

# Bachelor-Thesis

## Forensische Herausforderungen bei der Erkennung generativer Deepfake-Bilder

**Eingereicht am:** 27. 02. 2024  
**von:** Thomas Reimann  
**Matrikelnummer:** XXXXXX

- Betreuer:** Herr Prof. Dr. Matthias Kreuseler
- Betreuer:** Frau Prof. Dr. Antje Raab-Düsterhöft

---

## **Aufgabenstellung**

Ziel der Bachelor-Thesis ist die Untersuchung und Bewertung eines auf Deep Learning basierenden Ansatzes zur Erkennung von generativen Deepfake-Bildern. Die Erkennung ist dabei auf personendarstellende Bilder beschränkt, wobei als Erkennungsmerkmal Hände verwendet werden.

Dabei ist zu untersuchen, ob binär klassifizierende Deep Learning Modelle dazu geeignet sind, ausschließlich anhand der Darstellung der Hände zu bestimmen, ob ein Bild echt oder synthetisch ist.

Experimentiert wird mit Diffusions-synthetisierten Deepfake-Bildern. Dabei erfolgt das Model-Training mit den zuvor extrahierten Händen aus einer Menge an Trainingsbildern mit jeweils gleichen Anteilen an realen und gefälschten personendarstellenden Bildern.

---

## **Kurzreferat**

Falsch- und Desinformationen werden in sozialen Netzwerken zunehmend mit Hilfe von synthetisch generierten fotorealistischen Deepfake-Bildern verbreitet.

Diese Arbeit untersucht die Möglichkeit, personendarstellende Deepfake-Bilder über Deep Learning Modelle zu detektieren.

Als Erkennungsmerkmal wurden Hände verwendet. Dabei wurden die Modelle mit den Hand-darstellenden Bildausschnitten von echten und gefälschten personendarstellenden Fotos trainiert. Es wurde ein Model von Grund auf neu (From-Scratch Learning) und ein weiteres auf Basis eines existierenden Hand-erkennenden Models (Transfer Learning) trainiert.

Es zeigte sich, dass das auf Transfer Learning basierende Modell echte und gefälschte Bilder zu über 80% korrekt als fake oder real klassifizieren konnte.

## **Abstract**

False and disinformation are increasingly being spread on social networks using synthetically generated photorealistic deepfake images.

This work explores the possibility of detecting deepfake images depicting persons through deep learning models.

Hands were used as the detection feature. The models were trained with image sections depicting hands from real and fake photos of individuals. One model was trained from scratch (From-Scratch Learning) and another was trained based on an existing hand-detecting model (Transfer Learning).

It was found that the model based on Transfer Learning could correctly classify real and synthetic images as fake or real with an accuracy of over 80%.

---

# Inhalt

1. Einleitung .....	3
2. Selbstlernende Systeme und KI.....	5
2.1 Machine Learning.....	5
2.2 Deep Learning .....	6
2.3 Deep Learning: Convolutional Neural Networks .....	9
3. Deep Learning und Deepfake .....	12
3.1 Generierung von Deepfakes durch Deep Learning Systeme .....	12
3.1.1 Generative Adversarial Networks .....	13
3.1.2 Diffusionsmodelle .....	14
3.2 Text-to-Image Bilder .....	16
4. Deepfake-Forensik .....	17
4.1 Gesellschaftliche Aspekte der Verbreitung von Deepfake-Bildern .....	17
4.2 Lösungen und Ansätze automatisierter Erkennung.....	20
4.2.1 Generalistische Ansätze .....	21
4.2.1.1 Analyse von Rausch-Charakteristik in GAN-basierte Deepfake-Bilder .....	21
4.2.1.2 Identifikation von Rekonstruktionsfehlern bei Diffusions-basierten Deepfake-Bildern.....	23
4.2.1.3 Frequenzraumanalyse bei GAN-basierten Deepfake-Bildern.....	24
4.2.2 Merkmalsbasierte Ansätze .....	26
4.2.2.1 Detektion von manipulierten Gesichtsausdrücken.....	26
4.2.2.2 Detektion von unnatürlichen Lichtreflektion in Augen .....	28
4.3 Fazit zur Deep Learning-basierten Deepfake Detektion .....	29

---

5. Deep Learning Experiment: Hände als Biomarker zur Erkennung von Diffusionsbasierten Deepfake-Bildern .....	31
5.1 Motivation und Ziel .....	31
5.2 Datenbeschaffung und Vorverarbeitung .....	34
5.3 Modellaufbau .....	41
5.3.1 Bibliotheken, Entwicklungsumgebung und Hardware .....	41
5.3.2 Implementierung und Parametrisierung des Modells .....	42
5.4 Auswertung der Ergebnisse .....	51
6. Fazit .....	56
7. Ausblick .....	57
Literaturverzeichnis .....	58
Abkürzungsverzeichnis .....	64
Abbildungsverzeichnis.....	65
Selbstständigkeitserklärung.....	66
Anhang .....	67

# 1. Einleitung

Spätestens seit der Veröffentlichung von ChatGPT im November 2022 ist generative künstliche Intelligenz auf dem Weg, fester Bestandteil der Internetnutzung zu werden. Einige sprechen sogar von einer neuen Ära des Internets (z. B. Plank 2023). ChatGPT steht für „Chat Generative Pre-trained Transformer“ und ist ein Chatbot, der mit Hilfe von künstlicher Intelligenz (KI) in der Lage ist, auf Fragen oder Aufforderungen passende Antworten zu liefern. Er generiert aus einer gigantischen Menge von Daten präzise und strukturierte Inhalte aus Texten und Bildern. Die dabei generierten Inhalte und Konversationen sind dabei häufig in ihrer Qualität denen von Menschen gleichwertig (vgl. Eckert 2023).

Einige Zeit später, Anfang 2023, wurde die breite Öffentlichkeit mit einer weiteren auf generative KI basierten Technologie überrascht. Die Rede ist von vollsynthetisch generierten Bildern basierend auf einfache Textvorgaben. Einige dieser Bilder enthielten fotorealistische Darstellungen von prominenten Personen und sind auf dem ersten Blick nicht von echten Bildern zu unterscheiden. So wurde zum Beispiel der Papst in einer modischen weißen Daunenjacke erzeugt<sup>1</sup>, eine tumultartige Verhaftung von Donald Trump<sup>2</sup> dargestellt oder ein Bild generiert, das Vladimir Putin kniend vor dem chinesischen Präsidenten zeigt<sup>3</sup>. Zwar gab es vorher schon ähnlich realistisch aussehende Bildfälschungen, jedoch mussten diese aufwendig über spezielle Bildverarbeitungsprogramme und durch Experten erstellt werden. Mit Hilfe von generativer KI können sich Internetnutzer nun beliebige Bilder einfach über kurze Anweisungen generieren lassen. Sie müssen dafür lediglich ein paar Worte eingeben, die grob beschreiben, was auf dem Bild zu sehen sein soll. Innerhalb weniger Minuten ist das Bild generiert und landet oft genauso schnell in den sozialen Netzwerken.

Es liegt auf der Hand, dass diese Art von Fälschungen potenziell der Verbreitung von Falsch- und Desinformation eine neue Dimension verleihen. Dementsprechend

---

<sup>1</sup> <https://www.nau.ch/news/digital/bild-von-papst-franziskus-im-daunenjacken-look-geht-viral-66461685>

<sup>2</sup> <https://www.nn.de/politik/wenn-die-kunstliche-intelligenz-plotzlich-donald-trump-verhaftet-1.13183159>

<sup>3</sup> <https://www.dw.com/en/fact-check-no-putin-did-not-kneel-before-xi-jinping/a-65099092>

wichtig ist es, solche über KI gefälschte Bilder (Deepfake Bilder) zuverlässig automatisiert zu erkennen.

Das Ziel der vorliegenden Arbeit ist die Entwicklung eines experimentellen Deep Learning Models, welches anhand der Darstellung menschlicher Hände generative Deepfake-Bilder detektiert. Es kann dazu verwendet werden, um Bilder, welche Personen mit Gesten und Handlungen zeigen, als gefälscht oder echt zu klassifizieren.

Zunächst erfolgt eine allgemeine Heranführung an die Themen Deep Learning und Deepfakes. Der Schwerpunkt wird verstärkt auf Bilder liegen. Insbesondere liegen hier generative Bildsynthese-Modelle im Fokus. Es folgt eine kurze gesellschaftliche und forensische Einordnung der Verbreitung von Deepfakes und welche KI-gestützten Ansätze es gibt, um Deepfake-Bilder zu detektieren. Anschließend erfolgt die Dokumentation der Vorgehensweise des Experiments mit abschließender Bewertung, Fazit und Ausblick.

## 2. Selbstlernende Systeme und KI

Durch die zunehmend dominante Präsenz des Begriffs „künstliche Intelligenz“ im menschlichen Alltag erfährt der Begriff eine gewisse Abnutzung und führt zu der allgemeinen Annahme, dass eine Art Supersoft oder -hardware existiert, die menschenähnlich handelt oder Probleme löst (vgl. z. B. Pietras 2020). Gleichzeitig wird der Begriff insbesondere durch Marketingmaßnahmen stark „verwässert“ (ebd.). Ein realistischer, aber weniger attraktiv klingender Begriff wäre beispielsweise „selbstlernende Systeme“, denn die Intelligenz und die beeindruckenden Ergebnisse von ChatGPT und Co. basieren im Wesentlichen darauf, dass Algorithmen verwendet werden, die selbstständig in riesigen Datenbeständen Muster und Zusammenhänge erkennen und diese benutzen, um Probleme zu lösen, Vorhersagen zu treffen oder multimediale Inhalte zu generieren. Dabei handelt es sich nicht um statische („hardcoded“) Software-Programme, welche anhand einprogrammierter Regeln versuchen, menschliche Denkprozesse nachzubilden (Mester 2023).

### 2.1 Machine Learning

Machine Learning (ML) bedeutet einfach gesagt, dass Computersysteme in der Lage sind, selbstständig aus Daten zu lernen und dabei ihre Performance stetig verbessern. Eine präzisere Definition lautet:

*“A computer program is said to learn from experience  $E$  with respect to some class of tasks  $T$  and performance measure  $P$ , if its performance at tasks in  $T$ , as measured by  $P$ , improves with experience  $E$ ”* (Mitchell 1997, zitiert nach Géron 2019: S. 4).

Machine Learning Computersysteme benötigen dafür eine große Menge an Daten. Andernfalls würden für die meisten Anwendungen keine brauchbaren Ergebnisse erzielt werden. Dieser Ansatz ist als „Big Data“ bekannt und meint im Wesentlichen das (exzessive) Sammeln von Daten z. B. aus den Bereichen Internet, Mobilfunk, Soziale Netzwerk, Finanzen oder Medizin (vgl. Wuttke 2023).



Ein klassisches Beispiel für Machine Learning sind Spam-Filter. Aus einer Menge an normalen und als Spam markierten Emails lernt der Spam-Filter vorherzusagen, ob es sich bei einer eingehenden Email um Spam handelt oder nicht. In traditioneller Programmieretechnik würden hierfür die in Spam-Emails häufig vorkommenden Wörter oder Formulierungen als Marker in den Quellcode einprogrammiert und dem Detektionsalgorithmus übergeben werden (Géron 2019: S. 5). Der Algorithmus und die Marker müssten dabei stetig unter großen Aufwand manuell gepflegt und erweitert werden. Algorithmen, die auf maschinelles Lernen basieren, lernen in diesem Fall hingegen die verdächtigen Wörter und Formulierungen aus den Emails selbst (ebd.). Hierfür müssen Beispiel-Emails als Trainingsdaten zur Verfügung stehen, in denen Unterscheidungsmerkmale zur Bestimmung Spam/kein Spam maschinell erkannt werden.

## **Überwachtes und unüberwachtes Lernen**

Maschinelles Lernen lässt sich in überwachtes und unüberwachtes Lernen unterteilen. Sind die gewünschten Ergebnisse bekannt, wie im obigen Beispiel die Spam-Klassifikation, spricht man von überwachtem Lernen. Der Algorithmus identifiziert die Merkmale aus den Trainingsdaten, die für die Einordnung oder Vorhersage der Ergebnisse relevant sind (Cleve, Lämmel 2020: S. 59). Überwachtes Lernen spielt für Deep Learning eine wichtige Rolle, worauf im nächsten Kapitel eingegangen wird.

Im Gegensatz dazu sind beim unüberwachten Lernen keine Ergebnisse oder Vorhersagen bekannt. Dieser Ansatz dient hauptsächlich dazu, unsortierte Daten in Cluster zu unterteilen, also ähnliche Objekte in Gruppen abzubilden (ebd.).

## **2.2 Deep Learning**

Deep Learning (DL) ist wohl der bekannteste Ansatz im Bereich des maschinellen Lernens und hat in den letzten Jahren massiv an Bedeutung gewonnen. Das Besondere an dieser Technologie ist, dass Lernalgorithmen auf Basis von

neuronalen Abläufen im menschlichen Gehirn konzipiert und nachgebaut werden, „um so eine vergleichbare Leistungsfähigkeit zu erzielen“ (Cleve, Lämmel 2020: S. 51). Ausschlaggebend hierfür ist die Nachbildung von Neuronen. Durch die Verknüpfung von Neuronen erhält man ein neuronales Netz (ebd.). Ein Neuron ist im Gehirn die Grundeinheit für sämtliche Arten der Informationsverarbeitung. Im Gehirn gibt es hunderte Milliarden davon. Bei Deep Learning Modellen werden künstliche Neuronen erzeugt, die mit Hilfe mathematischer Funktionen Erregungswerte berechnen und an verknüpfte Neuronen weiterleiten. Dies erfolgt unter der Einbeziehung von Gewichten und Schwellwerten (ebd.: S. 52). Die Neuronen sind dabei in verschiedenen Schichten aufgebaut. Die Input-Schicht (Input Layer) nimmt die Informationen auf und übergibt sie den versteckten Schichten (Hidden Layer). In dieser findet die Auswertung der Daten statt. Am Ende stehen die Ausgabeschichten (Output Layer), die das Ergebnis repräsentieren (Tiedemann 2018).

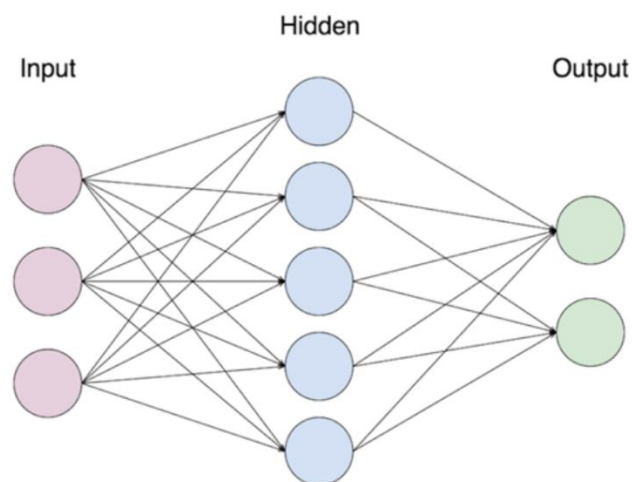


Abbildung 1: Schichtenmodell Neuronale Netze<sup>4</sup>

Mit Hilfe dieser gehirnähnlichen Architektur ist Deep Learning in der Lage selbständig komplexe Muster aus den Rohdaten zu erlernen. Dieser Vorgang erfolgt inkrementell innerhalb der Hidden Layer (Mahapatra 2018). So können zum Beispiel in Bildern anhand der Pixel Muster bestimmt werden, anhand derer wiederum

<sup>4</sup> <https://towardsdatascience.com/step-by-step-guide-to-building-your-own-neural-network-from-scratch-df64b1c5ab6e>

Gesichter erkannt werden. Dabei erfolgt zunächst eine einfache Musterbestimmung durch Kantenerkennung. Diese werden dann in den folgenden Schichten dazu benutzt, um komplexere Muster zu identifizieren, die der Erkennung von Gesichtsmerkmalen dienen. Am Ende dieses inkrementellen Prozesses stehen hoch komplexe Muster, die schließlich eine zuverlässige Gesichtserkennung ermöglichen (ebd.).

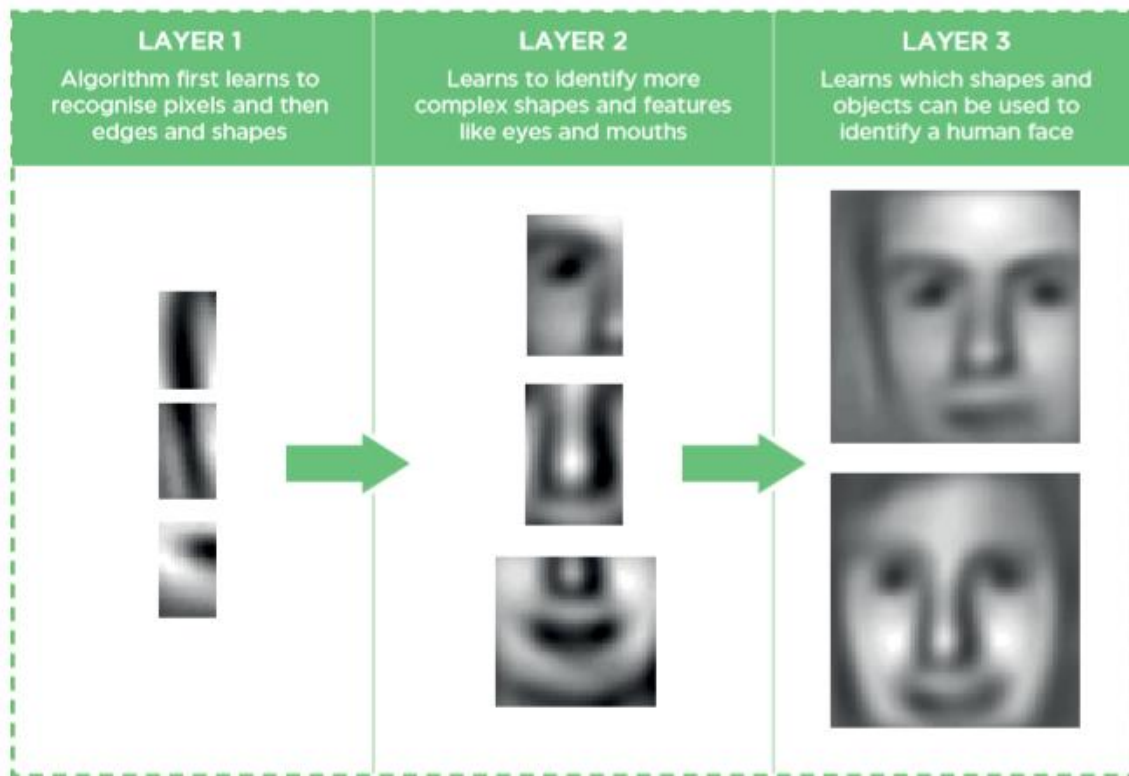


Abbildung 2: Inkrementelle Mustererkennung<sup>5</sup>

Für den Menschen sind die hochkomplexen Abläufe innerhalb der Hidden Layer eine Blackbox. Es ist häufig nicht mehr nachvollziehbar, wie das Gelernte erlernt wurde. „Statt das Gelernte fein säuberlich als Datenblock im digitalen Speicher abzulegen, verteilen sie die Informationen in einer Art und Weise, die außerordentlich schwierig zu entschlüsseln ist“ (Castelvecchi 2016). Das Wissen, was über Deep Learning generiert wird, befindet sich in dem vom Algorithmus erzeugten Modell und ist dem Menschen nicht zugänglich, wie z. B. eine Datenbank (ebd.).

<sup>5</sup> <https://towardsdatascience.com/why-deep-learning-is-needed-over-traditional-machine-learning-1b6a99177063>

Deep Learning wird über verschiedenen Architekturen neuronaler Netze realisiert. Sie unterscheiden sich je nach Anwendungsgebiet. Zu den wesentlichen gehören Feedforward Neuronal Networks, Convolutional Neural Networks, Recurrent Neural Networks und Autoencoder.

Für die vorliegende Arbeit hat die Architektur der Convolutional Neural Networks (CNN) besondere Relevanz, da ihr Anwendungsgebiet in der Bild- und Videoverarbeitung liegt. Über diese Architektur können die für Bildmotive charakteristischen Pixelmuster erlernt und für die automatische Bilderkennung verwendet werden. CNNs sind dabei auch in der Lage, weitaus subtilere Muster zu erkennen. Solche Muster können als kaum wahrnehmbare Artefakte in Bildern vorkommen und für weitere Bildklassifizierungsprozesse verwendet werden, wie z. B. die Klassifizierung eines Bildes in „real“ oder „gefälscht“.

## **2.3 Deep Learning: Convolutional Neural Networks**

Convolutional Neural Networks, zu Deutsch „Gefaltete Neurale Netzwerke“, basieren, wie der Name schon sagt, auf dem Prinzip der Faltung. Die Faltung findet in den verborgenen Schichten statt. Sie werden auch als „Convolutional Layer“ bezeichnet (Pathak 2023).

### **Das Prinzip der Faltung bei Bildverarbeitung**

Bei einer Faltung werden Filtermatrizen auf jeden Pixel eines Bildes angewendet. Innerhalb einer Filtermatrix werden sowohl anhand des jeweiligen Pixels als auch anhand der benachbarten Pixel Gewichtungen vorgenommen. Die Größe der Matrizen ist unterschiedlich. Je größer die Matrix, umso mehr Nachbarpixel fließen in die Berechnung der Gewichtungen mit ein. Somit lassen sich beispielsweise Kanten bestimmen oder Bilder weichzeichnen (Oldenburg 2006). Allgemein spricht man hier von „Feature Extraction“, zu Deutsch „Merkmalsextraktion“. Die folgende Abbildung zeigt die Kantendetektion mit Hilfe des sogenannten Sobel-Operators.

Dabei handelt es sich um eine 3x3 Filtermatrix, wobei nur die unmittelbar angrenzenden Nachbarpixel in die Berechnung miteinfließen. Vereinfacht gesagt, beruht die Kantendetektion darauf, dass die Nachbarpixel eine signifikante Helligkeitsdifferenz aufweisen und mit Hilfe von Gewichtungen sichtbar gemacht können („Sobel-Operator“ 2022).

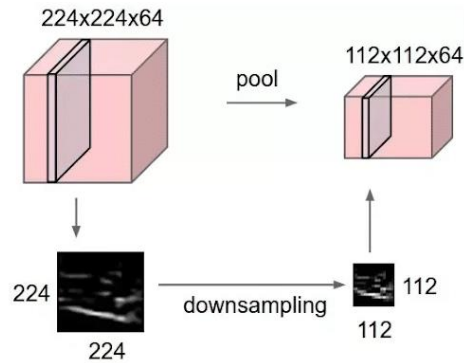


Abbildung 3: Kantendetektion mit Sobel Operator<sup>6</sup>

## Convolution und Pooling

Merkmale, die über Convolutional Layer extrahiert werden, sind zunächst an der unmittelbaren Position des Merkmals des Input-Bildes gebunden. Bei einer geringen Veränderung des Bildes (Rotation, Zuschneiden, Zoom usw.) können die auf den Merkmalen basierten Muster nicht mehr erkannt werden (Brownlee 2019). Um dieses Problem zu umgehen, erfolgt im Pooling Layer ein Downsampling-Prozess. Downsampling bedeutet im Wesentlichen, dass Informationen aus einem Bild entfernt werden, indem die Auflösung verringert wird. Dieser Prozess erfolgt über die Filtermatrizen. Von allen Pixelwerten innerhalb der Matrix wird entweder der Durchschnitt (Average Pooling) oder der Maximalwert (Max Pooling) genommen (ebd.). Dadurch werden individuelle Details des Input-Bildes entfernt und erkannte Muster lassen sich besser auf ähnliche Bilder anwenden.

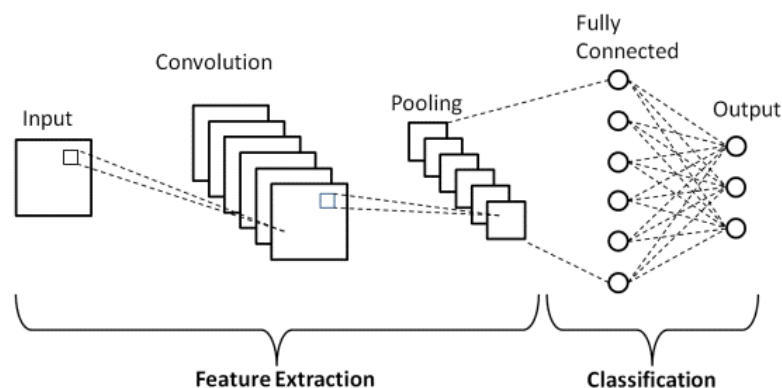
<sup>6</sup> <https://analyticsindiamag.com/different-edge-detection-techniques-with-implementation-in-opencv>

Abbildung 4: Downsampling<sup>7</sup>

Üblicherweise besteht ein CNN-Modell aus mehreren nacheinander geschalteten Convolutional und Pooling Layer mit unterschiedlich großen Filtermatrizen.

### Fully Connected Layer

Am Ende der CNN-Architektur befindet sich der „Fully Connected Layer“ (auch als „Dense Layer“ bezeichnet). Hier findet die neuronale Verknüpfung aller vorhergehender Layer statt. Dabei gehen die eingegangenen und ausgegangenen Werte aller Neuronen aus Convolutional und Pooling Layer in jedes einzelne Neuron des Fully Connected Layer ein (Stadler 2021). Letztendlich erfolgt hier der Klassifizierungsprozess.

Abbildung 5: CNN mit Fully Connected Layer<sup>8</sup>

<sup>7</sup> <https://datascientest.com/de/convolutional-neural-network-2>

<sup>8</sup> <https://www.analyticsvidhya.com/blog/2022/03/basic-introduction-to-convolutional-neural-network-in-deep-learning>

### 3. Deep Learning und Deepfake

Der Begriff „Deepfake“ setzt sich aus „Deep Learning“ und „Fake“ zusammen. Demnach werden Deepfake Inhalte mithilfe von Deep Learning Technologien erstellt. In den Begriffsdefinitionen wird fast immer von manipulierten Medieninhalten<sup>9</sup> gesprochen, die als Fälschung oder Täuschung kaum erkennbar sind. Der Begriff der Manipulation impliziert jedoch, dass z. B. echte Bilder oder Videos so verändert werden, dass sie etwas anderes zeigen. Für Deepfake-Bilder, die durch generative KI erzeugt werden, trifft die Definition häufig nicht mehr zu, da hier gänzlich neue Bilder erzeugt werden. Im Folgenden wird auf die Grundlagen eingegangen, wie Deep Learning Modelle Bilder generieren.

#### 3.1 Generierung von Deepfakes durch Deep Learning Systeme

Um Bilder fälschen zu können, benötigen Deep Learning Systeme zunächst eine gigantische Menge an Bildmaterial aus denen erlernt wird, ein neues gefälschtes Bild *vorherzusagen*. Wie bereits erwähnt erfolgt das Generieren von Informationen durch selbstlernende Systeme anhand der Auswertung von riesigen Datenbeständen. Das Generieren von Deepfake-Bildern basiert vereinfacht gesagt auf Vorhersagen, welche Werte Pixel annehmen müssen um die gewünschte Information in dem Bild so realistisch wie möglich darzustellen. Die Bildgeneratoren leisten dabei Erstaunliches: „What makes them particularly remarkable is their ability to fuse styles, concepts, and attributes to fabricate artistic and contextually relevant imagery.“ (AI Image Generation Explained: Techniques, Applications, and Limitations 2023). Das folgende gefälschte Bild zeigt eine dynamische tumultartige Szene in der Donald Trump sich unter Widerstand von mehreren Polizisten verhaften lässt. Es zeigt, dass generative KI mittlerweile in der Lage ist komplexe Szenen darzustellen, die es aus ähnlichem Bildmaterial erlernt und mit hoher Präzision zu realistisch aussehenden Bildern zusammenfügt. Dabei spielt auch KI-basierte Sprachverarbeitung eine Rolle, auf die in einem späteren Kapitel eingegangen wird.

---

<sup>9</sup> Zum Beispiel die offizielle Definition der Bundesregierung unter: <https://www.bundesregierung.de/breg-de/schwerpunkte/umgang-mit-desinformation/deep-fakes-1876736>



Abbildung 6: Trumps Verhaftung als Deepfake-Bild<sup>10</sup>

### 3.1.1 Generative Adversarial Networks

Ein Generative Adversarial Network (GAN) besteht aus zwei neuronalen Netzen, die in Konkurrenz zueinander agieren: den Generator und den Diskriminator (Luber 2021). Das neuronale Netz des Generators arbeitet dabei als Deconvolutional Neural Network (DNN). Vereinfacht kann man sagen, dass es gegensätzlich zum CNN arbeitet und Faltungsprozesse rückgängig macht<sup>11</sup> und damit neue Daten generiert (Rouse 2018). Generatoren werden verwendet, um statistische Verteilungen von realen Daten zu erlernen. Aus diesen Verteilungen werden dann neue Daten generiert, die von den realen Daten nur schwer zu unterscheiden sind (Schmidt-Colberg 2021).

Um die Qualität der generierten Daten stetig zu verbessern, kommt der Diskriminator zum Einsatz. Hierbei handelt es sich um ein CNN. Es arbeitet klassifizierend und unterscheidet, „ob ein gegebener Inhalt wie echte Daten aus

<sup>10</sup> <https://www.the-independent.com/news/world/americas/us-politics/trump-deepfake-arrest-twitter-ai-b2307470.html>

<sup>11</sup> DNN als umgekehrtes CNN zu betrachten ist eine stark vereinfachte Sichtweise. Für ein allgemeines Verständnis von GAN-Architekturen sollte dies jedoch ausreichend sein.



dem Datensatz oder wie künstlich synthetisierte Daten aussieht“ (ebd.). Die GAN-Architektur bezieht also in einer automatisierten Form das Prinzip des überwachten Lernens ein, um maximale Ergebnisse zu produzieren. Diese werden in einem Zustand erreicht, in dem Generator oder Diskriminator keinen Fortschritt mehr machen können, ohne den anderen zu verändern (ebd.).

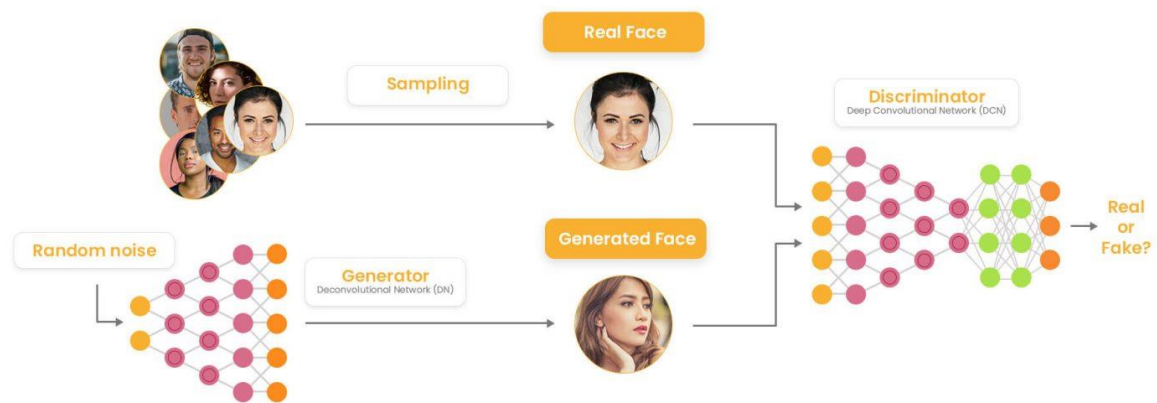


Abbildung 7: GAN-Architektur Erzeugung von Gesichtern<sup>12</sup>

### 3.1.2 Diffusionsmodelle

Die Bildsynthese über sogenannte Diffusionsmodelle basiert auf Diffusionsprozesse der Physik. Gemeint ist damit das thermodynamische Verhalten von Teilchen in Flüssigkeiten. „Wenn man etwa Tinte in ein Wasserglas tropft, ist es extrem schwierig vorauszusagen, in welchen Bereichen sich ein Tintenmolekül aufhalten wird“ (Bischoff 2023). Im Endzustand sind alle Tintenmoleküle gleichmäßig verteilt. Durch Wahrscheinlichkeitsrechnungen wird errechnet, wie der Ausgangszustand der Tintenmoleküle war. Der Ausgangszustand ist demnach der Zustand, den die Tinte unmittelbar im Moment des Eintretens im Wasser hat (vgl. ebd.).

<sup>12</sup> <https://www.clickworker.de/ki-glossar/generative-adversarial-networks>

Das Prinzip der Diffusionsprozesse lässt sich auch auf Bildsynthese anwenden. Es ergeben sich dabei folgende Analogien: der Anfangszustand (Tinte im Wasser kurz vor Beginn der Diffusion) ist das zu erlernende Bild. Der Endzustand (Tinte vollständig im Wasser verteilt) ist das zu erlernende Bild im maximal verrauschten Zustand. Diffusion bezieht sich bei Diffusionsmodellen auf Bildrauschen. Den Bildern wird schrittweise zufälliges Rauschen hinzugefügt, bis sie maximal verrauscht sind. Der Diffusionsalgorithmus lernt dabei aus den jeweiligen Schritten des Verrauschens, wie das Rauschen wieder entfernt werden kann, so dass am Ende ein neues Bild entsteht. Dabei werden beeindruckende Ergebnisse erzielt. So lassen sich nicht nur fotorealistische Bilder erzeugen, sondern auch Motive kombinieren und damit vielfältige Szenen darstellen (vgl. Linde 2023).



Abbildung 8: Erstelltes Testbild über den Dall-E Diffusionsgenerator mit folgender Szenenvorgabe: „Ein Forensiker analysiert ein Bild auf Twitter, welches ein Präsident zeigt, der eine Kapitulation unterzeichnet.“<sup>13</sup>

<sup>13</sup> Dem aufmerksamen Leser entgeht hier sicher nicht die thematische Nähe der dargestellten Szene zu der vorliegenden Arbeit.

### 3.2 Text-to-Image Bilder

Die Generierung von Bildern über GAN- oder Diffusionsmodelle erfolgt üblicherweise durch Text-Prompts. Mittlerweile stehen eine Vielzahl von Bezahl- und Kostenlosdiensten online zur Verfügung, über die man beliebige Bilder anhand von Texteingaben erzeugen kann. Ein optimaler Prompt besteht dabei aus drei Komponenten: Rahmen, Thema und Stil (Osztynowicz 2023). Der Rahmen bezieht sich auf den Typ des zu generierenden Bildes. Denkbar wären hier zum Beispiel Foto, 3D-Cartoon, Ölmalerei oder Illustration (ebd.). Das Thema beschreibt den Hauptinhalt des Bildes. Dies können beliebige Inhalts- oder Szenendarstellungen sein. Die Stilkomponente beschreibt „Aspekte wie Beleuchtung, Thema, Zeitraum und Kunststil. Je präziser die Beschreibung des Stils ist, desto besser können Diffusionsmodelle die Vorstellungskraft der Menschen visualisieren“ (ebd.).

Damit Bilder aus den Texteingaben generiert werden können, müssen die Modelle mit Bildern trainiert werden, die mit entsprechenden Beschreibungen versehen sind. Text-to-Image Modelle kombinieren Sprachmodelle mit generativen Modellen. Sprachmodelle kommen allgemein in der KI zum Einsatz, um natürliche Sprache zu verstehen und zu erzeugen (Wuttke 2023). Bei der Text-to-Image Bildgenerierung dienen Sprachmodelle dazu, Bilder anhand ihrer oftmals komplexen Beschreibungen einordnen und klassifizieren zu können. Sprachmodelle müssen dabei Textzusammenhänge oder Sachverhalte erfassen und dürfen dabei nicht nur einzelne isolierte Wörter berücksichtigen (ebd.). Dabei werden Bilder in großen Mengen vom Internet gecrawlt (Vincent 2022). Die verfügbare Datenmenge ist gigantisch. So kann Googles Text-to-Image Model „Imagen“ praktisch mit allen Bildern und den dazugehörigen Beschreibungen trainiert werden, die Google ohnehin bereits indiziert hat (vgl. ebd.).

Die Zahl der Anbieter kostenloser oder bezahlpflichtiger Text-to-Image Dienste ist im letzten Jahr rasant gewachsen. Zu den derzeit Besten gehören unter anderem Midjourney, Dall-E (OpenAI), StableDiffusion oder DreamShaper. Die Entwicklung geht in einem rasanten Tempo weiter und die insbesondere die Qualität generierter fotorealistischer Bilder verbessert sich zunehmend.

## 4. Deepfake-Forensik

In den vorangegangenen Kapiteln wurde beschrieben, wie selbstlernende Systeme aus Trainingsdaten Modelle kreieren, die in der Lage sind, Vorhersagen in den verschiedensten Formen zu treffen. Der Begriff „Vorhersagen“ ist dabei weitläufig zu verstehen und bedeutet unter anderem auch, dass multimediale Inhalte wie Bilder auf Basis von kurzen Nutzereingaben synthetisiert werden. Für Internetnutzer sind die entsprechenden Tools online jederzeit verfügbar und leicht bedienbar. Damit steigt auch das Missbrauchspotential.

In diesem Kapitel soll zunächst auf die gesellschaftlichen Auswirkungen der Verbreitung von Deepfake-Bildern eingegangen werden, gefolgt von der Betrachtung einiger forensischer auf Deep Learning basierte Ansätze zur automatischen Erkennung von Deepfake-Bildern.

### 4.1 Gesellschaftliche Aspekte der Verbreitung von Deepfake-Bildern

Grundsätzlich lässt sich sagen, dass Deepfake-Bilder potenziell dann problematisch sind, wenn sie kaum noch als solche erkennbar sind. Das in Kapitel 3.1 gezeigte Bild, auf dem die Verhaftung von Donald Trump dargestellt wird, lässt bereits erahnen, welchen Einfluss solche Bilder auf die Generierung und Verbreitung von Informationen haben können. Dabei geht es im Wesentlichen um die Verbreitung von Des- und Fehlinformationen. Die offizielle Seite der Bundesregierung warnt mit dem folgenden Hinweis:

*„Deepfakes können eine große Gefahr für die Gesellschaft und Politik darstellen. Insbesondere dann, wenn sie genutzt werden, um die öffentliche Meinung zu manipulieren und politische Prozesse gezielt zu beeinflussen. Desinformationskampagnen könnten künftig immer häufiger durch Deepfakes begleitet und ihr Effekt damit verstärkt werden, zumal die vermeintliche Glaubwürdigkeit durch Bewegtbilder hoch ist. Die schnelle Verbreitung über die Sozialen Medien erhöht das Gefahrenpotential.“<sup>14</sup>*

---

<sup>14</sup> <https://www.bundesregierung.de/breg-de/schwerpunkte/umgang-mit-desinformation/deep-fakes-1876736>

Angesichts der hohen Qualität, der durch generative KI erzeugten, fotorealistischen Bilder und der Tatsache, dass diese praktisch ohne Expertenwissen innerhalb von Minuten erzeugt und im Internet verbreitet werden können, wirkt dieser Warnhinweis bereits veraltet oder zumindest unvollständig. Es wird impliziert, dass insbesondere das Bewegtbild Einfluss auf die vermeintliche Glaubwürdigkeit von Deepfake-begleiteten Falschinformationen hat. Dies ist womöglich nicht länger der Fall. Aktuell begegnet uns eine Flut von synthetischen Bildern (vgl. Wesolowski et. al. 2023), die dazu benutzt werden, um Falsch- oder Fehlinformation zu erzeugen oder zu belegen oder um verifizierte Fakten oder Informationen zu diskreditieren. Die folgenden zwei Beispiele, die im Rahmen des eskalierenden kriegesischen Nahostkonflikts in sozialen Medien geteilt wurden, zeigen wie leicht es ist, Bildfälschungen zu erzeugen, die passend zu Thema und Stil Szenarien realistisch darstellen. Dabei werden insbesondere Personen in Situationen abgebildet, die in der Form nie stattgefunden haben. Zunehmend werden auch Menschen dargestellt, welche gar nicht existieren. Sie werden komplett neu aus den Trainingsbildern generiert.



Abbildung 9: Angebliches von Trümmern verletztes Baby im Gaza<sup>15</sup>

<sup>15</sup> <https://www.jpost.com/israel-news/article-769264>



Abbildung 10: Gefälschte Aufnahme von Soldaten und Panzern der israelischen Armee in einer zerstörten Stadt (vorgeblich Gaza).<sup>16</sup>

Durch die Verbreitung solcher politisch aufgeladenen Bilder in sozialen Netzwerken können Wahrnehmungen und Meinungen manipuliert werden. Dabei handelt es sich immer häufiger um Propaganda mit dem Ziel, Gesellschaften zu destabilisieren (Edwards 2023). Dementsprechend wichtig ist es, Maßnahmen und Technologien zu entwickeln um Deepfake-Bilder automatisiert und schnell zu erkennen.

<sup>16</sup> <https://www.dw.com/en/fact-check-ai-fakes-in-israels-war-against-hamas/a-67367744>

## 4.2 Lösungen und Ansätze automatisierter Erkennung

Detektionssoftware, die im Rahmen der Deepfake-Forensik zum Einsatz kommt, liefert sich mit den Algorithmen bilderzeugender KI ein regelrechtes Katz-und-Maus-Spiel (Bischoff 2023). Forensiker stehen dabei vor dem Problem, dass „[j]eder Algorithmus, [...] der in der Lage ist, Deepfakes aufzuspüren, [...] sich wiederum dafür verwenden [lässt], die bilderzeugenden KIs zu trainieren, damit sie noch überzeugendere Ergebnisse liefern“ (ebd.). Es ist ein KI-gegen-KI Kampf. Deepfake Forensiker müssen, um überhaupt eine Chance zu haben, ebenfalls mit Deep Learning Ansätzen arbeiten.

Im Folgenden sollen einige Ansätze und Lösungen zur Erkennung von DL-synthetisierten Bildern vorgestellt werden. Ansätze, die nur bei Bewegtbildern funktionieren, wie z. B. Intels Deepfake Detektor, der auf Merkmale wie, Blutfluss oder Reflektionen basiert<sup>17</sup>, werden nicht thematisiert.

Die Ansätze lassen sich im Wesentlichen in zwei Kategorien unterscheiden: generalistische und merkmalsbasierte Ansätze (vgl. Delaney 2023: S. 9). Bei der generalistischen Herangehensweise wird davon ausgegangen, dass Deepfake-Bilder im Gegensatz zu echten Bildern eine andere (für den Menschen nicht wahrnehmbare) Charakteristik oder Muster aufweisen und sich so als Fälschungen entlarven lassen. Gemeint sind damit z. B. Rausch- oder Frequenzmuster. Bei merkmalsbasierten Ansätzen hingegen werden Ausschnitte oder Fragmente eines Bildes identifiziert, welche Rückschlüsse zulassen, ob es sich um ein synthetisches Bild handelt oder nicht. Insbesondere ist hierbei die Analyse von biometrischen Merkmalen gemeint (ebd.). Somit ist dieser Ansatz auf fotorealistische Darstellungen von Personen beschränkt.

---

<sup>17</sup> <https://www.intel.com/content/www/us/en/research/blogs/trusted-media.html>



## 4.2.1 Generalistische Ansätze

Die meisten generalistischen Detektionsmodelle sind darauf trainiert, GAN-basierte Deepfakes zu erkennen. Dies ist nachvollziehbar, da diese Technologie bereits einige Jahre länger existiert als Diffusionsmodelle. Detektoren für Diffusionsmodelle befinden sich aktuell noch in der Entwicklung. Sowohl konzeptionell als auch praktisch.

### 4.2.1.1 Analyse von Rausch-Charakteristik in GAN-basierte Deepfake-Bilder

Jiameng Pu et. al. von der Virginia Tech Universität beschreiben in „*NoiseScope: Detecting Deepfake Images in a Blind Setting*“ eine Methode, die in der Lage ist, für GAN-generierte Bilder typische Rauschmuster zu erkennen (Pu et. al. 2020). Die Idee ist nicht gänzlich neu und findet bereits Anwendung in der digitalen Bildforensik, um Digitalkameras zu identifizieren, mit der ein Bild aufgenommen wurde. Jede Digitalkamera weist ein individuelles Sensorrauschen auf und hinterlässt so einen „Fingerabdruck“ in ihren Bildern (vgl. Kirchner o. J.). GAN-Generatoren hinterlassen ebenfalls einen auf Rauschen basierten Fingerabdruck. Dieser ist je nach GAN-Modell unterschiedlich. Die Vorgehensweise von Pu et. al. zur Extraktion der Fingerabdrücke besteht zunächst darin, Bilder des Testdatensatzes zu clustern und den jeweiligen GAN-Modellen (Deepfake-Bilder) aber auch Kamera-Modellen (echte Bilder) zuzuordnen, um anschließend die Fingerabdrücke anhand der jeweiligen Cluster zu identifizieren (Pu et. al. 2020: S. 5).

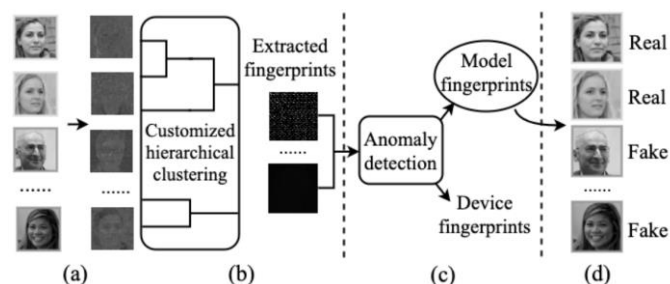


Abbildung 11: NoiseScope Detektor nach Pu et. al.<sup>18</sup>

<sup>18</sup> <https://people.cs.vt.edu/~reddy/papers/ACSAC20.pdf>



Das NoiseScope-Modell besteht aus 4 Komponenten:

### **1. Extraktor für Restrauschen**

Mit Hilfe der Wavelet-Transformation<sup>19</sup> wird Restrauschen aus den Bildern extrahiert (ebd.: S. 6).

### **2. Extraktor für Fingerabdrücke**

Alle Fingerabdrücke (Kamera oder GAN-Modell) werden extrahiert und Clustern zugeordnet. Das Clustern erfolgt hier inkrementell, um den Einfluss von zufälligem individuellem Rauschen der einzelnen Bilder zu reduzieren. Vereinfacht gesagt, werden dabei zunächst ähnliche Bilder geclustert und mit jedem Cluster-Update der Fingerabdruck neu berechnet (ebd.).

### **3. Fingerabdruck-Klassifizierung**

Hier werden die berechneten Fingerabdrücke den GAN-Generatoren zugeordnet. Nach Pu et. al. unterscheiden sich die GAN-Fingerabdrücke signifikant von den Kamera-Fingerabdrücken. Letztere kommen daher bei der Detektion nicht zum Einsatz. Mit Hilfe der sog. Haralick Features<sup>20</sup> wird die Textur der Fingerabdrücke identifiziert. Anschließend können sie dem NoiseScope Modell übergeben werden, um die Klassifizierung nach GAN-Modell vorzunehmen (ebd.).

### **4. Detektor für GAN-Bilder**

Ein Bild wird dann als Fälschung klassifiziert, wenn es mit mindestens einem GAN-Fingerabdruck assoziiert wurde. Dabei werden die in Schritt 2 extrahierten Fingerabdrücke mittels der PCE-Korrelation zwischen jedem Fingerabdruck und den Bildern des Testdatensatzes berechnet. Sobald die Korrelation einen Schwellwert erreicht, werden die Bilder als Fälschung klassifiziert (ebd.).

---

<sup>19</sup> Eine Methode, die in der Bildverarbeitung zum Einsatz kommt, um bspw. Rauschen sichtbar/messbar zu machen (siehe z. B. „Wavelet“ 2024).

<sup>20</sup> Eine von Robert M. Haralick entwickelte Merkmalsbeschreibung von Bildern unter Zuhilfenahme einer Matrix (siehe Haralick et. al. 1973)

#### 4.2.1.2 Identifikation von Rekonstruktionsfehlern bei Diffusions-basierten Deepfake-Bildern

Wang et. al. beschreiben einen vielversprechenden Ansatz zur Detektion von Diffusions-generierten Deepfakebildern in *DIRE for Diffusion-Generated Image Detection* (Wang et. al. 2023). DIRE steht für „Diffusion Reconstruction Error“ (ebd.) und bezieht sich auf die Bildrepräsentation von Rekonstruktionsabweichungen zwischen einem realen Bild und dessen Rekonstruktion über das Diffusionsmodell. Diese Abweichungen lassen sich darstellen und messen.



Abbildung 12: DIRE-Repräsentation von realen Bildern und deren synthetischer Version<sup>21</sup>

Die Autoren gehen dabei von der Grundannahme aus, dass Diffusions-generierte Bilder präziser durch vortrainierte Diffusionsmodelle rekonstruiert werden können als reale Bilder (ebd.). Diese Annahme lässt sich anhand der obigen Abbildung nachvollziehen. Die jeweiligen Rekonstruktionen der Diffusionsmodelle sind näher am Original als die Rekonstruktion des realen Bildes. Diesen Unterschied machen sich die Autoren zu Nutze. Sie benutzen Diffusionsmodelle, um Diffusions-

<sup>21</sup> [https://openaccess.thecvf.com/content/ICCV2023/papers/Wang\\_DIRE\\_for\\_Diffusion-Generated\\_Image\\_Detection\\_ICCV\\_2023\\_paper.pdf](https://openaccess.thecvf.com/content/ICCV2023/papers/Wang_DIRE_for_Diffusion-Generated_Image_Detection_ICCV_2023_paper.pdf)

generierte Bilder zu identifizieren. Dafür wird ein Deep Learning Modell mit binärer Klassifizierung verwendet.

Trainiert wird es mit dem DIRE-Wert, welcher dann die Klassifizierung real/synthetisch vornimmt (ebd.). Im Detail bedeutet das, dass alle Bilder des Trainingsdatensatzes durch Diffusion rekonstruiert werden müssen, um aus den Rekonstruktionsfehler zwischen dem Quellbild und dessen Replikat den DIRE-Wert zu ermitteln. Bei Wang et. al. kommen verschiedene Diffusionsmodelle zum Einsatz, da jedes Modell eine spezifische DIRE-Charakteristik aufweist und der Detektor mit diesen trainiert werden muss. Der DIRE-Wert ist bei allen getesteten Diffusions-Modellen signifikant anders, wenn das Ausgangsbild ein reales Bild ist. In ihren Tests kommen die Autoren auf Genauigkeitswerte zwischen 93.4% und 99.4% (ebd.).

#### **4.2.1.3 Frequenzraumanalyse bei GAN-basierten Deepfake-Bildern**

Die Überführung von Bildern in den Frequenzraum hat sich als beliebtes Werkzeug der digitalen Bildbearbeitung etabliert. Einige Operation, wie zum Beispiel Rauschreduktion, lassen sich in Frequenzraum-transformierten Bildern besser durchführen (Steinbrecher 2005: S. 79). Vereinfacht gesagt, wird durch die Frequenzraumdarstellung des Bildes Rauschen sichtbar gemacht, welches sich anschließend leichter reduzieren oder entfernen lässt, indem man das Signal, welches das Rauschen repräsentiert, bearbeitet (ebd.: S. 95).

Alkishri et. al. gehen in ihrem Ansatz zu Erkennung von Deepfake-Bildern davon aus, dass GAN-basierte Bilder verborgene Charakteristiken aufweisen, die durch die Überführung in den Frequenzraum mittels Deep Learning erkannt werden können und sich somit diese Bilder als Fälschungen entlarven lassen (Alkishri et. al. 2023). Die Bilder werden mit Hilfe der diskreten Fourier-Transformation in ihre Frequenz- bzw. Signaldarstellung überführt und anschließend binär klassifiziert.

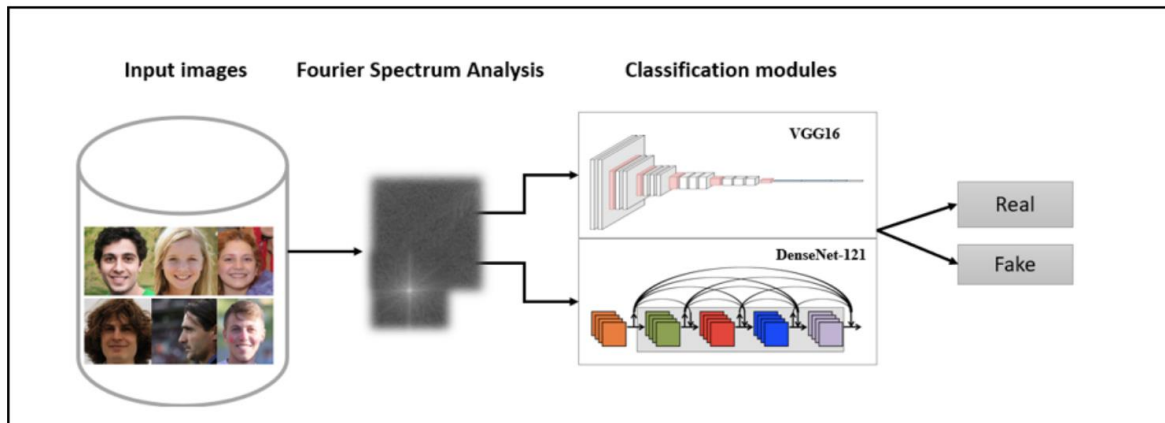


Abbildung 13: Fournier-Transformation und Klassifizierung<sup>22</sup>

Zur Klassifizierung kommen VGG16 und DenseNet-121 zum Einsatz (ebd.). Dabei liegt der Schwerpunkt auf hochfrequenten Komponenten. Mit diesen werden die Klassifizierungsmodelle trainiert, um am Ende zwischen real/fake unterscheiden zu können. Die experimentellen Ergebnisse dieses Ansatzes sind vielversprechend. So kommen die Autoren auf Genauigkeitswerte von 99% für den VGG16-Klassifizierer und auf 92% für DenseNet-121 (ebd.). Allerdings lässt sich dieses Modell nicht auf Diffusions-basierte Deepfake Erkennung anwenden, da es für GAN-synthetische Bilder konzipiert ist.

<sup>22</sup> <https://www.researchgate.net/publication/370592004>

## 4.2.2 Merkmalsbasierte Ansätze

Die Analyse von bestimmten Merkmalen zur Erkennung von Deepfake-Bildern bezieht sich wie bereits erwähnt hauptsächlich auf biometrische Merkmale. Insbesondere werden Details menschlicher Gesichter analysiert. Zumeist werden hierfür Bewegtbilder benötigt, da sich biometrische Auffälligkeiten oder fehlerhafte Darstellungen oft nicht aus Standbildern ableiten lassen. Zwei Ansätze, die sich auch auf einzelne Bilder anwenden lassen, werden im Folgenden kurz betrachtet.

### 4.2.2.1 Detektion von manipulierten Gesichtsausdrücken

G. Mazaheri und A. K. Roy-Chowdhury beschreiben in ihrem Ansatz eine Methode zur Erkennung von manipulierten Gesichtsausdrücken in Bildern und Videos. Mit Hilfe von Deep Learning Modellen zur Erkennung von Gesichtsausdrücken (Facial Expression Recognition, FER) werden die zugrundeliegenden Merkmale verschiedener Gesichtsausdrücke herangezogen, um manipulierte Regionen im Gesicht zu lokalisieren (Mazaheri, Roy-Chowdhury, 2022). Die Autoren benutzen ein „two-stream network“ (ebd.) für die Detektion. Im ersten Stream werden mittels FER die Regionen des Bildes, welche für die Codierung der dargestellten Gesichtsausdrücke relevant sind, aus verschiedenen Datensätzen mit realen und gefälschten Gesichtern extrahiert.

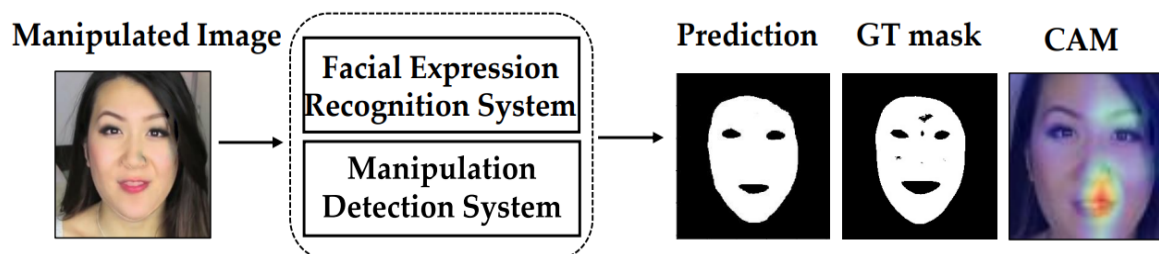


Abbildung 14: FER und Manipulationsdetektion für Deepfake-Erkennung bei Gesichtern<sup>23</sup>

<sup>23</sup> [https://openaccess.thecvf.com/content/WACV2022/papers/Mazaheri\\_Detection\\_and\\_Localization\\_of\\_Facial\\_Expression\\_Manipulations\\_WACV\\_2022\\_paper.pdf](https://openaccess.thecvf.com/content/WACV2022/papers/Mazaheri_Detection_and_Localization_of_Facial_Expression_Manipulations_WACV_2022_paper.pdf)

Im zweiten Stream kommt eine Encoder-Decoder-Architektur<sup>24</sup> für die Manipulationsdetektion zum Einsatz. Der Encoder kombiniert dabei die extrahierten Merkmale der Gesichtsausdrücke aus dem FER mit den dimensionsreduzierten Bildern<sup>25</sup>. Anschließend wird über den Decoder vorhergesagt, ob auf dem Bild manipulierte Bereiche vorhanden sind.

In ihrem Experiment kommen die Autoren je nach verwendeten Testdatensätzen auf Ergebnisse zwischen 81,46 % und 99,03 % (ebd.). Der Ansatz ist jedoch auf die Erkennung von manipulierten Bildern beschränkt. Das bedeutet, dass ein Originalbild verändert wird und diese lokalen Veränderungen detektiert und somit das Bild als Fälschung entlarvt werden. Auf vollsynthetische Bilder ist dieser Ansatz nicht ohne weiteres anwendbar.

---

<sup>24</sup> Diese Architektur ist auch als „Transformer“ bekannt. Dabei werden über den Encoder Eingabedaten in „Feature Vektoren“ umgewandelt. Diese repräsentieren die relevanten/wichtigen Merkmale der Eingabedaten (vgl. z. B. Munding 2021). Der Dekoder ist das Gegenstück. Er verwandelt die Feature-Vektoren zurück in das Ausgangsdatenformat zurück. So können zum Beispiel anhand extrahierter Feature Vektoren aus Bildern neue/veränderte Bilder generiert werden.

<sup>25</sup> Das Vorgehen der Dimensionsreduktion dient dazu, die Anzahl der Merkmale und Variablen von Daten zu reduzieren. Wesentliche für die Daten essenzielle Merkmale bleiben erhalten. Dadurch werden Daten besser interpretierbar und Deep Learning Modelle performanter (vgl. Pramoditha 2021).

#### 4.2.2.2 Detektion von unnatürlichen Lichtreflektion in Augen

Eine weitere Methode zur merkmalsbasierten Erkennung beschreiben Hu et. al. von der Universität von Buffalo USA. Sie schlagen vor, GAN-generierte Gesichter anhand der unnatürlichen Darstellung der Lichtreflektionen in menschlichen Augen zu identifizieren (Hu et. al. 2020). Diese Reflexionen können von den GAN-Modellen nicht ausreichend natürlich dargestellt werden, da die Modelle geometrische Beziehungen zwischen Gesichtsteilen nur unzureichend berücksichtigen (ebd.). So sind die von der Hornhaut erzeugten Reflexionen auf realen Bildern in beiden Augen nahezu gleich, während sich sie in den GAN-Bildern stark unterscheiden.

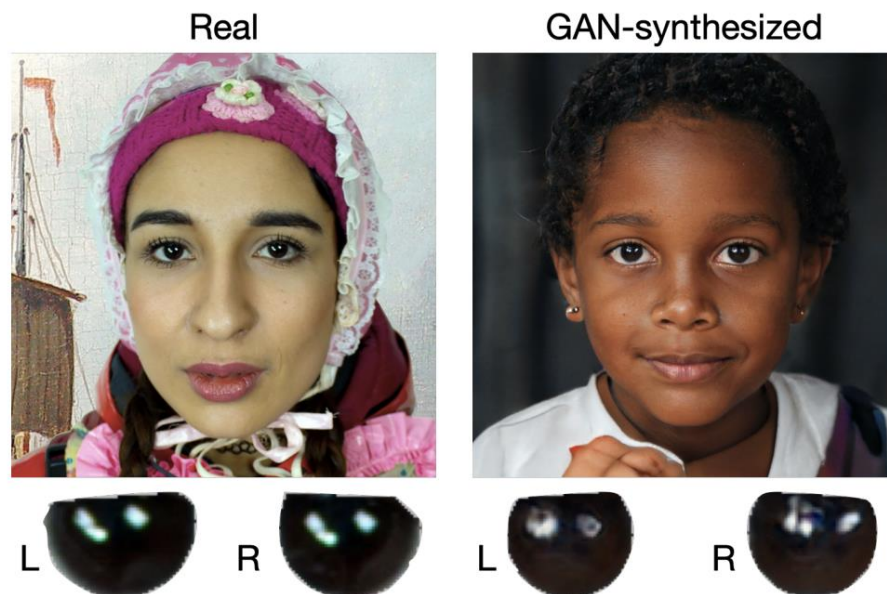


Abbildung 15: Hornhaut-Reflexionen GAN-erzeugte und reale Gesichter<sup>26</sup>

Die Hornhautreflexionen werden dabei in mehreren Schritten extrahiert. Zunächst kommt eine Gesichtserkennung zum Einsatz. Aus dieser werden die sogenannten Landmarker<sup>27</sup> extrahiert, die für die Lokalisierung der Augen zum Einsatz kommen. Nach der Lokalisierung werden die Pupillen mittels Canny-Kantendetektion und

<sup>26</sup> <https://arxiv.org/pdf/2009.11924.pdf>

<sup>27</sup> Dabei handelt es sich um Referenz- oder Orientierungspunkte in Bildern, welche zur Hilfe genommen werden, um bestimmte Muster oder Objekte zu erkennen (vgl. z. B. Sester 2022)

Hough-Transformation extrahiert. Im letzten Schritt werden die Reflektionen extrahiert. Dies erfolgt über eine einfache Schwellwert-Methode, da die Reflektionen immer heller sind als die Iris (ebd.). Die extrahierten Reflexionen werden in IoU-Punkte<sup>28</sup> überführt. Die Bestimmung, ob ein Gesicht GAN-synthetisiert ist, erfolgt anhand der Interpretationen der IoU-Werte des jeweiligen Augenpaars. Dieser liegt zwischen 0 und 1. Ein Wert nahe 1 impliziert ein reales Bild, während ein Wert nahe 0 auf ein Deepfake hinweist.

### **4.3 Fazit zur Deep Learning-basierten Deepfake Detektion**

Die hier vorgestellten Methoden und Ansätze für eine auf Deep Learning basierende Technologie zur Detektion von Deepfake-Bildern sind durchaus innovativ und beeindrucken mit guten Laborergebnissen. Dennoch werden sie der Entwicklung generativer KI aktuell noch nicht gerecht. Wie in Kapitel 4.2 erwähnt, liefert sich die Deepfake Forensik ein Katz-und-Maus Spiel. Derzeit hat die Maus einen bedeutenden Vorsprung. Die neuartigen und performanten Technologien Diffusions-basierter generativer KI sind seit gut einem Jahr praktisch für jeden Internetnutzer verfügbar. Mit Hilfe von Text-to-Image Generatoren erleben wir eine regelrechte Flut von Deepfake-Bildern. Immer mehr Unternehmen und Start-Ups drängen im Bereich generativer KI auf den Markt und tragen zu einer erheblichen Dynamik, Innovation und Weiterentwicklung generativer Bild- und Videosynthese bei. Um diesen technologischen Vorsprung aufzuholen, sollte sich Deepfake Forensik verstärkt auf Diffusionsmodelle konzentrieren. Nur wenige vielversprechende Ansätze wurden bisher veröffentlicht. Das oben beschriebene DIRE-Modell von Wang et. al. ist eine der wenigen Ausnahmen.

Meines Erachtens sollten Deepfake-Forensiker auch in Betracht ziehen, dass generalistische Ansätze nur für bestimmte Diffusionsmodelle funktionieren und aufgrund der rasanten Weiterentwicklung nur kurzzeitig eine zuverlässige Erkennungsrate haben. Weiterhin erscheint es fraglich, ob ein Modell, was in der

---

<sup>28</sup> Auch Jaccard-Koeffizient genannt. Dieser dient zur Bestimmung von Ähnlichkeiten von Mengen und liegt immer zwischen 0 und 1 („Jaccard-Koeffizient“ 2023).



Lage ist, jedes generative Bild zuverlässig zu erkennen, überhaupt sinnvoll ist. Deepfake-Bilder, die eine hohe forensische Bedeutung haben und potenziell durch ihre Verbreitung gesellschaftlichen Schaden anrichten, zeigen vor allem eines: Personen mit gefälschter Mimik, Gestik und Hintergrundgeschehen. Von daher plädiere ich dafür, auch forensische Deep Learning Ansätze zu entwickeln, die auf menschliche Biomarker basieren und damit auf personendarstellende Deepfake-Bilder beschränkt sind. Ein solcher Ansatz wird im folgenden Experiment verfolgt.

## **5. Deep Learning Experiment: Hände als Biomarker zur Erkennung von Diffusions-basierten Deepfake-Bildern**

In diesem Kapitel geht es um einen praktischen Versuch, mit Hilfe von Deep Learning Diffusions-basierte Deepfake-Bilder zu erkennen, die Personen darstellen. Der Ansatz ist merkmalsbasiert, wobei menschliche Hände als Biomarker dienen. Die natürliche Darstellung von menschlichen Händen ist für generative KI eine große Herausforderung. Oft sehen die synthetisierten Hände unnatürlich aus, sind anatomisch inkorrekt (z. B. zu viele oder zu wenige Finger, falsche Proportionen) oder wirken aufgrund fehlender Details wie gemalt.

### **5.1 Motivation und Ziel**

Die Motivation, Hände als Biomarker zu nutzen, um damit automatisiert Deepfake-Bilder zu entlarven, entspringt direkt aus der Erkenntnis, dass die Deepfake-Forensik massiven Aufholbedarf hat und die Forschungen und Experimente insbesondere bei Diffusions-synthetisierten Deepfakes noch am Anfang stehen. Hinzu kommt, dass Deepfake-Bilder zunehmend prominente Personen zeigen und diese aufgrund ihrer fotorealistischen Qualität, sowie ihrer rasanten Verbreitung in sozialen Netzwerken, verstärkt für Desinformationen missbraucht werden. Dabei lassen sich viele dieser Bilder bei genauerem Hinsehen als Fälschungen erkennen. In der Schnelllebigkeit des Internets schauen die Nutzer jedoch oftmals nicht so genau hin und halten die Deepfake-Bilder für echt. Es folgen zwei Beispielbilder, die auf dem ersten Blick echt aussehen, sich bei genauerer Betrachtung der Hände (und anderer Details) als Fälschungen entlarven lassen.



Abbildung 16: Streitender Papst, Diffusions-synthetisiert über Midjourney<sup>29</sup>



Abbildung 17: Donald Trump und Joe Biden beim Teetrinken, Diffusions-synthetisiert über Midjourney<sup>30</sup>

<sup>29</sup> <https://discord.gg/midjourney>

<sup>30</sup> Ebd.

Die Erzeugung realistischer Hände ist für Diffusionsmodelle eine der größten Herausforderungen. Zwar gibt es hier Fortschritte (Cuthbertson 2023), bei der Auswertung aktueller Bilder fällt jedoch auf, dass die Darstellung häufig noch nicht ausreichend realitätsnah erfolgt. So lange diese Schwäche besteht, liegt es nahe, sich als Forensiker diese zu Nutze zu machen. Wenn man einen großen Teil von Personen-darstellenden Deepfake-Bilder entlarven kann, indem man die Hände genauer betrachtet, so müsste es möglich sein, dies auch automatisiert durch ein binär klassifizierendes (real/fake) Deep Learning Modell durchzuführen.

Möglicherweise können durch Deep Learning auch für den Menschen nicht mehr sichtbare Strukturen herangezogen werden, welche eine zuverlässige Erkennung auch dann ermöglichen, wenn die Darstellung der Hände augenscheinlich ausreichend realitätsnah erfolgt.

Mit dem Experiment wird demnach folgendes Ziel verfolgt:

*Es soll anhand eines Deep Learning Models untersucht werden, inwieweit Hände als Biomarker geeignet sind, um Diffusions-basierte Bilder, welche Personen darstellen, als Deepfakes zu entlarven.*

Dazu wird ein eigenes Deep Learning Model mit realen und gefälschten Händen trainiert. Das Modeltraining erfolgt in verschiedenen Varianten, wobei mit verschiedenen Konfigurationen und Layer-Tiefen experimentiert wird. Darüber hinaus erfolgte das Training auch unter Einbeziehung eines existierenden Modells, welches in der Lage ist, menschliche Körperteile zu klassifizieren.

## 5.2 Datenbeschaffung und Vorverarbeitung

Die Datenbeschaffung und Vorverarbeitung erwiesen sich als große Herausforderung. Ziel war es, mindestens 500 jeweils reale und gefälschte Handbilder zu beschaffen, um somit ein Minimum an ausreichenden Trainingsdaten zur Verfügung zu haben.

### Datenbeschaffung realer Bilder

Im Internet gibt es eine Vielzahl von Datensätzen mit den unterschiedlichsten Bildinhalten. Eine der größten Datensätze, welche Bilder von Personen in den unterschiedlichsten Situationen beinhalten, ist das „flickr30k\_images“<sup>31</sup> Dataset mit mehr als einunddreißigtausend Bildern von u. a. Personen und Tieren in einer Vielzahl von Szenen und Umgebungen.



Abbildung 18: Beispielbilder von Personen aus dem "flickr30k\_images" Datensatz

Zusätzlich erfolgte eine manuelle Suche im Internet nach Bildern, die explizit Handgesten, wie z. B. ein Händeschütteln zeigen.

### Datenbeschaffung diffusionsgenerierter Bilder

Die Beschaffung von synthetischen Bildern, auf denen Personen mit Händen ausreichend fotorealistisch dargestellt werden, erwies sich anfangs als kaum lösbare Aufgabe. Ziel war es größtenteils Bilder zu verwenden, die durch den Dienst

---

<sup>31</sup> <https://www.kaggle.com/datasets/hsankesara/flickr-image-dataset>

„Midjourney“ erzeugt wurden. Das Unternehmen erzeugt meines Erachtens die realistischsten durch Diffusion synthetisierten Bilder. Diese sind jedoch weder in Form von Datasets noch über eine einfache Internetsuche auffindbar. Wenn man über Midjourney Bilder erzeugen will oder Zugang zu bereits erzeugten Bildern erhalten will, funktioniert dies ausschließlich über den Community-Dienst „Discord“. Dort können Nutzer mittels Text-to-Image Bilder erzeugen und in den verschiedenen Kanälen bereits erzeugte Bilder anschauen. Der Beitritt zur Midjourney-Community in Discord ist kostenlos und es ist möglich mittels Suchfunktionen bestimmte Bilder zu suchen. Problem hierbei ist, dass die Suchergebnisse häufig viel zu unpassend zur Suchabfrage waren. Dadurch kommt ein enormer manueller Suchaufwand hinzu, um aus den Suchergebnissen entsprechend geeignete fotorealistische Hand-darstellende Bilder zu finden. Somit musste die Suche in irgendeiner Form automatisiert werden. Über den „DiscordChatExporter“<sup>32</sup> ist es möglich, ganze Chats bzw. Kanäle der Discord-Community in HTML-Format zu exportieren. Die Exports enthalten die zu den Bildern gehörigen CDN-Domänen (Content Delivery Network), über welche man die Bilder direkt und ohne vorherige Authentifizierung herunterladen kann. Damit bestand immerhin die Möglichkeit sämtliche generierte Bilder der Midjourney-Community lokal zu speichern und diese dann performanter zu durchsuchen.

Die Herausforderung bestand zunächst darin, aus den mehreren hunderttausend Zeilen Code der HTML-Exporte die CDN-Domänen zu extrahieren und anschließend die Bilder herunterzuladen. Dies erfolgte mithilfe eines auf Regex basierenden Bash-Kommandos auf einen Ubuntu-System. Die extrahierten CDN-Domänen wurden zeilenweise in Text-Dateien geschrieben, mussten aber aufgrund der riesigen Menge mit ca. zweihunderttausend Ergebnissen aufgesplittet werden, so dass kleinere Textdateien mit ca. zwanzigtausend CDN-Domänen generiert wurden. Anschließend wurden die Textdateien mit Hilfe eines Bash-Skripts zeilenweise gelesen und die Bilder heruntergeladen. Dies dauerte mehrere Stunden je Textdatei. Aus den heruntergeladenen Bildern mussten nun noch diejenigen

---

<sup>32</sup> <https://github.com/Tyrrrz/DiscordChatExporter>

herausgefiltert werden, die eine oder mehrere Hände beinhalten. Dieses Vorgehen wird im Folgenden beschrieben.



Abbildung 19: Extrahierte fotorealistische Midjourney-Bilder, die Hände zeigen

## Datenaufbereitung

Da das experimentelle Deep Learning Modell mit den Händen und nicht mit dem gesamten Bild trainiert werden sollte, mussten nun die Hände aus sämtlichen Bildern extrahiert und jeweils als neues Bild abgespeichert werden. Dies musste ebenfalls automatisiert geschehen, da der manuelle Aufwand bei mehreren tausend Bildern nicht vertretbar war. Hierzu war es notwendig, ein Python-Skript zu erstellen und darin Handerkennungs-Bibliotheken zu implementieren. Als passend erwies sich Googles „MediaPipe“ Framework<sup>33</sup>. Das Framework kann ohne größeren Aufwand in Python implementiert werden und ist sowohl in der Lage Hände als auch dazugehörige Gesten zu erkennen.

---

<sup>33</sup> [https://developers.google.com/mediapipe/solutions/vision/hand\\_landmarker](https://developers.google.com/mediapipe/solutions/vision/hand_landmarker)





Abbildung 20: Selbstversuch Handerkennung via MediaPipe inklusive der Darstellung der Landmarker

Damit ließen sich schnell die Bilder aussortieren, auf denen keine Hände erkennbar waren. MediaPipe ist dabei auch in der Lage Hände zu erkennen, wenn diese größtenteils verdeckt oder nur sehr klein abgebildet sind. Das Skript prüft dabei zunächst, ob mindestens eine Hand auf dem Bild erkannt wird. Ist das nicht der Fall wird das Bild gelöscht. Anhand der detektierten Hände wird über die Landmarker ermittelt, welche Größe das Bild mit der Hand nach der Extraktion haben soll. Dazu müssen zwei Landmarker bestimmt werden: der mit den kleinsten X und Y-Werten und der mit den größten X und Y-Werten. Diese werden benötigt, um den Bildausschnitt zu bestimmen, welcher die Hand enthält. Da die Landmarker der Hände praktisch in allen Bildern nie am äußersten Rand der Hand lagen, musste der Hand-beinhaltende Bildausschnitt um 10% vergrößert werden. Somit wurde in den meisten Fällen sichergestellt, dass die gesamte Hand auf dem extrahierten Bild vorhanden ist. Zu beachten war diesbezüglich, dass eine pauschale Vergrößerung des Handbildes nicht möglich war, da es häufig vorkam, dass die Hände zu nah am Bildrand waren. Einen Bildausschnitt zu generieren, der über die Weite oder Höhe des Ursprungsbildes hinausgeht, führt zu einem Fehler.



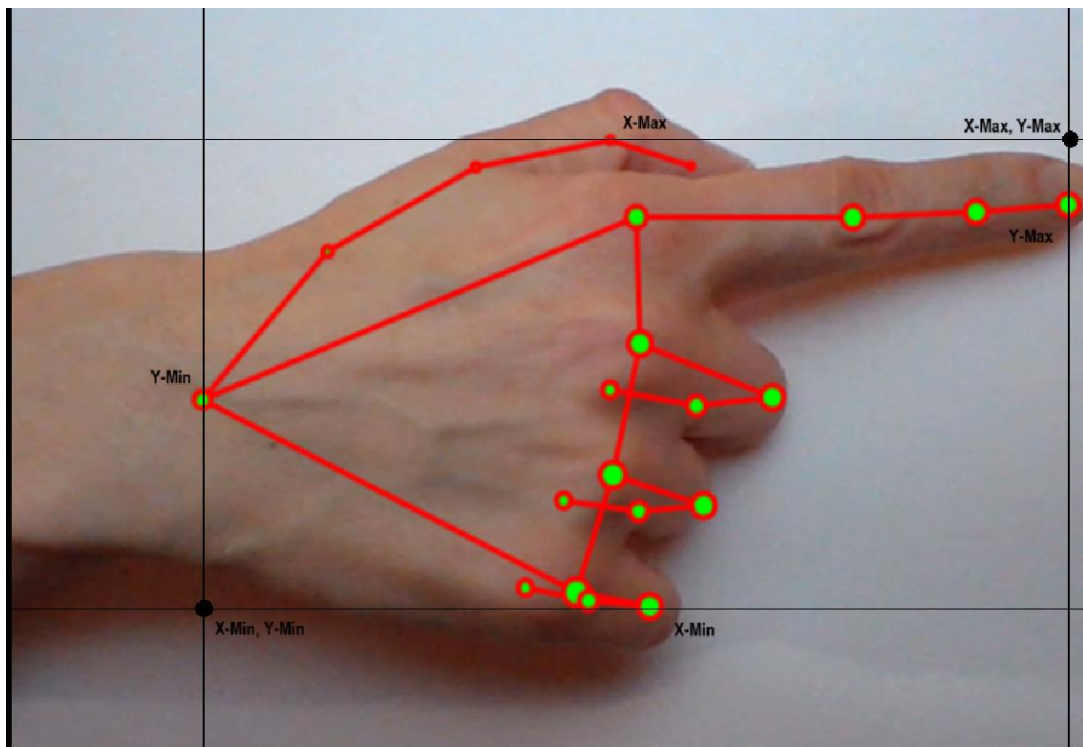


Abbildung 21: Hand-Landmarker und die Bestimmung des Handausschnitts (ohne 10% Erweiterung)

Im obigen Bild ist dargestellt, wie der Bildausschnitt der Hand über die Landmarker bestimmt wird. Hier sieht man auch, dass der äußerste Landmarker im Zeigefinger fast am äußeren rechten Rand des Bildes ist. In diesem Fall ist eine Vergrößerung um 10 Prozent in der Y-Achse nicht möglich. Es wird stattdessen der maximale Y-Wert des Bildes genommen. Der fertig vergrößerte Bildausschnitt musste eine Größe von mindestens 35 x 35 Pixeln aufweisen, bevor er als Bild gespeichert wird. Diese Mindestgröße ist notwendig, damit ein Minimum an Hand-identifizierenden Merkmalen für das Deep Learning Model erhalten bleibt. Im Folgenden ist noch einmal der Ablauf der automatisierten Datenaufbereitung der Midjourney-Bilder dargestellt.

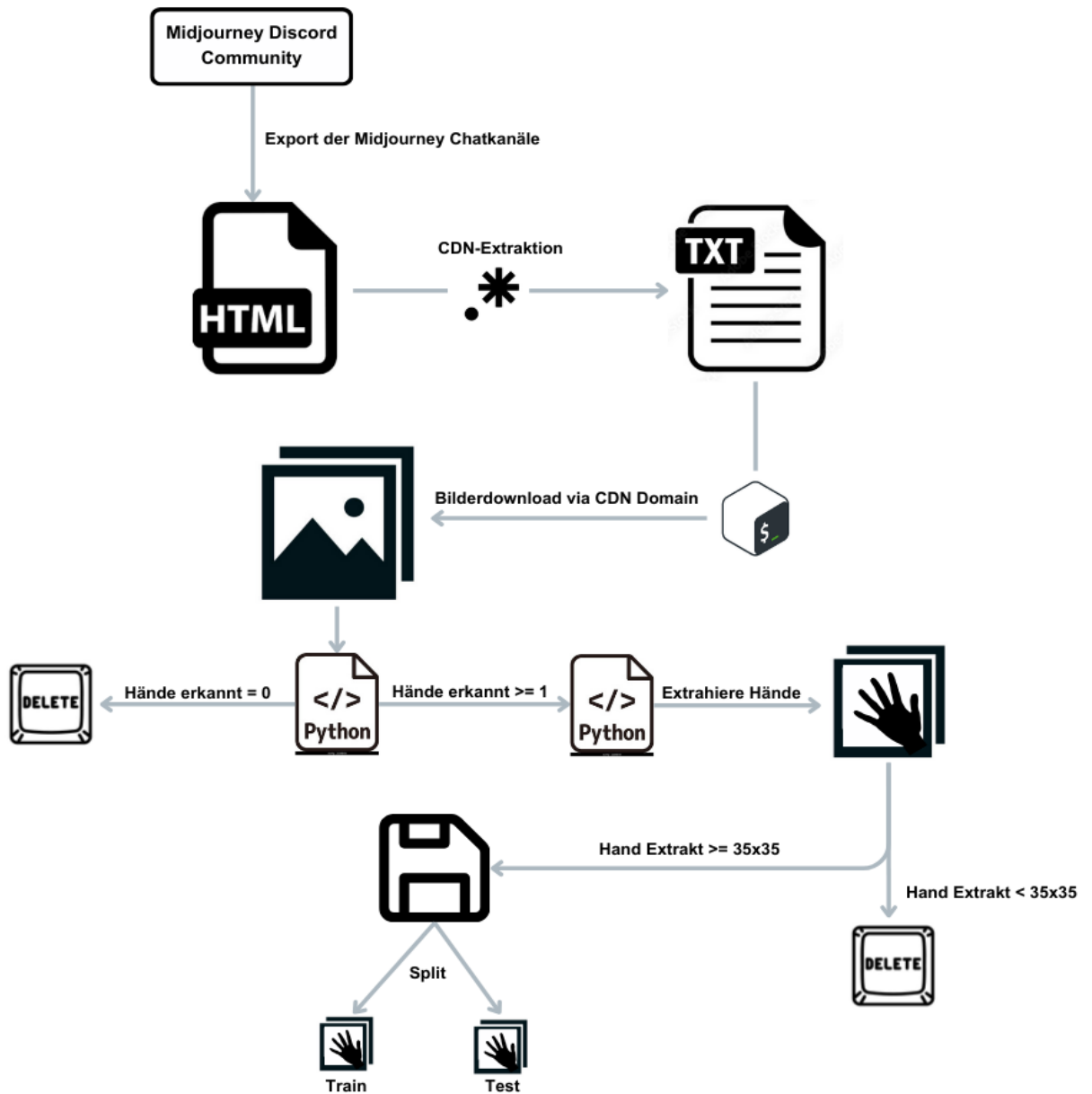


Abbildung 22: Ablaufdiagramm zur Datenbeschaffung und Aufbereitung der Midjourney-Bilder

Der gesamte Prozess bis hin zu den fertigen Trainings- und Testbildern ließ sich leider nicht automatisieren. Nachdem die Bilder verarbeitet wurden und die Handausschnitte als neue Bilder abgespeichert waren, erfolgte ein erheblicher manueller Aufwand, um die Handbilder zu bereinigen. Folgende Bilder wurden aussortiert:

- Hände, die nicht (ausreichend) fotorealistisch dargestellt waren (Bsp. Cartoons)
- Hände, die in hohem Maß anatomisch und proportional inkorrekt abgebildet waren
- Hände, die als solche erkannt wurden, aber keine waren
- Duplikate

Abschließend mussten die gewonnenen Handbilder noch in Trainings- und Testdaten aufgeteilt werden. Hierzu wurde ebenfalls ein kleines Python-Skript programmiert. Es wählt per Zufall Bilder für den Test- und Trainingsordner aus und verschiebt diese dorthin.

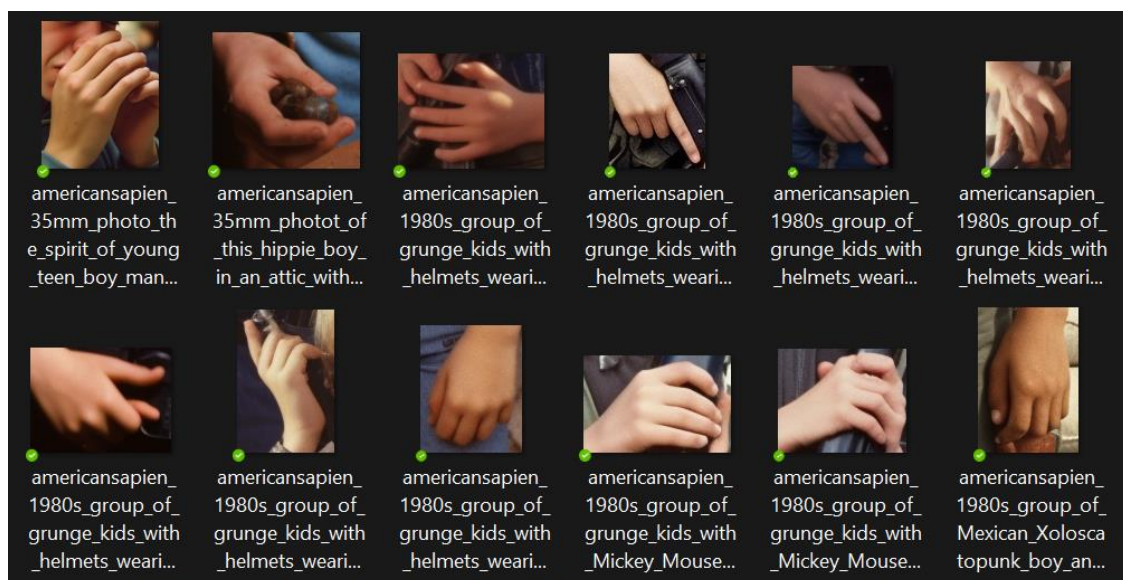


Abbildung 23: Extrahierte Hände aus den Midjourney-generierten Bildern

Für das Experiment standen schließlich 1000 Handbilder zur Verfügung. 500 reale und 500 falsche Hände.

## 5.3 Modellaufbau

### 5.3.1 Bibliotheken, Entwicklungsumgebung und Hardware

Die Implementierung des Deep-Learning Modells erfolgte mit dem Open-Source DL-Framework „Keras“. Keras agiert als High-Level-API<sup>34</sup> für „TensorFlow“<sup>35</sup>. TensorFlow wiederum ist ein von Google entwickeltes Open-Source-Framework für Machine Learning.

Neben den Keras- und TensorFlow-Bibliotheken kommen „Matplotlib“<sup>36</sup>, „Scikit-Learn“<sup>37</sup> und „Pandas“<sup>38</sup> zum Einsatz. Sie dienen der Visualisierung und dem Erfassen von Metriken.

Die Programmierung des Modells erfolgte in der browserbasierten Entwicklungsumgebung „Jupyter Notebook“<sup>39</sup> und dem Umgebungs- und Bibliotheken-Manager „Anaconda Navigator“<sup>40</sup>.

Bibliotheken / Entwicklertools	Version
Keras	2.13.1
TensorFlow	2.13.0
Matplotlib	3.7.0
Scikit-Learn	1.2.1
Pandas	1.5.3
Jupyter Notebook	6.5.2
Anaconda Navigator	2.4.2

*Tabelle 1: Versionsübersicht der verwendeten Software*

---

<sup>34</sup> <https://www.tensorflow.org/guide/keras>

<sup>35</sup> <https://www.tensorflow.org/>

<sup>36</sup> <https://matplotlib.org/>

<sup>37</sup> <https://scikit-learn.org/>

<sup>38</sup> <https://pandas.pydata.org/>

<sup>39</sup> <https://jupyter.org/>

<sup>40</sup> <https://www.anaconda.com/>

Als Hardware stand ein Laptop mit 16 GB RAM, AMD Ryzen 5-Prozessor (5600-H) mit integrierter 7-Kern-GPU zur Verfügung. Auf diesem wurde das Modell entwickelt und auch trainiert.

### **5.3.2 Implementierung und Parametrisierung des Modells**

Die Implementierung orientiert sich zu einem großen Teil an dem Keras-*Tutorial* „*Building powerful image classification models using very little data*“ von Francois Chollet (Chollet 2016)<sup>41</sup>.

#### **Trainingsmethoden: From-Scratch und Transfer Learning**

Grundsätzlich besteht beim Trainieren von Deep Learning Modellen die Möglichkeit, das Model von Grund auf neu („from scratch“) zu trainieren oder vortrainierte Modelle in das Training mit einzubeziehen, um deren erlernte Klassifizierungsparameter zu transferieren (Transfer Learning).

Im Experiment erfolgte das Modeltraining nach diesen beiden Methoden. Das From-Scratch Model hat dabei die zur Klassifizierung notwendigen Parameter allein aus den Hand-Trainingsdaten erlernt. Für die Transfer Learning Variante wurde das vortrainierte Model „VGG16“ verwendet. Es ist in der Lage eine Vielzahl von Körperteilen (inkl. Hände) zu erkennen und zu klassifizieren. Somit bietet das Model die Möglichkeit, erlernte Parameter zu transferieren, welche für die Identifikation realer Hände ausschlaggebend sind, um diese somit effektiver von gefälschten Händen zu unterscheiden.

#### **Durchschnittsgröße ermitteln**

Im ersten Schritt wird die Durchschnittsgröße aller verfügbaren Bilder ermittelt. Diese liegt gerundet bei 140 x 130. Alle Bilder werden dann auf diese Größe skaliert, bevor sie dem Model übergeben werden. Bildverarbeitende Deep Learning Modelle können nur mit Bildern gleicher Größe trainiert werden.

---

<sup>41</sup> Das Tutorial wurde bereits 2016 veröffentlicht und enthält den Hinweis, dass es veraltet ist. In Bezug auf den verwendeten Code und der Syntax der Modelimplementierung für binäre Klassifizierung ist die Anleitung nach wie vor aktuell.

## Trainingsbilder künstlich erhöhen

Im zweiten Schritt wird die Anzahl der Bilder künstlich erhöht. Hierzu bietet Keras ein Bild-Generator an. Die „ImageDataGenerator“ Bibliothek erstellt dabei aus jedem Bild weitere Bilder, die mit verschiedenen Funktionen verändert werden können. Zu diesen Funktionen gehören unter anderem: Rotation, Zoom, vertikale oder horizontale Spiegelung oder Abschneiden der Ränder. Für das Model können diese zusätzlich generierten Bilder die Mustererkennung für die Klassifizierung verbessern.

Folgende Parameter zur Bildgenerierung wurden dem Modell übergeben:

- `Width_shift_range=0.10`: Verschiebung des Bildes nach rechts oder links um maximal 10% (Bildgröße verändert sich dadurch nicht, nur das Motiv verschiebt sich)
- `Height_shift_range=0.10`: Verschiebung des Bildes nach oben oder unten um maximal 10% (auch hier keine Veränderung der Bildgröße)
- `Rotation_range=20`: Rotation um 20°
- `Shear_range=0.1`: Neigung um maximal 10% (ähnlich wie Rotation)
- `Zoom_range=0.1`: Hereinzoomen um 10%
- `Horizontal_flip=True`: horizontal spiegeln
- `Vertical_flip=True`: vertikal spiegeln

Es folgen einige Beispielbilder aus dem Image-Generator.

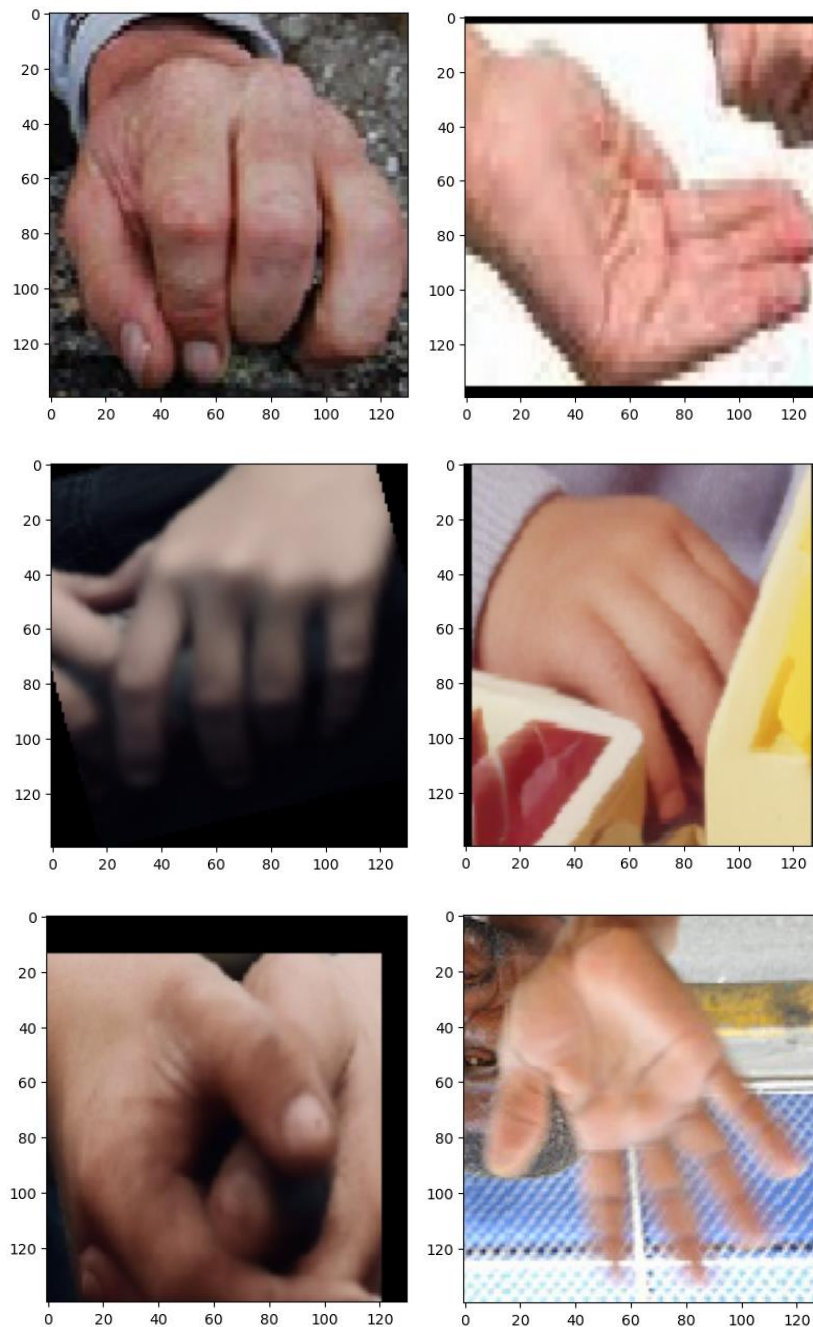


Abbildung 24: Beispiele aus dem Keras Image-Generator

## Overfitting

Bevor auf den Layer-Aufbau der Modelle eingegangen wird, soll noch kurz auf den Begriff des „Overfitting“ eingegangen werden. Ein Overfitting bedeutet, dass das Modell zwar bei den Trainingsdaten sehr gute Vorhersageergebnisse erzielt, jedoch bei unbekanntem Daten versagt (vgl. z. B. Oppermann o. J.). Es ist also „überangepasst“ an die Trainingsdaten und somit nicht anwendbar auf unbekanntem Daten. Das Verständnis und Bewusstsein für diesen möglichen Modellzustand ist essenziell, um ein performantes Deep Learning Modell zu entwickeln.

## Layer Implementierung: From-Scratch Model

### *Layerdesign*

Der Aufbau der Layer erfolgte sequenziell. Dabei wurden zwischen drei und zehn Convolutional Layer mit unterschiedlichen Kernel-Größen implementiert, um zu testen, welchen Einfluss die Anzahl der Convolutional Layer auf das Klassifizierungsergebnis hat.

Im nachfolgenden Dense Layer (Fully Connected Layer) wurde ebenfalls mit verschiedenen Werten (zwischen 32 und 2048) experimentiert. Hier hatte ein hoher Neuronenwert von 2048 die besten Auswirkungen auf das Klassifizierungsergebnis.

Um die Gefahr eines „Overfitting“ zu reduzieren, wurde anschließend ein „Dropout Layer“ mit 0,5 implementiert. Dieser bewirkt, dass per Zufallsprinzip die Hälfte der Neuronen im Dense Layer abgeschaltet werden. Da es sich um ein Modell für binäre Klassifizierung handelt, muss nach dem Dropout Layer ein weiterer Dense Layer mit einem Neuron implementiert werden. Damit wird gewährleistet, dass die extrahierten Merkmale aller vorherigen Layer zu einer binären Klassifizierung zusammengeführt werden.

Im letzten Schritt wurde explizit der Standardwert ( $1e-03$  bzw. 0.001) der Learning Rate des Optimizers verändert. Hier erzielte eine Reduzierung auf  $1e-04$  (0.0001) eine signifikante Verbesserung der Klassifizierungsergebnisse.



```

Model: "sequential"
-----
Layer (type)                Output Shape                Param #
-----
conv2d (Conv2D)             (None, 136, 126, 32)       2432
max_pooling2d (MaxPooling2D) (None, 68, 63, 32)         0
conv2d_1 (Conv2D)           (None, 64, 59, 64)         51264
max_pooling2d_1 (MaxPooling2D) (None, 32, 29, 64)         0
conv2d_2 (Conv2D)           (None, 28, 25, 64)         102464
max_pooling2d_2 (MaxPooling2D) (None, 14, 12, 64)         0
conv2d_3 (Conv2D)           (None, 10, 8, 64)          102464
max_pooling2d_3 (MaxPooling2D) (None, 5, 4, 64)           0
flatten (Flatten)           (None, 1280)                0
dense (Dense)                (None, 2048)                2623488
activation (Activation)      (None, 2048)                0
dropout (Dropout)           (None, 2048)                0
dense_1 (Dense)              (None, 1)                   2049
activation_1 (Activation)    (None, 1)                   0
-----
Total params: 2884161 (11.00 MB)
Trainable params: 2884161 (11.00 MB)
Non-trainable params: 0 (0.00 Byte)
-----

```

Abbildung 25: From-Scratch Model Zusammenfassung

Die obige Abbildung zeigt die Übersicht des finalen Models. Insgesamt ergaben sich 2.884.141 trainierbare Parameter. Da es sich bei den Trainingsbildern um kleine Bilder handelt, erwies es sich als kontraproduktiv die Anzahl der Trainingsparameter noch weiter zu erhöhen (z. B. durch Hinzufügen weiterer Layer), da dies zu einem Overfitting führte.

## Layer Implementierung: Transfer Learning Model

Die Anzahl der Layer des vortrainierten VGG16 Models ist insoweit vorbestimmt, als dass diese nicht ohne weiteres reduziert werden können. Es können aber neue Layer hinzugefügt werden. In diesem Fall war es lediglich notwendig, einen weiteren Dense Layer mit einem Neuron hinzuzufügen, um sicherzustellen, dass die Klassifizierung binär erfolgt.<sup>42</sup>

Um die vom VGG16 Model ursprünglich erlernten Merkmale zu erhalten, war es außerdem notwendig, dessen Layer auf „trainable = false“ zu setzen. Anderenfalls würde dies entgegen dem Konzept des Learning Transfers laufen, da es darum geht, von den bereits fertig erlernten Merkmalen zu profitieren.

```

Model: "model"

```

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 140, 130, 3)]	0
block1_conv1 (Conv2D)	(None, 140, 130, 64)	1792
block1_conv2 (Conv2D)	(None, 140, 130, 64)	36928
block1_pool (MaxPooling2D)	(None, 70, 65, 64)	0
block2_conv1 (Conv2D)	(None, 70, 65, 128)	73856
block2_conv2 (Conv2D)	(None, 70, 65, 128)	147584
block2_pool (MaxPooling2D)	(None, 35, 32, 128)	0
block3_conv1 (Conv2D)	(None, 35, 32, 256)	295168
block3_conv2 (Conv2D)	(None, 35, 32, 256)	590080
block3_conv3 (Conv2D)	(None, 35, 32, 256)	590080
block3_pool (MaxPooling2D)	(None, 17, 16, 256)	0
block4_conv1 (Conv2D)	(None, 17, 16, 512)	1180160
block4_conv2 (Conv2D)	(None, 17, 16, 512)	2359808
block4_conv3 (Conv2D)	(None, 17, 16, 512)	2359808
block4_pool (MaxPooling2D)	(None, 8, 8, 512)	0
block5_conv1 (Conv2D)	(None, 8, 8, 512)	2359808
block5_conv2 (Conv2D)	(None, 8, 8, 512)	2359808
block5_conv3 (Conv2D)	(None, 8, 8, 512)	2359808
block5_pool (MaxPooling2D)	(None, 4, 4, 512)	0
flatten (Flatten)	(None, 8192)	0
dense (Dense)	(None, 1)	8193

```

=====
Total params: 14722881 (56.16 MB)
Trainable params: 8193 (32.00 KB)
Non-trainable params: 14714688 (56.13 MB)

```

Abbildung 26: VGG16 Transfer Learning Model Zusammenfassung

<sup>42</sup> VGG16 kann 1000 Bilder klassifizieren (=1000 Klassen), u. a. Szenen, Objekte oder Körperteile.

In der Model-Übersicht mit dem Transfer von VGG16 werden die potenziellen Vorteile des Transfer Learning sichtbar. Lediglich 8193 Parameter müssen trainiert werden, während über 14.7 Millionen bereits trainierte Parameter aus VGG16 zur Verfügung stehen.

### **Epochen und Early Stopping**

Beiden Modellen wurde zusätzlich eine Methode implementiert, um den Verlustwert der Test- bzw. Validierungsdaten während des Trainings zu überwachen. Bei einer binären Klassifizierung liegt der Verlustwert zwischen 0 und 1 und ist eine wichtige Kennzahl zur Leistungsbeurteilung des Trainingsvorgangs des Modells. Er gibt an, inwieweit die vorhergesagten Klassen von den tatsächlichen Klassen abweichen.

Mit der Early Stopping Methode wurde sichergestellt den Trainingsprozess vorzeitig zu beenden, wenn der Verlustwert (Loss) der Validierungsdaten über eine definierte Anzahl von Epochen (Trainingsdurchläufe) hinweg nicht weiter sinkt oder sogar steigt. Zum einen bringt dies eine Zeitersparnis, da frühzeitig unterbrochen wird, wenn weitere Epochen keine Verbesserung mehr bewirken, zum anderen kann man damit ebenfalls Overfitting entgegensteuern. Sollte der Verlustwert der Trainingsdaten weiterhin sinken, während der Verlustwert der Testdaten gleichbleibt oder steigt, ist das Model am Ende zu sehr an die Trainingsdaten angepasst und damit nicht generalisierbar auf unbekannte Daten. Early Stopping wurde bei beiden Modellen mit „patience=10“ implementiert, was einen vorzeitigen Trainingsabbruch nach 10 aufeinanderfolgenden Epochen mit gleichbleibenden oder steigenden Verlustwert der Testdaten bewirkte. Der Wert wurde bewusst höher angesetzt, da die Modelle während des Trainings über einige Epochen hinweg schlechtere Performance aufweisen können und diese sich dann im Verlauf des weiteren Trainings wieder verbessert.

Die Gesamtanzahl der Epochen wurde auf 100 gesetzt. Die folgenden Abbildungen repräsentieren die Änderungen der Verlustwerte für die Trainings- und Testdaten im Verlauf des Trainingsprozesses.

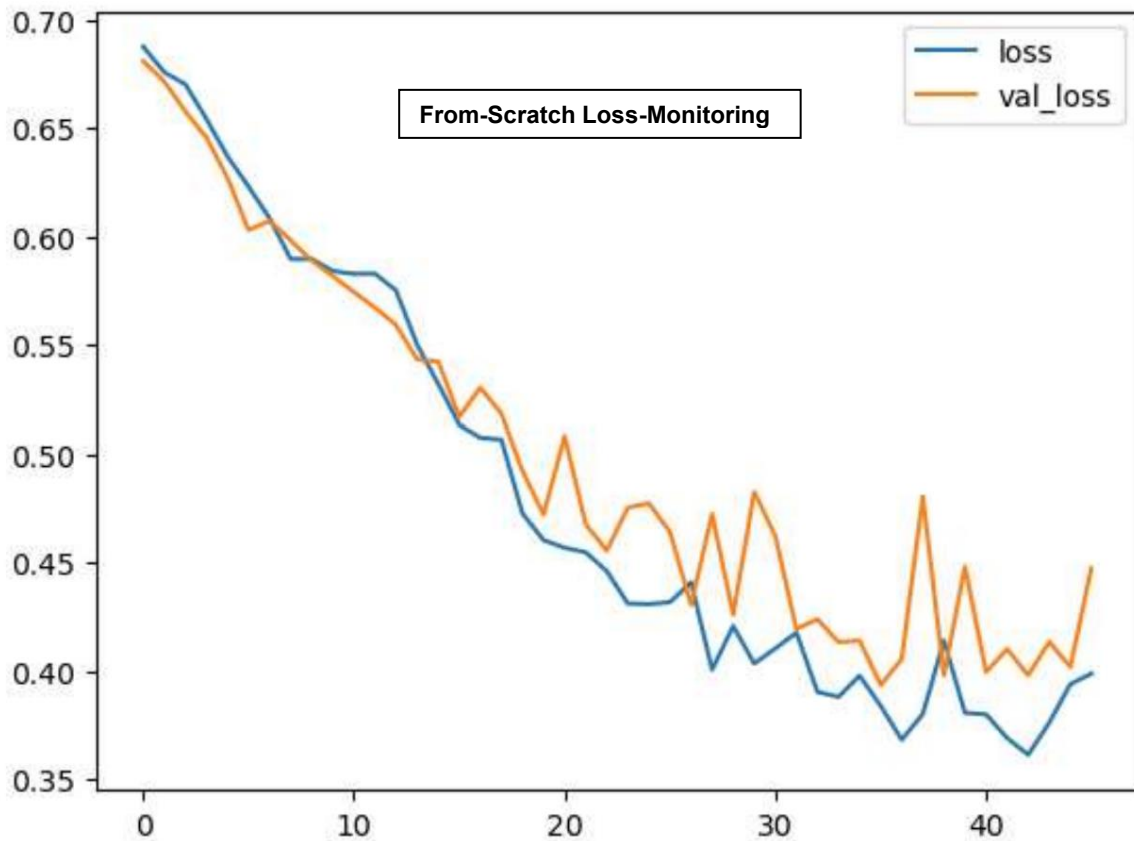


Abbildung 27: Verlustwerte im Verlauf der Epochen beim From-Scratch Model

Im obigen Diagramm des From-Scratch Trainings wird deutlich, dass „val\_loss“ (Verlustwert der Validierungs- bzw. Testdaten) im Verlauf des Trainings über einige Epochen hinweg vorübergehend steigt (das Modell also schlechter performt), sich dann aber wieder dem „loss“ Wert (Verlustwert der Trainingsdaten) annähert. Erkennbar wird auch, dass das Training schließlich nach der 46. Epoche beendet wird, da der val\_loss Wert sich seit der 36. Epoche nicht verbessert hat.

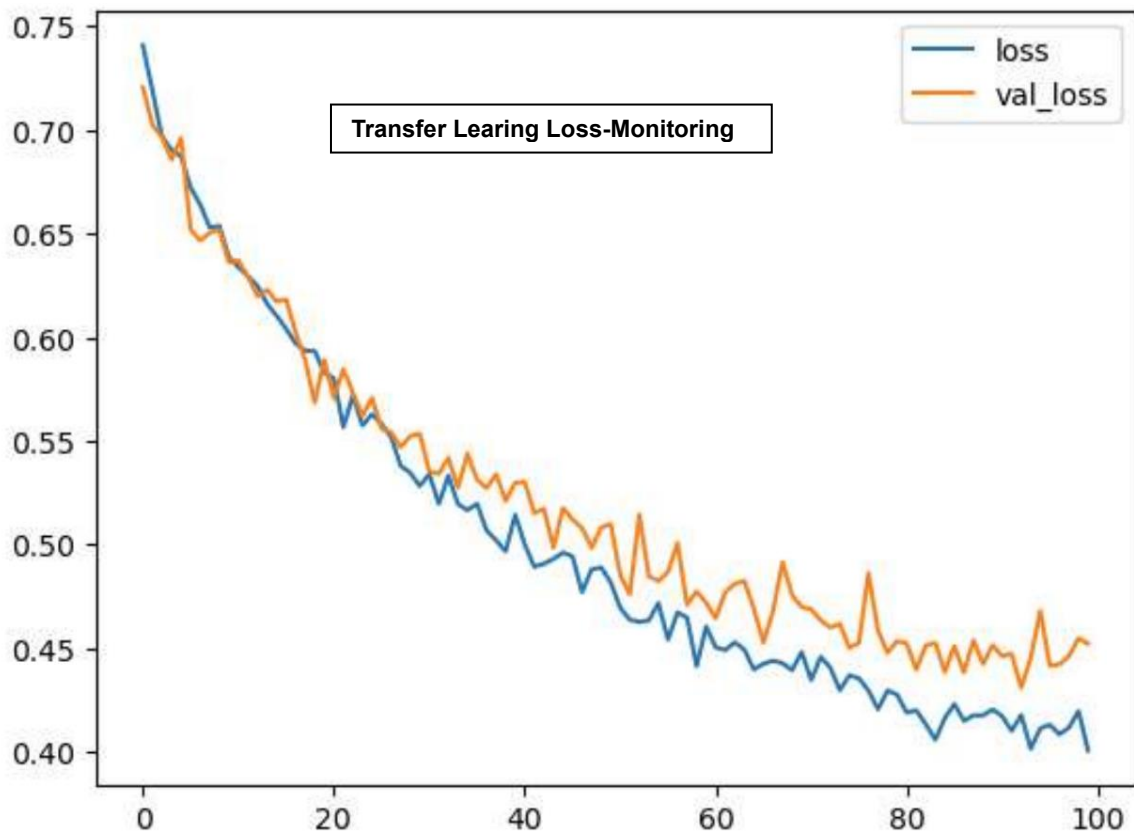


Abbildung 28: Verlustwerte im Verlauf der Epochen beim Transfer Learning Model

Im Loss-Monitoring Diagramm für das Transfer Learning Training lassen sich ebenfalls Schwankung des val\_loss Wertes beobachten. Ein vorzeitiger Abbruch des Trainings erfolgt nicht, da sich der Wert innerhalb von 10 Epochen immer wieder verbesserte. Anzumerken ist auch, dass sich der loss Wert ab der 82. Epoche nicht mehr signifikant verbessert. Insofern war eine Gesamtanzahl von 100 Epochen für eine erste Model-Evaluierung ausreichend.

## 5.4 Auswertung der Ergebnisse

Die im Folgenden vorgestellten Ergebnisse repräsentieren Tendenzen zur Performance der verwendeten Modelarchitekturen (From-Scratch und Transfer Learning) in Bezug auf einige oben beschriebene getestete Parameter-Variationen. Die Möglichkeiten der Anpassungen wurden hier nicht ausgeschöpft, insbesondere was die Anzahl und Kombinationen der Layer und die Größe der Filtermatrizen anbetrifft.

Die größten Herausforderungen brachte das From-Scratch Model mit sich. Hier sollte zumindest eine Vorhersagegenauigkeit von ca. 75% erreicht werden. Dies ist in den meisten Modellen mit binärer Klassifizierung ein guter Anfangswert. Zu beachten ist in dem Zusammenhang, dass eine Vorhersagegenauigkeit bei binär klassifizierenden Modellen von 50% praktisch nutzlos ist, da es einem Raten gleichkäme.

Für das Transfer Learning Model wurde eine Vorhersagegenauigkeit von mindestens 80% angestrebt. Dieser Wert war realistisch erreichbar, da bereits das erste Training (ohne angepasster Learning Rate und geringer Early Stopping Patience) ein Ergebnis nahe der 80% lieferte.

Eines der größten Probleme war die Anzahl der verfügbaren Trainingsdaten, da dies ebenfalls ein Overfitting insbesondere des From-Scratch Models begünstigt. Dennoch boten die 500 Bilder je Klasse (real/fake) eine solide Basis für die Durchführung des Experiments. Dies zeigt sich vor allem daran, dass das Austesten von verschiedenen Parametern und Layer-Designs signifikante Verbesserungen der Vorhersagegenauigkeit bewirken.

## Trainingsdauer

In der folgenden Übersicht ist die Trainingsdauer im Verhältnis zu der Anzahl der Model-Parameter<sup>43</sup> beider Modelle in ihrer finalen Konfiguration dargestellt. Obwohl das From-Scratch Model weniger als die Hälfte aller Epochen durchlaufen hat, läuft es auch unter Einbeziehung der übersprungenen Epochen fast doppelt so schnell wie das Transfer Learning Model. Einen erheblichen Einfluss auf die Dauer haben hierbei die Model-Parameter. Obwohl die trainierbaren Parameter den größten Einfluss haben, ist hier die Gesamtanzahl entscheidend. Während es ca. 2.9 Millionen im From-Scratch Model waren, war die Anzahl mit ca.14.7 Millionen im Transfer Learning Model fast fünfmal so hoch.

Modelvariante	Durchlaufene Epochen	Model-Parameter gesamt	Trainierbare Model-Parameter	Trainingsdauer
From-Scratch	46/100	2.884161	2.884161	0:07:50
Transfer Learning	100/100	14.722881	8193	0:39:48

## Accuracy, Precision und Recall

Für die Modelle ergaben sich folgende Accuracy-Werte:

**From-Scratch Model:** 0,81 (81%)

**VGG16 Transfer Learning Model:** 0,83 (83%)

Der Accuracy-Wert (Genauigkeitswert) ist insbesondere bei binär klassifizierenden Modellen mit Vorsicht zu genießen. So kann man bei der Evaluierung des Modells theoretisch auf eine Gesamtgenauigkeit von 75% kommen, obwohl das Model eine der beiden Klassen überhaupt nicht vorhersagen kann, also für eine Klasse auf einen Accuracy-Wert von 0,5 (50%) verharrt, während es die andere Klasse zu 100% richtig vorhersagt. In diesem Fall wäre das Model praktisch nutzlos, obwohl eine Gesamtgenauigkeitswert von 75% impliziert, dass in nur 25% der Fälle Klassen falsch vorhergesagt werden.

---

<sup>43</sup> „Model-Parameter“ meint hier die Gewichte und Biases der Neuronen innerhalb des neuronalen Netzes.

Die Accuracy-Formel setzt sich aus den falschen und richtigen Vorhersagen der Klassen zusammen:

$$Accuracy = \frac{\text{korrekt positiv (TP)} + \text{korrekt negativ (TN)}}{TP + TN + \text{falsch positiv (FP)} + \text{falsch negativ (FN)}}$$

Für eine bessere Beurteilung des Modells ist es notwendig, sich zusätzlich die Precision-Werte (Präzisionswerte) aufzuschlüsseln. Diese sind in den folgenden Tabellen dargestellt.

*Precision From-Scratch Model:*

	precision	recall	f1-score	support
0	0.93	0.67	0.78	150
1	0.74	0.95	0.83	150
accuracy			0.81	300
macro avg	0.84	0.81	0.81	300
weighted avg	0.84	0.81	0.81	300

*Precision VGG16 Transfer Learning Model:*

	precision	recall	f1-score	support
0	0.86	0.80	0.83	150
1	0.81	0.87	0.84	150
accuracy			0.83	300
macro avg	0.83	0.83	0.83	300
weighted avg	0.83	0.83	0.83	300

Der Präzisionswert gibt an, welcher Anteil der als fake (0) oder real (1) klassifizierten Hände tatsächlich fake oder real sind. Das From-Scratch Model erzielte für die korrekte Vorhersage von gefälschten Bildern eine sehr hohe Präzision von 93%, während reale Bilder nur zu 74% korrekt klassifiziert wurden.



$$Precision = \frac{correct\ predicted}{correct\ predicted + incorrect\ predicted}$$

Hier ist zu beachten, dass die isolierte Betrachtung der jeweilige Precision-Werte keine verlässliche Aussagekraft liefert, da in dessen Berechnung nicht das Verhältnis der Gesamtanzahl der verfügbaren Testdaten berücksichtigt wird. Diese liegt bei 150 je Klasse.

Der Recall-Wert hingegen bezieht die Gesamtanzahl der verfügbaren Daten je Klasse mit ein. Dieser unterscheidet sich je Klasse signifikant mit 0.67 für die fake-Klasse und 0.95 für die real-Klasse.

$$Recall = \frac{correct\ predicted}{total\ number\ of\ instances\ in\ dataset}$$

Für die Interpretation der Präzision kann auch der F1-Score herangezogen werden. Er kombiniert Precision und Recall.

Deutlich ausgewogener sind die Werte beim VGG16 Transfer Learning Model. Precision, Recall und F1-Score liegen alle über 0.8. Dies zeigt, dass das Model über beide Klassen hinweg zu über 80% korrekt klassifiziert. Demnach ist dieses Model das eindeutig performanter und dem From-Scratch Model zu bevorzugen.

## Confusion Matrix

Die Confusion Matrix stellt die absoluten Werte der vorhergesagten Klassen gegenüber. In den grünen Zellen befinden sich die korrekt vorhergesagten Klassen, während in den grauen Zellen die falsch vorhergesagten abgebildet sind. Auch hier wird wieder die bessere Performance des Transfer Learning Modells sichtbar. Echte Bilder werden weitaus weniger als Fakes klassifiziert als im From-Scratch Model.

*From-Scratch Model:*

		Actual	
		Real	Fake
Predicted	Real	101	49
	Fake	8	142

*VGG16 Transfer Learning Model:*

		Actual	
		Real	Fake
Predicted	Real	120	30
	Fake	20	130

## 6. Fazit

Nachdem das Jahr 2023 als das Jahr der künstlichen Intelligenz ausgerufen wurde (Buxmann 2023) ist es für die digitale Forensik jetzt längst an der Zeit sich diesen Umstand anzupassen und KI – insbesondere Deep Learning – in den forensischen Werkzeugkasten mitaufzunehmen. Die Enttarnung und Bekämpfung von Deepfakes können effektiv nur mit Hilfe von KI-Modellen erfolgen, da insbesondere Deepfake-Bilder zu jedem Thema innerhalb von Minuten generiert und in sozialen Netzwerken verbreitet werden können.

Der hier vorgestellte Deep Learning Ansatz, der Hände als Biomarker heranzieht, um personendarstellende Deepfake-Bilder zu entlarven, bietet eine mögliche Grundlage für die Erweiterung des forensischen Werkzeugkastens. Insbesondere die Testergebnisse des Transfer Learning Modells sind vielversprechend, da reale und gefälschte Bilder mit einer über 80-prozentigen Genauigkeit erkannt wurden. Die Genauigkeit lässt sich wahrscheinlich noch erhöhen, indem man das Model mit vielen weiteren Bildern trainiert und weiter die Convolutional Layer optimiert.

Da sich die digitale Forensik immer in einer Art KI Katz-und-Maus Spiel befinden wird, plädiere ich für pragmatische Ansätze, die sich auf die jeweilige Schwächen von Deepfake-Generatoren konzentrieren. Hierzu sollten insbesondere merkmalsbasierte Detektoren zum Einsatz kommen, da bestimmte Bildmotive (noch) nicht ausreichend realistisch gefälscht werden können. Denkbar wäre auch eine Kombination von merkmalsbasierten und generalistischen Ansätzen. In jedem Fall scheint es aufgrund der rasanten Entwicklung ratsam, flexible Deep Learning Ansätze zu entwickeln, ohne dabei den Anspruch zu erheben einen generalistischen Detektor zu erschaffen, der in der Lage ist alle Deepfake-Bilder motivunabhängig zu erkennen, da insbesondere fotorealistische Fälschungen forensische Relevanz haben.

## 7. Ausblick

Ziemlich genau ein Jahr nachdem Diffusions-basierte Text-to-Image Bilder uns mit fotorealistischen Fälschungen vom Papst und Donald Trump überrascht haben, hat OpenAI im Februar 2024 „Sora“<sup>44</sup> vorgestellt. Sora kann anhand von Text-Prompts fotorealistische Videos generieren („Text-to-Video“). Damit ist zu erwarten, dass Falsch- und Desinformationen in Zukunft auch mit Videoclips verbreitet werden. Es bleibt abzuwarten, welchen Einfluss diese Videos auf Falschmeldungen und Propaganda haben werden und ob der Effekt stärker sein wird als über Deepfake-Bilder. Sicher ist nur, dass IT-Forensiker immer mehr zu tun haben werden, während sich soziale Netzwerke zunehmend mit kaum mehr wahrnehmbaren Foto- und Videofälschungen füllen. Immerhin haben die Text-to-Video Deepfakes das gleiche Problem wie die Text-to-Image Bilder: die Hände sind häufig anatomisch inkorrekt dargestellt.



Abbildung 29: Ausschnitt aus einem Text-to-Video Clip von Sora mit anatomisch inkorrekt Handdarstellung<sup>45</sup>

<sup>44</sup> <https://openai.com/sora>

<sup>45</sup> Ebd.

## Literaturverzeichnis

„AI Image Generation Explained: Techniques, Applications, and Limitations“  
(2023): In: alexsoft.com (10.07. 2023). <https://www.altexsoft.com/blog/ai-image-generation> [27.12. 2023]

**Alkishri, Wasin / H. Yousif, Jabar / Mahmood, Al-Bahri (2023):**

Deepfake Image Detection Methods using Discret Fournier Transform Analysis and Convolutional Neural Network.

[https://www.researchgate.net/publication/370592004\\_DEEPPFAKE\\_IMAGE\\_DETECTION\\_METHODS\\_USING\\_DISCRETE\\_FOURIER\\_TRANSFORM\\_ANALYSIS\\_AND\\_CONVOLUTIONAL\\_NEURAL\\_NETWORK](https://www.researchgate.net/publication/370592004_DEEPPFAKE_IMAGE_DETECTION_METHODS_USING_DISCRETE_FOURIER_TRANSFORM_ANALYSIS_AND_CONVOLUTIONAL_NEURAL_NETWORK)

**Bischoff, Manon (2023):** Wie lassen sich KI-generierte Bilder enttarnen? In: spektrum.de.

<https://www.spektrum.de/news/deepfake-wie-lassen-sich-ki-generierte-bilder-enttarnen/2127222> [02. 01. 2024]

**Brownlee, Jason (2019):** A Gentle Introduction to Pooling Layers for Convolutional Neural Networks. In: machinelearningmastery.com (05. 07. 2019).

<https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks> [19. 12. 2023]

**Buxmann, Peter (2023):** 2023 – Das Jahr der künstlichen Intelligenz. In: faz.netz (26. 12. 2023). <https://www.faz.net/pro/d-economy/kuenstliche-intelligenz/das-jahr-der-kuenstlichen-intelligenz-19407937.html> [04. 01. 2024]

**Castelvecchi, Davide (2016):** Eine tückische Blackbox. In: spektrum.de (16. 11. 2016).

<https://www.spektrum.de/news/eine-tueckische-blackbox/1429906> [18.12. 2023]

**Chollet, Francois (2016):** Building powerful image classification models using very little data. In: keras.io (05. 06. 2016). URL: <https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html> [13. 10. 2023]

**Cleve, Jürgen/Lämmel, Uwe (De Gruyter Verlag 2020):** Data Mining, 3. Auflage, Berlin/Boston

**Cuthbertson, Anthony (2023):** AI has finally figured out how to draw hands – removing the one easy way to identify deepfake images. In: independent.co.uk (28. 03. 2023). <https://www.independent.co.uk/tech/ai-hands-midjourney-deepfake-images-b2308670.html> [09. 02. 2024]

**Delaney, Conrad (2023):** A Machine Learning Approach to Deepfake Detection. <https://cornerstone.lib.mnsu.edu/cgi/viewcontent.cgi?article=1005&context=undergrad-theses-capstones-all>

**Eckert, Svea (2023):** ChatGPT - wichtige Fragen und Antworten zur KI-App. In: ndr.de (15.03.2023). <https://www.ndr.de/ratgeber/ChatGPT-wichtige-Fragen-Antworten-zur-KI-App,chatgpt138.html> [10.12.2023]

**Edwards, Benj (2023):** Among AI dangers, deepfakes worry Microsoft president most. In: arstechnica.com (26. 05. 2023). <https://arstechnica.com/information-technology/2023/05/microsoft-president-declares-deepfakes-biggest-ai-concern> [20. 12. 2023]

**Géron, Aurélien (O`Reilly Verlag 2019):** Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2. Auflage, Sebastopol CA

**„Jaccard-Koeffizient“ (2023):** In: Wikipedia – Die freie Enzyklopädie. (29. 09. 2023). <https://de.wikipedia.org/w/index.php?title=Jaccard-Koeffizient&oldid=237730426> [02. 02. 2024]

**Kirchner, Matthias (o. J.):** Digitale Forensik – Spuren in Digitalfotos. [https://fahrplan.events.ccc.de/congress/2006/Fahrplan/attachments/1117-23C3\\_Kirchner.pdf](https://fahrplan.events.ccc.de/congress/2006/Fahrplan/attachments/1117-23C3_Kirchner.pdf)

**Linde, Helmut (2023):** So funktionieren KI-Bildgeneratoren. In: golem.de (30. 05. 2023). <https://www.golem.de/news/kuenstliche-intelligenz-so-funktionieren-ki-bildgeneratoren-2305-174436.html> [10. 01. 2024]

**Lubner, Stefan (2021):** Was ist ein Generative Adversarial Network (GAN)? In: Bigdata-insider.de (19. 02. 2021). <https://www.bigdata-insider.de/was-ist-ein-generative-adversarial-network-gan-a-999817> [27. 12. 2023]

**Mahapatra, Sambit (2018):** Why Deep Learning over Traditional Machine Learning? In: towardsdatascience.com (21. 05. 2018). <https://towardsdatascience.com/why-deep-learning-is-needed-over-traditional-machine-learning-1b6a99177063> [18. 12. 2023]

**Mazaheri, Ghazal / K. Roy-Chowdhury, Amit (2022):** Detection and Localization of Facial Expression Manipulations. [https://openaccess.thecvf.com/content/WACV2022/papers/Mazaheri\\_Detection\\_and\\_Localization\\_of\\_Facial\\_Expression\\_Manipulations\\_WACV\\_2022\\_paper.pdf](https://openaccess.thecvf.com/content/WACV2022/papers/Mazaheri_Detection_and_Localization_of_Facial_Expression_Manipulations_WACV_2022_paper.pdf)

**Mester, Lukas (2023):** Artificial Intelligence vs Machine Learning vs Deep Learning- Was ist der Unterschied? In: iterac.com (28. 03. 2023). <https://explore.iteratec.com/blog/was-unterscheidet-artificial-intelligence-machine-learning-und-deep-learning> [13.12. 2023]

**Munding, Konrad (2021):** Visual Transformers: Wie eine NLP-Architektur auf Computer Vision angewendet wird. In: dida.do (05. 07. 2021). <https://dida.do/de/blog/visual-transformers> [11.02.2024]

**M. Haralick, Robert / Shanmugam, K. / Dinstein, Its`hak (1973):** Textural Features for Image Classification. <https://haralick.org/journals/TexturalFeatures.pdf>

**Oldenburg, Reinhard (Franzbecker 2006):** Die Mathematik der Bildverarbeitung. Hildesheim/Berlin. <https://d-nb.info/1225683793/34>

**Oppermann, Artem (o. J.):** Overfitting und Underfitting in neuronalen Netzen. In: artemoppermann.com (o. J.). URL: <https://artemoppermann.com/de/overfitting-und-underfitting-in-deep-learning> [05. 12. 2023]

**Oszynowicz, Aleksandra (2023):** Diffusionsmodelle: Eindrucksvolle Bilder mit Generative KI erstellen. In: theblue.ai (25. 04. 2023). <https://theblue.ai/blog-de/trends/diffusionsmodelle-generative-ki> [10. 01. 2024]

**Pathrak, Amrita (2022):** Convolutional Neural Networks (CNNs): Eine Einführung. In: geekflare.com (30. 08. 2022). <https://geekflare.com/de/convolutional-neural-networks> [20. 12. 2023]

**Pietras, Jake (2020):** KI, Blockchain, IoT – wie Buzzwords Produkte verkaufen. In: t3n.de (18.07.2020). <https://t3n.de/news/ki-blockchain-iot-buzzwords-1300368> [13.12. 2023]

**Plank, Barbara (2023):** ChatGPT ist der Beginn einer neuen NLP-Ära – aber es ist noch ein langer Weg. In: lmu.de. (20.02.2023). <https://www.lmu.de/de/newsroom/newsuebersicht/news/chatgpt-ist-der-beginn-einer-neuen-nlp-aera---aber-es-ist-noch-ein-langer-weg.html> [09.12.2023]

**Pramoditha, Rukshan (2021):** 11 Dimensionality reduction techniques you should know in 2021. In: towardsdatascience.com (14. 04. 2021). <https://towardsdatascience.com/11-dimensionality-reduction-techniques-you-should-know-in-2021-dcb9500d388b> [30. 01. 2024]

**Pu, Jiameng / Mangaokar, Neal / Wang, Bolun / K. Reddy, Chandan / Viswanath, Bimal (2020):** NoiseScope: Detecting Deepfake Images in a Blind Setting. <https://people.cs.vt.edu/~reddy/papers/ACSAC20.pdf>

**Rouse, Margaret (2018):** Deconvolutional Neural Network. In: techopedia.com (11. 04. 2018). <https://www.techopedia.com/definition/33290/deconvolutional-neural-network-dnn> [30.12. 2023]



**Schmidt-Colberg, Marius (Whitepaper 2021):** Generative Adversarial Networks (GAN): Eine Übersicht und Beispiele zum Thema GAN. [https://ai.hdm-stuttgart.de/downloads/student-white-paper/Sommer-2021/Ubbersicht\\_Generative\\_Adversarial\\_Networks.pdf](https://ai.hdm-stuttgart.de/downloads/student-white-paper/Sommer-2021/Ubbersicht_Generative_Adversarial_Networks.pdf)

**Sester, Sarah (2022):** Real-time Computer Vision: Gesichter erkennen mit einem Roboter. In: statworx.com (30. 11. 2022). <https://www.statworx.com/content-hub/blog/real-time-computer-vision-gesichter-erkennen-mit-einem-roboter> [08. 02. 2024]

**Shu, Hu / Li, Yuezun / Lyu, Siwei (2020):** Exposing GAN-Generated Faces Using Inconsistent Corneal Specular Highlights. <https://arxiv.org/pdf/2009.11924.pdf>

**Stadler, Max-Ludwig (2021):** Convolutional Neural Network (CNN). In: Mindsquare.de (29. 07. 2021). <https://mindsquare.de/knowhow/convolutional-neural-network> [24. 12. 2023]

**Steinbrecher, Rainer (Oldenbourg Verlag 2005):** Bildverarbeitung in der Praxis. München, Deutschland. <https://www.rst-software.com/dbv/DBV-Buch1.PDF>

**„Sobel-Operator“ (2022):** In: Wikipedia – Die freie Enzyklopädie. (17. 12. 2022). <https://de.wikipedia.org/w/index.php?title=Sobel-Operator&oldid=228942162> [20. 12. 2023]

**Tiedemann, Michaela (2018):** KI, künstliche neuronale Netze, Machine Learning, Deep Learning: Wir bringen Licht in die Begriffe rund um das Thema „Künstliche Intelligenz“. In: alexanderthamm.com (29. 05. 2018). [https://www.alexanderthamm.com/de/blog/ki\\_artificial-intelligence-ai-kuenstliche-neuronale-netze-machine-learning-deep-learning/](https://www.alexanderthamm.com/de/blog/ki_artificial-intelligence-ai-kuenstliche-neuronale-netze-machine-learning-deep-learning/) [17.12. 2023]

**Wang, Zhendong / Bao, Jianmin / Zhou, Wengang / Wang, Weilun / Hu, Hezhen / Chen, Hong / Li, Houqiang (2023):**

DIRE for Diffusion-Generated Image Detection.

[https://openaccess.thecvf.com/content/ICCV2023/papers/Wang\\_DIRE\\_for\\_Diffusion-Generated\\_Image\\_Detection\\_ICCV\\_2023\\_paper.pdf](https://openaccess.thecvf.com/content/ICCV2023/papers/Wang_DIRE_for_Diffusion-Generated_Image_Detection_ICCV_2023_paper.pdf)

**„Wavelet“ (2024):** In: Wikipedia – Die freie Enzyklopädie (02.01. 2024).

<https://de.wikipedia.org/w/index.php?title=Wavelet&oldid=240758249>

[09. 01. 2024]

**Wesolowski, Kathrin / Weber, Joscha / Sparrow, Thomas (2023):** Faktencheck:

Wie erkenne ich KI-generierte Bilder? In: dw.com (09. 04. 2023).

<https://www.dw.com/de/faktencheck-wie-erkenne-ich-ki-generierte-bilder/a-65252413> [17. 12. 2023]

**Wuttke, Laurenz (2023):** Big Data: einfach erklärt! Definition und Technologie. In

[datasolut.com](https://datasolut.com) (02. 06 .2023). <https://datasolut.com/was-ist-big-data> [15.12. 2023]

**Wuttke, Laurenz (2023):** Natural Language Processing (NLP): Funktionen, Aufgaben und Anwendungsbereiche. In [datasolut.com](https://datasolut.com) (11. 12. 2023).

<https://datasolut.com/natural-language-processing-einfuehrung> [11.01. 2024]

## Abkürzungsverzeichnis

<b>CDN</b>	Content Delivery Network
<b>ChatGPT</b>	Chat Generative Pre-trained Transformer
<b>CNN</b>	Convolutional Neural Network
<b>DIRE</b>	Diffusion Reconstruction Error
<b>DL</b>	Deep Learning
<b>DNN</b>	Deconvolutional Neural Network
<b>FER</b>	Facial Expression Recognition
<b>GAN</b>	Generative Adversarial Network
<b>IoU</b>	Intersection over Union
<b>KI</b>	Künstliche Intelligenz
<b>ML</b>	Machine Learning
<b>PCE</b>	Peak to Correlation Energy

## Abbildungsverzeichnis

Abbildung 1: Schichtenmodell Neuronale Netze.....	7
Abbildung 2: Inkrementelle Mustererkennung.....	8
Abbildung 3: Kantendetektion mit Sobel Operator.....	10
Abbildung 4: Downsampling.....	11
Abbildung 5: CNN mit Fully Connected Layer.....	11
Abbildung 6: Trumps Verhaftung als Deepfake-Bild.....	13
Abbildung 7: GAN-Architektur Erzeugung von Gesichtern .....	14
Abbildung 8: Erstelltes Testbild über den Dall-E Diffusionsgenerator mit folgender Szenenvorgabe: „Ein Forensiker analysiert ein Bild auf Twitter, welches ein Präsident zeigt, der eine Kapitulation unterzeichnet.“ .....	15
Abbildung 9: Angebliches von Trümmern verletztes Baby im Gaza .....	18
Abbildung 10: Gefälschte Aufnahme von Soldaten und Panzern der israelischen Armee in einer zerstörten Stadt (vorgeblich Gaza).....	19
Abbildung 11: NoiseScope Detektor nach Pu et. al. ....	21
Abbildung 12: DIRE-Repräsentation von realen Bildern und deren synthetischer Version .....	23
Abbildung 13: Fournier-Transformation und Klassifizierung .....	25
Abbildung 14: FER und Manipulationsdetektion für Deepfake-Erkennung bei Gesichtern .....	26
Abbildung 15: Hornhaut-Reflexionen GAN-erzeugte und reale Gesichter .....	28
Abbildung 16: Streitender Papst, Diffusions-synthetisiert über Midjourney .....	32
Abbildung 17: Donald Trump und Joe Biden beim Teetrinken, Diffusions-synthetisiert über Midjourney.....	32
Abbildung 18: Beispielbilder von Personen aus dem "flickr30k_images" Datensatz.....	34
Abbildung 19: Extrahierte fotorealistische Midjourney-Bilder, die Hände zeigen.....	36
Abbildung 20: Selbstversuch Handerkennung via MediaPipe inklusive der Darstellung der Landmarker .....	37
Abbildung 21: Hand-Landmarker und die Bestimmung des Handausschnitts (ohne 10% Erweiterung).....	38
Abbildung 22: Ablaufdiagramm zur Datenbeschaffung und Aufbereitung der Midjourney-Bilder ....	39
Abbildung 23: Extrahierte Hände aus den Midjourney-generierten Bildern.....	40
Abbildung 24: Beispiele aus dem Keras Image-Generator.....	44
Abbildung 25: From-Scratch Model Zusammenfassung.....	46
Abbildung 26: VGG16 Transfer Learning Model Zusammenfassung .....	47
Abbildung 27: Verlustwerte im Verlauf der Epochen beim From-Scratch Model .....	49
Abbildung 28: Verlustwerte im Verlauf der Epochen beim Transfer Learning Model.....	50
Abbildung 29: Ausschnitt aus einem Text-to-Video Clip von Sora mit anatomisch inkorrekt er Handdarstellung .....	57

## **Selbstständigkeitserklärung**

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Wörtliche und sinngemäße Übernahmen wurden durch korrektes Zitieren kenntlich gemacht.

Diese Arbeit wurde weder vollständig noch in Auszügen im Rahmen einer anderen Prüfungsleistung eingereicht.

Berlin, den 27. 02. 2024

Thomas Reimann

# Anhang

## hand\_extractor.py

```
1 import mediapipe as mp
2 import numpy as np
3 import math
4 import cv2
5 from os import listdir, remove
6 from mediapipe.tasks import python
7 from mediapipe.tasks.python import vision
8 from pathlib import Path
9
10 corepath = 'C:/Users/nerdl/PycharmProjects/deepfake_ba/'
11 img_path_real_hands = corepath + 'im_hands_real/'
12 img_path_real_hands_crop = corepath + 'im_hands_real_crop/'
13 img_path_fake_hands = corepath + 'im_hands_fake/'
14 img_path_fake_hands_crop = corepath + 'im_hands_fake_crop/'
15 img_path_real_hands_crop_sort = corepath + 'im_hands_real_crop_sort/'
16 img_path_fake_hands_crop_sort = corepath + 'im_hands_fake_crop_sort/'
17
18
19 def scale_img(img_convert, hand_landmarks=[]):
20     height, width, _ = img_convert.shape
21     #Get all x and y coordinates form hand landmarks into array
22     x_coord = [landmark.x for landmark in hand_landmarks]
23     y_coord = [landmark.y for landmark in hand_landmarks]
24     #Find the min and max of x and y and scale with width and height to revert normalization
25     start_x = int(min(x_coord) * width)
26     start_y = int(min(y_coord) * height)
27     end_x = int(max(x_coord) * width)
28     end_y = int(max(y_coord) * height)
29     #Get the distance of start and end pixel. This is needed calculate the margin
30     start_end_dist = math.dist((start_x, start_y), (end_x, end_y))
31     #Based on the distance copped image will be enlarged by 10 percent
32     margin = int(10 * start_end_dist / 100)
33     start_x = start_x - margin
34     start_y = start_y - margin
35     end_x = end_x + margin
36     end_y = end_y + margin
37
38     #Make sure x and y are not below 0 and not exceed width and height
39     if start_x < 0:
40         start_x = 0
41     if start_x > width:
42         start_x = width
43     if start_y < 0:
44         start_y = 0
45     if start_y > height:
46         start_y = height
47
48     crop_img = img_convert[start_y:end_y, start_x:end_x]
49
50     return crop_img
51
52
```

```

53 def hand_landmarker_detect(img_path, img_path_crop):
54
55     base_options = python.BaseOptions(model_asset_path='models/gesture_recognizer.task')
56     options = vision.GestureRecognizerOptions(base_options=base_options, num_hands=4)
57     recognizer = vision.GestureRecognizer.create_from_options(options)
58     min_shape_crop = (35,35,3)
59
60     for image in listdir(img_path):
61         img = img_path + image
62         try:
63             img_in = mp.Image.create_from_file(img)
64         except:
65             print("Error Image create for detection: " +img)
66
67         detection_result = recognizer.recognize(img_in)
68         if len(detection_result.hand_landmarks) == 0:
69             remove(img)
70
71         for i in range(len(detection_result.hand_landmarks)):
72             hand_landmarks = detection_result.hand_landmarks[i]
73
74             try:
75                 img_convert = cv2.imread(img)
76             except:
77                 print("Error reading: " + img)
78
79             gesture = detection_result.gestures[0][0].category_name
80
81             if not gesture:
82                 continue
83
84             crop_img = scale_img(img_convert, hand_landmarks)
85             crop_img_shape = crop_img.shape
86
87             if crop_img_shape < min_shape_crop:
88                 print("too small")
89                 continue
90             file_path = Path(img)
91             file_extension = file_path.suffix
92             img_hand_split = image.split(file_extension)[0].replace('.', '-') + '-' + str(i) + '-' + str(gesture)
93             str(crop_img_shape[1]) + file_extension
94
95             try:
96                 cv2.imwrite(img_path_crop + img_hand_split, crop_img)
97             except:
98                 print("Write error: " + img_path_crop + img_hand_split)
99
100     return
101

```

```

102 def average_shape(img_path):
103     dim1 = []
104     dim2 = []
105     for image in listdir(img_path):
106         img = cv2.imread(img_path + image)
107         d1, d2, colors = img.shape
108         dim1.append(d1)
109         dim2.append(d2)
110
111     print(str(np.mean(dim1)) + " x " + str(np.mean(dim2)))
112     print(str(np.min(dim1)))
113     print(str(np.min(dim2)))
114
115     return
116
117 average_shape(img_path_real_hands_crop_sort)
118 average_shape(img_path_fake_hands_crop_sort)

```

## Jupyter Notebook Python Code und Ausgabe: From-Scratch Model

```
In [1]: import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.image import imread
from sklearn.metrics import classification_report, confusion_matrix

from keras import optimizers

from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Activation, Dropout, Flatten, Dense, Conv2D, MaxPooling2D
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image

import warnings
warnings.filterwarnings('ignore')

from timeit import default_timer as timer
from datetime import timedelta

In [2]: src_dir = os.path.join('C:\\Users\\nerdl\\PycharmProjects\\deepfake_ba', 'hands')

In [3]: test_path = os.path.join(src_dir, 'test')
train_path = os.path.join(src_dir, 'train')

In [4]: dimension1 = []
dimension2 = []
for image in os.listdir(os.path.join(test_path, 'real')):

    img = imread(os.path.join(test_path, 'real', image))
    d1,d2,colors = img.shape
    dimension1.append(d1)
    dimension2.append(d2)

In [5]: np.mean(dimension1)

Out[5]: 142.35333333333332

In [6]: np.mean(dimension2)

Out[6]: 133.24666666666667

In [7]: img_shape = (140,130,3)

In [8]: image_generator = ImageDataGenerator(rotation_range=20,
                                             width_shift_range=0.10,
                                             height_shift_range=0.10,
                                             rescale=1/255,
                                             shear_range=0.1,
                                             zoom_range=0.1,
                                             horizontal_flip=True,
                                             vertical_flip=True,
                                             fill_mode='constant'
                                             )
```



```
In [9]: model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(5,5),input_shape=img_shape, activation='relu',))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(filters=64, kernel_size=(5,5),input_shape=img_shape, activation='relu',))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(filters=64, kernel_size=(5,5),input_shape=img_shape, activation='relu',))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(filters=64, kernel_size=(5,5),input_shape=img_shape, activation='relu',))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(2048))
model.add(Activation('relu'))

model.add(Dropout(0.5))

model.add(Dense(1))
model.add(Activation('sigmoid'))

opt = optimizers.Adam(learning_rate=1e-04)
model.compile(loss='binary_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])
```

```
In [10]: model.summary()
```

```
Model: "sequential"
-----
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 136, 126, 32)	2432
max_pooling2d (MaxPooling2D)	(None, 68, 63, 32)	0
conv2d_1 (Conv2D)	(None, 64, 59, 64)	51264
max_pooling2d_1 (MaxPooling2D)	(None, 32, 29, 64)	0
conv2d_2 (Conv2D)	(None, 28, 25, 64)	102464
max_pooling2d_2 (MaxPooling2D)	(None, 14, 12, 64)	0
conv2d_3 (Conv2D)	(None, 10, 8, 64)	102464
max_pooling2d_3 (MaxPooling2D)	(None, 5, 4, 64)	0
flatten (Flatten)	(None, 1280)	0
dense (Dense)	(None, 2048)	2623488
activation (Activation)	(None, 2048)	0
dropout (Dropout)	(None, 2048)	0
dense_1 (Dense)	(None, 1)	2049
activation_1 (Activation)	(None, 1)	0

```
-----
Total params: 2884161 (11.00 MB)
Trainable params: 2884161 (11.00 MB)
Non-trainable params: 0 (0.00 Byte)
```

```
In [11]: early_stop = EarlyStopping(monitor='val_loss',patience=10)
```

```
In [12]: batch_size = 256
```

```

In [13]: train_image_gen = image_generator.flow_from_directory(train_path,
                                                             target_size=img_shape[:2],
                                                             color_mode='rgb',
                                                             batch_size=batch_size,
                                                             class_mode='binary')

Found 700 images belonging to 2 classes.

In [14]: test_image_gen = image_generator.flow_from_directory(test_path,
                                                             target_size=img_shape[:2],
                                                             color_mode='rgb',
                                                             batch_size=batch_size,
                                                             class_mode='binary', shuffle=False)

Found 300 images belonging to 2 classes.

In [15]: train_image_gen.class_indices

Out[15]: {'fake': 0, 'real': 1}

In [16]: start = timer()

In [17]: results = model.fit_generator(train_image_gen, epochs=100,
                                       validation_data=test_image_gen,
                                       callbacks=[early_stop])

ccuracy: 0.8200
Epoch 35/100
3/3 [=====] - 10s 3s/step - loss: 0.3978 - accuracy: 0.8329 - val_loss: 0.4140 - val_a
ccuracy: 0.8233
Epoch 36/100
3/3 [=====] - 11s 3s/step - loss: 0.3838 - accuracy: 0.8429 - val_loss: 0.3934 - val_a
ccuracy: 0.8100
Epoch 37/100
3/3 [=====] - 10s 3s/step - loss: 0.3682 - accuracy: 0.8457 - val_loss: 0.4055 - val_a
ccuracy: 0.8133
Epoch 38/100
3/3 [=====] - 10s 4s/step - loss: 0.3801 - accuracy: 0.8443 - val_loss: 0.4803 - val_a
ccuracy: 0.7500
Epoch 39/100
3/3 [=====] - 10s 3s/step - loss: 0.4139 - accuracy: 0.8086 - val_loss: 0.3979 - val_a
ccuracy: 0.8067
Epoch 40/100
3/3 [=====] - 10s 3s/step - loss: 0.3806 - accuracy: 0.8257 - val_loss: 0.4480 - val_a
ccuracy: 0.7733
Epoch 41/100
3/3 [
] 11s 3s/step loss: 0.3799 accuracy: 0.8286 val loss: 0.3994 val a

In [18]: end = timer()
print(timedelta(seconds=end-start))

0:07:50.544498

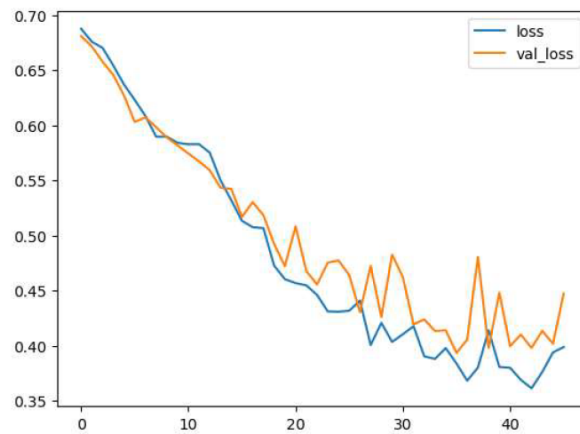
In [19]: model.save('hand_fake_detector.h5')

In [20]: losses = pd.DataFrame(model.history.history)

```

```
In [21]: losses[['loss', 'val_loss']].plot()
```

```
Out[21]: <Axes: >
```



```
In [23]: model = load_model('hand_fake_detector.h5')
```

```
In [24]: test_image_gen.class_indices
```

```
Out[24]: {'fake': 0, 'real': 1}
```

```
In [25]: pred_probabilities = model.predict_generator(test_image_gen)
```

```
In [26]: predictions = pred_probabilities > 0.5
```

```
In [27]: print(classification_report(test_image_gen.classes, predictions))
```

	precision	recall	f1-score	support
0	0.93	0.67	0.78	150
1	0.74	0.95	0.83	150
accuracy			0.81	300
macro avg	0.84	0.81	0.81	300
weighted avg	0.84	0.81	0.81	300

```
In [28]: confusion_matrix(test_image_gen.classes, predictions)
```

```
Out[28]: array([[101, 49],
               [ 8, 142]], dtype=int64)
```

## Jupyter Notebook Python Code und Ausgabe: VGG16 Transfer Learning Model

```
In [1]: import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from matplotlib.image import imread
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Activation, Dropout, Flatten, Dense, Conv2D, MaxPooling2D
from tensorflow.keras.callbacks import EarlyStopping
from keras import optimizers
from tensorflow.keras.models import load_model, Model
from tensorflow.keras.preprocessing import image
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input
from sklearn.metrics import classification_report, confusion_matrix

import warnings
warnings.filterwarnings('ignore')

from timeit import default_timer as timer
from datetime import timedelta

In [2]: src_dir = os.path.join('C:\\Users\\nerd1\\PycharmProjects\\deepfake_ba', 'hands')

In [3]: test_path = os.path.join(src_dir, 'test')
train_path = os.path.join(src_dir, 'train')

In [4]: dimension1 = []
dimension2 = []
for image in os.listdir(os.path.join(test_path, 'real')):

    img = imread(os.path.join(test_path, 'real', image))
    d1,d2,colors = img.shape
    dimension1.append(d1)
    dimension2.append(d2)

In [5]: img_shape = (140,130,3)

In [6]: image_generator = ImageDataGenerator(rotation_range=20,
                                             width_shift_range=0.10,
                                             height_shift_range=0.10,
                                             rescale=1/255,
                                             shear_range=0.1,
                                             zoom_range=0.2,
                                             horizontal_flip=True,
                                             vertical_flip=True,
                                             fill_mode='constant'
                                             )

In [7]: vgg = VGG16(input_shape = img_shape, weights = 'imagenet', include_top = False)

In [8]: for layer in vgg.layers:
        layer.trainable = False
```

```
In [9]: flat_layer = Flatten()(vgg.output)
pred = Dense(1, activation = 'sigmoid')(flat_layer)
model = Model(inputs = vgg.input, outputs = pred)
model.summary()
```

```
Model: "model"
-----
Layer (type)                 Output Shape              Param #
-----
input_1 (InputLayer)        [(None, 140, 130, 3)]    0
block1_conv1 (Conv2D)       (None, 140, 130, 64)     1792
block1_conv2 (Conv2D)       (None, 140, 130, 64)     36928
block1_pool (MaxPooling2D)  (None, 70, 65, 64)       0
block2_conv1 (Conv2D)       (None, 70, 65, 128)      73856
block2_conv2 (Conv2D)       (None, 70, 65, 128)      147584
block2_pool (MaxPooling2D)  (None, 35, 32, 128)      0
block3_conv1 (Conv2D)       (None, 35, 32, 256)      295168
block3_conv2 (Conv2D)       (None, 35, 32, 256)      590080
block3_conv3 (Conv2D)       (None, 35, 32, 256)      590080
block3_pool (MaxPooling2D)  (None, 17, 16, 256)      0
block4_conv1 (Conv2D)       (None, 17, 16, 512)      1180160
block4_conv2 (Conv2D)       (None, 17, 16, 512)      2359808
block4_conv3 (Conv2D)       (None, 17, 16, 512)      2359808
block4_pool (MaxPooling2D)  (None, 8, 8, 512)        0
block5_conv1 (Conv2D)       (None, 8, 8, 512)        2359808
block5_conv2 (Conv2D)       (None, 8, 8, 512)        2359808
block5_conv3 (Conv2D)       (None, 8, 8, 512)        2359808
block5_pool (MaxPooling2D)  (None, 4, 4, 512)        0
flatten (Flatten)           (None, 8192)              0
dense (Dense)                (None, 1)                  8193
-----
Total params: 14722881 (56.16 MB)
Trainable params: 8193 (32.00 KB)
Non-trainable params: 14714688 (56.13 MB)
-----
```

```
In [10]: opt = optimizers.Adam(learning_rate=1e-04)
model.compile(loss='binary_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])
```

```
In [11]: early_stop = EarlyStopping(monitor='val_loss',patience=15)
```

```
In [12]: batch_size = 256
```

```
In [13]: train_image_gen = image_generator.flow_from_directory(train_path,
                                                             target_size=img_shape[:2],
                                                             color_mode='rgb',
                                                             batch_size=batch_size,
                                                             class_mode='binary')
```

Found 700 images belonging to 2 classes.

```
In [14]: test_image_gen = image_generator.flow_from_directory(test_path,
                                                            target_size=img_shape[:2],
                                                            color_mode='rgb',
                                                            batch_size=batch_size,
                                                            class_mode='binary',shuffle=False)
```

Found 300 images belonging to 2 classes.

```

In [15]: train_image_gen.class_indices
Out[15]: {'fake': 0, 'real': 1}

In [16]: start = timer()

In [17]: results = model.fit_generator(train_image_gen, epochs=100,
                                       validation_data=test_image_gen,
                                       callbacks=[early_stop])

Epoch 1/100
3/3 [=====] - 24s 9s/step - loss: 0.7409 - accuracy: 0.4843 - val_loss: 0.7206 - val_a
ccuracy: 0.4867
Epoch 2/100
3/3 [=====] - 23s 9s/step - loss: 0.7192 - accuracy: 0.4800 - val_loss: 0.7026 - val_a
ccuracy: 0.5200
Epoch 3/100
3/3 [=====] - 23s 9s/step - loss: 0.6969 - accuracy: 0.5257 - val_loss: 0.6966 - val_a
ccuracy: 0.5233
Epoch 4/100
3/3 [=====] - 23s 8s/step - loss: 0.6905 - accuracy: 0.5614 - val_loss: 0.6858 - val_a
ccuracy: 0.5567
Epoch 5/100
3/3 [=====] - 24s 8s/step - loss: 0.6873 - accuracy: 0.5757 - val_loss: 0.6964 - val_a
ccuracy: 0.5400
Epoch 6/100
3/3 [=====] - 23s 8s/step - loss: 0.6723 - accuracy: 0.6057 - val_loss: 0.6521 - val_a
ccuracy: 0.6100
Epoch 7/100
3/3 [=====] - 23s 9s/step - loss: 0.6642 - accuracy: 0.6000 - val_loss: 0.6468 - val_a
ccuracy: 0.6100

In [18]: end = timer()
print(timedelta(seconds=end-start))

0:39:48.979625

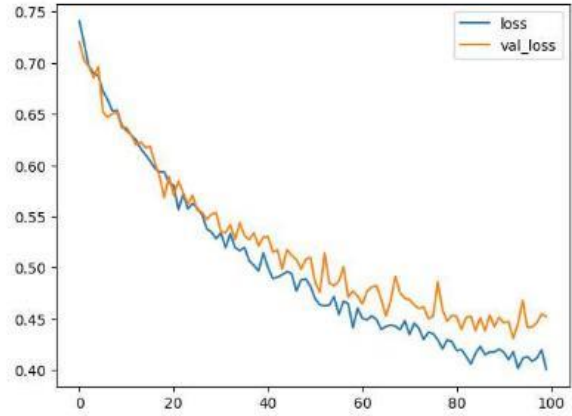
In [19]: model.save('hand_fake_detector_transfer_learning.h5')

In [20]: losses = pd.DataFrame(model.history.history)

In [21]: losses[['loss', 'val_loss']].plot()

Out[21]: <Axes: >

```



```

In [31]: model = load_model('hand_fake_detector_transfer_learning.h5')

In [32]: test_image_gen.class_indices
Out[32]: {'fake': 0, 'real': 1}

In [33]: pred_probabilities = model.predict_generator(test_image_gen)

```

```
In [34]: predictions = pred_probabilities > 0.5
```

```
In [35]: print(classification_report(test_image_gen.classes, predictions))
```

	precision	recall	f1-score	support
0	0.86	0.80	0.83	150
1	0.81	0.87	0.84	150
accuracy			0.83	300
macro avg	0.83	0.83	0.83	300
weighted avg	0.83	0.83	0.83	300

```
In [36]: confusion_matrix(test_image_gen.classes, predictions)
```

```
Out[36]: array([[120, 30],  
               [ 20, 130]], dtype=int64)
```