogen. Der Gradient wird

gewählt zu:

$$\vec{g_i} = \frac{1}{\tilde{N}} \sum_{j \in \tilde{\mathfrak{P}}_t} (F(i) - F(j)) \cdot \left(\begin{bmatrix} c_i - c_j \\ d_i - d_j \end{bmatrix} \right). \tag{2}$$

Anschließend wird das Induviduum i angepasst:

$$\begin{bmatrix} c_i \\ d_i \end{bmatrix} \leftarrow \text{srd} \left(\begin{bmatrix} c_i \\ d_i \end{bmatrix} + r \cdot \vec{g_i} \right), \tag{3}$$

wobei $\text{srd}(\cdot)$ die stochastische Rundungsoperation meint und $r \sim U(0, \frac{1}{2})$ eine gleichverteilte Zufallsvariable ist.

Im zweiten Schritt werden jeweils zwei Individuen aus der gleichen Gattung $\mathfrak{G}_t^{(c,d)}$ – falls vorhanden – durch Roulette-Wheel-Selektion [12] ausgewählt und gekreuzt. Die Anzahl an Filterkerneln und Neuronen des Nachkommen wird entsprechend den Gleichungen 2 und 3 berechnet. Die Schrittweiten von Faltungen und von Max-Pooling werden durch eine Zwei-Punkte-Kreuzung bestimmt, d. h. die Schrittweiten werden abschnittsweise von den Eltern übernommen. Der Nachkomme ersetzt das am wenigesten fitte Individuum aus seiner Gattung, wenn seine Fitness nach den b_t Trainingsepochen höher ist.

Eine zufällige Mutation stellt den letzten Schritt dar. Hierbei werden die besten zwei Individuen von $\mathfrak{P}_t$ ausgewählt. Anschließend werden die optimierbaren Werte zufällig verändert. Die so generierten Nachkommen ersetzen die schlechtesten Individuen der Population.

3 Versuchsaufbau

Die vorgestellte Metaheuristik wird an den Datensätzen, die in diesem Kapitel vorgestellt werden, fünfmal evaluiert. Der Algorithmus wird jeweils mit $\frac{N}{2} = 15$ Individuen initialisiert, die $b_0 = 8$ Epochen trainiert werden. Die Parameter für die Fitnessfunktion aus Gl. 1 sind $C = 10$, $M = 10^6$ und $s_{\text{max}} = 27$ MB in Anlehnung an die verfügbaren Ressourcen moderner FPGAs. Ferner ist $\tilde{N} = \left\lfloor \frac{N}{3} \right\rfloor$. Die Bündelgröße (engl. *batch size*) wird zu 64 gesetzt. Um die Konvergenz zu beschleunigen, werden am Ende jeder der $T = 10$ Iterationen die beiden schlechtesten Individuen aus der Population entfernt (vgl. [10]) und die Anzahl der Trainingsdurchläufe linear von $b_1 = 4$ zu Beginn auf $b_{10} = 6$ erhöht.

Der NEU-Oberflächendefekt-Datensatz [13] besteht aus sechs Klassen von Oberflächendefekten auf Stahl mit jeweils 300 Grauwertbildern der Größe 200×200 Pixeln. Die Bilder werden auf 192×192 Bildpunkte interpoliert und es wird ein Testdatensatz von 180 und ein Validierungsdatensatz von 360 Bildern angelegt. Die restlichen 1.260 Bilder werden durch Rotation, Translation, Rauschen und Variation in der Beleuchtung um den Faktor 4 augmentiert und als Trainingsdatensatz verwendet.

Im RSDDs-Datensatz [14] sind Aufnahmen von Defekten auf Eisenbahnschienen enthalten. Die 128 verwendeten Bilder haben eine Größe von 55×1250 Pixeln. Für die Zwecke der Defekterkennung werden Ausschnitte von 55×55 Pixeln aus den Bildern extrahiert und entsprechend der Ground-Truth klassifiziert. Damit ergibt sich ein Trainingsdatensatz von 2.342, ein Validierungsdatensatz von 563 und ein Testdatensatz von 302 Bildausschnitten, von denen jeweils ca. 12 % einen Defekt enthalten.

Die optische Erkennung von Defekten an Löt- und Schweißstellen ist die Klassifikationsaufgabe des Welds-Datensatzes [15], welcher sowohl Röntgenbilder der Lötstellen als auch die Ground-Truth der Defekte enthält. Ähnlich wie beim RSDDs-Datensatz werden insgesamt etwa 9.000 Ausschnitte der Größe 64×64 aus den Röntgenbildern als Grauwertbild extrahiert und klassifiziert, bevor sie als Datensatz benutzt werden können.

Der MNIST-Datensatz [16] besteht aus 60.000 Trainings- und 10.000 Testbildern von handgeschriebenen Ziffern. Der Trainingsdatensatz wird unterteilt in 47.000 Bilder zum Trainieren und 13.000 Bilder für die Validierung. Jedes Bild wird von 28×28 auf 32×32 hochskaliert.

Als weiterer Datensatz wird der Fashion-MNIST [17] verwendet. Dieser stellt zwar keine industrielle Anwendung dar, gilt aber als ein Benchmarkdatensatz und ist mit industriellen Datensätzen vergleichbar. Der Datensatz zeigt Grauwertbilder von zehn verschiedenen Klassen von Kleidung. Der Datensatz enthält 70.000 Bilder der Größe 28×28, die identisch wie der MNIST-Datensatz aufgeteilt und hochskaliert werden.

Zuletzt wird noch ein künstlicher Datensatz der DAGM untersucht. Dieser ist der fünfte Datensatz aus dem Wettbewerb für die optische Inspektion aus dem Jahr 2007 [18]. Es stehen 1.150 Grauwertbilder einer synthetischen Textur zu Verfügung, wobei 150 Bilder einen Defekt enthalten. Die Größe der Bilder beträgt 512×512 Pixel.

4 Ergebnisse

Datensatz	NEU	RSDDs	Welds	MNIST	Fashion	DAGM
Max. Genauigkeit	>0,999	0,989	0,974	0,994	0,923	>0,999
Mittlere Genauigkeit	0,997	0,985	0,966	0,992	0,913	0,997
Mittlere Laufzeit	251 min	240 min	159 min	151 min	176 min	142 min
Anzahl Parameter	12.868	26.561	25.266	133.980	134.162	9.724
MAC-Operationen	2,87 Mio.	0,78 Mio	2,19 Mio.	0,72 Mio.	1,01 Mio.	0,86 Mio.

Tabelle 1: Zusammenfassung der Ergebnisse des Verfahrens für die sechs Datensätze zusammen. Aus den fünf Durchläufen sind der minimale und der mittlere Klassifizierungsfehler bzgl. des Testdatensatzes, die mittlere Laufzeit und die im Mittel benötigten Ressourcen dargestellt.

Der vorgestellte Algorithmus wurde in Tensorflow (Python) implementiert. Das Training wurde auf einer *GeForce 1080 Ti* durchgeführt.

Die Ergebnisse sind in Tabelle 1 zusammengefasst und der Verlauf der Median-Genauigkeit in Abb. 3. Auffällig hierbei ist, dass die Median-Fitness beim NEU- und beim DAGM-Datensatz stärker steigt als bei den anderen und dass CNNs für Datensätze mit größeren Bildern tendenziell mehr MAC-Operationen und weniger Parameter benötigen, da die zugehörige Inferenz einen höheren Ressourcenverbrauch aufweist. Die Fitness von Welds und Fashion-MNIST steigt nur sehr langsam, da die erreichbare Genauigkeit deutlich geringer ist (s. Tab. 1).

Beim NEU-Datensatz werden im Median Genauigkeiten von 99,7 % erreichbar, bei relativ hohen Laufzeiten der Heuristik von im Mittel 251 min. Die Anzahl der MAC-Operationen variiert zwischen 1,7 Mio. und 4,6 Mio und die der Parameter zwischen 7.158 und 57.034, wobei der letzte Wert einen Ausreißer darstellt. Das korrespondierende CNN erreicht als einziges nicht eine Genauigkeit von 1.

Beim RSDD-Datensatz werden Genauigkeiten von 98 % bis 99 % erreicht. Die resultierenden Netze haben 5 bis 6 Faltungs- und 1-2 FC-Schichten. Der Welds-Datensatz verhält sich ähnlich, erreicht aber eine etwas geringere Testgenauigkeit.

Mit 99,38 % Testgenauigkeit erzielt das beste CNN bzgl. des MNIST-Datensatz eine Genauigkeit, die sich im Bereich des Standes der Technik bewegt. Das CNN besitzt acht Faltungs- und zwei FC-Schichten, hat 133.980 Parameter und die Inferenz benötigt ca. 0,72 Mio. MAC-

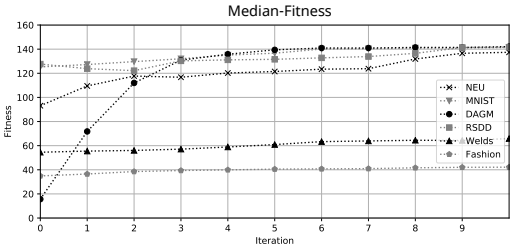

Abbildung 3: Verlauf der mittleren Median-Fitness bzgl. der fünf Durchläufe für jeden der sechs Datensätze.

Operationen. Damit stellt dieses Netz einen Ausreißer bzgl. der benötigten Ressourcen und der Genauigkeit dar. Die übrigen Netze besitzen lediglich 4 oder 5 Faltungsschichten.

Ebenso stellen Topologien, die für den Fashion-MNIST-Datensatz Genauigkeiten von über 91 % erreichen, einen Erfolg dar. Alle gefundenen Netz haben 3-6 Faltungs und eine FC-Schicht.

Die Defekterkennung im DAGM-Datensatz ist eine vergleichsweise einfache Herausforderung, bei der weniger Parameter notwendig sind (s. Tabelle 1). Nach Abb. 3 steigt die mediane Fitness stark und kontinuierlich an, während die mittlere Maximalfitness stufenförmig ansteigt. Dies liegt daran, dass der zu detektierende Fehler ein rotierter und verschobener Patch ist, dessen jeweilige Erscheinungsform sukzessive durch den Klassifikator erlernt wird.

Für ein weiteres Experiment werden die besten CNNs extrahiert, die Gewichte neu initialisiert und anschließend 100 Epochen trainiert. Für kein CNN kann die Genauigkeit aus der Metaheuristik erreicht werden. Beispielsweise erreicht das beste CNN des Fashion-MNIST Datensatzes zwar eine Trainingsgenauigkeit von über 99,95 %, aber in der Validierung und beim Testen können keine Genauigkeiten von über 89 % erreicht werden. In diesem Fall neigt die Topologie bei einer Neuinitialisierung zu starker Überanpassung gegenüber dem Resultat aus der Heuristik. Dies zeigt, dass die Metaheuristik nicht nur nach einer Topologie sucht, sondern implizit auch nach einer geeigneten Initialisierung der Gewichte (vgl. [19]), da durch die evolutionären Operationen das jeweilige *basin of attraction* aufgrund des Hinzufügens bzw. Entfernens

von Gewichten verlasssen wird.

5 Fazit

In diesem Beitrag wird eine neue Metaheuristik, die auf differentieller Evolution beruht, für die Optimierung einer CNN-Topologie vorgestellt. Für relativ einfache Datensätze, wie sie in industriellen Anwendungen oft vorliegen, können Topologien mit hohen Genauigkeiten, die nahe am Stand der Technik liegen, mit relativ kurzer Rechenzeit gefunden werden. Desweiteren garantiert diese Methode einen geringen Ressourcenbedarf, sodass das Ergebnis auf mobilen Geräten oder FPGAs implementiert werden kann.

Für komplexere Aufgaben muss zwischen zusätzlicher Rechendauer und langsamerer Konvergenz abgewogen werden. Beispielsweise kann die Anzahl der Trainingsepochen, die Größe der Population oder die Anzahl an Iterationen erhöht werden. Ferner kann durch Verwenden vortrainierter Netze oder Transferlernen Rechenzeit eingespart werden.

Literatur

1. A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.
2. K. O. Stanley and R. Miikkulainen, "Evolving Neural Networks through Augmenting Topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002.
3. F. H.-F. Leung, H.-K. Lam, S.-H. Ling, and P. K.-S. Tam, "Tuning of the Structure and Parameters of a Neural Network Using an Improved Genetic Algorithm," *IEEE Transactions on Neural Networks*, vol. 14, no. 1, pp. 79–88, 2003.
4. D. Floreano, P. Dürr, and C. Mattiussi, "Neuroevolution: from Architectures to Learning," *Evolutionary Intelligence*, vol. 1, no. 1, pp. 47–62, Mar 2008. [Online]. Available: https://doi.org/10.1007/s12065-007-0002-4
5. B. Zoph and Q. V. Le, "Neural Architecture Search with Reinforcement Learning," *arXiv preprint arXiv:1611.01578*, 2016.
6. R. Ren, T. Hung, and K. C. Tan, "A Generic Deep-Learning-based Approach for Automated Surface Inspection," *IEEE Transactions on Cybernetics*, 2017.

7. J.-K. Park, B.-K. Kwon, J.-H. Park, and D.-J. Kang, "Machine Learning-based Imaging System for Surface Defect Inspection," *International Journal of Precision Engineering and Manufacturing-Green Technology*, vol. 3, no. 3, pp. 303–310, 2016.
8. D. Weimer, B. Scholz-Reiter, and M. Shpitalni, "Design of Deep Convolutional Neural Network Architectures for Automated Feature Extraction in Industrial Inspection," *CIRP Annals*, vol. 65, no. 1, pp. 417–420, 2016.
9. R. Kuo and F. E. Zulvia, "The Gradient Evolution Algorithm: A new Metaheuristic," *Information Sciences*, vol. 316, pp. 246–265, 2015.
10. K.-L. Du and M. Swamy, *Search and Optimization by Metaheuristics*. Springer, 2016.
11. S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in *International Conference on Machine Learning*, 2015, pp. 448–456.
12. J. H. Holland, *Adaptation in Natural and Artificial Systems: an Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. MIT press, 1992.
13. K. Song, S. Hu, and Y. Yan, "Automatic Recognition of Surface Defects on Hot-Rolled Steel Strip using Scattering Convolution Network," *Journal of Computational Information Systems*, vol. 10, no. 7, pp. 3049–3055, 2014.
14. J. Gan, Q. Li, J. Wang, and H. Yu, "A Hierarchical Extractor-Based Visual Rail Surface Inspection System," *IEEE Sensors Journal*, vol. 17, pp. 7935–7944, 2017.
15. D. Mery, V. Riffo, U. Zscherpel, G. Mondragón, I. Lillo, I. Zuccar, H. Lobel, and M. Carrasco, "GDXray: The Database of X-ray Images for Nondestructive Testing," *Journal of Nondestructive Evaluation*, vol. 34, no. 4, p. 42, 2015.
16. Y. LeCun, C. Cortes, and C. Burges, "MNIST Handwritten Digit Database," *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, vol. 2, 2010.
17. H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms," *arXiv preprint arXiv:1708.07747*, 2017.
18. Deutsche Arbeitsgemeinschaft für Mustererkennung. (2007) Weakly Supervised Learning for Industrial Optical Inspection. [Online]. Available: https://resources.mpi-inf.mpg.de/conference/dagm/2007/prizes.html#industry
19. D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, "Why Does Unsupervised Pre-Training Help Deep Learning?" *Journal of Machine Learning Research*, vol. 11, no. Feb, pp. 625–660, 2010.