



## **KOMPENSATIONSARBEIT IM MODUL SOFTWARE ENTWICKLUNG**

MASTER:

“DATA SCIENCE & INTELLIGENT ANALYTICS”

Betreuer:

Stefan Huber

Autor:

Jochen Hollich | 1810837475

Datum

22.12.2019

## Inhalt

Einleitung .....	3
Abgrenzung der Begrifflichkeiten.....	3
Abgrenzung Machine-Learning, Artificial-Intelligence und Deep-Learning.....	4
Theorie Neuronaler Netze .....	5
Biologische Neuronale Netze.....	5
Biologische Neuronale Netze   Nervensysteme .....	6
Lernen biologischer Nervensysteme.....	6
Technische Neuronale Netze.....	6
Arten Neuronaler Netze .....	12
Anwendung Neuronaler Netze im Kontext Data-Science .....	14
Rückschluss auf die Umsetzung in Python mit Keras .....	15
Convolutional Neuronal Network (CNN).....	18
CNN Hauptkomponenten.....	18
Python Implementierung Hyperparameter CNN.....	23
Auswertung Neuronaler Netze.....	25
Accuracy .....	25
Loss .....	25
Praktische Implementierung in Python .....	26
Datensätze .....	26
MNIST .....	26
CIFAR10 .....	27
Gewählte Implementierung .....	27
Ergebnisse.....	27
MNIST .....	27
CIFAR10 .....	29
Ausblick.....	30
Abbildungsverzeichnis .....	31
Literaturverzeichnis .....	31

## Einleitung

Viele Technologien der Menschheit finden ihren Ursprung in der Beobachtung natürlicher Phänomene. So inspirierten uns Vögel hinsichtlich der Luftfahrt, die Lotuspflanze war das Vorbild für Ingenieure bei der Entwicklung von Autolacken und die Fähigkeit des Geckos kopfüber entlang zu spazieren stand Pate für ultrastarke Haftfolien. Die gegenwärtige Phase wird auch als das digitale Zeitalter bezeichnet. *„Das „Digitale Zeitalter“ zeichnet sich durch mehrere zentrale Eigenschaften aus, die mit den Begriffen Digitalisierung, Vernetzung, Mobilität und Miniaturisierung erklärt werden können.“* (Karl-Heinz Bartling, Das Digitale Zeitalter). Mit den Konzepten der Bionik und den digitalen Treibern unserer Zeit erscheint es somit schlüssig eine Maschine nach dem Vorbild des menschlichen Gehirns intelligent zu gestalten.

Die Begriffe Artificial Intelligence, Machine-Learning und Deep-Learning sind Technologien, welche zugleich intensiv in den Medien diskutiert werden. Im Grunde genommen sind diese Disziplinen die praktische Implementierung und die technische Optimierung von mathematischen Algorithmen in einer rechenstarken, IT-gestützten Umwelt. Folglich werden unabhängig von der Komplexität der logischen und physischen Implementierung die Ziele der Algorithmik mit der Regression, Klassifikation (Binär, Multiclass, Multilabel, Multioutput), Clusterings oder Anomalie-Detection verfolgt. Auch wenn dies zunächst für den Leser abstrakt wirken mag, können viele Aufgaben aus dem privaten wie beruflichen Umfeld u.a. mit dieser Technologie automatisiert und beschleunigt werden.

Veranschaulicht an dem aktuell diskutierten Thema „Autonomes Fahren“ spielt die Technologie von Computer Vision (Teilgebiet des Deep-Learning) eine zentrale Rolle. Die Computer Vision befasst sich damit, aus digitalen Bildern und Videos Informationen herauszuarbeiten und diese Informationen für weitere Ver- und Bearbeitungsschritte bereitzustellen. Analog zum Beispiel des Autonomen Fahrens, kann diese Technologie in zahlreichen anderen Anwendungsgebieten / Domänen wie bspw. der Biomedizin, Überwachung, Umweltwissenschaften, Sozialwissenschaften verwendet werden.

Vorliegende Arbeit befasst sich mit der Verwendung von Neuronalen Netzen und im speziellen mit Convolutional Neuronalen Netzwerken (CNN), vorwiegend im Bereich der Computer Vision. CNNs können jedoch zugleich mit einer hohen Erfolgsquote in weiteren Klassifizierungsfragestellungen verwendet werden. Um die theoretisch beschriebenen Aspekte zu untermauern, wurden praktische Implementierungen der Konzepte in Jupyter-Notebooks auf Basis der Programmiersprache Python erstellt und an den entsprechenden Stellen dieser Arbeit auf sie verwiesen. Das gesamte Repository ist öffentlich zugänglich und kann unter [folgendem Link](#) heruntergeladen werden.

## Abgrenzung der Begrifflichkeiten

Bevor in der folgenden Ausarbeitung auf die nähere Erläuterung von biologischen Nervensystemen, technischen Neuronalen Netzwerken (NN) und anschließend auf die CNNs eingegangen wird, werden noch grundlegende Begrifflichkeiten geklärt.

## Abgrenzung Machine-Learning, Artificial-Intelligence und Deep-Learning

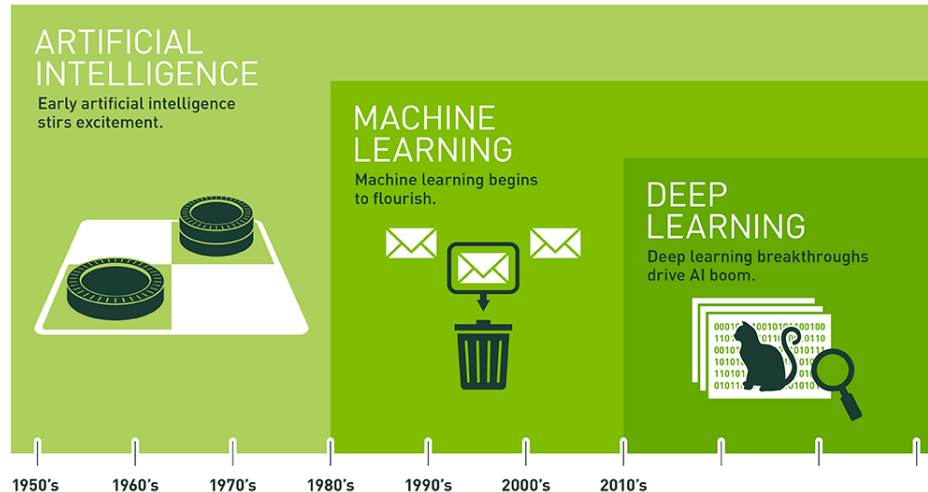


Abbildung 1: Historische Entwicklung AI, ML, DL  
(<https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/>)

Die obenstehende Grafik veranschaulicht die Entwicklung der Begriffe anhand der zeitlichen Historie.

### Artificial Intelligence (AI / KI)

AI / KI zielt darauf ab menschliches Verhalten durch Verwendung der Gesetze aus der Logiklehre, den Entscheidungsbäumen und den Konditionalverzweigungen zu imitieren. Berühmte Beispiele sind IBM's Deep Blue (Schachautomatisierung) Schlug im Jahr 1996 den damaligen Weltmeister Garry Kasparov) oder Googles AlphaGo, welches ausschließlich für die Automatisierung des Spiels Go dient, und ebenfalls erfolgreiche Go-Titel-Inhaber schlug. Der Unterschied der beiden Technologien lag darin, dass IBM's Deep Blue jeden weiteren möglichen Spielzug im Laufe eines Schachspiels „brute-forced“. Dieser „einfache“ Ansatz war jedoch durch die höhere Komplexität des Spieles Go nicht in dieses übertragbar. Die „Intelligenz“ von Alpha Go liegt in der Art wie das Spiel gelernt wurde. Hierbei wurde ein Reinforcement-Algorithmus implementiert, in welchem Alpha-Go lernte Go effizient zu spielen.

### Machine-Learning (ML)

ML bietet für ein System die Möglichkeit aus bestehenden Daten zu lernen, ohne explizit auf den möglichen Output hin programmiert zu sein. Dabei werden die Vorgehensweise der Algorithmik praktisch implementiert.

Im Gegensatz zum Deep-Learning kann im Machine-Learning ein Mensch in die Datenanalyse (noch) eingreifen.

### Deep-Learning (DL)

DL ist ein Teilbereich des ML. Hierbei werden Muster aus Datensätzen mithilfe von Neuronalen Netzwerken extrahiert. Neuronale Netze wiederum basieren auf der Funktionsweise des biologischen Nervensystems. Somit kann der Prozess des Lernens automatisiert werden. Neuronale Netze adaptieren die eigenen Parameter (Gewichte) während des Lernvorgangs, dem sogenannten „Trainings“. Diese entstehenden Modelle werden anschließend in der Verwendung Neuronaler Netze für die Prognose verwendet.

Der Kern der vorliegenden Arbeit befasst sich mit dem Thema Neuronale Netze und im Speziellen mit Convolutional Neural Networks (CNN). CNN ist ein Vertreter des Deep-Learning. Mit diesem Setup bietet sich grundsätzlich die Analyse von Bilddaten an.

Im praktischen Teil werden zu den Beispiel-Datensätzen MNIST und CIFAR10 entsprechend zunächst Multilayer-Perzeptrone und anschließend CNNs aufgebaut.

### Theorie Neuronaler Netze

Neuronale Netze – sowohl bei biologischer als auch bei technischer Betrachtungsweise – verfolgen das Ziel zu lernen, um auf Basis des Erlernten Prognosen abgeben zu können. Dabei wird eine sich verändernde aber ähnliche Umwelt analysiert, um mit dem Erlernten Aussagen über ähnliche Bedingungen oder Situationen tätigen zu können. Die Komplexität der untenstehenden Analyse biologischer Nervensysteme / Neuronen übersteigt die hier ausgeführte Schilderung bei Weitem. Das folgende stark vereinfachte biologische Beispiel wurde lediglich gewählt, um zu der Funktionsweise technischer Neuronaler Netze hinzuführen. Somit wurde nicht tiefer als nötig auf die Funktionsweise biologischer Neuronen und Nervensysteme eingegangen.

### Biologische Neuronale Netze

Vereinfacht gesprochen werden im Nervensystem Reize in Form von elektrischen Signalen aufgenommen, über das „Neuronengeflecht“ mittels elektrischer Signale zu den entsprechenden Destinationen hin weitergeleitet und zuletzt an entsprechender Stelle verarbeitet ausgegeben. Dabei bilden die Neuronen in unserer Betrachtung in dieser Arbeit die kleinsten individuellen Einzelsysteme für diese Signal-Weiterleitung.

In einem nächsten Schritt wird, vor der Betrachtung des „großen Ganzen“, bzw. der Funktionsweise des Nervensystems, auf die Funktionsweise eines einzelnen biologisch bipolaren Neurons eingegangen.

Ein einzelnes biologisches Neuron vereint die nachfolgenden Bestandteile mit den entsprechenden Funktionen in sich:

#### 1) Nervenzelle | Neuron

Diese Bezeichnung beschreibt eine Zusammensetzung bestehend aus den folgenden Komponenten:

##### a. Dendriten | Input

Dendriten sind die Teilbereiche bei welchen die elektrischen Signale (bspw. kommend von vorgelagerten Neuronen) eintreffen und zum Zellkern weitergeleitet werden. In unsere Betrachtungsweise hat ein Neuron mehr als ein eingehendes Dendrit. Die eintreffenden Signale können reizend oder hemmend sein.

##### b. Zellkörper | Schoma | Verarbeitung

Alle eintreffenden Signale von den Dendriten werden in der Zelle verarbeitet. Abhängig von dem Outcome dieser Verarbeitung wird der Status des Zellkörpers entweder in Form des Aktionspotentials aktiviert oder deaktiviert über das Axon ausgegeben. Die Höhe des Schwellwerts für die Zustandsänderung ist anhängig von dem jeweiligen Neuron. Die Ausgabe eines Neurons hängt somit einerseits von den eintreffenden Signalen und der Verarbeitung der jeweiligen Nervenzelle ab.

##### c. Axon | Output

Axonen sind Teilbereiche des Neurons welche das Signal des gegenwärtig fokussierten Neurons ausgeben.

#### 2) Synapse

Als Synapsen bezeichnet man die Schnittstellen zwischen Axon und Dendrit.

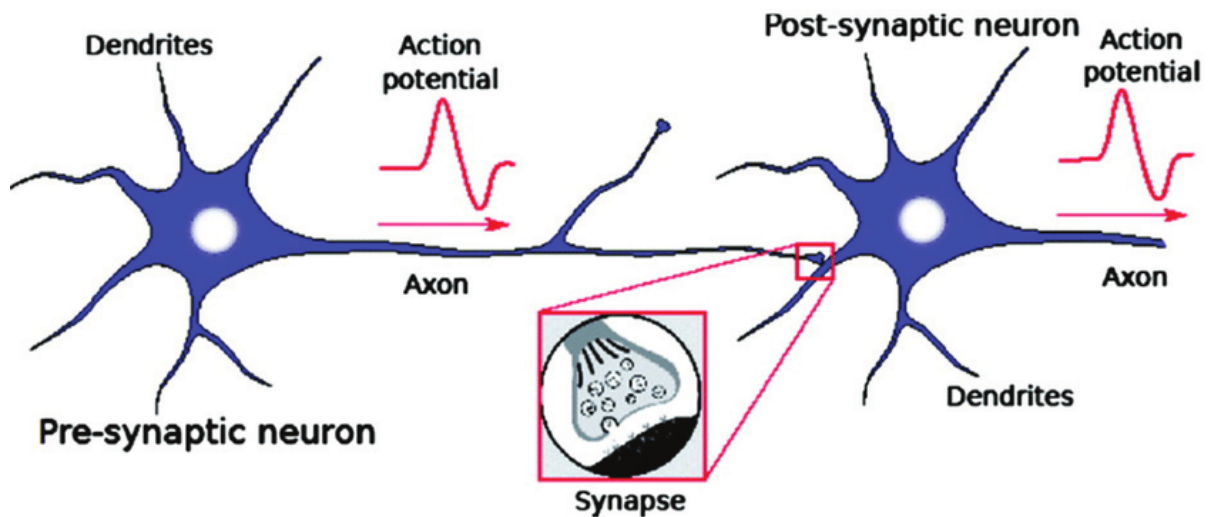


Abbildung 2, Neuron-Aufbau (<https://medicalxpress.com/news/2018-07-neuron-axons-spindly-theyre-optimizing.html>)

Zusammenfassend kommt über eine Synapse ein eingehendes elektrisches Signal am Dendriten unseres betrachteten Neurons an. Dieses Signal (und alle weiteren eintreffenden Signale kommend von weiteren dritten Neuronen) wird an den verarbeitenden Zellkern weitergeleitet. Abhängig von der Aktivierung des Zellkerns gibt dieser ein entsprechendes Aktionspotential an das Axon, welches wiederum über eine Synapse mit der nachgelagerten Nervenzelle verbunden ist.

#### Biologische Neuronale Netze | Nervensysteme

Wie oben geschildert können Neuronen über die Synapsen an dem Axon und Dendriten miteinander verbunden sein. Durch entsprechende parallele und sequentielle Anordnung von Neuronen entstehen somit komplexe Neuronen-Netzwerke, die sogenannten Nervensysteme. „Das menschliche Gehirn besteht aus etwa  $10^{10}$  bis  $10^{12}$  miteinander vernetzten Nervenzellen, den Neuronen“. (Selle, Künstliche Neuronale Netze, S.1)

#### Lernen biologischer Nervensysteme

„Synapsen übertragen nicht nur elektrische Signale von einer Nervenzelle zur nächsten, sie können die Intensität des Signals auch verstärken oder abschwächen. Diese sogenannte synaptische Plastizität ist die Grundlage von Lernen und Gedächtnis“ (Max-Planck-Institut, SYNAPTISCHE PLASTIZITÄT - WIE DAS GEHIRN LERNT). Konkret bedeutet dies das, wenn ein biologisches Individuum einen Vorgang (gleich ob kognitiver oder motorischer Art) übt, währenddessen die entsprechenden Synapsen und somit der nachgelagerte Zellkern häufiger getriggert werden. Durch diese frequente Reizung lassen diese Neuronen, durch das Üben und somit häufigere Nutzung der entsprechenden Synapse und Nervenzelle, ein Signal schneller durch das Neuron „passieren“. Dieses Phänomen nennt man Langzeitpotenzierung. Lernen findet somit in den Synapsen und dem Zellkern statt und bewirkt das Impulse effizienter von einem Neuron zum nächsten übertragen werden.

#### Technische Neuronale Netze

Auf Basis des Grundverständnisses biologischer Netzwerke werden nachfolgend technische Neuronale Netzwerke betrachtet. Der Transfer des zuvor gewählten Beispiels - der Beschreibung der Funktionsweise eines einzelnen biologischen Neurons - in das technische Umfeld, lässt das „einfachste“ Perzeptron-Neuronale-Netzwerk entstehen. Die weitere Ausarbeitung und somit folgende Erklärung befasst sich mit der Annahme von einem

Supervised-Learning Ansatz. Die anschließende praktische Ausarbeitung verwendet das in den vorliegenden Kapiteln beschriebene Setup.

Ein Neuronales Netz, im nachfolgend beschriebenen Fall das Perzeptron, besteht ausschließlich aus einem einzelnen Neuron. Die Erklärung erfolgt anhand folgender Darstellung:

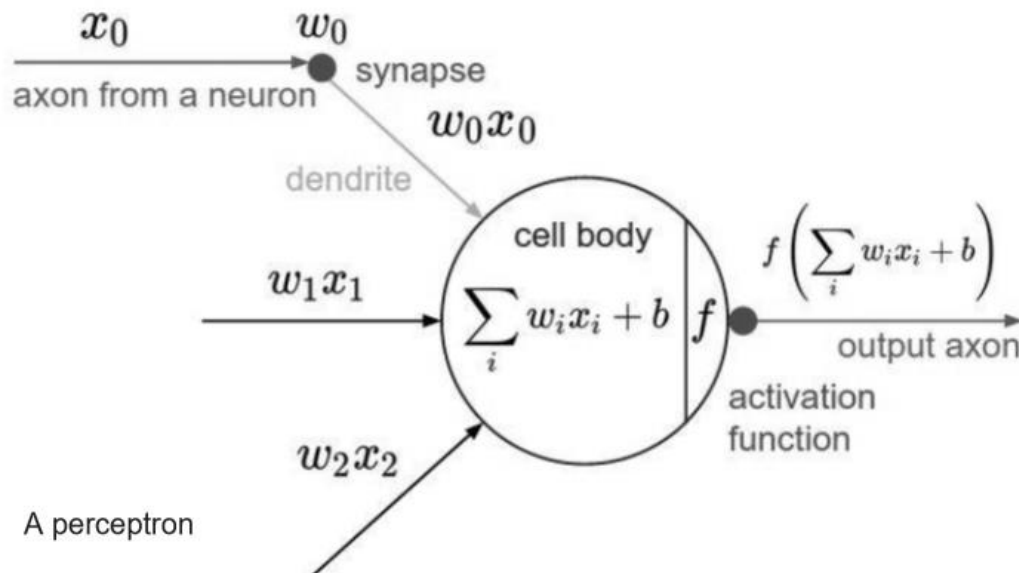


Abbildung 3, Veranschaulichung Perceptron, Vorlesungsunterlagen Machine-Learning/ Teil 4 Deep-Learning – FH Kufstein SS19

Die gerichteten Bestandteile eines Perzeptron-Neuronalen-Netzwerks lauten wie folgt:

- a) Input-Layer  
Dies ist der Startpunkt eines Neuronalen Netzes. In diesem Punkt werden die ggf. bereits vorverarbeiteten Daten in das NN „gespeist“. Neuronale Netze erwarten meist eine Anordnung der eingehenden Elemente ausgedrückt durch einen mathematischen Vektor oder einer Matrix.
- b) Input eines Neurons |  $x$   
Der Input eines spezifischen Neurons ist die Summe aller einkommenden Signale multipliziert mit dem Gewicht, welches spezifisch für das einkommende Dendrit / Synapse gilt.
- c) Gewicht |  $w$   
Technische Neuronale Netze sind durch die Gewichtung der gerichteten Dendriten geprägt. Diese Gewichte werden während des Trainings von Neuronalen Netzen adaptiert.
- d) Zellkörper | Activation-Function  $f$   
In der Aktivierungsfunktion fließen alle einkommenden Produkte (Input \* Gewicht) aus den existenten Dendriten zusammen. Abhängig von der Höhe der Summe aller Produkte und der gewählten Aktivierungsfunktion (bspw.- „Sigmoid-Funktion“, Hyperbolic Tangent, „Rectified Linear Unit“) wird ein Signal weitergeben oder nicht. Je nachdem wie stark ein Neuron aktiviert wurde gibt es ein entsprechend hohes oder niedriges Output-Signal weiter.



Die Auswahl der Aktivierungsfunktion ist Modellabhängig. Die wohl bekanntesten Aktivierungsfunktionen sind:

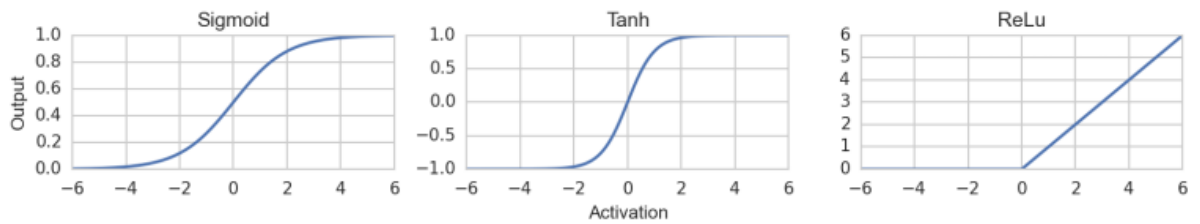


Abbildung 4, Darstellung gängiger Aktivierungsfunktionen, <https://www.cbcity.de/tutorial-neuronale-netze-einfach-erklart>

e) Output Axon | y

Ob und in welcher Höhe ein Neuron ein Signal weitergibt (feuert) hängt von der Höhe des Inputs in Zusammenhang mit der Activation-Function ab. Die Weitergabe an das nachgelagerte Neuron erfolgt auf dem Output-Axon.

Im konkreten Fall des Perzeptron-Neuronalen-Netzwerkes können lediglich die Werte 0 oder 1 weitergegeben werden.

f) Hidden Layer | Zwischenschicht

Zwischen Input und Output Layer befindet sich in einem Neuronalen Netzwerk mindestens ein Hidden-Layer. Je mehr Hidden Layer innerhalb eines Neuronalen Netzes bestehen, desto tiefer ist das Modell. Dabei hat jedes Hidden-Layer während des Lifecycles des Modelles eine konstante Anzahl an Neuronen parallel geschaltet. Innerhalb eines Modells mit mehreren Hidden Layern können die einzelnen Hidden-Layer eine unterschiedliche Anzahl an Neuronen besitzen. „Wenn jedes Neuron mit jedem Neuron des nächsten Layers verbunden ist, dann spricht man von ‘Fully Connected Layer’ (auch ‘Dense-Layer’). Es gibt dann so viele Gewichte  $w$ , wie es Verbindungen gibt“ (Balzer, Neuronale Netze einfach erklärt).

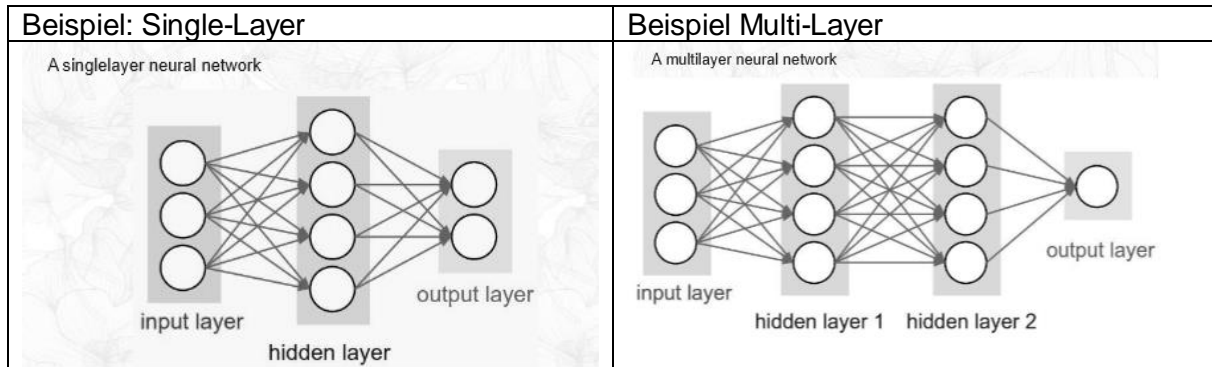
g) Der Output-Layer

Der Output-Layer „liegt hinter den Zwischenschichten und bildet die letzte Schicht in einem künstlichen neuronalen Netzwerk“ (Moser, KÜNSTLICHE NEURONALE NETZE – AUFBAU & FUNKTIONSWEISE). Ob lediglich ein einzelner Output-Wert oder mehrere mögliche Output-Werte ausgegeben werden, hängt von der Art der Implementierung bzw. des zugrundeliegenden Use-Cases ab.



## Technische Neuronale Netze | Architekturen

Ähnlich den biologischen Neuronen können technische Neuronen ebenfalls parallel und sequentiell aufgespannt werden, dabei entstehen Neuronale Netzwerke. Auch wenn sich hierdurch unterschiedliche Architekturen bilden können, gibt es folgende Gemeinsamkeiten: „Die Neuronen (auch Knotenpunkte) eines künstlichen neuronalen Netzes sind schichtweise in sogenannten Layern angeordnet und in der Regel in einer festen Hierarchie miteinander verbunden. Die Neuronen sind dabei zumeist zwischen zwei Layern verbunden (Inter-Neuronlayer-Connection), in selteneren Fällen aber auch innerhalb eines Layers (Intra-Neuronlayer-Connection).“ (Moser, KÜNSTLICHE NEURONALE NETZE – AUFBAU & FUNKTIONSWEISE)



Werden die beiden obenstehenden NNs verglichen, so werden folgende Gemeinsamkeiten ersichtlich:

- a) Jedes Neuronale Netz hat ein Input-Layer. Dieser Layer kann beispielsweise ein Vektor oder eine Matrix sein.
- b) Architektur
  - a. Einerseits besteht die Architektur aus der Anzahl der Hidden-Layer (Hidden Layer = weder Input noch Output-Layer).
  - b. Jeder Layer wiederum besteht aus einer Anzahl von Neuronen.
- c) Jedes Neuronale Netz hat einen Output-Layer.
  - a. Dies kann ein Single-Output (bspw. Implementierung einer binären Klassifikation) ...
  - b. ...oder ein Multioutput (bspw. Implementierung einer Multi-Output-Klassifikation/ MultiClass-Klassifikation) sein.

Vice-Versa können folgende Unterschiede von Neuronalen Netzen detektiert werden:

- a) Anzahl der Hidden Layer
- b) Anzahl der einkommenden Dendriten eines individuellen Neurons
- c) Anzahl der ausgehenden Axone eines individuellen Neurons.

Des Weiteren können Neuronalen Netze auf unterschiedliche Arten, abhängig vom Status Ihrer Verwendung, genutzt werden. Hierzu fokussiert man sich zunächst auf folgende Durchlaufmöglichkeiten:

- a) **Forward-Propagation**  
Innerhalb der Forward-Propagation wird das Neuronale Netz vom Input bis hin zum Output einmalig durchlaufen. Hierbei bleiben die Gewichte des Neuronalen Netzes **nicht** verändert.
- b) **Backward-Propagation**  
Bei der Backward-Propagation wird das Neuronale Netz rückwärts ausgehend vom

Output hin zum Input durchlaufen. Hierbei werden die Gewichte des Neuronalen Netzes adaptiert. Dies ist ein zentrales Element des Lernens.

Diese Möglichkeiten des Durchlaufens kommen unterschiedlich zum Einsatz:

### **Verwendung von Forward- & Backward-Propagation im Training**

Das Training eines Neuronalen Netzes ist der Prozess, in welchem die Gewichte des Neuronalen Netzes angepasst werden. Dies geschieht nach dem folgenden Konzept:

- 1) Die Gewichte eines neuronalen Netzes werden randomized gesetzt. Diesen Prozess wird auch als die Initialisierung des Netzwerkes bezeichnet.
- 2) Gelabelte Trainingsdaten durchlaufen das Neuronale Netzwerk (Zentrale Parameter Input  $X$ , Gewicht  $w$ , gewählte Aktivierungsfunktion  $f$ ).
- 3) Der entstehende Output  $y$  wird innerhalb der Cost- / Loss-Function ausgewertet. Konkret bedeutet dies, dass der vorhergesagte Wert mit dem tatsächlich bekannten Wert der gelabelten Trainingsdaten verglichen wird.
- 4) Die Cost-Function wird mittels der Backward-Propagation und einer Optimierungsfunktion (Anpassung der Gewichte der Neuronen) optimiert. Mögliche Optimierungsfunktionen werden in untenstehender Grafik veranschaulicht.

Dies bedeutet konkret, dass auf Basis der Ergebnisse der Cost-Function durch das NN mittels der Backpropagation geprüft wird, welche Neuronen, für den Fehler verantwortlich waren. Diese Neuronen werden anschließend adaptiert, sodass der Fehlerwert der Loss-Function geringer wird. Diese Suche nach der idealen Aktivierung bedeutet mathematisch das Finden von Minimalwerten im Fehlerraum.

Folgende Grafik veranschaulicht unterschiedliche Optimierungsstrategien grafisch (sofern diese nicht animiert ausgegeben wird, kann sie unter „<https://github.com/JHC90/SoftwareDev-Huber/blob/master/Sources/Loss-Functions.gif>“ eingesehen werden):

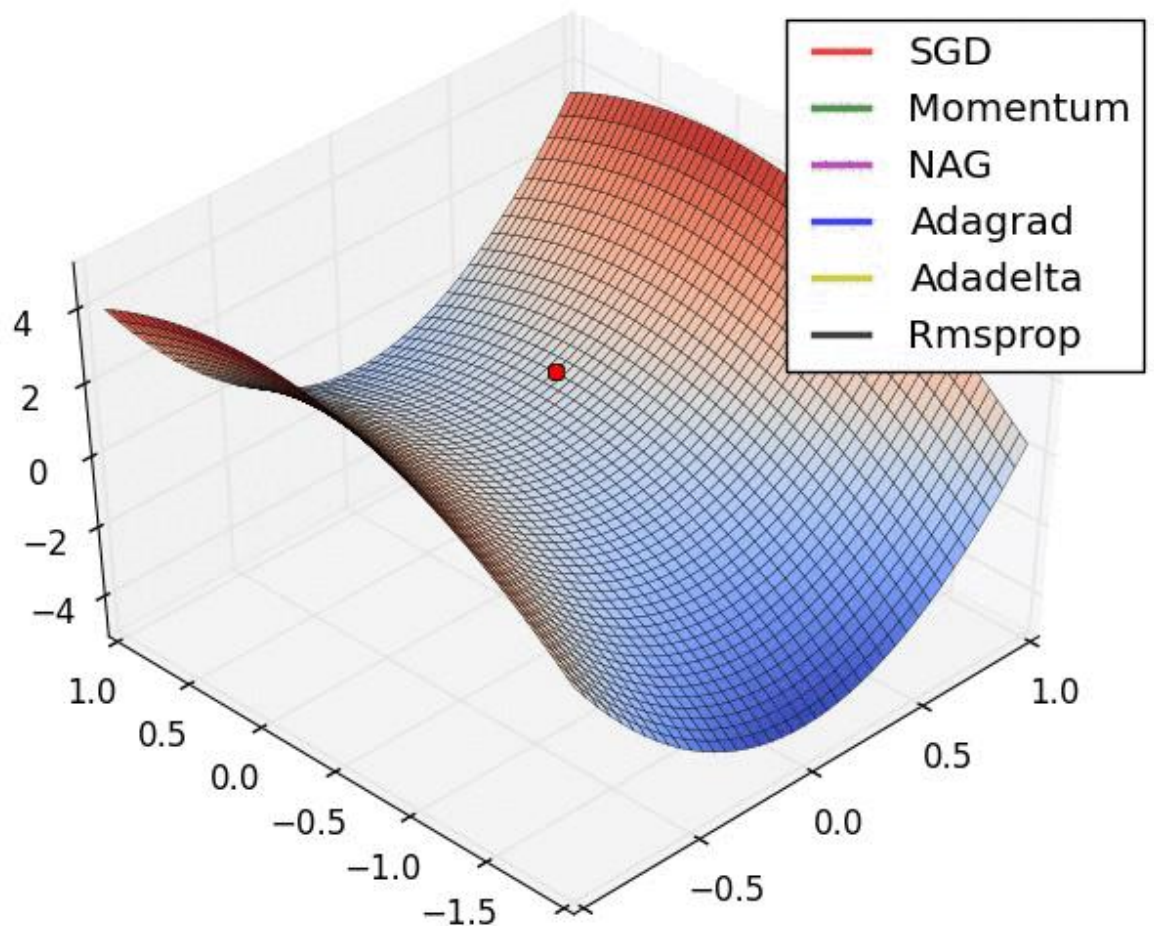


Abbildung 5, Mögliche Optimierungen der Fehlerfunktion, <https://www.cbcity.de/tutorial-neuronale-netze-einfach-erklart>

- 5) Dieser Vorgang wiederholt sich iterativ für alle Instanzen der Trainingsdaten, ist der gesamte Trainingsdatensatz einmalig durch das Neuronale Netz „geflossen“, so ist eine Epoche abgeschlossen.
- 6) Innerhalb des Trainings des Neuronalen Netzes können mehrere Epochen vorkommen.

Dieser Prozess der Anpassung der Gewichte wird das „Lernen“ eines Neuronalen Netzes genannt.

Zusammenfassend kann für den Lernprozess eines Perzeptrons folgendes beschrieben werden:

„ -Wenn die Ausgabe eins (aktiv) ist und eins sein soll oder wenn sie null (inaktiv) ist und null sein soll, dann werden die Gewichtungen nicht verändert.  
 - Wenn die Ausgabe null ist, aber eins sein sollte, werden die Gewichtungen für alle aktiven Eingabeverbindungen erhöht.  
 - Wenn die Ausgabe eins ist, aber null sein sollte, werden die Gewichtungen für alle aktiven Eingabeverbindungen verringert.“ (Selle, Künstliche Neuronale Netze, S.15)

a) **Verwendung der Forward-Propagation im trainierten Modell zur Prognose**

Wurde ein Neuronales Netz trainiert, so wird dieses trainierte Modell in der Regel für die Prognose verwendet. Prognostizieren bedeutet in diesem Fall, dass keine gelabelten Daten vorhanden sind. Ziel ist es mit den Input-Daten den Output

vorherzusagen.

Technisch ausgedrückt bedeutet dies, dass das trainierte Neuronale Netzwerk einmalig mit der Forward-Propagation durchläuft und eine Prognose für den Output Y liefert.

### Arten Neuronaler Netze

Wie aufgezeigt wurde, unterscheiden sich Neuronale Netze in der Tiefe (Anzahl der Hidden Layer) und in der Höhe (Anzahl der Neuronen in einem Layer).

Je nach Anordnung entstehen somit unterschiedliche Architekturen, welche wiederum anwendungsspezifisch verwendet werden. Eine Auflistung und Veranschaulichung geläufiger Architekturen findet sich in nachfolgender Grafik:

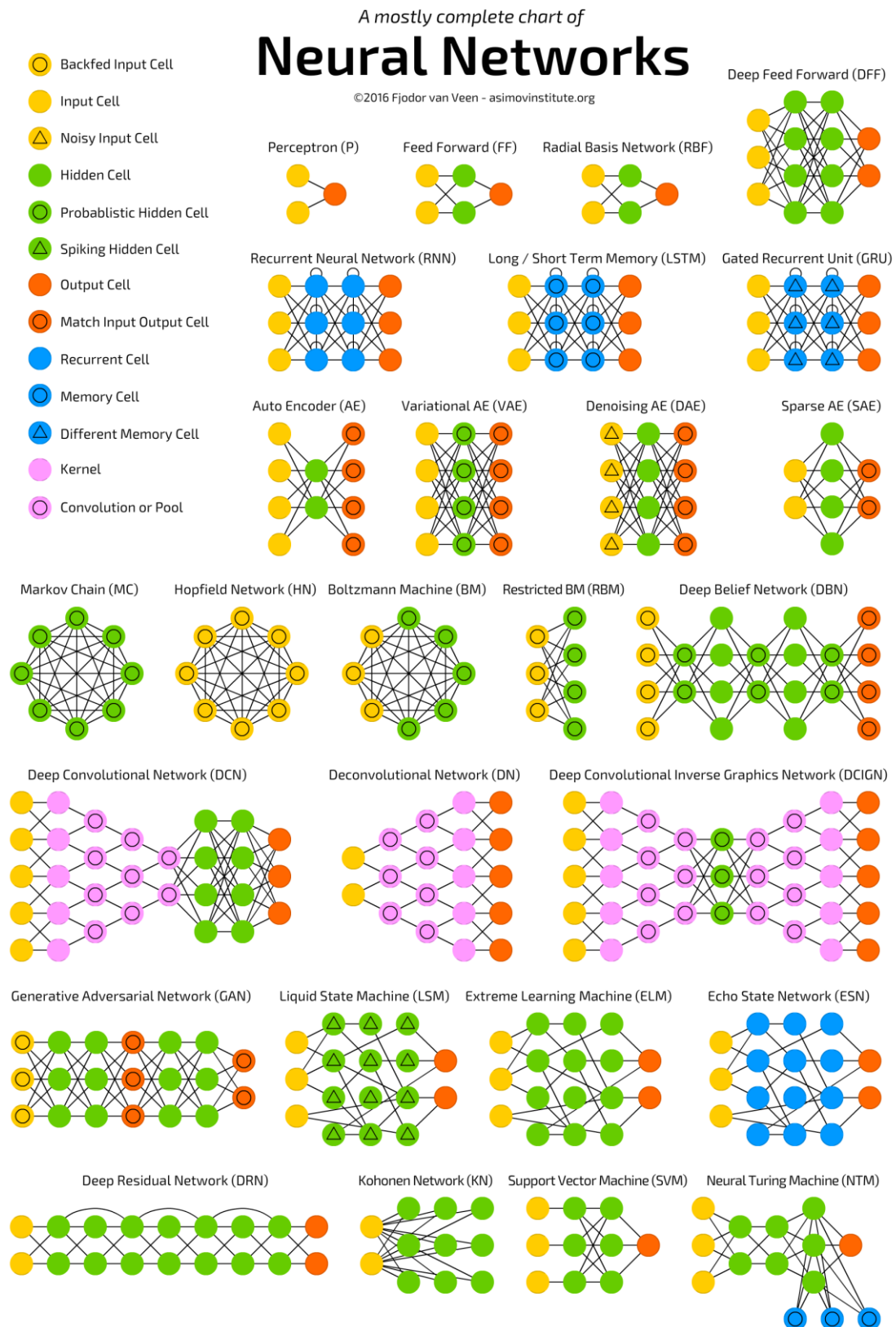


Abbildung 6, Veranschaulichung NN Architekturen, <https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464>

### Anwendung Neuronaler Netze im Kontext Data-Science

Wie bereits einleitend erwähnt, sind die Ziele des Deep-Learning entweder innerhalb der Regression, Assoziation, Klassifikation, Clustering oder Anomalie-Detection. Im Vergleich zu den Machine-Learning Technologien werden im Deep-Learning leistungsstärkere Ansätze mittels Neuronaler Netze verwendet. Diese individuell auf den Anwendungsfall angepassten Netzwerke besitzen die Charakteristik, dass sie autark von Datensätzen lernen. Lernen heißt im konkreten Beispiel, dass die Gewichte der einzelnen Neuronen innerhalb des Netzwerkes entsprechend angepasst werden.

Somit bieten Neuronal- & Deep-Networks durch das charakteristische Konzept des Lernens eine „andere“ Herangehensweise an die genannten Problemstellungen als die Algorithmen des Machine-Learnings.

## Rückschluss auf die Umsetzung in Python mit Keras

Dieses Unterkapitel dient als Cheat-Sheet für die Implementierung Neuronaler Netze mit Keras.

<u>Paramter</u>	<u>Beschreibung</u>	<u>BSP Paramter in Tensorflow</u>	<u>Mögliche Parameter</u>
Batch Size	In ML / DL Projekten werden häufig Datenmengen bearbeitet, welche so groß sind, dass diese nicht zeitgleich in einem NN verarbeitet werden können. Aus diesem Grund werden die gesamten Daten in Batches der Größe „batch Size“ aufgeteilt. Jeder Batch wird letztlich sequentiell isoliert am NN trainiert.	Batch_size = 128	
Iteration	Wird die Anzahl der Trainingsinstanzen geteilt, erhält man die Batch Anzahl. Nachdem jeder Batch einmal das gesamte Netz durchlaufen muss bevor der gesamte Datensatz verarbeitet wurde, ist somit die Anzahl der Iterationen gleich der Anzahl der Batches.	Die Iteration wird indirekt über die Batch Size und die „Anzahl“ der Trainingsinstanzen bestimmt.	
Epoche	Durchläuft ein Datensatz (oder einmal alle Batches) das Neuronale Netz, so entspricht dies einer Epoche. Während des Trainings eines NN werden im Grunde genommen über Forward- & Back-Propagation die Gewichte des Netzes angepasst. Somit gibt es einen Mehrwert, wenn der	epochs = 20	



	gesamte Datensatz „mehrfach“ (= mehrere Epochen) durch das Netz fließt und die Gewichte anpasst.		
Layer / Dense	Wie zuvor erläutert unterscheiden sich die Architekturen Neuronaler Netze durch die unterschiedliche Anzahl von Layern, der Anzahl der Neuronen je Layer und die Aktivierungsfunktion der Neuronen.	model.add(Dense(512, activation='relu', input_shape=(784,)))	Ein Layer wird mit 512 Neuronen zu dem Modell hinzugefügt, und anschließend die Aktivierungsfunktion aufgerufen. Relu(Alternativ übliche Aktivierungen = Hyperbolic tangent, sigmoid)
Dropout rate	Anzahl der Neuronen in jedem Layer welche während des Trainings (konkret während eines Durchlaufs, welche Neuronen konkret nicht berücksichtigt werden ist Zufall) nicht aktiv sind → dient dem Entgegenwirken des Overfittings (Reduktion der Komplexität), nach Ablauf Training wird die Dropout-Rate deaktiviert.	model.add(Dropout(0.2))	
compile	Mit diesem Befehl wird das bisher beschriebene NN initialisiert. Für die Initialisierung muss die Loss-Function definiert werden, der Optimizer und die Evaluierungsmetric	model.compile(loss='categorical_crossentropy', optimizer=RMSprop(), metrics=['accuracy'])	<p><b><u>Loss-Function</u></b>  Die Loss-Function wird je nach Business-Case geändert. In Klassifikationsfragestellungen sind mögliche Funktionen die Categorical crossentropy oder die Log-Loss.  Bei Regressionsfragestellung wird häufig die L1 oder L2 Norm verwendet. Mögliche Loss-Functions innerhalb von Keras befinden sich unter folgendem <a href="#">Link</a>.</p> <p><b><u>Optimizer</u></b>  Hier wird der Optimizer gewählt, nach welchem die Loss-Function optimiert werden soll</p>

			<b><u>Metrics</u></b> Die Metric ist grundsätzlich ähnlich der Loss-Funtion. Diese wird jedoch nicht während des Trainings sondern im späteren produktiven Einsatz angewendet (nach der Verwendung der Trainingsdaten, werden die Testdaten für die Ermittlung der Accuracy verwendet).
Fit	Hier wird das Modell entsprechend ausgeführt.	<pre>history = model.fit(x_train, y_train,                     batch_size=batch_size,                     epochs=epochs,                     verbose=1, # show training                     progress during training Possible = 0,1,2                     validation_data=(x_test, y_test))</pre>	

## Convolutional Neuronal Network (CNN)

Ein CNN ist ein Neuronales Netz, welches nach dem Vorbild des humanen visuellen Cortex entwickelt wurde. Das ursprüngliche Ziel von CNNs ist die Entwicklung der Computer Vision. Diese Netzwerke werden vorrangig für die Bearbeitung von Video- und Audiodateien verwendet, werden aber auch in anderen Klassifizierungsfragestellungen verwendet.

Die nun im Weiteren beschriebenen CNNs, dienen der Image-Classification.

Um die Funktionsweise eines Convolutional Neuronal Networks zu verstehen wird zunächst das Verständnis von Neuronalen Netzen, um die Eigenschaften, Layer und der Funktionsweisen der CNNs angereichert.

### CNN Hauptkomponenten

In der weiteren praktischen Ausarbeitung dieser Arbeit wird mit Bildmaterial aus den bekannten Datensätze MNIST und CIFAR10 gearbeitet. Daher wird in der Beschreibung der Funktionsweise von CNNs auch ein Beispiel der Bildanalyse verwendet.

Ein CNN ist durch folgende veranschaulichte Komponenten geprägt:

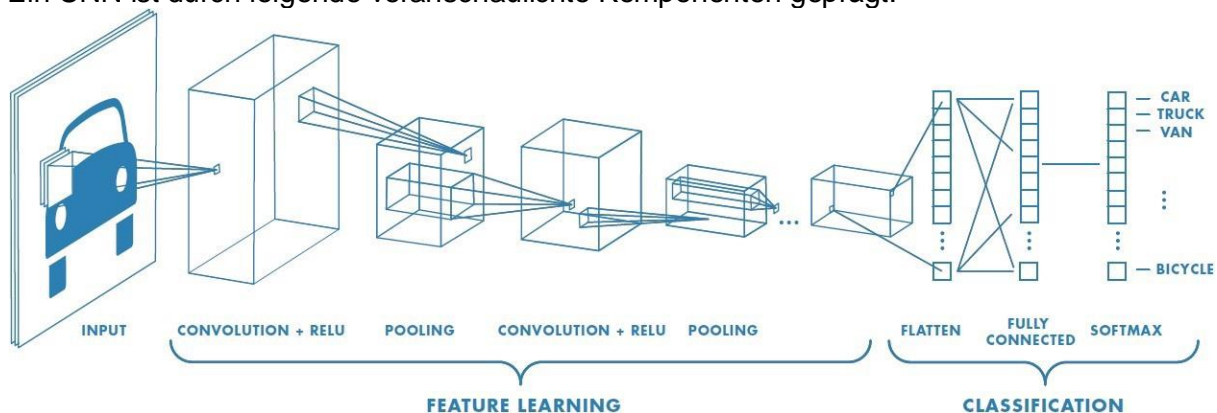


Abbildung 7, Aufbau CNN, <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>

Auf die entsprechenden Komponenten wird nachfolgend sequentiell eingegangen.

### Input

Ein markanter Unterschied zwischen „einfachen“ Neuronalen Netzen wie beispielsweise einem Multi-Layer-Perceptron und CNNs ist die Verarbeitung der Input-Daten. Ein Neuronales Netz erwartet einen Input-Vector, wohin gehend ein CNN mit Matrizen als Input-Data beginnt.

Digitale Bilder sind zunächst durch die Pixelanzahl (Pixelhöhe x Pixelbreite) und das verwendete Farbschema (Schwarz / weiß | Binär oder Farbbilder) geprägt. Durch eine Anordnung dieser Parameter entstehen Formen und Konturen, welche von Menschen als digitale Bilder wahrgenommen werden und in Form von Foto- oder Video-Material konsumiert werden können. Technisch kann die Darstellung als eine Matrix, aufgespannt aus Pixelhöhe und Pixelbreite, vorgestellt werden, welche die jeweiligen Farbwerte beinhaltet.

Eine technische Herausforderung der Computer Vision besteht darin, dass unterschiedlicher Bilder gleicher zugrundeliegender Motive (beispielsweise die Bilderserie eines Autos aus unterschiedlichen Blickwinkeln) für den Menschen klar als dasselbe Objekt identifizierbar sind, für technische Systeme jedoch „auf den ersten“ Blick gänzlich diverse Objekte vorliegen.

Veranschaulicht am einfachen Beispiel der Schwarz-Weiß-Bilder „X“ und „O“ können folgende Änderungen aus technischer Perspektive festgehalten werden:

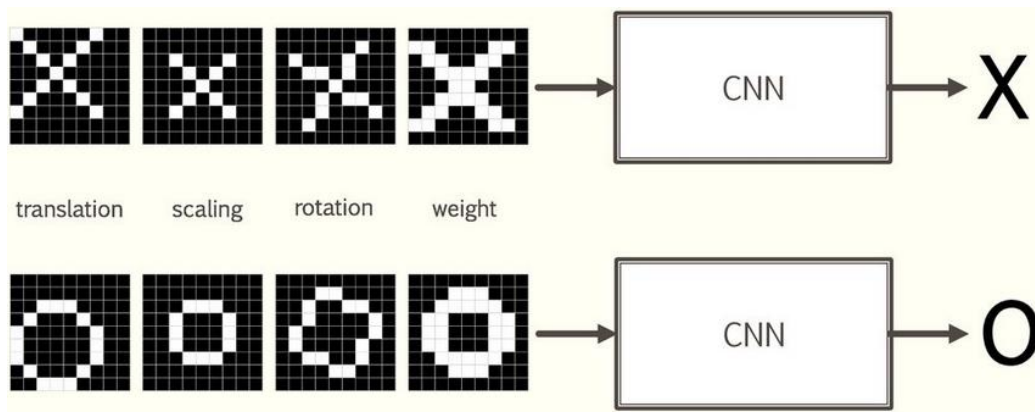


Abbildung 8, Image-Unterschiede Schwarz / Weiß einfacher Objekte, <https://slideplayer.com/slide/14855329/>

Um das Verständnis für die Bildbearbeitung und die oben beschriebenen Bildunterschiede aus technischer Perspektive zu festigen wurden folgende Jupyter-Notebook Ausarbeitungen angefertigt. Die Aufgabenstellungen stammen aus dem Buch „[Programmieren-Trinieren](#)“ von Luigi Lo Iacono, Stephan Wiefling und Michael Schneider

- 1) [Invert & Switch](#) | Translation
- 2) [Shaddow](#) | Weight
- 3) [Sizing](#) | Scaling
- 4) [Rotation](#) | Rotation.

Ziel eines CNNs ist es, Bilder gleicher Motive jedoch aus unterschiedlichen Parametern dennoch mit einem Modell zu erfassen und mit einer höchst-möglichen Accuracy zu klassifizieren.

In dem bisher beschriebenen Beispiel wurde lediglich auf Verformung von Schwarz-Weiß-Bildern von vergleichsweise einfachen Bilddateien konzentriert. Jedoch kann dieses Grundprinzip problemlos in komplexere Farbbilder übertragen werden.

Mögliche Unterschiede bei Farbbildern:

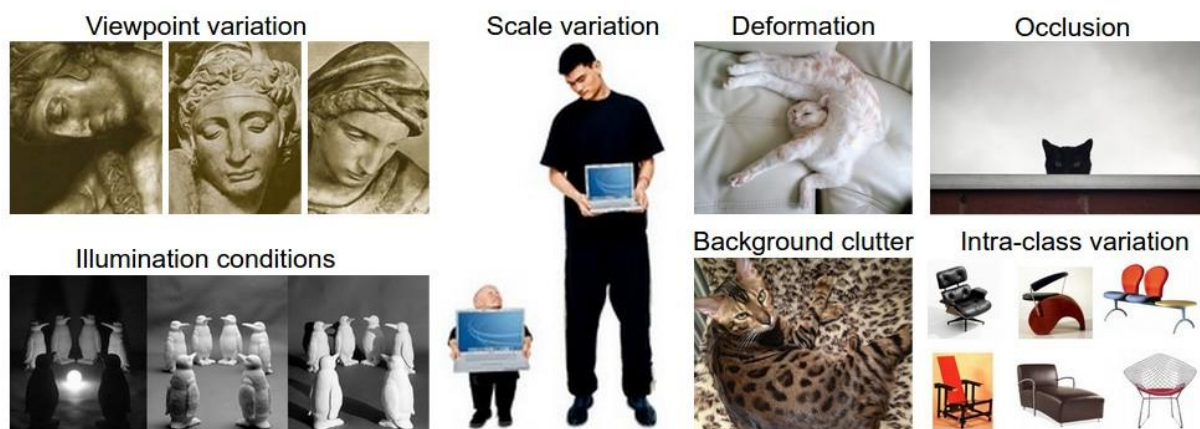


Abbildung 9, Beispiel Verformung von Farbbildern, <http://cs231n.github.io/classification/>

### Convolutional Layer

Im ersten Schritt der Bearbeitung innerhalb eines CNN wird das Bild im Convolutional Layer gefaltet (engl. to convolute = falten). Konkret bedeutet dies, dass das Input-Image zunächst nach bestimmten Kriterien (ausgedrückt durch Filter / Kernel / Faltungsmatrizen) durchsucht und die Ergebnisse in Form von Bildpatches in einem Stack abgelegt werden. Die Anzahl der Bildpatches ist somit gleich der Anzahl der gewählten Filter. Jedes entstehende Imagepatch

innerhalb des Stapels spiegelt das Vorkommen und die Lokalisation des jeweiligen Filters im ursprünglichen Input-Bild.

Um diese Faltung zu vollziehen wird nach dem folgenden Schema vorgegangen:

- 1) Erstellen von Filtern aus vertikalen und horizontalen Strukturen. Diese Filter werden vom CNN automatisch erstellt. Jeder einzelne Filter ist für sich gesehen statisch und ändert somit seine Werte nicht.
- 2) Die Filter werden sequenziell auf das ursprüngliche Input-Bild angewendet.
  - a. Jeder Filter gleitet über das Input-Image mit einer definierten Schrittweite (Stride) hinweg, diesen Prozess nennt man Padding.
  - b. Bei jedem „Zwischen-Stopp“ während des Paddings wird der Filter mit dem betrachteten Ausschnitt des Input-Bildes verglichen. Konkret werden die Werte des Filters mit den Werten des Bildausschnittes multipliziert und daraus ein Durchschnitt berechnet.
- 3) Durch jeden Filter innerhalb des Padding entsteht somit eine komprimierte Darstellung des Input-Bildes unter dem „Blickwinkel“ des entsprechenden Filters. Dabei wird nicht mehr jeder einzelne Pixel des Input-Images, sondern je nach Filtergröße eine Gruppe von Pixeln innerhalb des Input-Bildes im Bildpatch repräsentiert.
- 4) Durch das Convolutional Layer entsteht aus dem Input-Bild ein Stapel von Bildern geprägt durch den gewählten Filter.
- 7) Ziel des Convolutional-Layers ist es die High-Level-Feature (ausgedrückt durch ortsunabhängige Ecken, Kanten, Linien und Farbtupfer (in Englisch: Feature)) innerhalb des Input-Bildes mittels des Filters in den entstehenden Bildpatches unabhängig der Lokalisation im Originalbild zu detektieren. Im beschriebenen Beispiel findet zugleich eine Dimensionsreduktion zwischen dem Input-Image und dem Bildpatch im Convolutional-Stack statt. Diese Reduktion hängt von der Dimension und der Art der Implementierung des verwendeten Filters ab.

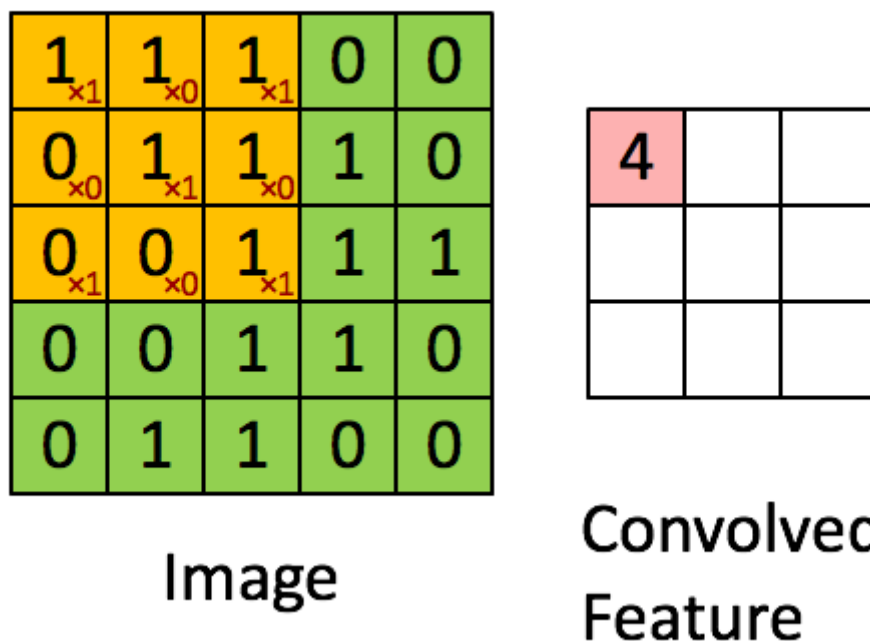


Abbildung 10, Veranschaulichung Convolutional Layer am Beispiel eines Filters, <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53> 1) (sofern diese nicht animiert ausgegeben wird, kann sie

unter [„https://github.com/JHC90/SoftwareDev-Huber/blob/master/Sources/Concoluted-Layer.gif“](https://github.com/JHC90/SoftwareDev-Huber/blob/master/Sources/Concoluted-Layer.gif) eingesehen werden)

### Rectified-Linear Unit

Zwischen den einzelnen Layern wird häufig eine erste Optimierung in Form von ReLUs verwendet. Ziel der ReLU-Funktion ist es alle negativen Werte innerhalb der Bildpatches des vorgelagerten Stacks zu detektieren und diese dem Wert 0 zuzuweisen. Durch dieses Vorgehen wird die „Nicht-Linearität“ innerhalb der Bilder erhöht. Input-Bilder sind grundsätzlich nicht linear. Durch diese Funktion wird diese Nicht-Linearität jedoch zusätzlich verstärkt. Mit diesem Ausbau wiederum werden die zu extrahierenden Merkmale technisch deutlicherer zu detektieren.

Die ReLU-Funktion kann sowohl nach dem Convolutional als auch nach dem Pooling Layer angesetzt werden.

### Pooling Layer

Je größer die Dimension eines Bildes (unabhängig ob Input oder Convoluted Image) desto intensiver ist die benötigte Rechenpower für eine performante Bearbeitung. Das Pooling Layer verfolgt das Ziel jedes Bild innerhalb des Convolutional-Image-Stack (Output aus dem Convolutional-Layer) in den Dimensionen zu reduzieren, die Ergebnisse somit zu aggregieren. Folglich ist der Output des Pooling Layers die Dimensionsreduktion der Images im Convolutional-Stack, jedoch keine Änderung der Anzahl der Images in dem Stack.

Für diese Dimensionsreduktion wird nach dem folgenden Schema vorgegangen:

- 1) Auswahl der Parameter (Fenstergröße, Schrittweite)
- 2) Für jedes Bild innerhalb des Convolutional Stacks
  - a. Sliding
    - i. Innerhalb jedes Zwischenstopps während des Slidings Auswahl des Maximal-Wertes innerhalb des betrachteten Fensters
  - b. Abspeichern der entstanden Dimensionsreduktion im Pooling-Stack.

Der Pooling Layer kann durch folgende animierte Grafik veranschaulicht werden:

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

Abbildung 11, Veranschaulichung Pooling Layer am Beispiel eines Filters, <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53> 1) (sofern diese nicht animiert ausgegeben wird, kann dieser unter [„https://github.com/JHC90/SoftwareDev-Huber/blob/master/Sources/Pooling-Layer.gif“](https://github.com/JHC90/SoftwareDev-Huber/blob/master/Sources/Pooling-Layer.gif) eingesehen werden)



### Zwischeneinschub

Die bisher beschriebenen Stationen aus Convolutional-Layer, ReLU und Pooling Layer innerhalb eines CNNs können beliebig oft wiederholt werden. Hierbei entsteht auch das Grundkonzept des Deep-Learning und die möglichen Architekturen.

### Fully Connected Layer

Der Fully Connected Layer greift die Bildpatches des zuletzt vorgelagerten Pooling-Stacks auf und flacht den Stack zu einem Vektor hin ab. Diesen Prozessen nennt man auch „ausrollen“ oder „flatten“. Somit sind *„Die Output-Signale der Filter-Schichten [] unabhängig von der Position eines Objektes, daher sind zwar keine Positionsmerkmale mehr vorhanden, dafür aber ortsunabhängige Objektinformationen“* (Becker, Convolutional Neural Networks - Aufbau, Funktion und Anwendungsgebiete). Mit diesen Eigenschaften sind die Probleme des MLP aufgehoben.

Ab dem Zeitpunkt des Vektor-Ausrollens werden die Technologien des bereits beschriebenen Neuronalen Netzes verwendet, um im vorliegenden Falle eine Klassifikation zu erstellen. Somit bildet der abgeflachte Vektor den Input eines Neuronalen Netzes, welches in unserem Falle zuletzt durch die Output-Neuronen klassifiziert wird. In praktischen Beispielen von MNIST und CIFAR10 sind die möglichen Output-Klassen bekannt und entsprechen in beiden Fällen jeweils einem Wert von 10. Die Anzahl der Output-Neuronen wird infolgedessen 10 sein.

### Softmax Im Fully-Connected Layer

Die Softmax-Funktion ist Teil der Softmax-Regression (oder Multinomial Logistic Regression) die wiederum eine Verallgemeinerung der logistischen Regression ist. Sie wird häufig bei MultiClass-Klassifizierungsproblemen verwendet. Im Kontext der verwendeten Beschreibung bedeutet dies, dass der Output der letzten Neuronen im beschriebenen CNN kumuliert 1 beträgt. *„Die Softmax-Funktion kann auch als normierte Exponentialfunktion bezeichnet werden. In Klassifikationsaufgaben kommt diese Funktion häufig für die Einheiten der Ausgabeschicht zum Einsatz, weil die Ausgabe dann als die jeweilige Klassenwahrscheinlichkeit  $p_i$  der  $K$  Klassen interpretiert werden kann“* (Selle, Künstliche Neuronale Netze, S.29). In Bildklassifikationsaufgaben wird diese Cost-Function gerne auf Höhe der Outputs verwendet.

Die Softmax-Funktion ist Teil der Softmax-Regression (oder Multinomial Logistic Regression) die wiederum eine Verallgemeinerung der logistischen Regression ist. Sie wird häufig bei MultiClass-Klassifizierungsproblemen verwendet. Im Kontext der verwendeten Beschreibung bedeutet dies, dass wenn die Anzahl ( $N$ ) der möglichen Output-Klassen bekannt ist, der Outputvektor der Softmax-Funktion  $N$ -Dimensionen hat. Kumuliert man die Werte der Output-Vektoren auf, so erhält man den Wert 1.



## Python Implementierung Hyperparameter CNN

In diesem Cheat-Sheet werden lediglich die CNN-Spezifischen Parameter für die Implementierung in Keras beschrieben

<u>Parameter</u>	<u>Beschreibung</u>	<u>BSP Parameter in Tensorflow</u>	<u>Mögliche Parameter</u>
Erstellen der Convolutional Layer	Hier wird dem CNN ein Conv2d vorweggestellt. Es wird ein Filter (kernel) der Größe (3 * 3) erstellt und die Aktivierungsfunktion „relu = Rectified linear Unit“ gewählt.	<code>modelcnn.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_shape</code>	
Data Augmentation	Dies ist eine Sammlung an Funktionen um die Input-Data jeweils während des Trainings zu „verzerrern“. Ziel ist es hierbei dem Overfitting des Algorithmus entgegen zu wirken.	<code>datagen = ImageDataGenerator( featurewise_center=False, # set input mean to 0 over the dataset samplewise_center=False, # set each sample mean to 0 featurewise_std_normalization=False, # divide inputs by std of the dataset samplewise_std_normalization=False, # divide each input by its std zca_whitening=False, # apply ZCA whitening zca_epsilon=1e-06, # epsilon for ZCA whitening rotation_range=0, # randomly rotate images in the range (degrees, 0 to 180) # randomly shift images horizontally (fraction of total width) width_shift_range=0.1, # randomly shift images vertically (fraction of total height) height_shift_range=0.1, shear_range=0., # set range for random shear zoom_range=0., # set range for random zoom</code>	

		channel_shift_range=0., # set range for random channel shifts # set mode for filling points outside the input boundaries fill_mode='nearest', cval=0., # value used for fill_mode = "constant" horizontal_flip=True, # randomly flip images vertical_flip=False, # randomly flip images # set rescaling factor (applied before any other transformation) rescale=None, # set function that will be applied on each input preprocessing_function=None, # image data format, either "channels_first" or "channels_last" data_format=None, # fraction of images reserved for validation (strictly between 0 and 1) validation_split=0.0)	
--	--	--	--

## Auswertung Neuronaler Netze

Nach dem Aufbau, Training und Predicting von Neuronalen Netzen müssen diese im nächsten Schritt ausgewertet werden. Hierzu gibt es folgende Optionen:

### Accuracy

Es wird versucht, durch die Implementierungen von Neuronalen Netzen für Klassifikationsfragestellungen eine möglichst hohe Accuracy zu erreichen. Somit können nach dem Training eines Neuronalen Netzes diese zunächst gezeichnet werden:

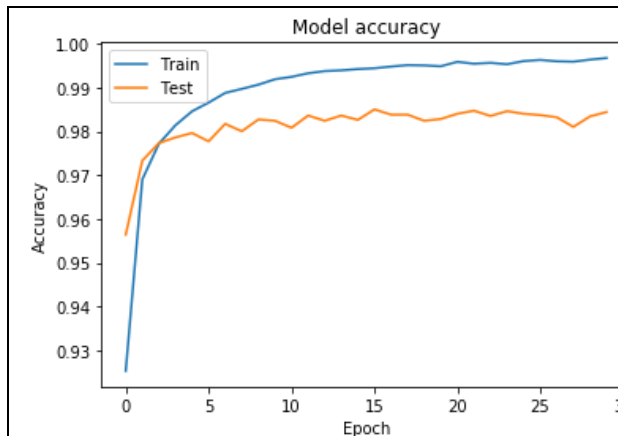


Abbildung 12; Eigene Grafik

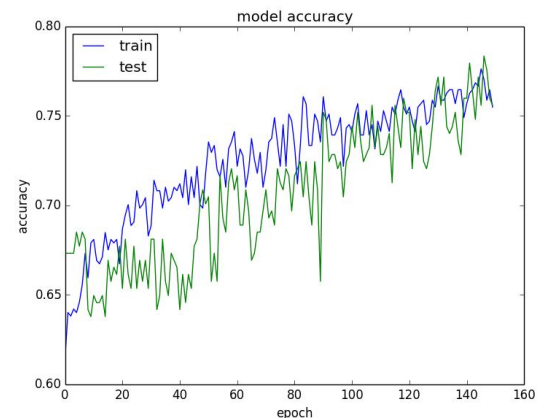


Abbildung 13, Referenzgrafik Accuracy, <https://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/>

Werden die beiden dargestellten Grafiken 12 und 13 gegenüber gestellt, so ist in der linken Darstellung erkenntlich, dass sich sowohl bezogen auf die Train- und Testdaten ab einer gewissen Epoche kaum „spürbare“ Verbesserungen in dem Modell geschehen. Somit ist keine höhere Accuracy durch eine Erhöhung der Epochen zu erwarten.

Eine Analyse der rechten Referenzgrafik im Direktvergleich zeigt hingegen „den anhaltenden Trend“, einer Erhöhung der Epochenanzahl sowie folgend eine höhere Accuracy. In diesem Beispiel würde man für eine Erhöhung der Accuracy das Modell mit einer höheren Epochen Anzahl weitertrainieren.

Die Accuracy zeigt somit auf wie akkurat die Vorhersage des Modelles im Vergleich zu der Realität ist.

### Loss

Innerhalb Neuronalen Netze wird versucht, durch die die Loss Function zu optimieren(minimieren). Ziel ist es mit jedem Update der Gewichte innerhalb der Epochen die Loss-Function so niedrig wie möglich zu gestalten. Dazu werden gezielt innerhalb der Backpropagation die entsprechend „fehlerhaften“ Neuronen adaptiert.



Werden die beiden Grafiken 14 und 15 verglichen so erkennt man in der linken Darstellung eine durchgehende Verringerung des Loss-Values gemessen an den Trainingsdaten. Das ist auch zu erwarten, denn je höher die Epoche des Trainings gewählt, desto eher passt sich der Algorithmus an die Trainingsdaten an (Overfitting). Werden die Test-Daten der linken Grafik berechnet, so ist zunächst eine Verringerung (bis ca. Epoche 4) und anschließend eine Steigung des Loss-Values zu erkennen. Somit birgt eine Erhöhung der Epochen hier wenig Verbesserungspotential.

Betrachtet man nun die Loss-Function der rechten Referenz-Grafik Abbildung 15, so ist sowohl in den Trainings- und Testdaten zu erkennen, dass mit steigender Epochenanzahl die jeweiligen Loss-Werte sinken. Somit ist der Ausbau der Epochen im Training durchaus sinnvoll.

### Praktische Implementierung in Python

Um die hier behandelten, theoretischen Konzepte zu Neuronalen Netzen und zu CNNs zu untermauern wurden die diskutierten Technologien in folgenden Case-Studies verwendet. Diese Ausarbeitungen wurden in Jupyter Notebook ausgeführt, sodass

- die Darstellung im Online-Git-Repo gewährleistet ist,
- die einzelnen Teilaspekte durch Kommentare versehen werden konnten.

### Datensätze

Die bisher theoretische Abhandlung befasst sich überwiegend mit der Fragestellung der Klassifikation. Für diese Aufgabenstellung gibt es die bekannten Image-Datensätze MNIST und Cifar10. Diese wurden als Datengrundlage für die jeweiligen Ausarbeitungen gewählt.

### MNIST

MNIST (Modified National Institute of Standards and Technology Database) ist der Name für eine Datenbank handgeschriebener einzelner Ziffern. Insgesamt sind 70.000 gelabelte Einzelbilder in der Auflösung 28 x 28 zu 10 unterschiedlichen Klassen vorzufinden. MNIST ist aus dem Bereich der Image-Klassifizierung ein vergleichsweise einfacher Datensatz und wird häufig als „Hello-World“-Projekt im Bereich des Machine-Learnings und Deep-Learnings verwendet.

Ziel ist es einen Klassifikator zu finden, welcher die handgeschriebenen Ziffern korrekt den numerischen Ziffern zuordnet.

## CIFAR10

CIFAR10 ist ebenfalls ein Datensatz, welcher vorwiegend im Machine-Learning und Deep-Learning für Schulungszwecke verwendet wird. Dieser Datensatz wiederum besteht aus 60.000 gelabelten 32 x 32 Farbbildern zu 10 unterschiedlichen Klassen.

Ziel ist es erneut einen Klassifikator zu finden, welcher die Bilder korrekt den Klassen zuordnet.

## Gewählte Implementierung

Für die Darstellung der Konzepte aus dem Bereich Neuronaler Netze wurden folgende Ausarbeitungen angefertigt:

- 1) Implementierung eines Multi-Layer-Perzeptron Netzwerkes für die Klassifikation der MNIST Data: [Link](#)
- 2) Implementierung eines Multi-Layer-Perzeptron Netzwerkes für die Klassifikation der CIFAR10 Data: [Link](#).

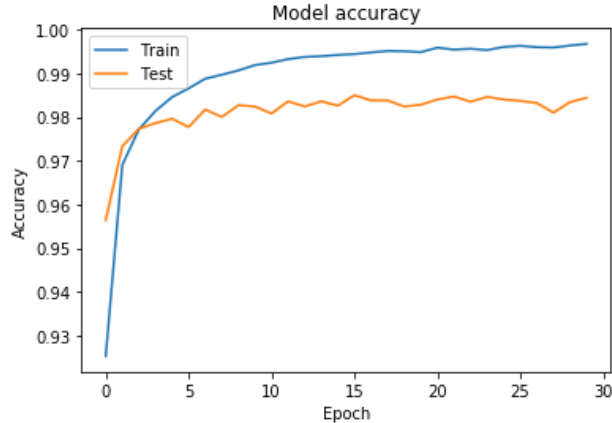
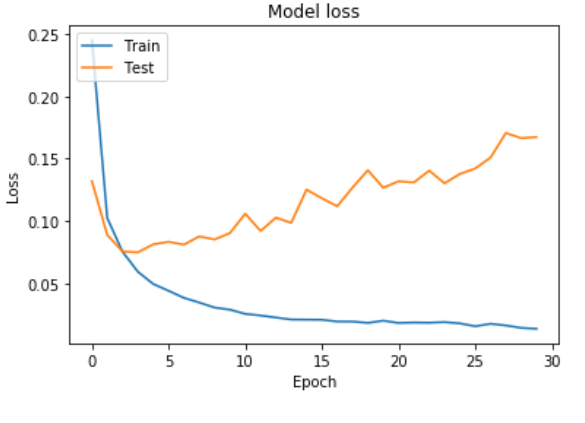
Für die Darstellung der Konzepte aus dem Bereich des Convolutional Neuronal Networks wurden folgende Ausarbeitungen angefertigt:

- 1) Implementierung eines CNNs für die Klassifikation der MNIST Data: [Link](#)
- 2) Implementierung eines CNNs für die Klassifikation der CIFAR10 Data: [Link](#).

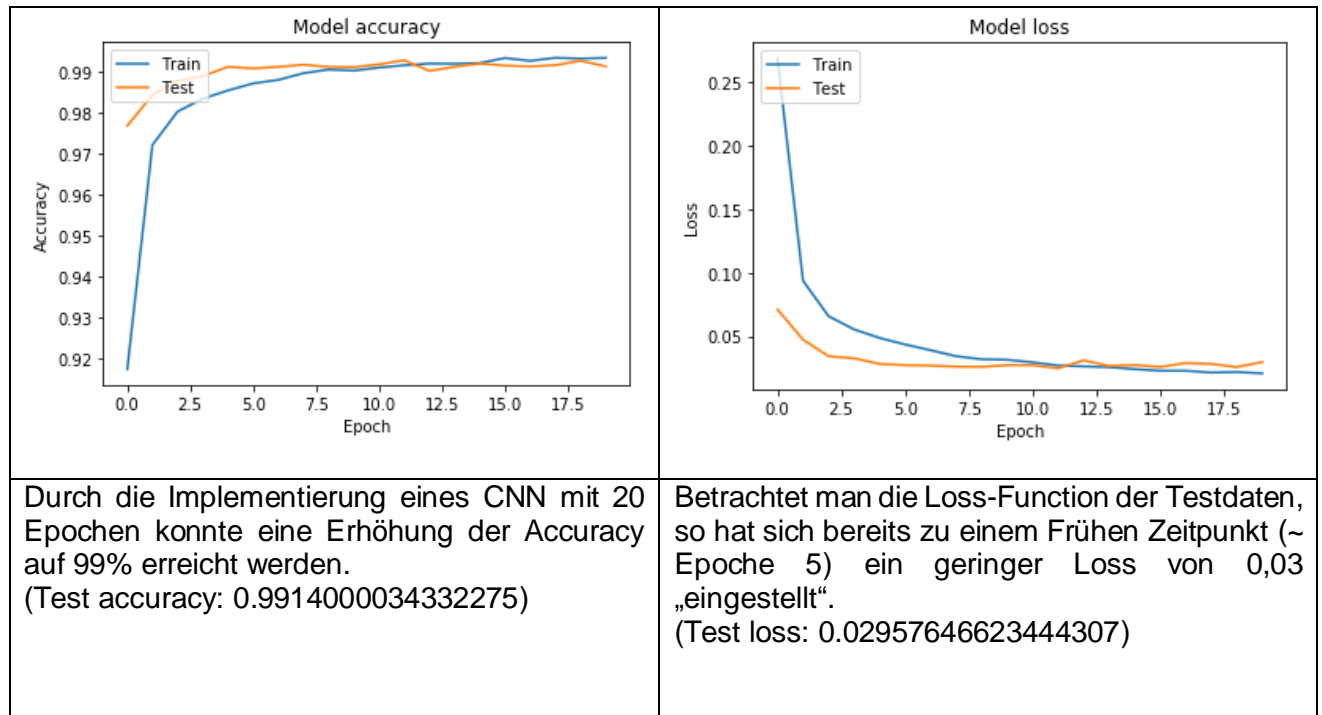
## Ergebnisse

### MNIST

#### MLP

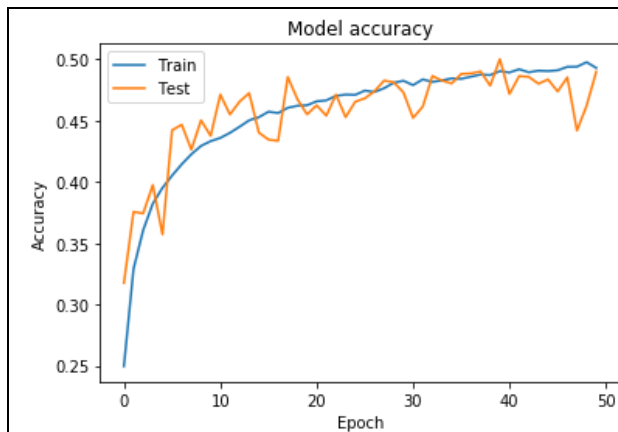
 <p>Model accuracy</p> <p>Y-axis: Accuracy (0.93 to 1.00)</p> <p>X-axis: Epoch (0 to 30)</p> <p>Legend: Train (blue), Test (orange)</p>	 <p>Model loss</p> <p>Y-axis: Loss (0.05 to 0.25)</p> <p>X-axis: Epoch (0 to 30)</p> <p>Legend: Train (blue), Test (orange)</p>
<p>Die Genauigkeit gemessen an den Testdaten beträgt ~98%, somit besteht eine Fehlerrate von 2%.</p> <p>(Test Accuracy: 0.9836999773979187)</p>	<p>Betrachtet man die Loss-Function an den Testdaten, so ist das Minimum in etwa bei Epoche 3 oder 4 somit führt eine Erhöhung der Epochen nicht notwendig zu einem besseren Ergebnis. Die Daten overfitten in diesem Falle an den Trainingsdaten</p> <p>(Test Loss: 0.2103530801721231)</p>

## CNN



Folglich ist gemessen an der Accuracy und der Loss Function für die MNIST- Aufgabenstellung die CNN Implementierung zu wählen.

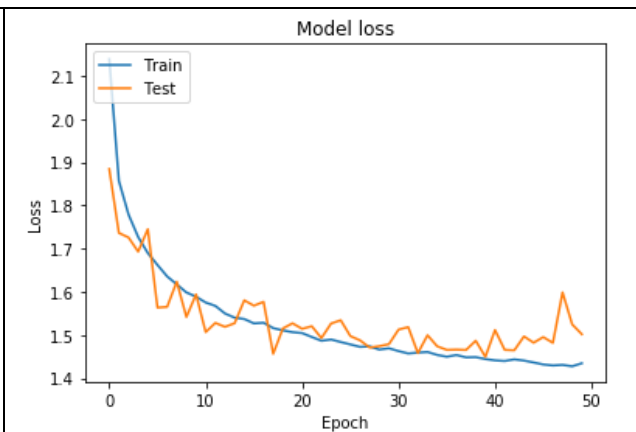
## CIFAR10 MLP



Verwendet man den MLP im Kontext der CIFAR10-Daten so erhält man eine geringe Accuracy von ~50%.

Mögliche Maßnahmen um die Performance des Algorithmus zu erhöhen wären mithilfe des Image-Preprocessings oder des Finetunings des Netzes möglich.

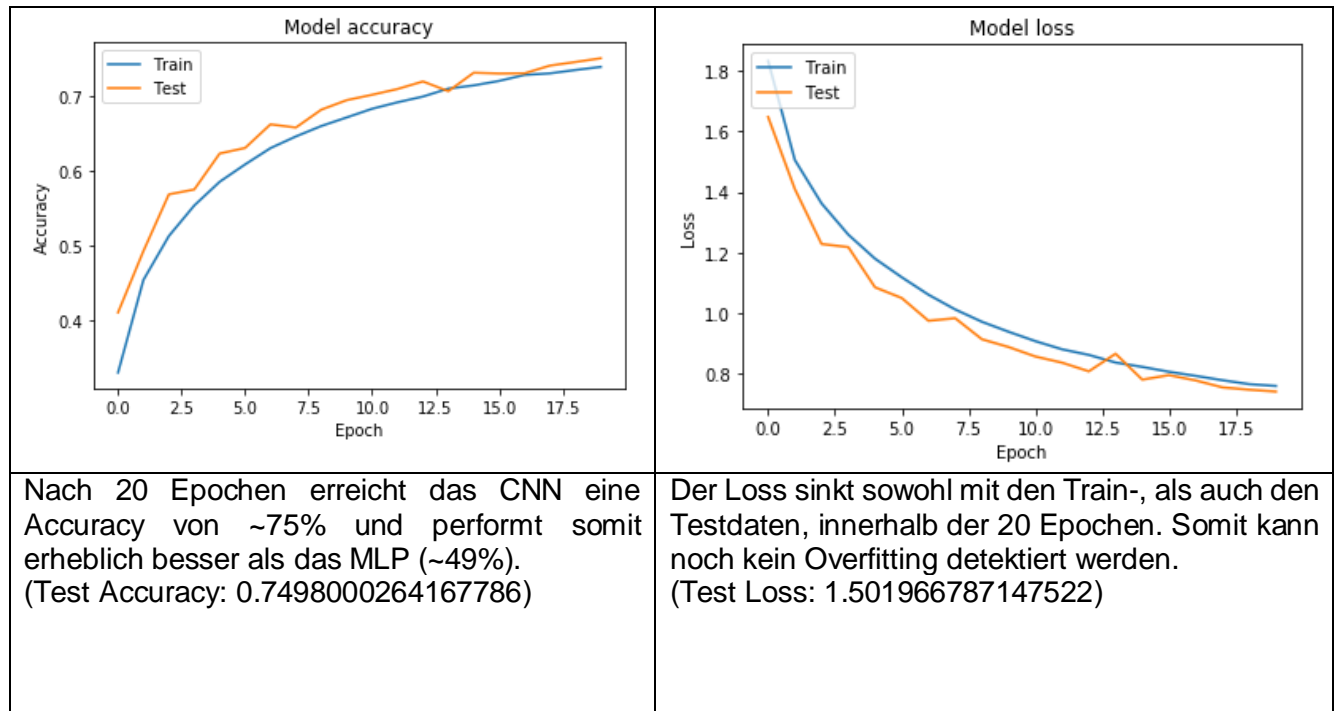
Eine Erklärung für die verhältnismäßig „viel schlechtere“ Performance des MLP im Vergleich zu der Anwendung im MNIST-Dataset, liegt im Algorithmus selbst. MLP performt vergleichsweise weniger ausgereift wie CNN, sobald die Input Bilder mehr skalieren, rotieren oder unterschiedlich gewichtet sind. Diese Änderungen innerhalb der Klassen treten im CIFAR10 Datensatz erheblich häufiger auf als im MNIST-Datensatz (Test Accuracy: 0.4900999963283539)



Der Test-Loss-Wert sinkt zügig innerhalb der ersten 15 Epochen von etwa 2% auf 1,5% und pendelt mit der steigenden Epochen Zahl um diesen Wert. Ab Epoche ~45 ist ein Anstieg zu erkennen. Eine mögliche Erklärung ist ein Einsetzen des Overfittings ab diesem Punkt. (Test Loss: 1.501966787147522)



## CNN



Auch im CIFAR10-Datensatz erweist sich CNN als höher performant als das Multi-Layer-Perzeptron.

## Ausblick

„Digitalisierung bedeutet im engeren Sinn die Transformation von analogen in digitale Daten.“ (Selle, Künstliche Neuronale Netze, S.102). Somit sind die Technologien der Computer Vision eine Verkörperung der Digitalisierung.

Für mich persönlich ist es interessant zu sehen, dass durch die Digitalisierung in der Industrie viele Unternehmen ihre „Tore“ zu den verwendeten Technologien öffnen, um die Entwicklung dieser voranzutreiben. Ein tagesaktuelles Beispiel ist die Veröffentlichung der [Bilderkennungssoftware innerhalb der Produktion](#) des deutschen Automobilherstellers BMW. BMW ist mit dieser öffentlichen Politik kein Einzelfall, auch Versicherungen verwenden die automatisierte Bildererkennung. Somit haben viele unterschiedliche Teilnehmer ein Interesse an der Weiterentwicklung dieser Technologie, welche zunehmend in unseren privaten und beruflichen Alltag einziehen wird.

## Abbildungsverzeichnis

Abbildung 1: Historische Entwicklung AI, ML, DL ( <a href="https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/">https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/</a> ) .....	4
Abbildung 2, Neuron-Aufbau ( <a href="https://medicalxpress.com/news/2018-07-neuron-axons-spindly-theyre-optimizing.html">https://medicalxpress.com/news/2018-07-neuron-axons-spindly-theyre-optimizing.html</a> ) .....	6
Abbildung 3, Veranschaulichung Perceptron, Vorlesungsunterlagen Machine-Learning/ Teil 4 Deep-Learning – FH Kufstein SS19 .....	7
Abbildung 4, Darstellung gängiger Aktivierungsfunktionen, <a href="https://www.cbcity.de/tutorial-neuronale-netze-einfach-erklart">https://www.cbcity.de/tutorial-neuronale-netze-einfach-erklart</a> .....	8
Abbildung 5, Mögliche Optimierungen der Fehlerfunktion, <a href="https://www.cbcity.de/tutorial-neuronale-netze-einfach-erklart">https://www.cbcity.de/tutorial-neuronale-netze-einfach-erklart</a> .....	11
Abbildung 6, Veranschaulichung NN Architekturen, <a href="https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464">https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464</a> .....	13
Abbildung 7, Aufbau CNN, <a href="https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53">https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53</a> .....	18
Abbildung 8, Image-Unterschiede Schwarz / Weiß einfacher Objekte, <a href="https://slideplayer.com/slide/14855329/">https://slideplayer.com/slide/14855329/</a> .....	19
Abbildung 9, Beispiel Verformung von Farbbildern, <a href="http://cs231n.github.io/classification/">http://cs231n.github.io/classification/</a> .....	19
Abbildung 10, Veranschaulichung Convolutional Layer am Beispiel eines Filters, <a href="https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53">https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53</a> 1) (sofern diese nicht animiert ausgegeben wird, kann sie unter „ <a href="https://github.com/JHC90/SoftwareDev-Huber/blob/master/Sources/Concoluted-Layer.gif">https://github.com/JHC90/SoftwareDev-Huber/blob/master/Sources/Concoluted-Layer.gif</a> “ eingesehen werden) .....	20
Abbildung 11, Veranschaulichung Pooling Layer am Beispiel eines Filters, <a href="https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53">https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53</a> 1) (sofern diese nicht animiert ausgegeben wird, kann dieser unter „ <a href="https://github.com/JHC90/SoftwareDev-Huber/blob/master/Sources/Pooling-Layer.gif">https://github.com/JHC90/SoftwareDev-Huber/blob/master/Sources/Pooling-Layer.gif</a> “ eingesehen werden) .....	21
Abbildung 12; Eigene Grafik .....	25
Abbildung 13, Referenzgrafik Accuracy, <a href="https://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/">https://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/</a> .....	25
Abbildung 14, Eigene Grafik .....	26
Abbildung 15, Referenzgrafik Accuracy, <a href="https://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/">https://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/</a> .....	26

## Literaturverzeichnis

- Balzer, P. (19. 03 2016). Neuronale Netze einfach erklärt. Von [cbcity.de/tutorial-neuronale-netze-einfach-erklart](https://www.cbcity.de/tutorial-neuronale-netze-einfach-erklart) abgerufen
- Bartling, K.-H. (2017). Das digitale Zeitalter. Von <https://www.teamnext.de/academy/artikel/das-digitale-zeitalter/> abgerufen
- Becker, R. (06. 02 2019). Convolutioal Neuronal Networks. Abgerufen am 19. 12 2019 von <https://jaai.de/kuenstliche-neuronale-netze-aufbau-funktion-291/>
- BECKER, R. (06. 02 2019). Convolutional Neural Networks - Aufbau, Funktion und Anwendungsgebiete. Von <https://jaai.de/convolutional-neural-networks-cnn-aufbau-funktion-und-anwendungsgebiete-1691/> abgerufen

Learning, K. N. (2018). Künstliche Neuronale Netzwerke und Deep Learning.  
*Wirtschaftswissenschaften HTW SAAR.*

MAX-Planck-Institut. (kein Datum). SYNAPTISCHE PLASTIZITÄT - WIE DAS GEHIRN LERNT. Von  
<https://www.max-wissen.de/132855/Synapsen> abgerufen

MOESER, J. (27. 09 2018). KÜNSTLICHE NEURONALE NETZE – AUFBAU & FUNKTIONSWEISE. Von  
<https://jaai.de/kuenstliche-neuronale-netze-aufbau-funktion-291/> abgerufen

Selle, S. (12. 05 2018). Künstliche Neuronale Netzwerke und Deep Learning. Von  
[https://www.htwsaar.de//wiwi/fakultaet/personen/profile/selle-stefan/Selle2018e\\_Kuenstliche\\_Neuronale\\_Netzwerke.pdf](https://www.htwsaar.de//wiwi/fakultaet/personen/profile/selle-stefan/Selle2018e_Kuenstliche_Neuronale_Netzwerke.pdf) abgerufen