



# Neurale Netze

Künstliche Intelligenz zur Bildererkennung am Beispiel von Obst mithilfe von paralleler  
Programmierung auf Basis von Grafikarten

Copyright © Sven Fredrik Maibaum – All Rights Reserved.

Unauthorized copying of this file, via any medium is strictly prohibited.

# Inhaltsverzeichnis

<b>EINLEITUNG .....</b>	<b>3</b>
<b>THEORIE.....</b>	<b>5</b>
<b>BIOLOGISCHE NEURONEN .....</b>	<b>5</b>
<i>Aufbau eines Künstlichen Neurons .....</i>	<i>7</i>
<i>Netzarchitektur .....</i>	<i>8</i>
Perceptron .....	10
Feed Forward .....	11
Mehrschichtiges Perceptron .....	12
Convolutional Neural Network .....	14
<i>Lernverfahren .....</i>	<i>14</i>
Verschiedene Lernverfahren .....	14
Forward Propagation .....	15
Backpropagation .....	17
<b>IMPLEMENTIERUNG .....</b>	<b>19</b>
<i>Ziel .....</i>	<i>19</i>
Programmiersprachenwahl .....	19
Implementierung vom Netzwerk .....	20
Warum Convolutional Neural Network? .....	20
Architektur .....	22
Implementierung von Kernel (Filterung) .....	22
Convolutional Filter .....	22
Pooling Layer .....	25
Neurales Netz .....	28
Matrizen .....	31
Verarbeitung von Daten .....	32
Probleme bei der Durchführung .....	35
Ergebnisse .....	35
<i>Zwischenfazit der bisherigen Ergebnisse .....</i>	<i>36</i>
Wurden alle Entwicklungsziele erfüllt? .....	36
Mögliche Optimierungsprozesse .....	37
<b>FAZIT .....</b>	<b>37</b>
<i>Optimierungspotenzial mittels Tensor Cores .....</i>	<i>37</i>
<i>Ausblicke .....</i>	<i>40</i>
<b>LITERATURVERZEICHNIS.....</b>	<b>44</b>
<b>ANHANG .....</b>	<b>45</b>

## Einleitung

Künstliche Intelligenz (KI) ist ein sich ständig weiterentwickelnder Teilbereich der Informatik, der zunehmend an Bedeutung gewinnt. Im Kern geht es darum, ein künstliches menschliches Gehirn zu schaffen, um abstrakte Problemlösungen zu finden und in Zukunft die menschliche Forschung zu ersetzen oder zu unterstützen. Bisher kommen dabei künstliche neuronale Netze (KNN) zum Einsatz, die versuchen in der Bilderkennung, der Sprachverarbeitung und der Vorhersage von Ereignissen komplexe Muster und Zusammenhänge zu erkennen.

Mittlerweile gibt es jedoch schnell wachsende Systeme, wie die großen Sprachmodelle und KI-Generierungssysteme von "Open AI", "ChatGPT" oder "DALLE-E". Anfang dieses Jahres ist zudem auch ChatGPT-4 erschienen, das erste multimodale Modell.

(OpenAI, Introducing ChatGPT, 2021), (OpenAI, DALL-E: Creating Images from Text., 2021)

Ein künstliches neuronales Netz besteht in der Regel aus einer Vielzahl miteinander verbundenen Neuronen, die Informationen ähnlich wie im menschlichen Gehirn verarbeiten. Jedes Neuron verarbeitet einen bestimmten Teil der Eingangsdaten und gibt die verarbeiteten Informationen an das nächste Neuron weiter. Dieser Prozess wiederholt sich bis zum letzten Neuron, wo das mathematische Ergebnis interpretiert und klassifiziert wird. Die Verbindungen zwischen den Neuronen werden durch Gewichte charakterisiert, die während des Lernprozesses angepasst werden, um das neuronale Netz zu optimieren. Künstliche neuronale Netze werden in einer Vielzahl von Anwendungen eingesetzt, darunter wie bereits erwähnt in der Bilderkennung, Spracherkennung und Datenvorhersage.

Neuronale Netze werden in der Regel auf der Grundlage traditioneller Musterverarbeitungs- und Lernverfahren entwickelt. Dabei wird das Netz mit einer großen Menge von Trainingsdaten gefüttert, die es über einen bestimmten Zeitraum analysiert und dabei Muster und Beziehungen lernt. Anschließend kann das Netz ähnliche Daten auf diese Muster hin untersuchen und identifizieren. Es gibt aber auch innovative Lernverfahren wie "Open AI", bei denen zwei KIs gleichzeitig zusammenarbeiten. Eine zuvor trainierte KI generiert zu zufälligen Fragen (ebenfalls von KIs generiert) mögliche

Lösungstexte oder -bilder, die dann von der zweiten KI analysiert und bewertet werden. Je nach Qualität des generierten Ergebnisses passt sich die erste KI aufgrund der Bewertung wieder an und optimiert ihre Netzwerkparameter.

Allerdings kann das Thema künstliche Intelligenz große Probleme für unsere Zukunft darstellen, die wir noch nicht sehen. Einerseits könnte die Weiterentwicklung eines eigenständig lernenden Systems einen grundsätzlichen „Tod“ für unseren Arbeitsmarkt darstellen. Die Weiterentwicklung solcher Systeme wie z.B. von „Open AI“ könnten in Gebieten der freien Erstellung von Content auf lange Sicht die Menschen ersetzen. Des Weiteren besitzen große Systeme keinen richtigen Einblick mehr, um zu erkennen, warum sie sich entscheiden wie sie sich entscheiden. Außerdem müssen wir auch noch hinterfragen, wie diese Systeme einige Situationen bezüglich der Moral einschätzen können, beispielsweise in dem Bereich des Autonomen Fahrens. Die Frage hierbei ist, ob ein Computer im Endeffekt entscheiden kann und soll, ob er beispielsweise in die Bauarbeitergruppe fährt oder in die Kindergartengruppe. Und wem geben wir dann die Schuld für diesen Vorfall? Dem Computer?

Trotz dieser Herausforderungen und moralischen Bedenken spielt künstliche Intelligenz eine zentrale Rolle in der heutigen Informatik und wird in zunehmendem Maße in verschiedenen Bereichen eingesetzt, da sie wie gesagt, eine sehr gute Möglichkeit bietet, komplexe Muster und Zusammenhänge zu erkennen. Des Weiteren kann es sein, dass sie nebenbei auch noch weniger Rechenleistung verbrauchen als ein herkömmliches System.

Zur besseren Ergründung der Prozesse, die den neuronalen Netzwerken zugrunde liegen, beabsichtige ich in meiner Arbeit ein vollständig selbstständiges neuronales Netzwerk zu konstruieren und dessen Leistungsfähigkeit zu verbessern. Dabei versuche ich, anhand von trainierten Daten weitere Bilder zu klassifizieren, um auf diese Weise die Effektivität des Verfahrens darzustellen.

## *Theorie*

### **Biologische Neuronen**

Im Laufe der Geschichte haben zahllose Gelehrte und Wissenschaftler versucht, die komplexe Funktionsweise der biologischen Neuronen zu enträtseln. Diese komplizierten Netzwerke sind in fast allen lebenden Organismen, einschließlich des menschlichen Gehirns, allgegenwärtig und spielen eine wesentliche Rolle bei der Verarbeitung von Sinneseindrücken, der Regulierung der motorischen Aktivität und der Unterstützung der Kognition. Sie sind für ein breites Spektrum kognitiver Vorgänge, vom Lernen und Gedächtnis bis zur Entscheidungsfindung, von grundlegender Bedeutung.

Die Struktur biologischer Neuronen ist komplex und besteht aus mehreren wesentlichen Komponenten, darunter der Zellkörper, die Dendriten und das Axon. Der kernhaltige Zellkörper, das Soma, dient als Kern des Neurons und ist für die Aufrechterhaltung der grundlegenden Zellfunktionen verantwortlich. Die baumartigen Dendriten, die sich in eine Reihe von spezialisierten Fortsätzen verzweigen, empfangen Signale von anderen Neuronen und leiten diese Informationen an den Zellkörper weiter. Das Axon, ein langgestreckter Fortsatz, leitet Signale aus dem Zellkörper an andere Gruppen von Neuronen, Muskelzellen oder Drüsenzellen weiter.

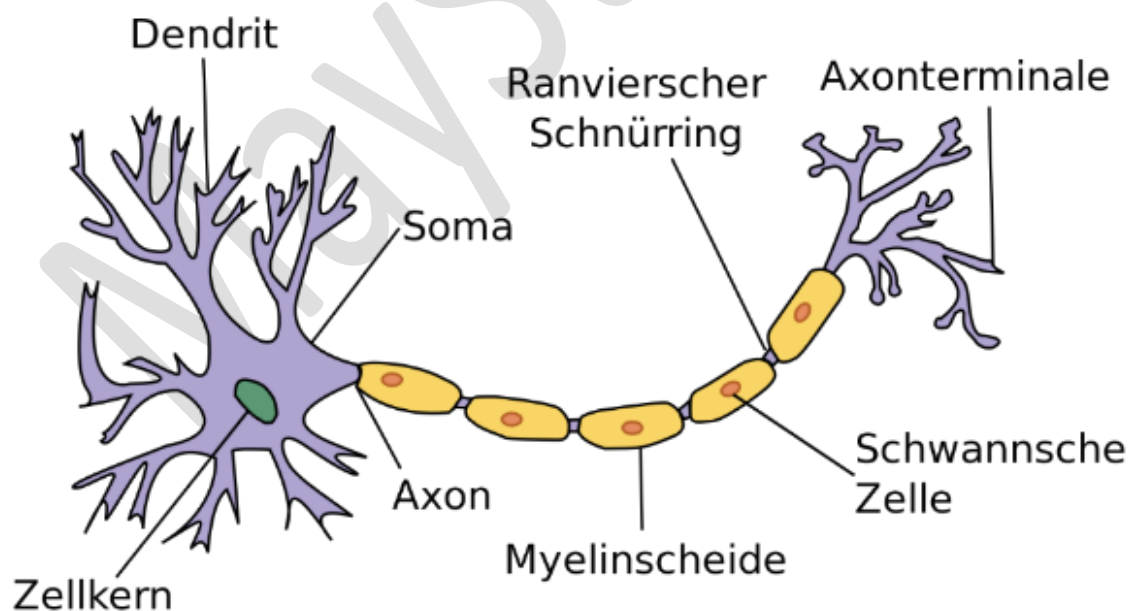
Der Ort der Kommunikation zwischen den Neuronen ist der synaptische Spalt, eine winzige Lücke zwischen den Nervenzellen. In diesem mikroskopisch kleinen Raum überträgt ein Neuron Nachrichten an ein anderes. Diese Übertragung erfolgt, wenn ein Neuron chemische Signalmoleküle - Neurotransmitter - freisetzt, die sich an spezifische Rezeptoren des anderen Neurons binden. Wenn der Reiz stark genug ist, löst er ein Aktionspotenzial aus, einen elektrischen Impuls, der das Axon des Neurons entlangwandert.

Die Stärke und Wirksamkeit der Verbindungen zwischen den Neuronen kann sich im Laufe der Zeit verändern, ein Phänomen, das als synaptische Plastizität bekannt ist. Diese Veränderungen tragen wesentlich zu den Lern- und Gedächtnisprozessen des Gehirns bei und ermöglichen es ihm, sich anzupassen und neue Informationen und Erfahrungen zu verarbeiten.

Die Fortschritte in unserem Verständnis der biologischen Neuronen waren von entscheidender Bedeutung und haben Aufschluss darüber gegeben, wie diese Zellen Informationen verarbeiten und welche Mechanismen dem Lernen und Gedächtnis zugrunde liegen. Dieses Wissen hat auch den Weg für die Entwicklung künstlicher neuronaler Netze geebnet, die darauf abzielen, die kognitiven Funktionen des menschlichen Gehirns nachzubilden, was erhebliche Auswirkungen auf den Bereich der künstlichen Intelligenz hat.

Insgesamt bilden biologische Neuronen das Fundament des Nervensystems und spielen eine entscheidende Rolle bei der Informationsverarbeitung, der Verhaltensregulierung und der Kognition. Das komplexe Zusammenspiel neuronaler Signale, sowohl chemischer als auch elektrischer Art, ermöglicht die Kommunikation und Koordination zwischen Neuronen. Ohne diese Prozesse wäre das Funktionieren des Gehirns insgesamt nicht möglich.

(Kirsanov, 2023)



(wiktionary, 2023) (Ognjanovski, 2019)

## Aufbau eines Künstlichen Neurons

Wie gerade schon erwähnt basiert ein Künstliches Neuron auf dem Prinzip der Synapsen bzw. der Verbindungsstellen durch das Axon und der Dendriten, und verarbeitet möglichen Dateneingaben durch die Verbindungen der einzelnen Neuronen.

Jedoch im Gegensatz zu biologischen Neuronen haben künstliche Neuronen keine Zellkörper, Axone, Dendriten oder Synapsen. Stattdessen bestehen sie aus einer Reihe von Eingabe- und Ausgabeneuronen, die untereinander mithilfe von Gewichten verbunden sind. Diese Verbindungen repräsentieren die Beziehungen zwischen den Neuronen und stellen bestimmte Muster, Strukturen und Eigenschaften da.

Um zu verstehen, wie künstliche Neuronen funktionieren, muss man die beiden Hauptkomponenten kennen, einerseits die Gewichte und andererseits die Aktivierungsfunktion und Bias Werte. Anstatt nämlich eine Signalstärke wie beim Biologischen Neuron zu senden, werden beim künstlichen Neuronalen Netz Zahlen gesendet, die mithilfe des Gewichts angepasst werden. Dieses Gewicht einer Verbindung wird mit dem Wert des Neurons multipliziert, um zum Beispiel eine niedrige Signalstärke bei kleinen Gewichten zu repräsentieren und eine große bei größeren Gewichten. Der Bias Wert eines Neurons stellt zu den Verbindungen der Gewichte noch eine halb statische Fein Anpassung des Neurons da, da der Bias Wert nicht von den Eingabedaten abhängig ist.

Die Aktivierungsfunktion eines Neurons dient dazu, die Werte des Neurons noch bestmöglich anzupassen und zu bestimmen, ob ein Neuron aktiviert wurde oder nicht. Beispielsweise können durch eine sogenannte ReLu Funktion negative Werte verhindert werden, oder durch die Sigmoid Funktion nur Werte nahe 0 stärker relevant werden, da die Sigmoid Funktion nur Werte zwischen  $-0.9$  und  $0.9$  erlaubt.

Wenn mehrere solcher künstlichen Neuronen miteinander verbunden werden, bilden diese ein neuronales Netzwerk, das dann mithilfe von Beispieldaten trainiert werden kann. Anhand der Differenz zwischen der erwarteten und der tatsächlichen Ausgabe wird dann die Anpassung der Gewichte und Bias-Werte angepasst, diesen Prozess nennt man auch „Backpropagation“.

## Netzarchitektur

Die Struktur eines künstlichen neuronalen Netzwerks definiert, auf welche Weise es Daten verarbeitet und aus ihnen Erkenntnisse gewinnt. Es gibt zahlreiche unterschiedliche Strukturen, von denen jede ihre eigenen Vor- und Nachteile aufweist. In diesem Essay wird analysiert, auf welche Art sich Strukturen künstlicher neuronaler Netzwerke auf Leistungsfähigkeit und Anwendbarkeit auswirken und warum es so viele verschiedene Strukturen gibt.

Eine weit verbreitete Struktur künstlicher neuronaler Netzwerke ist das Vorwärts-Netzwerk, auch als Multilayer-Perceptron (MLP) bekannt. Diese Struktur setzt sich aus einer Eingangsebene, verborgenen Ebenen und einer Ausgabebene zusammen. Die Eingangsebene empfängt die Daten, die verborgenen Ebenen verarbeiten die Informationen und die Ausgabebene stellt das abschließende Ergebnis zur Verfügung. MLPs kommen bei Problemlösungen wie Bilderkennung, Spracherkennung und Verarbeitung natürlicher Sprache zum Einsatz.

Eine weitere üblicherweise genutzte Struktur ist das Rückkopplungsneuronale Netzwerk (RNN), das Schleifen enthält, welche die Erhaltung von Informationen ermöglichen. RNNs werden für Tätigkeiten wie Sprachmodellierung, Spracherkennung und sequenzielles Lernen eingesetzt. Die LSTM-Struktur (Long Short-Term Memory) ist eine Art von RNN, die entwickelt wurde, um das Problem verschwindender Gradienten zu umgehen, welches auftreten kann, wenn die Gradienten zu klein werden, um die Netzgewichte wirksam zu aktualisieren.

Faltungsneuronale Netzwerke (Convolutional Neural Network, CNN) sind eine Art künstlicher neuronaler Netzwerkstrukturen, die für die Bildverarbeitung entwickelt wurden. CNNs verwenden Faltungsebenen, um Merkmale in Bildern zu identifizieren, die anschließend in vollständig verbundenen Ebenen zur Klassifizierung eingesetzt werden. CNN werden für Tätigkeiten wie Objekterkennung, Gesichtserkennung und Bildklassifizierung verwendet.

Generative gegnerische Netzwerke (GANs) sind eine Art künstlicher neuronaler Netzwerkstrukturen, die zur Generierung neuer Daten genutzt werden. GANs bestehen aus zwei künstlichen neuronalen Netzwerken: einem Generator und einem Diskriminator.



Der Generator erzeugt neue Daten, während der Diskriminator bewertet, ob die Daten authentisch oder gefälscht sind. GANs wurden bereits für Tätigkeiten wie Bildsynthese, Textsynthese und Musiksynthese eingesetzt.

Es gibt viele weitere künstliche neuronale Netzwerkstrukturen, wie zum Beispiel Autoencoder, Spiking Neural Networks und selbstorganisierende Karten. Jede Struktur besitzt ihre eigenen Stärken und Schwächen, und die Auswahl der Struktur hängt von der jeweiligen Problemstellung ab.

Der Einfluss der Struktur künstlicher neuronaler Netzwerke auf Leistung und Anwendung ist beträchtlich. Unterschiedliche Strukturen können sehr verschiedene Leistungsmerkmale aufweisen. Beispielsweise eignen sich CNNs deutlich besser für die Bildverarbeitung als MLPs, während RNNs besser für das sequenzielle Lernen geeignet sind als MLPs.

Die Wahl der Struktur beeinflusst auch die Komplexität und Skalierbarkeit des Systems. Einige Strukturen, wie MLPs, sind verhältnismäßig einfach und können auf kleinen Datensätzen trainiert werden. Andere Strukturen, wie zum Beispiel GANs, sind wesentlich komplizierter und erfordern umfangreiche Daten- und Rechenkapazitäten.

Die Wahl der Struktur beeinflusst auch die Interpretierbarkeit des Modells. Einige Strukturen, wie zum Beispiel MLPs, sind relativ einfach zu interpretieren, da die Netzgewichte direkt auf die Eingabemerkmale bezogen werden können. Andere Strukturen, wie zum Beispiel CNNs, sind deutlich schwieriger zu interpretieren, da die erlernten Merkmale über das Netzwerk verteilt sind.

Zusammenfassend spielt die Struktur künstlicher neuronaler Netzwerke eine entscheidende Rolle für die Leistungsfähigkeit und Anwendbarkeit dieser Technologie. Es gibt zahlreiche verschiedene Strukturen, jede mit ihren eigenen Vor- und Nachteilen. Die Wahl der Struktur hängt von der Problemstellung, der Menge der verfügbaren Daten, den bereitstehenden Rechenressourcen und der Interpretierbarkeit des Modells ab. Mit der kontinuierlichen Weiterentwicklung künstlicher neuronaler Netzwerke werden wahrscheinlich neue Strukturen entwickelt, um aktuellen und zukünftigen Herausforderungen gerecht zu werden.

(Team, 2022)

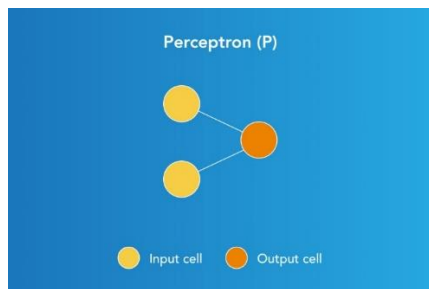
## Perceptron

Das Perceptron ist ein frühes Beispiel eines künstlichen neuronalen Netzes, das in den 1950er Jahren von Frank Rosenblatt entwickelt wurde. Es ist ein einfaches Modell, das aus einer Schicht miteinander verbundener künstlicher Neuronen besteht. Diese Neuronen empfangen Eingangsdaten, berechnen daraus eine gewichtete Summe und erzeugen durch Anwendung einer Aktivierungsfunktion ein binäres Ausgangssignal. Dadurch eignet sich das Perceptron besonders für einfache binäre Klassifikationsaufgaben.

Um dem Perceptron beizubringen, wie es Aufgaben lösen soll, werden die Gewichte, die den Eingaben zugeordnet sind, mit Hilfe von markierten Trainingsdaten angepasst. Dabei wird für jede Eingabe eine Vorhersage getroffen. Ist die Vorhersage falsch, werden die Gewichte entsprechend angepasst. Dieses Lernverfahren wird als überwachtes Lernen bezeichnet.

Obwohl das Perceptron einfach zu handhaben ist und sich gut für binäre Klassifikationsaufgaben eignet, hat es auch seine Grenzen. Es kann nur lineare Entscheidungsgrenzen erlernen, was seine Anwendbarkeit auf einfache binäre Klassifikationsaufgaben beschränkt. Bei komplexeren Problemen mit nichtlinearen Entscheidungsgrenzen stößt das Perceptron an seine Leistungsgrenzen. Außerdem benötigt das Perceptron beschriftete Trainingsdaten und eine große Anzahl von Beispielen, um effektiv verallgemeinern zu können.

Trotz seiner Schwächen hat das Perceptron als grundlegendes Modell für künstliche neuronale Netze in der Vergangenheit eine wichtige Rolle gespielt und die Forschung auf diesem Gebiet angeregt. Heutzutage werden jedoch leistungsfähigere Modelle wie Deep Neural Networks oder Convolutional Neural Networks (CNN) verwendet, die besser mit komplexeren, nichtlinearen Entscheidungsgrenzen umgehen können. Dies hat dazu geführt, dass maschinelles Lernen und künstliche Intelligenz erfolgreich in einer Vielzahl von Anwendungen eingesetzt werden, wie z.B. Bild- und Spracherkennung, autonomen Fahren, Spielstrategien und vieles mehr.



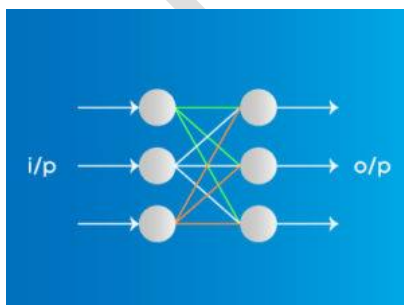
(Team, 2022)

## Feed Forward

Das einfachste System eines Neuralen Netzes, stellt das Fließen der Eingabedaten in eine Richtung da, dabei durchlaufen die Daten dann mehrere künstliche Neuronen und die errechnete Summe eines Neurons wird durch eine Aktivierungsfunktion weiter angeglichen. Solche Netze können als einschichtiges Modell existieren oder auch mit sogenannten versteckten Schichten als mehrschichtiges Modell.

Des Weiteren hängt die Anzahl an Schichten von der gewünschten Komplexität des Netzes ab und der gewählten Aktivierungsfunktion. Jedoch ist bei solch einem Netz zu beachten, dass es nur vorwärts durch die Schichten geht und keine sogenannte „Backpropagation“ besitzt. Dementsprechend sind die Gewichte von Anfang an statisch und können sich nicht weiter verändern.

Wenn dann dieser Prozess abgeschlossen ist, wird algorithmisch ein bestimmtes Ergebnis ausgewählt und je nachdem um welches Ausgabe Neuron es sich handelt, dann interpretiert.



(Team, 2022)

**Einsatz:**

- Einfache Klassifizierung (wo traditionelle, auf maschinellem Lernen basierende Klassifizierungsalgorithmen ihre Grenzen haben)
- Gesichtserkennung (Einfache, unkomplizierte Bildverarbeitung]
- Computer Vision (Wenn die Zielklassen schwer zu klassifizieren sind]
- Spracherkennungssysteme

**Vorteile von Feed Forward Neural Networks**

- Weniger komplex, einfach zu entwerfen und zu pflegen
- Schnell und zügig (Einbahnstraßenausbreitung]
- Reagiert sehr gut auf verrauschte Daten
- Nachteile von neuronalen Feed Forward-Netzwerken:
- Kann nicht für tiefes Lernen verwendet werden (aufgrund des Fehlens dichter Schichten und der Rückvermehrung].

**Mehrschichtiges Perceptron**

Ein Multilayer Perceptron, kurz MLP, ist eine Art künstliche Intelligenz, die darauf abzielt, komplexe Zusammenhänge in Daten zu erkennen. Man kann es sich wie ein kleines künstliches Gehirn vorstellen, das aus verschiedenen Schichten besteht. Um besser zu verstehen, wie ein MLP funktioniert, muss man einen genaueren Blick auf seine Bestandteile und ihre Aufgaben werfen.

Am Anfang steht die Eingabeschicht, die als erste Ebene des Netzes fungiert und die zu untersuchenden Daten empfängt. Jedes Neuron, in dieser Schicht kümmert sich um eine bestimmte Information oder ein Detail der Daten. Die Eingabeschicht dient als Brücke zwischen den Rohdaten und dem eigentlichen MLP.

Das Herzstück eines MLPs sind die verborgenen Schichten, die sich zwischen der Eingabe- und der Ausgabeschicht befinden. Ihre Aufgabe ist es, Muster und Merkmale in den Daten zu erkennen und zu extrahieren. Je nach Schwierigkeit des Problems und der

gewählten Netzwerkarchitektur gibt es eine oder mehrere dieser Schichten. In den verborgenen Schichten haben die Neuronen Aktivierungsfunktionen, die bestimmen, wie stark sie auf die Eingaben reagieren.

Am Ende des Prozesses steht die Ausgabeschicht, die das Ergebnis der gesamten Rechenarbeit darstellt. Die Anzahl der Neuronen in dieser Schicht hängt davon ab, wie viele verschiedene Ergebnisse oder Kategorien das Netz vorhersagen soll. Je nach Anwendung kann die Ausgabeschicht Wahrscheinlichkeiten für verschiedene Kategorien ausgeben oder kontinuierliche Werte vorhersagen.

Die Neuronen in den verschiedenen Schichten sind durch Gewichte miteinander verbunden, die die Stärke ihrer Zusammenarbeit angeben. Während das Netz trainiert wird, werden diese Gewichte so angepasst, dass die Vorhersagen immer besser und genauer werden.

Ein praktisches Beispiel für MLP ist die Erkennung von handgeschriebenen Zahlen. Dabei werden die Pixelwerte der Bilder als Input für das Netzwerk verwendet und als Output die Wahrscheinlichkeiten für jede der zehn möglichen Ziffern (0-9) ausgegeben. Das Netz lernt, die Struktur und Form der verschiedenen Ziffern zu erkennen und kann so unbekannte handschriftliche Ziffern erfolgreich klassifizieren.

Ein anderes Beispiel ist die Erkennung von Spam-E-Mails. Hier könnte das MLP anhand von Textmerkmalen wie Worthäufigkeiten oder bestimmten Schlüsselwörtern darauf trainiert werden, Spam-E-Mails von legitimen E-Mails zu unterscheiden. Die Ausgabeschicht würde dann die Wahrscheinlichkeit ausgeben, mit der eine bestimmte E-Mail Spam ist.

Zusammenfassend lässt sich sagen, dass ein mehrschichtiges Perceptron ein leistungsfähiges Werkzeug ist, um komplexe Muster in Daten zu erkennen und Vorhersagen zu treffen. Durch das Verständnis der verschiedenen Schichten und ihrer Funktionen wird deutlich, wie diese künstliche Intelligenz

(Team, 2022)

# Convolutional Neural Network

Convolutional Neural Network, CNN ist ein Bereich der künstlichen Intelligenz, der speziell für die Verarbeitung komplexer Daten wie Bilder oder Sprache entwickelt wurde. Die Idee hinter CNN ist, dass es in der Lage ist, wichtige Informationen aus den Daten zu extrahieren, um Muster oder Zusammenhänge zu erkennen.

Eine wichtige Komponente von CNN ist die Verwendung von Filtern. Diese Filter werden über die Daten gelegt und führen mathematische Operationen durch, um relevante Merkmale zu identifizieren. Beispielsweise kann ein Filter so gestaltet werden, dass er Kanten oder Linien in einem Bild erkennt. Durch die Anwendung mehrerer Filter mit unterschiedlichen Merkmalen kann das CNN komplexe Muster und Merkmale in den Daten erkennen.

Eine weitere wichtige Komponente von CNN ist die Pooling-Schicht. Diese Schicht reduziert die Datengröße, indem wichtige Merkmale erhalten bleiben. Zum Beispiel kann ein Max-Pooling-Algorithmus das größte Merkmal aus einem bestimmten Bereich extrahieren, während ein Average-Pooling-Algorithmus den Durchschnitt der Merkmale berechnet.

Convolutional Neural Network wird häufig für die Bilderkennung und Klassifikation verwendet, um Bilder automatisch zu identifizieren und zu kategorisieren. Es gibt jedoch auch andere Anwendungen, wie z. B. die Videoverarbeitung, wo sie zur Erkennung von Bewegungen und Aktivitäten beitragen kann.

## *Lernverfahren*

### **Verschiedene Lernverfahren**

Ein gängiges Lernverfahren ist das "überwachte Lernen" (Supervised Learning). Hierbei werden dem neuronalen Netz Beispiele in Form von Eingabe-Ausgabe-Paaren präsentiert. Das Netz lernt, indem es die Gewichte der Verbindungen zwischen den Neuronen anpasst, um die Abweichungen zwischen den tatsächlichen und den

gewünschten Ausgaben zu minimieren. Der Lernprozess basiert meist auf dem sogenannten Backpropagation-Algorithmus, der den Fehler rückwärts durch das Netz propagiert und dabei die Gewichte entsprechend anpasst.

Ein weiteres Lernverfahren ist das "unüberwachte Lernen" (Unsupervised Learning). Hierbei werden dem Netz keine expliziten Ausgabewerte vorgegeben. Stattdessen versucht das Netz, Muster oder Strukturen in den Eingabedaten selbstständig zu erkennen. Eine beliebte Methode im unüberwachten Lernen ist die Clusterbildung, bei der ähnliche Eingaben in Gruppen zusammengefasst werden.

"Reinforcement Learning" (bestärkendes Lernen) ist ein weiteres Lernverfahren, bei dem das neuronale Netz Entscheidungen trifft, um eine Aufgabe zu lösen. Das Netz erhält Feedback in Form von Belohnungen oder Strafen, basierend auf der Qualität seiner Entscheidungen. Ziel ist es, die Entscheidungen so zu optimieren, dass die kumulative Belohnung maximiert wird. Reinforcement Learning eignet sich besonders gut für Probleme, bei denen es um die Optimierung von Entscheidungen in einer dynamischen Umgebung geht, wie z.B. bei Spieltheorien oder Robotik.

Schließlich gibt es noch das "tiefgehende Lernen" (Deep Learning), eine Untergruppe des überwachten Lernens. Tiefgehende neuronale Netze bestehen aus vielen Schichten von Neuronen und sind in der Lage, abstrakte Merkmale und Hierarchien in den Eingabedaten zu erkennen. Durch die Verwendung vieler Schichten können tiefgehende Netze komplexe Muster und Zusammenhänge in den Daten lernen, die mit flacheren Netzen nicht erfasst werden können.

## **Forward Propagation**

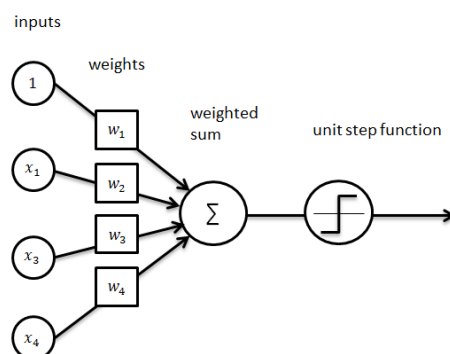
Der "Feed Forward Algorithmus" bezieht sich in der Regel auf das Vorwärtsdurchlaufen eines künstlichen neuronalen Netzwerks, insbesondere bei Feed Forward-Netzwerken. Im Folgenden erkläre ich den Algorithmus ausführlich und beziehe dabei mathematische Ausdrücke ein.

Beim Feed Forward Algorithmus werden die Eingabedaten durch das Netzwerk geschickt, um Vorhersagen oder Klassifikationen zu treffen. Hier ist eine schrittweise Erklärung des Prozesses:

1. Initialisierung: Die Eingabedaten werden in die Eingabeschicht eingespeist, wobei jedes Neuron einen Wert aus den Eingabedaten empfängt. Mathematisch gesehen sei  $x_i$  der i-te Eingabewert, und wir haben insgesamt n Eingabewerte ( $x_1, x_2, \dots, x_n$ ).
2. Gewichtete Summe: Jedes Neuron in der nächsten Schicht (versteckte oder Ausgabeschicht) berechnet eine gewichtete Summe der Werte der Neuronen aus der vorherigen Schicht, wobei die Gewichte der Verbindungen berücksichtigt werden. Die gewichtete Summe für das j-te Neuron in der k-ten Schicht ist:

$$z_{kj} = b_{kj} + \sum_{i=1}^{n_{k-1}} (w_{kji} \cdot x_{kji})$$

(Ognjanovski, 2019)



Dabei ist  $w_{kji}$  das Gewicht der Verbindung vom j-ten Neuron der k-ten Schicht zum Ausgabewert  $a_i$  der vorherigen Schicht. Hierbei ist  $i$  der Index des vorherigen Neurons. Jeder dieser Ausgabewerte wurde vorher mit einer Aktivierungsfunktion angepasst.

Anschließend wird noch der Bias Wert des j-ten Neurons der k-ten Schicht addiert, um einen möglichen Ausgleich des Neurons zu schaffen.

3. Aktivierung: Jedes Neuron wendet seine Aktivierungsfunktion auf die gewichtete Summe an, um den Aktivierungswert (Ausgangswert) für das Neuron zu berechnen. Eine übliche Aktivierungsfunktion ist die Sigmoid-Funktion, die durch die Formel definiert ist:

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



Oder die Tahn:

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

Oder die ReLu:

$$\sigma(z) = \max(0, z)$$

Der Aktivierungswert für das j-te Neuron in der k-ten Schicht ist dann:

$$a_{kj} = \sigma(z_{kj})$$

4. Fortschreiten durch die Schichten: Schritte 2 und 3 werden für jede versteckte Schicht wiederholt, bis die Ausgangsschicht erreicht ist.

(Brothcrunsher, Neuronale Netze - Backpropagation - Forwardpass, 2017)

(Ognjanovski, 2019)

## Backpropagation

Der Backpropagation-Algorithmus ist ein wesentlicher Bestandteil des maschinellen Lernens, insbesondere bei neuronalen Netzwerken. Er ist verantwortlich für die Optimierung der Gewichte in einem Netzwerk, um Fehler zu minimieren und die Vorhersagegenauigkeit zu verbessern. Backpropagation wird häufig in Verbindung mit einem Gradientenabstiegsverfahren verwendet, um die Gewichte der Neuronen interaktiv anzupassen.

Um den Backpropagation-Algorithmus besser zu verstehen, müssen wir uns zunächst das Konzept eines Feed Forward-Netzwerks vergegenwärtigen. In einem solchen Netzwerk fließt die Information von der Eingangsschicht über die versteckten Schichten bis zur

Ausgangsschicht. Jede Verbindung zwischen den Neuronen hat ein Gewicht, das den Einfluss des einen Neurons auf das andere bestimmt. Der Algorithmus besteht aus zwei Hauptphasen: Feed Forward und Backpropagation.

(Brotcrunsher, Neuronale Netze - Backpropagation - Backwardpass, 2017)

(Ognjanovski, 2019)

1. Feed Forward: In der Feed Forward-Phase wird eine Eingabe durch das Netzwerk geschickt, um eine Vorhersage zu erhalten. Die Aktivierung eines Neurons in einer Schicht wird durch eine gewichtete Summe der Ausgänge der vorhergehenden Schicht und einer Aktivierungsfunktion bestimmt.
2. Backpropagation: In dieser Phase berechnen wir den Fehler (oder die Verlustfunktion) der Vorhersage im Vergleich zum tatsächlichen Zielwert und aktualisieren die Gewichte entsprechend. Der Fehler wird rückwärts durch das Netzwerk propagiert, daher der Name Backpropagation. Zunächst berechnen wir den Fehler der Ausgangsschicht:

$$\delta_j = (y_j - a_j) \cdot f'(z_j)$$

Hierbei ist  $\delta_j$  der Fehler des j-ten Neurons,  $y_j$  der Zielwert,  $a_j$  die Aktivierung des k-ten Neurons,  $f'$  die Ableitung der Aktivierungsfunktion und  $z_j$  die gewichtete Summe der Eingaben des j-ten Neurons.

Anschließend berechnen wir den Fehler für die versteckten Schichten:

$$\delta_j = (\sum_k w_{jk} \cdot \delta_k) \cdot f'(z_j)$$

Hierbei ist  $\delta_j$  der Fehler des j-ten Neurons,  $w_{jk}$  das Gewicht zwischen Neuron j und k,  $\delta_k$  der Fehler des k-ten Neurons in der nächsten Schicht und  $z_j$  die gewichtete Summe der Eingaben des j-ten Neurons.

Schließlich aktualisieren wir die Gewichte im Netzwerk:

$$w_{ji} = w_{ji} + \eta \cdot \delta_j \cdot a_i$$

Dabei ist  $\eta$  die Lernrate, ein Hyperparameter, der die Schrittgröße der Gewichtsaktualisierung steuert.

## **Implementierung**

### ***Ziel***

In meinem Projekt würde ich gerne ein möglichst schnell arbeitendes CNN umsetzen, das Bilder klassifizieren kann (von Obst in diesem Fall), die dafür benötigten Trainingsdaten möchte ich selbst sammeln und auch in einer hohen Stückzahl. Durch das Einsetzen von Convolutional Filter und Pooling Ebenen erhoffe ich mir eine deutlich gesteigerte Leistung sowie Genauigkeit der Klassifizierung. Dementsprechend möchte ich den größten Teil aller Rechnungen parallel laufen lassen (Grafikkarte) um die Leistung zu vertausendfachen.

### **Programmiersprachenwahl**

Die Wahl der Programmiersprache ist für Projekte wie diesem von großer Bedeutung, da die Sprache der Künstlichen Intelligenz schnell verarbeiten und gleichzeitig eine hardwarenahe Struktur besitzen sollte. Eine spätere Integration von Grafikkarten wird ebenfalls von entscheidender Bedeutung sein.

Da ich hauptsächlich mit Unreal Engine arbeite, ist mir C++ am vertrautesten und bietet daher eine gute Grundlage für schnelle, hardwarenahe Programmierung. Eine weitere "low-level language" Alternative wäre RUST, die jedoch laut Benchmarks langsamer ausgeführt wird als C++.

Des Weiteren bietet C++ eine gute Möglichkeit NVIDIA Grafikkarten über das Framework CUDA zu integrieren und da das Framework direkt von NVIDIA entwickelt wurde, bietet es einige Features bezüglich Matrixmultiplikationen (Tensor Cores) an.

(Nvidia, 2023)

## **Implementierung vom Netzwerk**

### **Warum Convolutional Neural Network?**

Zunächst ist es wichtig zu verstehen, dass CNNs speziell für die Verarbeitung visueller Informationen entwickelt wurden. Im Gegensatz zu herkömmlichen neuronalen Netzen, die oft Schwierigkeiten haben, räumliche Beziehungen in Bildern zu erkennen, sind Convolutional Neuronale Netzwerke in der Lage, lokale Muster und Strukturen in Bildern effizient zu erfassen. Dies liegt an ihrer einzigartigen Architektur, die aus Faltungs- (Convolutional), Pooling- und Fully Connected-Schichten besteht. Diese Schichten arbeiten zusammen, um die räumlichen Merkmale von Bildern zu erfassen und eine hierarchische Repräsentation der Daten zu erzeugen, die für die Klassifizierung verwendet werden kann.

Ein weiterer entscheidender Vorteil von CNNs gegenüber herkömmlichen Klassifikationsverfahren ist, dass sie weniger anfällig für den "Fluch der Dimensionalität" sind. Bei herkömmlichen Ansätzen steigt die Anzahl der Parameter exponentiell mit der Anzahl der Eingabemerkmale, was zu einer erhöhten Komplexität und einer erhöhten Wahrscheinlichkeit des Overfittings führt. Im Gegensatz dazu verwenden CNNs Weight-Sharing und Pooling-Schichten, um die Anzahl der zu lernenden Parameter zu reduzieren, was zu einer effizienteren und weniger anfälligen Klassifikation führt.

Darüber hinaus ermöglichen CNNs die automatische Extraktion von Merkmalen aus den Eingabedaten. Während herkömmliche Klassifikationsverfahren in vielen Fällen auf manuell ausgewählte Merkmale angewiesen sind, lernen CNNs selbstständig, welche Merkmale für die Klassifikation relevant sind. Dies führt zu einer besseren Leistung und einer höheren Genauigkeit, da das Modell selbst die optimalen Merkmale identifizieren und verwenden kann. Dieser Prozess der automatischen Merkmalsextraktion ist besonders wichtig, wenn man bedenkt, dass die manuelle Merkmalsextraktion zeitaufwändig, kostenintensiv und oft fehleranfällig ist. Durch die automatische Merkmalsextraktion können CNNs auch mit komplexeren und vielfältigeren Datensätzen umgehen, was sie zu einer leistungsfähigeren Klassifikationsmethode macht.

Ein weiterer Aspekt, der die Überlegenheit von CNNs gegenüber herkömmlichen Klassifikationsmethoden unterstreicht, ist ihre Invarianz gegenüber Transformationen wie Skalierung, Rotation und Translation. Diese Invarianz ist wichtig, da sie es dem Modell ermöglicht, die gleichen Objekte oder Muster in verschiedenen Größen, Orientierungen und Positionen innerhalb eines Bildes zu erkennen. Herkömmliche Klassifikationsmethoden sind oft empfindlich gegenüber solchen Transformationen, was zu einer schlechteren Leistung und Genauigkeit führen kann.

Schließlich profitieren CNNs von ihrer inhärenten Fähigkeit zur Parallelisierung. Das bedeutet, dass ihre Berechnungen effizient auf Grafikprozessoren oder speziellen Hardwarearchitekturen ausgeführt werden können. Dies führt zu einer schnelleren Verarbeitung und Analyse der Eingabedaten, was besonders wichtig ist, wenn große Bildmengen oder Echtzeitanwendungen verarbeitet werden müssen. Im Vergleich dazu sind herkömmliche Klassifikationsverfahren oft weniger effizient in Bezug auf Rechenleistung und Skalierbarkeit.

Zusammenfassend lässt sich sagen, dass Convolutional Neuronale Netzwerke eine Reihe von Vorteilen gegenüber konventionellen Klassifikationsmethoden bieten, insbesondere bei der Verarbeitung und Klassifikation visueller Informationen. Ihre einzigartige Architektur, ihre Fähigkeit zur automatischen Merkmalsextraktion, ihre Invarianz gegenüber Transformationen und ihre Effizienz bei der Parallelisierung machen sie zu einer leistungsfähigeren und robusteren Methode für die Klassifikation von Bildern und anderen visuellen Daten.

(3Blue1Brown, 2022)

# Architektur

## Implementierung von Kernel (Filterung)

### Convolutional Filter

Convolutional Filter, auch bekannt als Faltungsmasken oder Faltungskerne, sind ein zentrales Element in Convolutional Neural Networks (CNNs). Um die Funktionsweise dieser Filter auf mathematischer Ebene zu verstehen, betrachten wir zunächst die grundlegenden Konzepte der Faltung.

Die Faltung ist eine mathematische Operation, bei der zwei Funktionen miteinander kombiniert werden, um eine dritte Funktion zu erzeugen. In unserem Fall sind dies ein Eingangsbild (oder ein Feature Map) und ein kleinerer Filter (oder Kernel). Der Filter besteht aus einer Matrix von Zahlen, den sogenannten Gewichten.

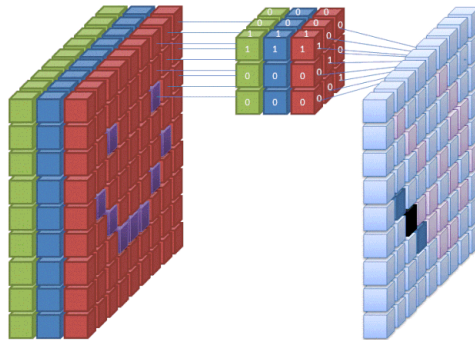
Um die Faltung zu berechnen, wird der Filter schrittweise über das Eingangsbild geschoben, und an jeder Position wird eine elementweise Multiplikation der Filter- und Bildwerte durchgeführt, gefolgt von einer Summierung der Ergebnisse. Das Ergebnis dieser Berechnung an jeder Position bildet eine neue Matrix, die sogenannte Feature Map oder Aktivierungskarte. Die Feature Map zeigt, wie stark das Eingangsbild auf den Filter reagiert, und gibt sozusagen Informationen über bestimmte Merkmale des Bildes wieder.

Nehmen wir als Beispiel ein einfaches graustufiges Bild der Größe 5x5 und einen Filter der Größe 3x3. Wir beginnen in der linken oberen Ecke des Bildes und platzieren den Filter so, dass er die ersten 3x3 Pixel des Bildes abdeckt. Nun multiplizieren wir jedes der neun Elemente des Filters mit dem entsprechenden Pixelwert im Bild und addieren diese Produkte. Der resultierende Wert wird in der linken oberen Ecke der Feature Map gespeichert.

(3Blue1Brown, 2022)

Um den nächsten Wert der Feature Map zu berechnen, verschieben wir den Filter horizontal um einen Schritt (in der Regel um einen Pixel) und wiederholen den Vorgang.

Wenn wir das Ende des Bildes erreichen, setzen wir den Filter in die nächste Zeile und beginnen wieder von der linken Seite. Dieser Prozess wird fortgesetzt, bis der Filter das gesamte Bild abgedeckt hat.



(Cecbur, 2019)

Ein wichtiger Aspekt der Convolutional Filter ist die Wahl der Schrittgröße, auch als "Stride" bezeichnet. Eine Schrittgröße von 1 bedeutet, dass der Filter bei jedem Schritt um einen Pixel verschoben wird. Eine größere Schrittgröße führt dazu, dass der Filter schneller über das Bild springt und somit eine kleinere Feature Map erzeugt. Durch die Anpassung der Schrittgröße kann man die Größe der Feature Map und die Rechenkomplexität des Netzwerks steuern. Eine weitere wichtige Komponente ist die Aktivierungsfunktion, die nach der Faltung auf die Feature Map angewendet wird. Eine häufig verwendete Aktivierungsfunktion ist die ReLU (Rectified Linear Unit), welche alle negativen Werte in der Feature Map auf null setzt. Dies führt zu einer nichtlinearen Transformation, die es dem Netzwerk ermöglicht, komplexere Muster zu lernen.

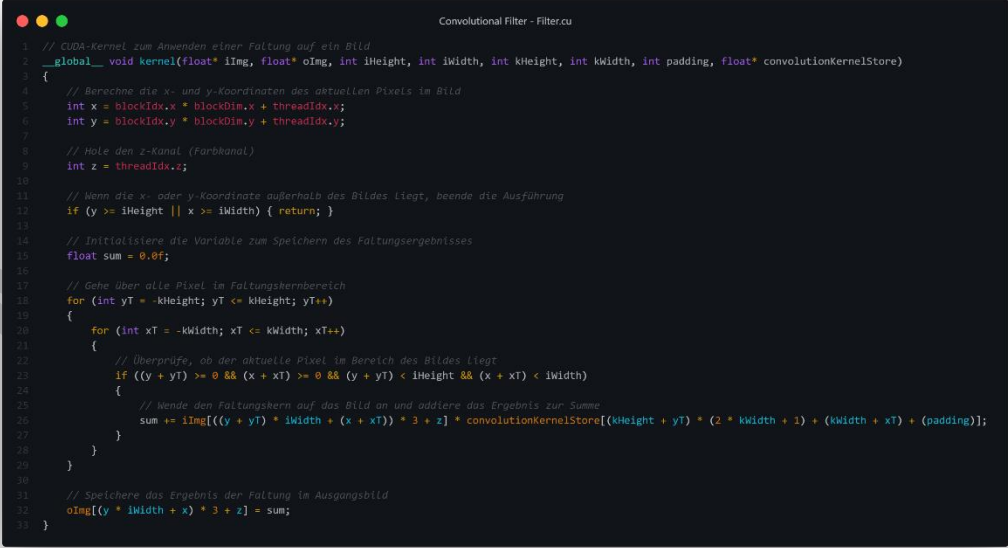
Es ist auch üblich, mehrere Filter in einem Convolutional Layer zu verwenden, um unterschiedliche Merkmale des Eingangsbildes zu erfassen. Jeder Filter kann auf verschiedene Aspekte des Bildes empfindlich sein, wie zum Beispiel Kanten, Texturen oder Farben. Die verschiedenen Feature Maps, die von den einzelnen Filtern erzeugt werden, werden dann von nachfolgenden Schichten des Netzwerks weiterverarbeitet und kombiniert, um eine abstraktere und höhere Repräsentation der Bilddaten zu erhalten.

Ein weiteres Konzept, das oft in Verbindung mit Convolutional Filtern verwendet wird, ist das sogenannte Padding. Padding bezieht sich auf das Hinzufügen von zusätzlichen

Randpixeln um das Eingangsbild, bevor die Faltung durchgeführt wird. Der Zweck des Paddings besteht darin, die räumliche Dimension der Feature Map beizubehalten und diese zu kontrollieren. Padding kann entweder mit Nullen (Zero-Padding) oder durch das Spiegeln der benachbarten Pixelwerte (Reflective Padding) erfolgen.

Es gibt zwei Haupttypen von Padding: "Same" und "Valid". Bei "Same" Padding wird so viel Padding hinzugefügt, dass die Größe der Feature Map gleich der Größe des Eingangsbildes ist. Bei "Valid" Padding wird kein zusätzliches Padding verwendet, was dazu führt, dass die Feature Map kleiner als das Eingangsbild ist.

In meinem Filter verwende ich ein sogenanntes „Same“ Padding mit Nullen, um die Dimensionen des Bildes beizubehalten. Außerdem sind in meiner Implementierung nur ungerade Filter möglich, da ich immer den mittleren Wert der Gewichte-Matrix als Zentrum des neuen Pixels verwende. Dementsprechend gibt es nur einen Wert für das Padding, der für die Dimensionen der Gewichte Matrix nach oben, unten, links und rechts steht.



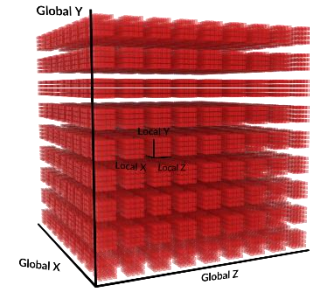
```
1 // CUDA-Kernel zum Anwenden einer Faltung auf ein Bild
2 __global__ void kernel(float* iImg, float* oImg, int iHeight, int iWidth, int kHeight, int kWidth, int padding, float* convolutionKernelStore)
3 {
4     // Berechne die x- und y-Koordinaten des aktuellen Pixels im Bild
5     int x = blockIdx.x * blockDim.x + threadIdx.x;
6     int y = blockIdx.y * blockDim.y + threadIdx.y;
7
8     // Hole den z-Kanal (Farbkanal)
9     int z = threadIdx.z;
10
11     // Wenn die x- oder y-Koordinate außerhalb des Bildes liegt, beende die Ausführung
12     if (y >= iHeight || x >= iWidth) { return; }
13
14     // Initialisiere die Variable zum Speichern des Faltungsergebnisses
15     float sum = 0.0f;
16
17     // Gehe über alle Pixel im Faltungskernbereich
18     for (int yT = -kHeight; yT <= kHeight; yT++)
19     {
20         for (int xT = -kWidth; xT <= kWidth; xT++)
21         {
22             // Überprüfe, ob der aktuelle Pixel im Bereich des Bildes liegt
23             if ((y + yT) >= 0 && (x + xT) >= 0 && (y + yT) < iHeight && (x + xT) < iWidth)
24             {
25                 // Wende den Faltungskern auf das Bild an und addiere das Ergebnis zur Summe
26                 sum += iImg[(y + yT) * iWidth + (x + xT)] * 3 + z * convolutionKernelStore[(kHeight + yT) * (2 * kWidth + 1) + (kWidth + xT) + (padding)];
27             }
28         }
29     }
30
31     // Speichere das Ergebnis der Faltung im Ausgangsbild
32     oImg[(y * iWidth + x) * 3 + z] = sum;
33 }
```

(GL, 2022)

In meinen „Kernel“ übergebe ich einerseits 2 Pointer als allokierte Grafikkarten Speicherstellen, in denen einerseits das Eingabebild gespeichert wird und andererseits das



Ausgabebild („oImg“). Diesen Kernel kann man sich auch in vereinfachter Weise darstellen, wobei jeder Thread ein kleiner Rechenblock ist. Jeder dieser kleinen Blöcke würde nun ein Pixel übernehmen und errechnen, jeder dieser kleine Thread Blöcke ist in größere Blöcke unterteilt die Maximal 1024 Threads beinhalten dürfen.



Insgesamt bilden dann diese Threads ein großes 3D Gitter bei dem die Höhe, Tiefe und Länge eingestellt werden können. In meinem Fall besitzt der Kernel nur eine Höhe von 3 Threads, da das Foto 3 verschiedene Farb-Ebenen hat.

(3Blue1Brown, 2022)

## Pooling Layer

Pooling-Filter sind eine wichtige Komponente in der Bildverarbeitung, insbesondere in Convolutional Neural Networks (CNNs). Sie dienen dazu, die räumlichen Dimensionen von Eingabedaten zu reduzieren und somit die Komplexität des Modells zu verringern, ohne dabei relevante Informationen zu verlieren. Es gibt verschiedene Arten von Pooling-Filtern, wie zum Beispiel Max-Pooling und Average-Pooling. In diesem Text werde ich mich auf die mathematische Funktionsweise von Pooling-Filtern konzentrieren.

Angenommen, wir haben eine Eingabematrix  $A$  (oftmals auch als Feature-Map bezeichnet) der Größe  $M \times N$ . Der Pooling-Filter hat eine bestimmte Größe (z.B.  $2 \times 2$  oder  $3 \times 3$ ) und eine Schrittweite (Stride), die bestimmt, wie weit der Filter bei jedem Schritt über die Eingabematrix gleitet. Für dieses Beispiel nehmen wir einen  $2 \times 2$  Pooling-Filter mit einer Schrittweite von 2.

Wir beginnen in der oberen linken Ecke der Eingabematrix  $A$  und wenden den Pooling-Filter auf die ersten  $2 \times 2$  Werte an. Bei Max-Pooling suchen wir den größten Wert innerhalb dieses  $2 \times 2$ -Bereichs, während wir bei Average-Pooling den Durchschnitt der vier Werte berechnen. Angenommen, wir verwenden Max-Pooling, so nehmen wir den größten Wert und speichern ihn in einer neuen Matrix  $B$ .

Danach verschieben wir den Pooling-Filter um die Schrittweite (in diesem Fall 2) nach rechts und wenden die Pooling-Operation erneut an. Wir wiederholen diesen Vorgang,

bis wir am rechten Rand der Eingabematrix A angekommen sind. Anschließend verschieben wir den Pooling-Filter zurück zur linken Seite und zwei Zeilen nach unten (entsprechend der Schrittweite) und setzen den Prozess fort, bis wir am unteren Rand der Eingabematrix angekommen sind.

Am Ende dieses Vorgangs erhalten wir eine neue Matrix B, die die reduzierten räumlichen Dimensionen der Eingabematrix A enthält. Die Größe von B ist abhängig von der Größe der Eingabematrix, der Größe des Pooling-Filters und der Schrittweite. Mathematisch gesehen können wir die Größe von B mit den folgenden Formeln berechnen:

$$B\_Höhe = (M - Filter\_Höhe) / Schrittweite + 1$$

$$B\_Breite = (N - Filter\_Breite) / Schrittweite + 1$$

Wobei M und N die Höhe und Breite der Eingabematrix A sind, und Filter\_Höhe und Filter\_Breite die Größe des Pooling-Filters.

Zusammenfassend ist die Funktion von Pooling-Filtern das Zusammenfassen von Informationen aus benachbarten Werten der Eingabematrix und die Reduzierung der räumlichen Dimensionen. Diese Reduktion hilft, die Komplexität des Modells zu verringern und Overfitting zu vermeiden, indem weniger Parameter im Netzwerk benötigt werden. Obwohl Pooling-Filter Informationen aus der Eingabematrix zusammenfassen und somit verkleinern, sind sie so gestaltet, dass sie die relevantesten Informationen beibehalten, die für die nachfolgenden Schichten im CNN wichtig sind.

Die Wahl des Pooling-Verfahrens (z.B. Max-Pooling oder Average-Pooling) hängt von der Anwendung und den zu lernenden Merkmalen ab. Max-Pooling ist oft hilfreich, um das Vorhandensein bestimmter Merkmale in einem Bereich der Eingabematrix zu erfassen, während Average-Pooling dazu beiträgt, die durchschnittliche Aktivierung in einem Bereich zu ermitteln, was bei bestimmten Aufgaben, wie zum Beispiel der Erkennung von Texturen, nützlich sein kann.

Es ist erwähnenswert, dass Pooling-Filter nicht auf isolierte Bereiche der Eingabematrix beschränkt sind. In einigen Fällen, insbesondere wenn es darum geht, globale Informationen über das gesamte Eingabebild zu erfassen, kann ein sogenanntes Global-

Pooling verwendet werden. Dabei wird der Pooling-Filter so groß wie die gesamte Eingabematrix gewählt, und die resultierende Matrix B hat die Größe  $1 \times 1$ .

Insgesamt ist die mathematische Funktionsweise von Pooling-Filtern recht einfach und intuitiv. Die grundlegende Idee besteht darin, Informationen aus benachbarten Werten der Eingabematrix zu aggregieren und eine neue Matrix mit reduzierten räumlichen Dimensionen zu erzeugen. Der Hauptvorteil dieser Reduktion besteht darin, dass sie die Komplexität des Modells verringert und somit das Training von CNNs beschleunigt und die Gefahr von Overfitting reduziert. Die Wahl des Pooling-Verfahrens hängt von den Anforderungen der jeweiligen Anwendung und den zu lernenden Merkmalen ab, wobei Max-Pooling und Average-Pooling die am häufigsten verwendeten Techniken sind.

In neueren Architekturen und Forschungsarbeiten werden jedoch auch alternative Ansätze zur Reduzierung der räumlichen Dimensionen und zur Informationsaggregation erforscht. Ein Beispiel dafür ist das sogenannte "strided convolution", bei dem die Faltungsschicht selbst die räumlichen Dimensionen reduziert, indem sie mit einer Schrittweite größer als eins arbeitet. Dieser Ansatz kann in einigen Fällen die Notwendigkeit von separaten Pooling-Schichten reduzieren oder sogar eliminieren.

Zusammenfassend ist die mathematische Funktionsweise von Pooling-Filtern in der Bildverarbeitung und insbesondere in Convolutional Neural Networks (CNNs) von zentraler Bedeutung, um die Komplexität des Modells zu verringern und dabei die relevanten Informationen zu erhalten. Die verschiedenen Pooling-Techniken und ihre Anwendungen ermöglichen es, unterschiedliche Aspekte der Eingabedaten auf effiziente Weise zusammenzufassen und für das Training von CNNs nutzbar zu machen.

(3Blue1Brown, 2022)

```

Convolutional Filter - Pooling.cu
1  __global__ void kernelMax(float* iImg, float* oImg, int iHeight, int iWidth, int poolSize)
2  {
3      int x = blockIdx.x * blockDim.x + threadIdx.x;
4      int y = blockIdx.y * blockDim.y + threadIdx.y;
5      int z = threadIdx.z;
6
7      // printf("%d \n", iImg[(y * iWidth + x) * 3 + z]);
8
9      if (y >= iHeight || x >= iWidth) { return; }
10
11     int poolStartX = x * poolSize;
12     int poolStartY = y * poolSize;
13
14     float maxValue = 0;
15     for (int i = poolStartX; i < poolStartX + poolSize; i++) {
16         for (int j = poolStartY; j < poolStartY + poolSize; j++) {
17             int inputIndex = (j * iWidth * poolSize + i) * 3 + z;
18             maxValue = (maxValue > iImg[inputIndex]) ? maxValue : iImg[inputIndex];
19         }
20     }
21
22     oImg[(y * iWidth + x) * 3 + z] = maxValue;
23 }

```

## Neurales Netz

Nach dem Durchlaufen der Convolutional Ebenen (Filter) und den Pooling Ebenen werden die momentan noch 2D Bilder in einen 1D Vektor umgewandelt. Hierbei spricht man auch vom Flattening. Dieser Eingabe Vektor soll nun das Normale Feed Forward Netzwerk durchlaufen. Um die Rechnungen, die schon im Theorie Teil genannt worden sind zu vereinfachen, greife ich in meiner Implementierung auf Matrizen zurück, da diese den Prozess der Gewichtung möglichst einfach und effizient beschreiben.

Meine Implementierung der Forward Propagation beginnt, indem die Eingabedaten (als Vektor  $x$  bezeichnet) an die Eingabeschicht des neuronalen Netzes übergeben werden. Die Neuronen existieren bei meiner Methode nun nicht mehr explizit, da sie nur noch in Form von Höhe und Breite der Gewichte-Matrizen gespeichert sind. Um die Implementierung noch weiter zu vereinfachen wird nun jede Schicht des Neurales Netzes in der Klasse „Layer“ dargestellt. Jede dieser Layer Klassen besitzt dann einerseits seine

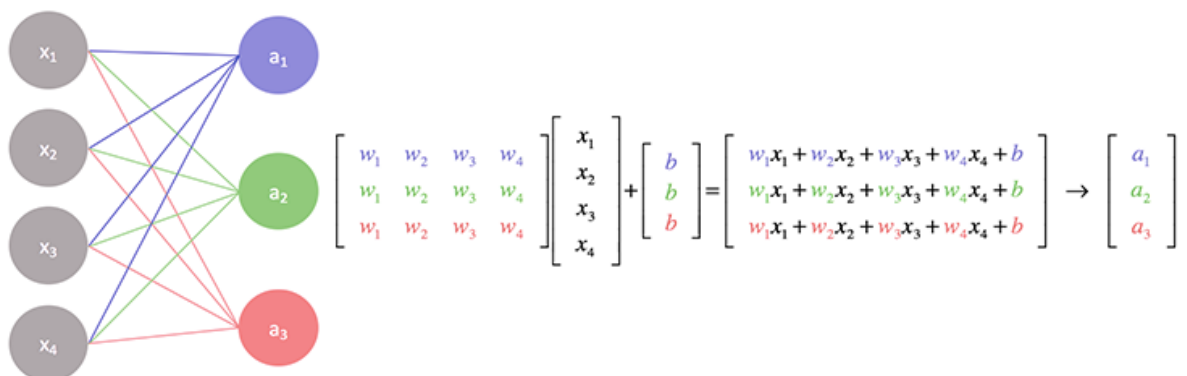
eigene Gewicht-Matrix und Bias-Vektor, sowie eine ausgewählte Aktivierungsfunktion. Um nun ein volles Netzwerk mit Eingabe-Schicht, Versteckten-Schichten und Ausgabe Schicht zu erzeugen werden die verschiedenen Layer in einer weiteren Klasse namens „Neural Network“ gespeichert.

Bei der Forward Propagation wird nun jeder der gespeicherten Layer hintereinander aufgerufen, wobei immer die Ausgabedaten des Vorherigen genutzt werden. In dem Layer wird dann die Gewichte-Matrix mit dem Eingabe-Vektor multipliziert und anschließend der Bias-Vektor addiert. Dieser Ausgabe-Vektor durchläuft anschließend noch die Aktivierungsfunktion und wird wieder ausgegeben.

```

Convolutional Filter - NeuralNetwork.cuh
1 DeviceMatrix forward(const DeviceMatrix& input) {
2     DeviceMatrix output = input;
3     for (Layer& layer : layers_) {
4         output = layer.forward(output);
5     }
6     return output;
7 }

```



Dementsprechend gilt: Um die Aktivierung der Neuronen in der ersten versteckten Schicht zu berechnen, führen wir eine gewichtete Summe der Eingabedaten durch, die wie folgt ausgedrückt werden kann:

$$z_1 = W_1 * x + b_1$$

Hier bezeichnet  $W_1$  die Gewichts-Matrix der Verbindungen zwischen der Eingabeschicht und der ersten versteckten Schicht,  $x$  ist der Eingabe-Vektor,  $b_1$  ist der Bias-Vektor für die erste versteckte Schicht und  $z_1$  ist der resultierende Vektor, der die gewichteten Summen für jedes Neuron in der ersten versteckten Schicht darstellt. Der Bias-Vektor repräsentiert eine Art von Offset, der zu den gewichteten Summen hinzugefügt wird, um die Aktivierung der Neuronen zu beeinflussen.

Nachdem die gewichteten Summen berechnet wurden, wenden wir eine Aktivierungsfunktion (bezeichnet als  $f$ ) auf den Vektor  $z_1$  an:

$$a_1 = f(z_1)$$

Der Vektor  $a_1$  repräsentiert nun die Aktivierung der Neuronen in der ersten versteckten Schicht. Dieser Prozess wird nun für jede aufeinanderfolgende versteckte Schicht wiederholt, indem die Aktivierungen der vorherigen Schicht als Eingabe für die nächste Schicht verwendet werden:

$$z_2 = W_2 * a_1 + b_2$$

$$a_2 = f(z_2)$$

Hier ist  $W_2$  die Gewichtsmatrix zwischen der ersten und der zweiten versteckten Schicht,  $b_2$  der Bias-Vektor für die zweite versteckte Schicht und  $a_2$  der Aktivierungsvektor für die zweite versteckte Schicht.

Nachdem alle versteckten Schichten durchlaufen wurden, erreichen die Aktivierungen die Ausgabeschicht.

(Broterunsher, Neuronale Netze - Backpropagation - Forwardpass, 2017)

(far1din, 2022)

(Broterunsher, Neuronale Netze - Backpropagation - Backwardpass, 2017)

## Matrizen

In meinem Projekt habe ich außerdem eine eigene Matrix-Klasse implementiert, die durch einfache Operatoren genutzt werden kann, jedoch im Hintergrund CUDA Kernel verwendet, um die gegebenen Matrizen zu multiplizieren, addieren und so weiter. Des Weiteren bietet diese Klasse eine Funktion, um zufällige Werte zu vergeben, was besonders wichtig ist beim Initialisieren des neuen Neuralen Netzwerks, damit es diese zufälligen Werte anschließend mithilfe der Backpropagation anpassen kann.

```
Convolutional Filter - Matrix.cuh
1 class DeviceMatrix {
2 public:
3     // default constructor
4     DeviceMatrix() : rows_(0), cols_(0), data_(nullptr) {}
5
6     DeviceMatrix(int rows, int cols);
7     DeviceMatrix(const DeviceMatrix& other);
8     ~DeviceMatrix();
9
10
11     void randomize();
12
13     DeviceMatrix transpose();
14     DeviceMatrix operator+(const DeviceMatrix& other);
15     DeviceMatrix operator-(const DeviceMatrix& other);
16     DeviceMatrix operator*(const DeviceMatrix& other);
17     DeviceMatrix operator*(const float& alpha);
18
19     void operator=(const DeviceMatrix& other);
20     void operator+=(const DeviceMatrix& other);
21     void operator-=(const DeviceMatrix& other);
22     void operator*=(const DeviceMatrix& other);
23
24     float* data() { return data_; };
25     int rows() { return rows_; };
26     int cols() { return cols_; };
27
28     void DeviceMatrix::print();
29     void set_values(float* values, int rows, int cols);
30     void set_data(float* data);
31
32     float norm();
33     DeviceMatrix sum_along_axis(int axis);
34
35 private:
36     float* data_;
37     int rows_;
38     int cols_;
39
40     int BLOCK_SIZE = 4;
41 };
```

Eine dazugehörige Operatoren Funktion geht nun hin und nutzt einerseits die eigenen Daten aus seinem eigenen Konstruktor sowie die Daten aus der anderen zu multiplizierenden Matrix. Hierbei müssen die Daten nicht weiter allokiert werden da

dieses schon beim Initialisierungsprozess der beiden Matrizen geschieht und dementsprechend die Daten der Matrizen schon auf der Grafikkarte gespeichert sind.

```
Convolutional Filter - Matrix.cu
1 DeviceMatrix DeviceMatrix::operator*(const DeviceMatrix& other) {
2     DeviceMatrix result(rows_, other.cols_);
3     dim3 grid((result.cols_ - 1) / BLOCK_SIZE + 1, (rows_ - 1) / BLOCK_SIZE + 1, 1);
4     dim3 block(BLOCK_SIZE, BLOCK_SIZE, 1);
5     mul_kernel << <grid, block >> > (data_, other.data_, result.data_, rows_, cols_, other.cols_);
6     return result;
7 }
```

```
Convolutional Filter - Matrix.cu
1 __global__ void mul_kernel(float* a, float* b, float* result, int a_rows, int a_cols, int b_cols) {
2     int col = blockIdx.x * blockDim.x + threadIdx.x;
3     int row = blockIdx.y * blockDim.y + threadIdx.y;
4     if (col < b_cols && row < a_rows) {
5         float sum = 0;
6         for (int i = 0; i < a_cols; ++i) {
7             sum += a[row * a_cols + i] * b[i * b_cols + col];
8         }
9         result[row * b_cols + col] = sum;
10    }
11 }
```

## Verarbeitung von Daten

Da ich auch selbstständig Daten für mein Neutrales Netz sammeln wollte, habe ich mir eine Apparatur überlegt, die über Nacht verschiedene Daten sammeln sollte, indem sie Tausende von Bildern macht. Bei diesem Projekt soll das Gerät das zu fotografierende Objekt umkreisen, um aus jedem möglichen Winkel ein Foto zu schießen und somit eine 360-Grad-Aufnahme des Objekts zu erstellen. Pro ein Grad Drehung soll das Gerät 30 verschiedene Fotos anfertigen, die das Objekt von oben bis unten abbilden. Als Inspirationsquelle diente das Konzept von 3D-Scans, bei denen das Objekt auf einer



Drehscheibe platziert und rotiert wird. Die Kamera ist dabei an einem beweglichen Arm befestigt, der sich um das Objekt herumbewegt.

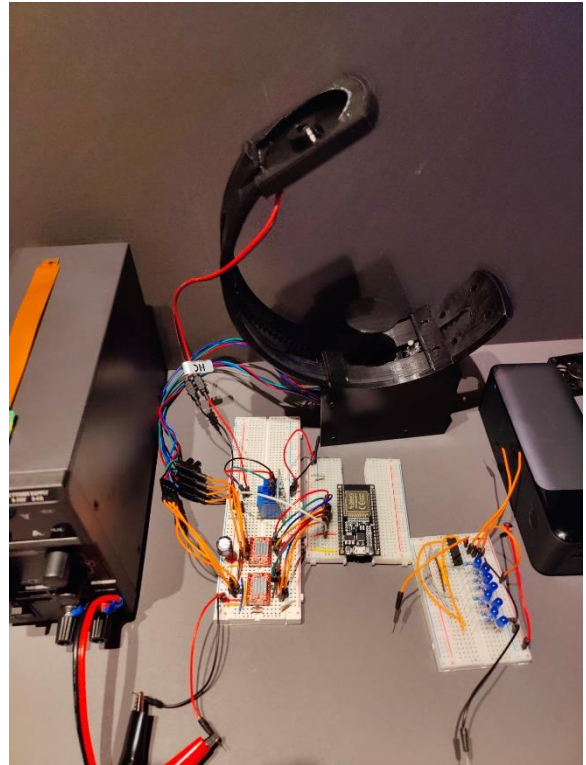
Mit Hilfe eines 3D ausgedruckten Objektes beziehungsweise Scanners sowie 2 Schrittmotoren einer Steuereinheit und einer Steuereinheit für eine Kamera habe ich diesen Scanner umgesetzt.

(Dmitriev, 2019)

Teileliste:

- ESP 32
- 2 x A4988
- 2x Nema 17 (Schrittmotor)
- Banana PI zero
- Raspberry PI zero
- Relais
- LED-Ring
- Kühlungs Lüfter
- Breadboard
- Raspberry Pi Kamera
- 30 Volt Netzteil
- 3D gedruckte Teile

Als Konzept habe ich mir überlegt, dass der ESP32 beide Schrittmotoren und den LED-Ring zur Beleuchtung des Objekts steuern soll. Der Raspberry Pi hingegen soll die Raspberry Pi Kamera steuern. Der ESP32 kommuniziert dabei über serielle Datenübertragung mit dem Raspberry Pi, um diesem mitzuteilen, wann ein Foto aufgenommen werden soll.



Grundsätzlich besteht die Idee darin, dass der ESP32 die beiden Schrittmotortreiber einerseits hinsichtlich der Schrittgröße ansteuert, sowie andererseits bezüglich Richtung und Schrittgeschwindigkeit. Zudem soll der ESP32 das Relais steuern, welches den LED-Ring schaltet. Der LED-Ring muss außerhalb des ESP-Stromkreises geschaltet werden, da dieser mit 12 Volt statt 5 Volt arbeitet.

Des Weiteren habe ich eine zusätzliche Platine entwickelt, auf der 8 LEDs als Fehlerausgabe dienen. Diese LEDs können mithilfe eines Shift-Registers angesteuert werden, sodass man nicht jede LED einzeln ansteuert, sondern lediglich eine Information in das Shift-Register schreibt. Dies spart im Endeffekt Output-Pins des ESP32.

Bei der Benutzung ist es wichtig, darauf zu achten, dass der Hintergrund hinter dem Objekt schwarz ist, damit nur das Objekt als relevanter Bildteil erfasst wird. Jeder schwarze Wert wird vom Computer als 0 gewertet bzw. besitzt keinen richtigen Farbwert. Dementsprechend steht das Objekt im Vordergrund.

(Dmitriev, 2019)

## **Probleme bei der Durchführung**

Ein größeres Problem, auf das ich gestoßen bin, war die Inkompatibilität der Raspberry Pi Kamera mit dem Banana Pi. Da beide Geräte unterschiedliche Formfaktoren haben und in China entwickelt wurden, entsprechen sie keiner einheitlichen Norm. Als mögliche Lösung habe ich mir einen Raspberry Pi Zero angeschafft, bei dem die Kamera den gleichen Formfaktor wie der Stecker am Raspberry Pi hatte. Allerdings führte dies zu weiteren Schwierigkeiten bei der Software des Raspberry Pi, um die Kamera zu steuern.

Die Kamera wurde zwar erkannt und mit korrekten Größenverhältnissen angezeigt, jedoch konnte sie aufgrund eines fehlenden Protokolls keine Bilder aufnehmen. Trotz längerer Bemühungen und Recherche im Internet konnte ich das Problem nicht beheben, da die dort gefundenen Lösungen keine Hilfe boten.

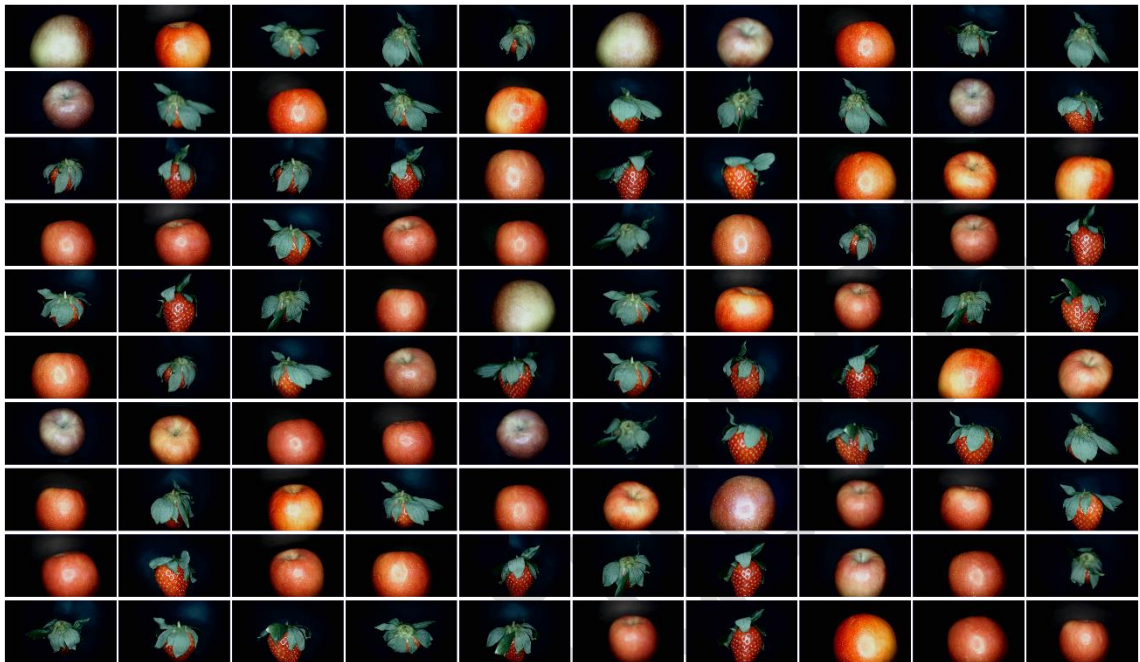
Um dennoch die Bilder des Scanners nutzen zu können, habe ich stattdessen meine normale Webcam verwendet, die an den PC angeschlossen wird und Bilder aufnimmt. Zwar bietet die Webcam eine geringere Auflösung, dies stellt jedoch in meinem Fall kein größeres Problem dar. Mein neuronales Netz verfügt über verschiedene Pooling-Ebenen, sodass ich das Bild auch in einer kleineren Variante einspeisen kann.

## **Ergebnisse**

In meinem Experiment habe ich zwei verschiedene Obstsorten untersucht, um sie zu klassifizieren: einen Apfel und eine Erdbeere. Des Weiteren wurden leere Bilder verwendet, die als keine Klassifizierung gelten. Aufgrund der Nutzung einer Webcam-Kamera konnten jedoch nicht so viele Testdaten aufgenommen werden wie ursprünglich geplant, da der PC während der Aufnahmen laufen musste, um die Bilder zu verarbeiten, und der Scanner kein eigenständiges Gerät ist, sondern nur in Verbindung mit dem PC funktioniert.

Daher wurden von jeder Obstsorte lediglich 900 Bilder erstellt, indem das Obst 36-mal in der Mitte um jeweils 10 Grad gedreht wurde und der Ring sich dabei 25-mal drehte,

wobei bei jeder Drehung ein Bild aufgenommen wurde. Um dennoch ausreichend Testdaten zu erhalten, habe ich zehn verschiedene Äpfel und zehn verschiedene Erdbeeren mit dem Gerät gescannt, indem ich jede Obstsorte zehnmal verarbeitet habe.



### *Zwischenfazit der bisherigen Ergebnisse*

#### **Wurden alle Entwicklungsziele erfüllt?**

Das Netzwerk konnte durch den Einsatz von CUDA und paralleler Programmierung relativ zügig trainiert werden. Anschließend erreichte es eine korrekte Klassifizierung der Bilder in 95% der Fälle. Um die Bilder jedoch präzise zu klassifizieren, war es notwendig, eine große Anzahl an Convolution-Filtern sowie eine hohe Anzahl an Neuronen pro Schicht einzusetzen.

Insgesamt liegen die Laufzeiten des Convolutional Neural Networks im Rahmen meiner Entwicklungsziele. Bei großen Bilddaten benötigen die meisten Filter maximal 2 Millisekunden, wobei diese Zeit exponentiell abnimmt, je kleiner das Bild ist. Am Ende

liegt die maximale Zeit für die Convolutional-Schicht bei etwa 100 Millisekunden. Dank der Matrix-Vereinfachung arbeitet das neuronale Netz in der Forward-Propagation ebenso schnell.

## **Mögliche Optimierungsprozesse**

Ein zentrales Problem des Netzwerks bestand in der Komplexität seiner Ebenen, da jede Ebene ihre Daten eigenständig speicherte. Dadurch war der gemeinsame Grafikkartenspeicher schnell erschöpft. Zudem führte der Arbeitsspeicher (RAM) zu weiteren Speicherproblemen, da die Bilddaten mithilfe von OpenCV gespeichert wurden und einige Bilder somit doppelt abgelegt waren.

Um die Komplexität des Netzwerks zu reduzieren, könnte man entweder eine asynchrone Speicherverteilung implementieren oder auf Shared Memory zurückgreifen, der parallel abrufbar ist. Dadurch müsste jedes Bild nur einmal gespeichert werden.

## **Fazit**

### **Optimierungspotenzial mittels Tensor Cores**

Um das Convolutional Neural Network weiter zu optimieren, könnte man zukünftig das Netzwerk selbst bestimmte Filter finden lassen. Dadurch könnten präzisere Texturen und Merkmale der Bilder erkannt und für genauere Klassifizierungen verwendet werden. Eine weitere Optimierung betrifft den Speicherbedarf, der in meiner Implementierung das Hauptproblem für die Größe darstellt. Da ich zunächst auf Einfachheit und Verständlichkeit Wert gelegt habe, habe ich nicht bedacht, dass Bilddaten erheblichen Speicherplatz benötigen.

Diese Speicherprobleme könnten durch den Einsatz von Shared Memory gelöst werden, das parallel abgerufen werden kann, oder durch die Verwendung von Asynchronität. Bei asynchronem Zugriff kann der Speicher von mehreren Funktionen gleichzeitig genutzt werden, wodurch eine doppelte Speicherung vermieden wird.

Um die Leistungsfähigkeit der neuronalen Netzwerkschicht weiter zu steigern, könnte man auf sogenannte „Tensor Cores“ zurückgreifen. Diese wurden von Nvidia speziell für

künstliche Intelligenz entwickelt und beherrschen insbesondere Matrixmultiplikationen sehr gut. Sie sind explizit für die Berechnung von:

$$z = W * x + b$$

ausgelegt.

Tensor Cores sind spezielle Hardware-Einheiten, die in modernen Grafikprozessoren (GPUs) und KI-Beschleunigern wie NVIDIA's Volta- und Turing-Architekturen zu finden sind. Sie wurden entwickelt, um die Berechnungen im Zusammenhang mit Deep Learning und Künstlicher Intelligenz (KI) deutlich zu beschleunigen. Um ihre Funktionsweise besser zu verstehen, sollten wir zunächst einen Blick auf Tensoren werfen.

Ein Tensor ist eine mathematische Struktur, die eine Verallgemeinerung von Skalaren, Vektoren und Matrizen ist. Tensoren können Daten in verschiedenen Dimensionen repräsentieren und sind daher ideal für die Verarbeitung von Daten in Deep-Learning-Anwendungen, bei denen es häufig um multidimensionale Daten geht.

Die Hauptfunktion von Tensor Cores ist die Beschleunigung von Matrixmultiplikationen und Akkumulationen. Matrixmultiplikation ist eine grundlegende Operation in Deep-Learning-Algorithmen, insbesondere bei neuronalen Netzen. Wenn wir zwei Matrizen A und B mit den Dimensionen  $A(m \times k)$  und  $B(k \times n)$  multiplizieren möchten, erhalten wir eine resultierende Matrix C mit den Dimensionen  $C(m \times n)$ . Die Berechnung von C kann wie folgt beschrieben werden:

$$C(i, j) = \text{sum}(A[i, p] * B[p, j]) \text{ für } p \text{ von } 1 \text{ bis } k$$

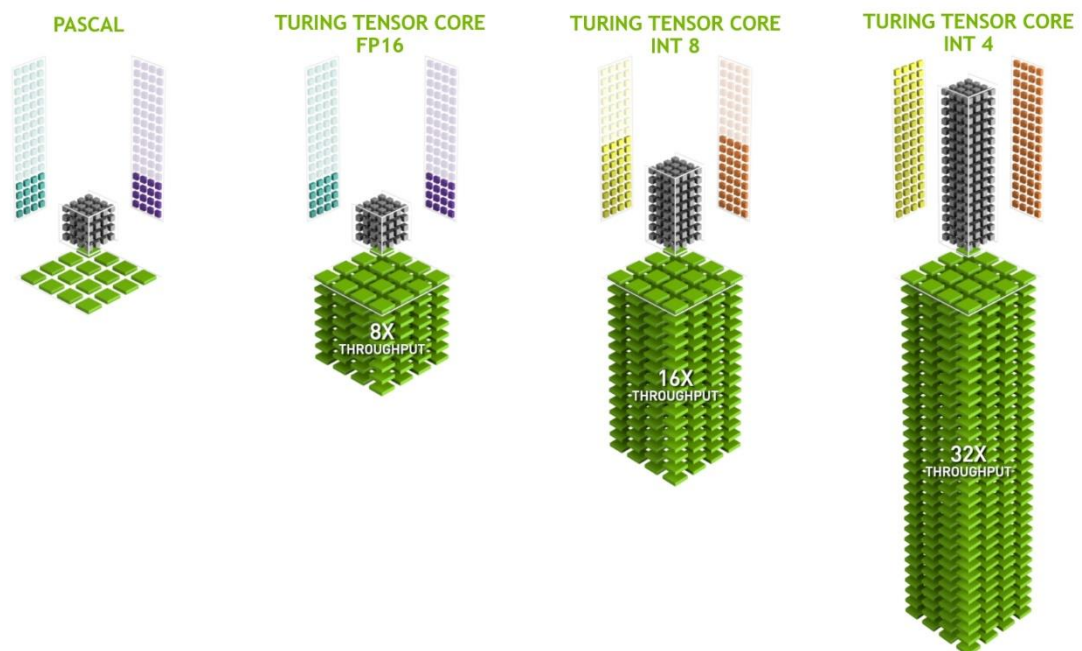
In Deep-Learning-Anwendungen können diese Matrizen sehr groß werden, und herkömmliche Hardware kann solche Berechnungen recht langsam ausführen. Genau hier kommen Tensor Cores ins Spiel. Sie sind speziell darauf ausgelegt, solche massiven Matrixmultiplikationen effizient und schnell durchzuführen.

Die Funktionsweise von Tensor Cores ist ziemlich interessant. Anstatt die gesamte Matrixmultiplikation in einem Schritt durchzuführen, teilen sie die Matrizen in kleinere

"Tiles" oder Blöcke auf. Jeder Tensor Core führt dann die Matrixmultiplikation für diese Blöcke parallel aus. Die resultierenden Teilmatrizen werden anschließend zu einer einzigen Ausgabematrix kombiniert.

Das beschleunigt die Berechnungen erheblich, da die parallele Verarbeitung viel schneller ist als die sequenzielle Verarbeitung. Durch die Verwendung von Tensor Cores können GPUs und KI-Beschleuniger komplexe Deep-Learning-Modelle und KI-Anwendungen in kürzerer Zeit und mit höherer Energieeffizienz trainieren und ausführen.

Ein weiteres bemerkenswertes Merkmal von Tensor Cores ist ihre Fähigkeit, mit gemischter Präzision zu arbeiten. Das bedeutet, dass sie sowohl mit niedriger Präzision (z.B. halber Präzision, FP16) als auch mit höherer Präzision (z.B. einfacher Präzision, FP32) arbeiten können. In vielen Fällen reicht eine geringere Präzision für Deep-Learning-Anwendungen aus, was zu schnelleren Berechnungen und einem geringeren Speicherbedarf führt. Tensor Cores können jedoch bei Bedarf auch auf höhere Präzision umschalten, um genauere Ergebnisse zu erzielen.



(Nvidia, 2023)

Zusammenfassend sind Tensor Cores spezialisierte Hardware-Einheiten, die für die effiziente Verarbeitung von massiven Matrixmultiplikationen und Akkumulationen in

Deep-Learning-Anwendungen entwickelt wurden. Sie ermöglichen parallele Berechnungen und gemischte Präzision und tragen dazu bei, die Leistung und Energieeffizienz von GPUs und KI-Beschleunigern erheblich zu verbessern. Durch die Integration von Tensor Cores in moderne GPUs und KI-Beschleuniger können Entwickler und Forscher schneller und effizienter komplexe Deep-Learning-Modelle und KI-Anwendungen trainieren und ausführen.

Die Zukunft von Tensor Cores und ähnlichen Technologien sieht vielversprechend aus. Mit der kontinuierlichen Weiterentwicklung und Verbesserung von Hardware-Architekturen und der wachsenden Nachfrage nach KI-Anwendungen ist es wahrscheinlich, dass wir noch leistungsfähigere und effizientere Lösungen für Deep-Learning-Berechnungen sehen werden.

Ich selbst bin begeistert von den Möglichkeiten, die Tensor Cores und ähnliche Technologien bieten. Sie ermöglichen es uns, anspruchsvollere und leistungsfähigere Modelle zu erstellen und die Grenzen dessen, was mit künstlicher Intelligenz möglich ist, zu erweitern. Gleichzeitig wird durch die ständige Weiterentwicklung der Technologie die Barriere für den Einstieg in die KI-Forschung und -Entwicklung gesenkt, sodass immer mehr Menschen Zugang zu diesen leistungsstarken Werkzeugen haben und ihre eigenen Ideen verwirklichen können.

## **Ausblicke**

Die KI-Technologie hat sich in den letzten Jahren rasant entwickelt und wird in vielen verschiedenen Bereichen unseres Lebens schon eingesetzt auch in Bereichen wo man es nicht erwarten würde, was sowohl Hoffnung als auch Bedenken hinsichtlich der Auswirkungen auf Moral und Jobsicherheit in der Gesellschaft weckt.

Einerseits hat die KI das Potential, unser Leben zu verbessern und eine Vielzahl von Möglichkeiten zu eröffnen, sie kann uns einerseits bei alltäglichen Aufgaben unterstützen, die Effizienz in der Industrie sowie Wirtschaft erhöhen und sogar dazu beitragen komplexe Probleme wie zum Beispiel den Klimawandel oder wissenschaftliche Konzepte zu lösen. Außerdem wird künstliche Intelligenz schon zunehmend in der



Medizin eingesetzt um Ärzte bei der Diagnostik von Krankheiten und der Entwicklung neuer Therapiemöglichkeiten zu unterstützen.

Andererseits gibt es jedoch auch starke Bedenken bezüglich der Anwendung dieser Technologie hinsichtlich ethischer sowie gesellschaftlicher Folgen. Hierbei ist die Frage der Moral besonders wichtig, da wir eigentlich nicht anstreben uns selbst zu ersetzen oder dass KI-Systeme unethische Entscheidungen treffen.

Ein gutes Beispiel für dieses ethische Dilemma ist das selbstfahrende Auto, welches im Zweifelsfall eine Entscheidung treffen muss, um auf eine Situation zu reagieren. Nehmen wir an, es gibt drei mögliche Entscheidungen, bei denen jedoch in jedem Fall mindestens eine Person zu Tode kommen würde. Die künstliche Intelligenz müsste beispielsweise entscheiden, ob sie in eine Gruppe von Bauarbeitern ausweicht, den Fahrer selbst gegen eine Wand fahren lässt oder in eine Gruppe von Kindergartenkindern hineinfährt.

In solch einer Situation würde die künstliche Intelligenz wahrscheinlich das kleinste Übel wählen, etwa indem sie den Fahrer in die Wand fahren lässt. Alternativ könnte sie jedoch auch so reagieren, dass sie vorrangig den eigenen Fahrer schützt, falls das System darauf trainiert wurde. In jedem Fall zeigt dieses Szenario die ethischen Herausforderungen, die sich bei der Programmierung und Nutzung von selbstfahrenden Autos ergeben.

Auch ein gutes Beispiel stellt eine Studie von Open AI dar, bei der Wölfe und Hunde klassifiziert werden sollten und die KI auch immer 100% richtig entschieden hat, was ein Hund und was ein Wolf ist, jedoch nicht auf Grundlage des Tieres, sondern auf Grundlage des Bildes. Da es sich bei der Klassifizierung um ein unüberwachtes System handelt, hat die KI gelernt, diese Bilder in den meisten Fällen an dem Hintergrund zu erkennen. Und da auf den meisten Wolfsbildern im Hintergrund Schnee zu sehen war und auf den Hundebildern nicht, hatte die KI in Wirklichkeit gelernt, dass jedes Bild wo Schnee im Hintergrund ist, ein Wolfsbild sein müsste. Dies legt die Vermutung nahe, dass beispielsweise eine Künstliche Intelligenz, die ein Kind in einem Kriegsgebiet wahrnimmt, dieses möglicherweise als Bedrohung einstuft. Der Grund dafür könnte sein, dass die KI primär das Kriegsgebiet erkennt und nicht gezielt das Kind als unschuldiges Individuum differenziert.

Eine weitere Studie von Open AI die kürzlich herausgegeben wurde, behandelt das Thema Jobsicherheit in unserer Gesellschaft in Hinblick auf vermehrten Einsatz von Künstlicher Intelligenz und der Wahrscheinlichkeit in einzelnen Jobsparten ersetzt zu werden. Hierbei stellt die Studie heraus (OpenAI, GPTs are GPTs: An Early Look at the Labor Market Impact Potential of Large Language Models, 2023), dass künstliche Intelligenz schon früher als gedacht Berufe wie Mathematiker oder Steuerberater als Erstes ersetzen könnte. Bei diesen Berufen geht es häufig um die Betrachtung und Analyse eines Gesamtbildes. Für einen Computer ein relativ einfaches Problem, da dieser blitzschnell Daten verarbeiten kann und dementsprechend eine Künstliche Intelligenz das Gesamtbild aus allen analysierten Daten betrachten könnte.

Des Weiteren könnte Künstliche Intelligenz alsbald Berufe im Bereich der Kreativität übernehmen, da Systeme wie DALL-E 2 von Open AI gezeigt haben, wie gut sie sie darin sind, Bilder auf Anfrage zu erstellen oder große Sprachmodelle wie GPT-4 die Aufgabe von Designern übernehmen könnten, da GPT-4 aus einer einfachen Skizze innerhalb von Sekunden eine gut strukturierte Webseite entwerfen kann. Somit würde es auch Menschen die keine Ahnung von Web-Development haben, möglich sein, schnell ihre eigenen Systeme aufzubauen.

(OpenAI, GPTs are GPTs: An Early Look at the Labor Market Impact Potential of Large Language Models, 2023) (Team, 2022)

Ich selbst würde sagen, dass Künstliche Intelligenz schneller als befürchtet viele Jobs übernehmen wird oder zumindest Teilbereiche dieser Berufe, da es eine kostengünstige und effiziente Variante zu Arbeitskräften darstellt. Wo normalerweise 100 Leute arbeiten, könnte man diese durch ein einfaches Programm ersetzen.

Ich selbst nutze auch schon Künstliche Intelligenz in meinem eigenen Workflow, da diese mich zum Beispiel dabei unterstützt Fehler zu finden, meinen Code zu verbessern, oder weitere Möglichkeiten aufzeigt, ein Problem zu lösen. Diese Integration von Künstlicher Intelligenz in meinem eigenen Workflow hat dazu geführt, dass ich fünf bis sechsmal so schnell entwickeln kann, bei einer besseren Umsetzung des Problems und dementsprechend deutlich mehr Ergebnisse in der gleichen Zeit erziele.

In Anbetracht der Herausforderungen, die durch die mögliche Ersetzung zahlreicher Arbeitsplätze entstehen, ist es von entscheidender Bedeutung, den Fokus auf die Umschulung und Weiterbildung der Belegschaft zu legen. Hierbei sollte ihnen die Gelegenheit geboten werden, sich mit den neuesten Technologien vertraut zu machen und sich an die sich wandelnde Arbeitswelt anzupassen.

In Zukunft wird die Schaffung neuer Arbeitsplätze in der KI-Branche und in verwandten Bereichen zunehmend wichtiger werden, da die Ethik hinter Künstlicher Intelligenz immer mehr in den Fokus rücken wird.

Ein weiterer Aspekt, der berücksichtigt werden muss, ist die Verantwortung für die Entscheidungen, die von KI-Systemen getroffen werden. Es ist wichtig, klare Regeln und Gesetze zu entwickeln, die bestimmen, wer für Fehler oder Schäden haftbar ist, die durch KI-Anwendungen verursacht werden. Dies kann helfen, Vertrauen in die Technologie aufzubauen und sicherzustellen, dass KI-Systeme nicht unkontrolliert eingesetzt werden.

Letztendlich liegt die Verantwortung für die zukünftigen Auswirkungen von KI auf die Gesellschaft und die Arbeitswelt bei uns, weil wir diese Technologien entwickeln und einsetzen. Es ist unsere Aufgabe, uns der ethischen und sozialen Herausforderungen bewusst zu sein und entsprechende Maßnahmen zu ergreifen, um sicherzustellen, dass KI zum Wohle der gesamten Menschheit eingesetzt wird.

Zusammenfassend ist es entscheidend, bei der Erforschung und Entwicklung von KI-Systemen Moral und Jobsicherheit im Fokus zu behalten. Wir müssen uns auf die Schaffung von Rahmenbedingungen konzentrieren, die den verantwortungsbewussten Einsatz von KI fördern und gleichzeitig den Schutz der Menschenrechte gewährleisten. Durch Zusammenarbeit zwischen Informatikern, Ethikern, Regierungen und Industriepartnern könnten wir Richtlinien und Gesetze entwickeln, die die ethischen und sozialen Aspekte der KI-Technologie berücksichtigen.

Die zukünftigen Ausblicke von Künstlicher Intelligenz sind vielversprechend, aber es liegt an uns, die richtigen Entscheidungen zu treffen, um die Chancen zu maximieren und die Risiken zu minimieren. Wir sollten uns darauf konzentrieren, bestehende Arbeitskräfte umzuschulen und neue Arbeitsplätze zu schaffen, um den Übergang in eine KI-gesteuerte Zukunft zu erleichtern. Gleichzeitig müssen wir innovative soziale

Sicherheitsnetze entwickeln, um diejenigen zu unterstützen, die von den Veränderungen am stärksten betroffen sind.

## Literaturverzeichnis

- 3Blue1Brown. (2022, November 18). *But what is a convolution?* Retrieved from <https://youtu.be/KuXjwB4LzSA>
- Brotcrunsher. (2017, February 2). *Neuronale Netze - Backpropagation - Backwardpass*. Retrieved from <https://youtu.be/EAtQCut6Qno>
- Brotcrunsher. (2017, February 1). *Neuronale Netze - Backpropagation - Forwardpass*. Retrieved from <https://youtu.be/YIqYBxpv53A>
- Cecbur. (2019, February 14). *File:Convolutional Neural Network with Color Image Filter.gif*. Retrieved from [https://commons.wikimedia.org/wiki/File:Convolutional\\_Neural\\_Network\\_with\\_Color\\_Image\\_Filter.gif](https://commons.wikimedia.org/wiki/File:Convolutional_Neural_Network_with_Color_Image_Filter.gif)
- Dmitriev, D. (2019, October 29). *VGG16 Neural Network Visualization*. Retrieved from <https://youtu.be/RNnKtNrsmg>
- far1din. (2022, December 11). *Backpropagation in Convolutional Neural Networks from Scratch*. Retrieved from <https://youtu.be/z9hJzduHToc>
- GL, L. O. (2022). *GPU Computing*. Retrieved from <https://learnopengl.com/Guest-Articles/2022/Compute-Shaders/Introduction>
- Kirsanov, A. (2023, January 29). *Dendrites: Why Biological Neurons Are Deep Neural Networks*. Retrieved from <https://youtu.be/hmtQPrH-gC4>
- Nvidia. (2023). *Discover How Tensor Cores Accelerate Your Mixed Precision Models*. Retrieved from <https://developer.nvidia.com/tensor-cores>
- Ognjanovski, G. (2019, January 14). *Everything you need to know about Neural Networks and Backpropagation — Machine Learning Easy and Fun*. Retrieved from <https://towardsdatascience.com/everything-you-need-to-know-about-neural-networks-and-backpropagation-machine-learning-made-easy-e5285bc2be3a>
- OpenAI. (2021). *DALL-E: Creating Images from Text*. Retrieved from <https://openai.com/blog/dall-e/>
- OpenAI. (2021). *Introducing ChatGPT*. Retrieved from <https://openai.com/blog/chatgpt/>

OpenAI. (2023, march 27). *GPTs are GPTs: An Early Look at the Labor Market Impact Potential of Large Language Models*. Retrieved from OpenAI:  
<https://arxiv.org/pdf/2303.10130.pdf>

Team, G. L. (2022, November 23). *Types of Neural Networks and Definition of Neural Network*. Retrieved from <https://www.mygreatlearning.com/blog/types-of-neural-networks/>

wiktionary. (2023, January 25). *Neuron*. Retrieved from Wiktionary:  
<https://de.wiktionary.org/wiki/Neuron>

## Anhang

USB-Stick:

- Visual Studio 2019 Projekt (Convolutional Neural Network)
- Videos vom Scanner
- Schaltkreisfoto
- Einige Test Bilddaten (1 Apfel, 1 Erdbeere)
- Abschlussarbeit in PDF-Format.