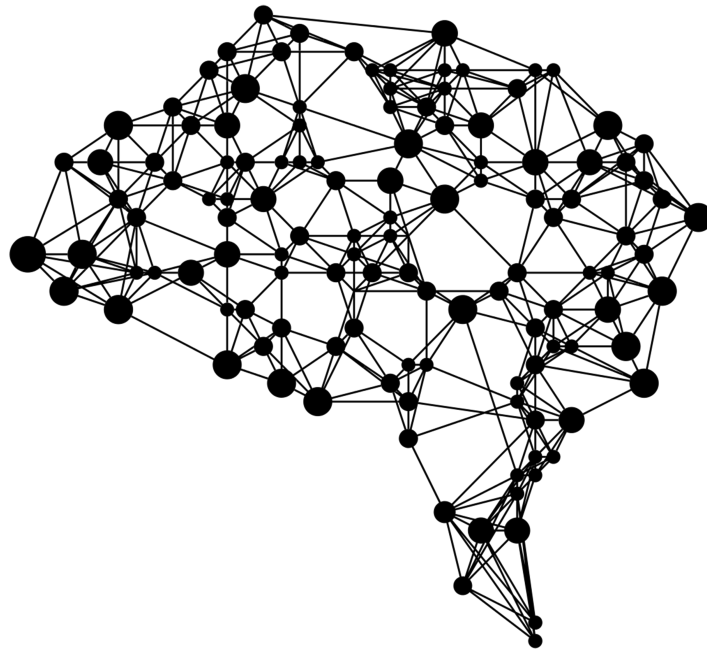


# Machine Learning in Smartphone Apps

ASGSG Informatik, 2020/2021

Phillip Bronzel 7. Januar 2021



cleanpng 2020

Hiermit versichere ich, dass ich die Arbeit selbstständig verfasst, dass ich keine anderen Quellen und Hilfsmittel als die angegebenen benutzt und die Stellen der Arbeit, die anderen Quellen dem Wortlaut oder Sinn nach entnommen sind, in jedem einzelnen Fall unter Angabe von Quellen kenntlich gemacht habe.

# Inhaltsverzeichnis

<b>1</b>	<b>Einführung</b>	<b>2</b>
1.1	Wahl des Themas . . . . .	2
1.2	Ziel der Arbeit . . . . .	2
<b>2</b>	<b>Neuronale Netzwerke</b>	<b>3</b>
2.1	Wichtigste Ereignisse in der Geschichte . . . . .	3
2.2	Aufbau . . . . .	4
2.2.1	Erstellung eines Neuronalen Netzwerks anhand eines Beispiels	5
2.3	Funktionsweise . . . . .	5
2.3.1	Trainieren des Neuronalen Netzwerks . . . . .	7
	Die Cost Function . . . . .	7
	Gradientenabstiegsverfahren . . . . .	8
	Ketten Regel . . . . .	9
	Backpropagation . . . . .	9
2.4	Abwandlungen . . . . .	9
2.4.1	Convolutional Neural Networks . . . . .	10
	Pooling . . . . .	11
<b>3</b>	<b>Labelcheck als Smartphone App</b>	<b>11</b>
3.1	Die Idee . . . . .	11
3.2	Die Label/Siegel . . . . .	12
3.3	Erstellen des Modells . . . . .	12
3.3.1	Trainieren des Modells mit TensorFlow und Python . . . . .	12
3.4	Entwickeln der App . . . . .	12
3.4.1	Das Framework: Flutter . . . . .	12
3.4.2	Importieren des Modells . . . . .	13
3.4.3	Veröffentlichen der App . . . . .	13
3.5	Testen der App . . . . .	14
3.6	Fazit . . . . .	14
<b>A</b>	<b>Anhang</b>	<b>15</b>
A.1	Weitere Aktivierungsfunktionen . . . . .	15
A.2	Code für das Beispiel aus 2.3 . . . . .	16
A.3	Labelcheck Code . . . . .	16
A.4	Die Flutter Architektur . . . . .	17
	<b>Literaturverzeichnis</b>	<b>18</b>

# 1 Einführung

Im Enterprise und Forschungsbereich spielt Machine Learning schon seit vielen Jahren eine bedeutende Rolle. Doch wie kann es dem Endnutzer, zum Beispiel in mobilen Apps, weiterhelfen?

## 1.1 Wahl des Themas

Seitdem ich im Jahr 2017 meinen ersten richtigen Kontakt mit der Programmierung von Microcontrollern (Arduinos) hatte, habe ich mich stark für die Entwicklung von Software interessiert. Dies war ein guter Einstieg, da man dort schnell und recht einfach Ergebnisse, wie zum Beispiel eine blinkende LED, erzielt.

Auch habe ich mich seitdem immer für die „neuen großen Technologien“ wie Blockchain oder Machine Learning interessiert. Zum Thema Machine Learning habe ich zuvor noch nicht viel gemacht, daher ergriff ich die Chance dieses Jahr meine Facharbeit über dieses Thema zu schreiben.

AI is profound, and we are at a point—and it will get better and better over time—where the GPU is getting so powerful there’s so much capability to do unbelievable things. What all of us have to do is to make sure we are using AI in a way that is for the benefit of humanity, not to the detriment of humanity.<sup>1</sup>

Ich persönlich finde dieses Zitat sehr wichtig; es ist jetzt über 3 Jahre alt und bis heute hat sich enorm viel in diesem Bereich getan. Wir haben nun GPU’s, welche speziell auf mathematische Berechnungen mit Tensoren optimiert sind und so das Trainieren von Neuronalen Netzen um ein Vielfaches beschleunigen.<sup>2</sup>

Des weiteren ist es mir, genauso wie Cook, wichtig, diese mächtige Technologie nicht zu missbrauchen<sup>3</sup>, sondern gute Dinge mit ihr zu schaffen: wie beispielsweise im Bereich der Medizin. In diesem Bereich wurden schon viele beachtliche Anwendungszwecke gefunden, so hat Google’s Tochterfirma DeepMind im Dezember 2020 eine Technologie<sup>4</sup> präsentiert, welche das Falten von Proteinen akkurat prognostizieren kann; dies war vorher nur sehr langsam und deutlich ungenauer möglich.<sup>5</sup>

## 1.2 Ziel der Arbeit

Mein persönliches Ziel ist es, mehr über den Aufbau von Neuronalen Netzen und die Funktionsweise von Machine Learning zu lernen. Außerdem möchte ich auch ein

---

<sup>1</sup>Byrnes 2017, Tim Cook (CEO von Apple) In einem Interview mit MIT Technology Review

<sup>2</sup>Hebert 2020, NVIDIA Grafikprozessoren mit integrierten Tensor Kernen

<sup>3</sup>Beispiel: Autonome Waffen, wie Drohnen, welche Ziele autonom erfassen können oder auch „Deep-fakes“

<sup>4</sup>Künstliche Intelligenz

<sup>5</sup>Rettner 2020

praktisches Ergebniss haben, dafür habe ich im Kapitel !TODO! eine App entwickelt, welche dem Nutzer mehr Informationen über Produkte beim einkaufen liefern soll.

## 2 Neuronale Netzwerke

### 2.1 Wichtigste Ereignisse in der Geschichte

Im Jahr 1943 wurde die erste Arbeit darüber geschrieben, wie Neuronen im Gehirn funktionieren könnten und die Autoren Warren McCulloch und Walter Pitts experimentierten sogar damit diese mit elektronischen Schaltkreisen nachzubauen.<sup>6</sup>

In den 1950er Jahren haben Forscher von IBM daran gearbeitet ein Neuronales Netzwerk mit einem Computer zu simulieren. Der Versuch scheiterte allerdings.<sup>7</sup>

Immer wieder gab es kleinere Forschungsprojekte, ein sehr großer Durchbruch war aber 1975 die Entwicklung eines „Backpropagation“ Algorithmus durch den Wissenschaftler Paul Werbos. Ähnliche Algorithmen wurden wiederholt und unabhängig entwickelt, aber Werbos’ Algorithmus war der erste mit großer Bedeutung.<sup>8</sup> Das Prinzip des Algorithmus wird auch heute noch verwendet, es ist dieser Algorithmus der dem Neuronalen Netzwerk das selbstständige Lernen ermöglicht.<sup>9</sup>

In 1998 veröffentlichte Yann LeCun und sein Team eine Arbeit über die Anwendung eines „Convolutional Neural Networks<sup>10</sup>“ zur Erkennung von geschriebenen Zeichen in einem Dokument.<sup>11</sup> Diese Arbeit gilt als Ursprung des, für beispielsweise Bilderkennungs Software gut geeignete, CNNs und Weiterentwicklungen werden auch heute noch verwendet.

Obwohl ein großes Potenzial erkannt wurde, war es über die nächsten Jahre wieder recht still. Der nächste große Durchbruch passierte in 2012 als Geoffrey Hinton ein Modell entwickelte, was die Fehlerquote in einer öffentlichen Challenge für Bilderkennung beinahe halbierte.<sup>12</sup> Der Grund dafür waren mehrere fundamentale Neuerungen aus dem Bereich Deep Learning; die wahrscheinlich größte Änderung: Starke Parallelisierung des Backpropagation-Prozesses, durch Verschiebung der Last von der CPU auf die GPU. Aufgrund der starken Überlegenheit eines Grafikprozessors in parallelisierten Prozessen, wie die benötigten Tensormultiplikationen durch die deutlich größere Anzahl an (dafür schwächeren) Kernen im Vergleich zu einer herkömmlichen CPU, kann ein Neuronales Netzwerk mehrere hundertmal schneller trainiert werden.

Heute gibt es (vergleichsweise) simple Frameworks, wie das im Jahr 2015 erscheinende TensorFlow oder PyTorch aus 2016, welche das erstellen, trainieren und ver-

---

<sup>6</sup>Warren McCulloch 1943

<sup>7</sup>Roberts unbekanntes Jahr, Absatz 3

<sup>8</sup>Werbos 1975

<sup>9</sup>Genaueres in Kapitel 2.3

<sup>10</sup>Ab jetzt als CNN bezeichnet

<sup>11</sup>Yann LeCun und Haffner 1998

<sup>12</sup>Geoffrey E. Hinton 2012

wenden von Neuronales Netzwerk enorm vereinfachen. Ihr Funktionsumfang wächst durch die große Open-Source Community ständig.

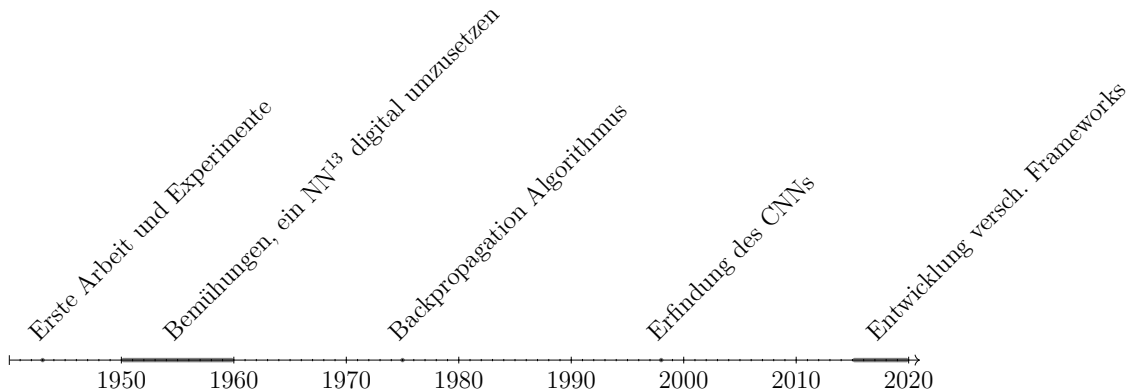


Abbildung 1: Zeitstrahl von 1940 bis 2020 mit den wichtigsten Ereignissen der Entwicklung künstlicher Neuronaler Netzwerke

## 2.2 Aufbau

In Abbildung 2 sieht man den Aufbau eines herkömmlichen künstlichen Neuronalen Netzwerks, so wie es noch vor 40 Jahren verwendet wurde. In der Grafik erkennt man drei Layer mit einer beliebigen Anzahl Neuronen, welche untereinander mit jeweils allen Neuronen der vorigen und nächsten Layer verbunden sind. Im Gegensatz zu einem biologischen Neuron, welches nur aktiv oder inaktiv sein kann, kann ein künstliches Neuron einen Zustand in Form eines Wertes von  $0 \leq x \leq 1$  haben. Jede Verbindung hat einen Weight Parameter und auch jedes Neuron hat einen Bias. Die Anzahl der Hidden Layer kann an das Ziel angepasst und ausgewählt werden und auch die Anzahl der einzelnen Neuronen ist erstmal beliebig, als Faustregel für gute Ergebnisse gilt aber:

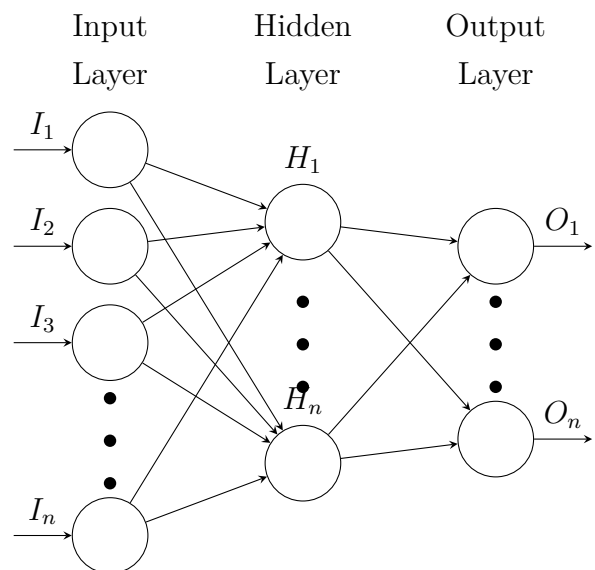


Abbildung 2: Vereinfachter Aufbau eines Neuronales Netzwerks

- Die Anzahl der Neuronen in dem Hidden Layer sollte zwischen der Größe des Input und Output Layers liegen.

- Die Anzahl der Neuronen in dem Hidden Layer sollte etwa  $\frac{2}{3}$  der Größe des Input Layers plus der Größe des Output Layers entsprechen.
- Die Anzahl der Neuronen in einem Hidden Layer sollte weniger als die Hälfte der Größe des Input Layers sein.<sup>14</sup>

### 2.2.1 Erstellung eines Neuronales Netzwerks anhand eines Beispiels

Als Beispiel für ein Neuronales Netzwerk, welches darauf ausgelegt ist, geschriebene Ziffern aus Bildern mit 24x24 Pixeln und nur Graustufen zu erkennen wäre dann: Ein Input Layer mit  $24^2$  Neuronen, jeweils für jeden Pixel, welche jeweils eine Aktivierung zwischen 0 (komplett weiß) und 1 (komplett schwarz) haben können, eines. Genau 10 Neuronen im Output Layer, für jedes Zahlzeichen eines. Schließlich muss die Anzahl der Hidden Layer und Neuronen festgelegt werden. Ich wähle als Beispiel 2 Layer mit jeweils 16 Neuronen, die Neuronen-Anzahl kann aber auch unterschiedlich sein. Auch die Weights und Biases werden zunächst zufällig ausgewählt, die Werte werden dann später im Trainingsprozess<sup>15</sup> angepasst. Die Eingabe in das Netzwerk ist also ein zweidimensionaler Tensor, oder auch eine Matrix, mit den Bilddaten und die Ausgabe des Netzwerks ist ein eindimensionaler Tensor, oder auch ein Vektor, mit den Prognosen.

$$\begin{bmatrix} i_{0,0} & \dots & i_{0,23} \\ \vdots & \ddots & \vdots \\ i_{23,0} & \dots & i_{23,23} \end{bmatrix} \Rightarrow \text{Hidden Layer} \Rightarrow \begin{bmatrix} o_0 \\ \vdots \\ o_9 \end{bmatrix} \quad (1)$$

## 2.3 Funktionsweise

Ein Neuronales Netzwerk kann man sich eigentlich als eine große verschachtelte mathematische Funktion vorstellen. In dem zuvor genannten Beispiel wäre es eine Funktion mit 576 Variablen und 10 Ergebnissen. Gibt man dieser Funktion nun ein Bild, beziehungsweise 576 Werte als Input, so werden von links nach rechts alle Weights  $w$  und Biases  $b$  zusammen mit dem vorigen Aktivierungswerten  $a$  berechnet. Da ein Neuron aber nur Werte im Bereich  $0 \leq x \leq 1$  haben kann<sup>16</sup>, so wird das Ergebniss noch mithilfe einer Aktivierungsfunktion in diesen Bereich umgewan-

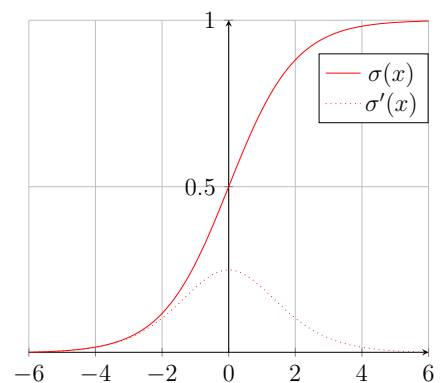


Abbildung 3: Die Sigmoidfunktion

<sup>14</sup>Heaton 2017, Alle drei Faustregeln

<sup>15</sup>siehe Kapitel 2.3.1

<sup>16</sup>Sanderson 2017

delt.<sup>17</sup> Eine früher Häufig verwendete Funktion ist dabei die Sigmoidfunktion, siehe Abbildung 3.<sup>18</sup> Es gibt aber auch noch eine Vielzahl weiterer Funktionen, wie die heute häufig verwendete ReLU Funktion<sup>19</sup>, welche den Trainingsprozess durch die einfachere Funktion beschleunigt.<sup>20</sup> Die daraus resultierende Funktion würde in etwa so aussehen:<sup>21</sup>

$$\sigma(w_1a_1 + w_2a_2 + w_3a_3 + \dots + w_na_n + b) \quad (2)$$

Um mit dieser Formel alle Aktivierungen auf einmal berechnen zu können verwendet man folgende Funktion, in welcher alle Weights und Biases in Spalten-Vektoren zusammengefasst werden. Die Hochzeichen sind keine Exponenten sondern gelten als Bezeichnung für den Layer, hier beispielsweise 0 und 1. Das Ergebniss dieser Funktion ist ein Vektor mit allen Aktivierungen des darauf folgenden Layers.

$$\sigma \left( \begin{bmatrix} w_{0,0} & w_{0,1} & \dots & w_{0,n} \\ w_{1,0} & w_{1,1} & \dots & w_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & \dots & w_{k,n} \end{bmatrix} \cdot \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_k \end{bmatrix} \right) = a^{(1)} \quad (3)$$

22

Auch diese Funktion kann wiederrum kompakter formuliert werden und diese Schreibweise wird auch für gewöhnlich verwendet:

$$a^{(1)} = \sigma(W \cdot a^{(0)} + b) \quad (4)$$

20

Theoretisch wenn ein Neuron einen hohen Aktivierungswert haben soll, wenn beispielsweise eine gerade Linie erkannt wird (um mit anderen Neuronen zusammen im späteren Verlauf aus den Mustern ganze Ziffern zu erkennen), so müssen die Weights der zu dem Neuron führenden Verbindungen alle möglichst niedrige Aktivierungen haben, ausser an den Stellen an denen die Linie sich befinden soll. Um sicherzustellen, dass es sich wirklich um eine gerade Linie handelt befindet sich direkt über dem Strich ein Bereich in dem keine Aktivierungen sein sollten, dieser ist rot markiert. Das erkennt man in Abbildung 4 sehr gut. In a erkennt man die zu erkennende Linie und in b sieht man die zugehörigen Weights der Input Nodes zu dem Neuron. Dabei stellt grün positive Weights da, rot negative und Weiß/Transparent ist 0. Der Bias des Neurons stellt eine Zusätzliche Hürde oder eine Verstärkung da, was auch in Formel 2 als  $b$  sichtbar ist.

<sup>17</sup>Google 2020 Schlüsselwort: Activation Function

<sup>18</sup>Sanderson 2017

<sup>19</sup>siehe Anhang A.1

<sup>20</sup>Harrison Kinsley 2020 Seite 76 folgende

<sup>21</sup>Harrison Kinsley 2020 Seite 185

<sup>22</sup>Gleichungen 3 und 4 von Sanderson 2017



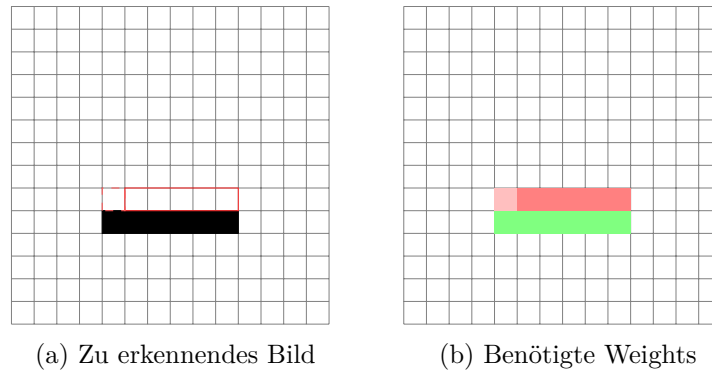


Abbildung 4: Visualisierung der gewünschten Formen a und die dazugehörigen Weights b (jeweils abgeschnitten)

### 2.3.1 Trainieren des Neuronalen Netzwerks

Dieser Prozess ist der wichtigste. Durch das Trainieren erzielt ein Neuronales Netzwerk den Effekt des selbstständigen Lernens. Und da die Werte der Weights und Biases zunächst zufällig ausgewählt wurden, muss das Netzwerk trainiert werden um nicht völligen Unsinn auszugeben.<sup>23</sup>

**Die Cost Function** Um herauszufinden wie gut oder schlecht ein Neuronales Netzwerk arbeitet, also auf das Beispiel bezogen wie genau oder ungenau es Ziffern erkennen kann, gibt es die Cost Function. Es gibt verschiedene Arten und Möglichkeiten ähnliche Funktionen anzuwenden, hier werde ich mich allerdings auf die Minimierung der Cost Function beziehen. Als Ergebniss kommt eine einzige Zahl heraus welche hoch ist, wenn das Netzwerk schlechte Ergebnisse erzielt und gegen 0 läuft, wenn das Netzwerk sehr gute Ergebnisse liefert. Es gibt mehrere verschiedene Cost Functions, aber ich fokussiere mich erstmal auf die MSE Funktion. MSE steht für „Mean squared Error“, sie berechnet den Cost Wert<sup>24</sup> aus dem durchschnitt der Summe der Vorhersagen und den erwarteten Ergebnissen zum Quadrat:

$$C = \frac{1}{m} \sum_{i=1}^m (x^{(i)} - y^{(i)})^2 \quad (5)$$

Wenn:

- $i$  = Index der Trainingsdate
- $x$  = Vorhersage des Netzwerk
- $y$  = Erwartetes (richtiges) Ergebnis
- $m$  = Anzahl der Trainingsdaten<sup>25</sup>

<sup>23</sup>Ein Code Beispiel, wie man ein solches Netzwerk mit modernen Frameworks erstellen und trainieren würde befindet sich im Anhang A.2.

<sup>24</sup>Manchmal auch Loss genannt, meint das gleiche.

<sup>25</sup>Formel 5 und Erklärung vergleiche Krzyk 2018

**Gradientenabstiegsverfahren** Leider ist es bei solch großen Funktionen nicht mehr möglich (stimmt das? !TODO!) das Globale Minimum explizit zu bestimmen.

Daher berechnet man die Steigung  $\Delta C^{26}$  der Funktion und bestimmt anschließend die Richtung  $-\Delta C$  in welche der Graph sinkt. In Abbildung 5 ist der Graph nur Zweidimensional und daher gibt es nur eine Richtung in welcher der Graph fallen kann. So wird die Eingabe immer weiter so verändert, dass sich die Cost Function minimiert. Dies passiert in mehreren Iterationen oder auch Epochen, in welchen die Veränderungen, also Schritte in Richtung  $-\Delta C$ , in Abhängigkeit von der Steigung, immer kleiner werden um einen Überschuss zu verhindern.<sup>27</sup> Das ist auch der Grund weswegen, wie in dem Ergebniss des Trainings von A.2 zu erkennen, Trainingszeiten exponentiell zur Genauigkeit ansteigen. Die Größe dieser Schritte wird auch Lernrate / Learning Rate genannt.<sup>28</sup>

Bei mehrdimensionalen Funktionen, wie auch den Neuronalen Netzwerken, gibt es mehr Möglichkeiten in welche Richtung der Graph am schnellsten Fallen könnte. Die oben beschriebene Technik die dafür verwendet wird nennt sich das Gradientenabstiegsverfahren. Da die Funktionen in echt allerdings deutlich komplizierter sind, gibt es sehr viele Extrema und da das Ziel ein möglichst tiefer Extrempunkt ist, versucht man zu verhindern, dass man in einem solcher hohen Minima „stecken bleibt“. Deswegen verwendet man meistens abgewandelte Formen des Gradientenabstiegsverfahrens, zum Beispiel mit einem Mo-

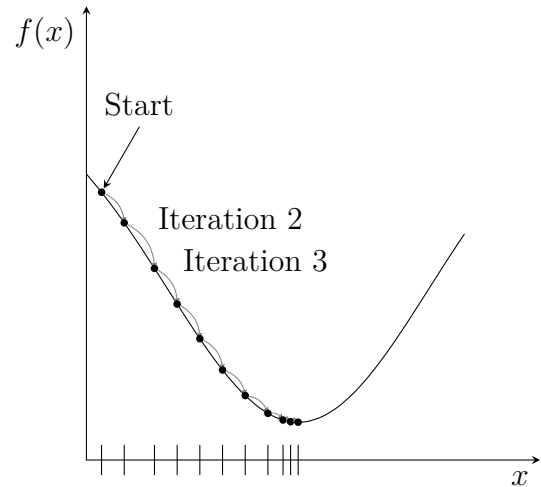


Abbildung 5: Vorgehen bei der Minimierung

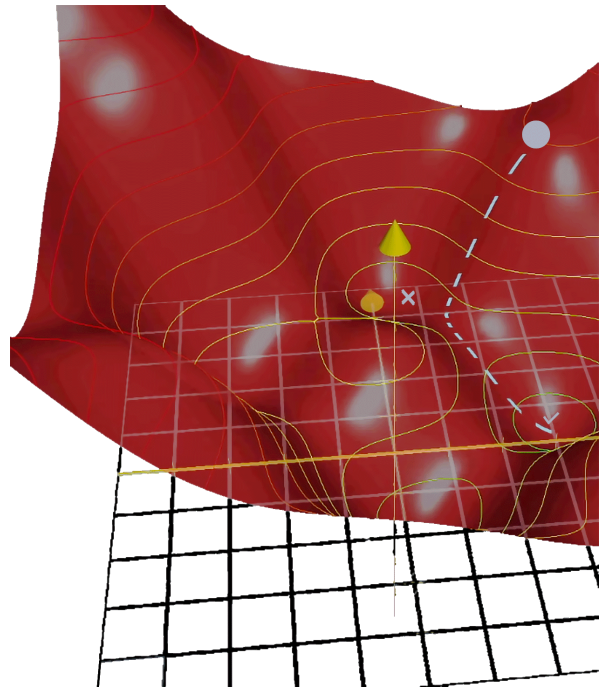


Abbildung 6: Visualisierung des Gradientenabstiegsverfahren im dreidimensionalen Raum (Sanderson 2017)

<sup>26</sup> $C$  bezieht sich hier auf die Cost function

<sup>27</sup>Sanderson 2017

<sup>28</sup>Fortuner 2017

difikator für Momentum. So fällt unter Umständen „der Ball“ weiter in ein tieferes Minimum, was in Abbildung 6 zu erkennen ist. Desweiteren wird für Machine Learning fast ausschließlich „Stochastic gradient descent“ verwendet, da das Gradientenabstiegsverfahren allein viel zu aufwendig ist. Mit dieser Form verliert man etwas Genauigkeit, der Prozess geht aber ein vielfaches schneller vonstatten. Anstatt den Gradienten von allen Trainingsdaten zusammen zu suchen, teilt man die Trainingsdaten in mehrere „Batches“ auf und wendet auf diesen das Gradientenabstiegsverfahren an. So werden häufiger/schneller Schritte richtung Extrempunkt gemacht, diese sind dafür aber ungenauer (sie repräsentieren nicht den schnellsten Weg).<sup>29</sup>

**Ketten Regel** Die Kettenregel vereinfacht das Ableiten von komplizierten Funktionen, zum Beispiel mit weiteren Funktionen im inneren. Wie man diese anwendet sieht man in Formel 6.<sup>30</sup>

$$\begin{aligned} f(x) &= u(g(x)) \\ f'(x) &= u'(g(x)) \cdot g'(x) \end{aligned} \tag{6}$$

Dieses Verfahren wird verwendet um die Ableitungen „verschachtelter“ Funktionen zu finden, in diesem Fall wird hiermit die Ableitung des Netzwerks und in späteren Schritten auch die Ableitung von Anteilen des Netzwerks gesucht.<sup>31</sup>

**Backpropagation** Jetzt ist klar was das Ziel des Trainingsprozesses ist, aber wie wird die Cost Function minimiert? Hier kommt der Backpropagation Algorithmus ins Spiel. In Kombination mit der Cost Function und den soeben besprochenen Verfahren bestimmt dieser Algorithmus welche und um wie viel die Weights und Biases des Neuronalen Netzwerks angepasst werden müssen. Dies passiert, wie der Name schon impliziert, rückwärts. Ziel ist es hier also den Einfluss der einzelnen Weights und Biases auf das Endergebniss herauszufinden, um so dann diese Werte anzupassen.

Der Algorithmus berechnet also die Ableitungen der einzelnen Teilstücke des Netzwerks mithilfe der Kettenregel und ist somit in der Lage herauszufinden welche Variablen (Weights und Biases) welchen Einfluss auf das Endergebniss haben.

## 2.4 Abwandlungen

Heute gibt es sehr viele verschiedene Abwandlungen von diesen Techniken und auch Neuronale Netzwerke, die zwar ähnlich aufgebaut sind, die besser für manche Zwecke sind als andere. Da gibt es zum Beispiel „Convolutional Neural Networks“<sup>32</sup>, welche

<sup>29</sup>Sanderson 2017 und Sra 2019

<sup>30</sup>Rudolph 2019

<sup>31</sup>Chen 2018

<sup>32</sup>auch CNN; deutsch: **faltendes** neuronales Netzwerk

besonders gut zum erkennen von Objekten in Bildern geeignet sind und daher auch eins in 3.3 verwendet wird. Oder „Long short Term memory Networks“<sup>33</sup>, welche speziell auf das erkennen von Stimmen ausgelegt sind.

### 2.4.1 Convolutional Neural Networks

Convolutional Neural Networks versuchen ein Problem der normalen Neuronalen Netzwerke im Bereich der Bildererkennung zu lösen. Während in einem normalen Netzwerk die Position des zu erkennenden Objektes im Bild eine Rolle spielt, wird dies zum Großteil in einem Convolutional Neural Network durch die veränderte Funktionsweise behoben/verbessert.

Während ein Mensch diese beiden Bilder aus Abbildung 7 direkt und ohne jegliche Mühen als Kreuz erkennen kann, ist dies für einen Computer eine große Schwierigkeit, auch für ein herkömmliches Neuronales Netzwerk. Die Werte gleichen sich schließlich nicht wirklich.

Hier kommt dann das CNN ins Spiel welches grob beschrieben nach Eigenschaften innerhalb des Bildes sucht, dieses Vorgehen ist in Abbildung 8 visualisiert. Convolutional NN's nutzen „Filtering“, ein Filter, wie in Abbildung 9 zu sehen, bewegt sich über das Bild und berechnet zu wie viel Prozent das Bild mit dem Filter übereinstimmt. Wendet man diese Berechnung, die auch „Convolution“ genannt wird, auf das erste Bild von Abbildung 8 an, so resultiert eine neue kleinere Matrix, wie in Abbildung 10 zu erkennen. Diese Matrix hat dort Werte nahe 1, wo der Filter zu 100% mit dem Bild übereinstimmt und kleine oder sogar negative Werte in Bereichen, in denen der Filter wenig oder keine Übereinstimmung mit dem Bild aufweist.<sup>34</sup> Zusammen mit mehreren anderen Convolutions (und daher auch Filtern) bildet sich ein Convolution Layer, welcher dann mit anderen Layern kombiniert wird, zum Beispiel herkömmliche Aktivierungsfunktio-

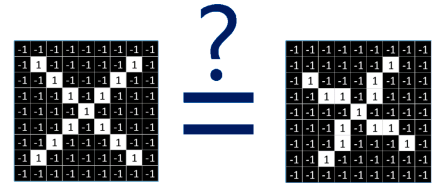


Abbildung 7: Repräsentationen zweier Kreuze durch zweidimensionale Arrays

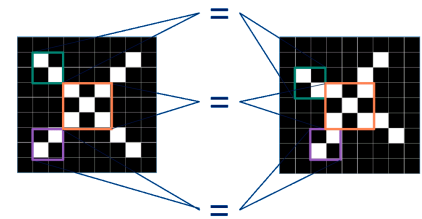


Abbildung 8: Vorgehen eines CNN's



Abbildung 9: Ein Filter innerhalb eines CNN's

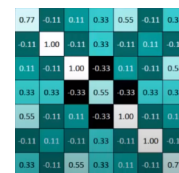


Abbildung 10: Resultierende Matrix (Abbildungen 7 bis 10 von Rohrer 2016)

<sup>33</sup>auch LSTMN; deutsch: Langes **Kurzzeitgedächtnis** Netzwerk

<sup>34</sup>Rohrer 2016

nen<sup>35</sup>, untereinander verbundene Layer<sup>36</sup> oder auch Pooling und Normalisierungs Layer, in letzterem werden einfach alle negativen Werte durch 0 ersetzt.<sup>37</sup>

**Pooling** Pooling wird verwendet um die aus einem Convolutional Layer resultierende Matrix weiter zu verkleinern, um so eine Beschleunigung des Trainings und auch der Anwendung zu erzielen.

Es gibt verschiedene Arten von Pooling, zum Beispiel das maxPooling, welches immer das Maximum eines bestimmten Bereichs übernimmt. Für das Pooling muss eine „Windows Size“ und ein „Stride“ Wert festgelegt werden. Window Size bestimmt die Größe des Bereichs in welchem das Pooling jeweils durchgeführt wird und Stride bestimmt wie weit dieser Bereich nach jedem mal verschoben wird. Führt man nun Pooling mit einer Window Size von 2 und einem Stride Wert von 2 auf der Matrix aus Abbildung 10 aus, so werden zuerst die Werte aus einem 2·2 Bereich oben links in der Matrix ausgewählt und das Maximum (hier: 1) in eine neue Matrix übernommen. Anschließend verschiebt sich der Bereich um 2 nach rechts und das Maximum wird erneut übernommen, das wird über die gesamte Matrix gemacht.<sup>38</sup> Daraus resultiert dann folgende Matrix:

1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77

Abbildung 11: Resultat des Poolings (Rohrer 2016)

### 3 Labelcheck als Smartphone App

In diesem Kapitel wird mithilfe von Python und TensorFlow ein Netzwerk erstellt und trainiert, sowie anschließend eine mobile App mit Dart und Flutter entwickelt, welche dann öffentlich für den Download bereit stehen soll.

#### 3.1 Die Idee

Die Ziel ist es, dass die App es ermöglicht im Supermarkt die verschiedenen Label der Produkte zu scannen und dem Nutzer dann Auskunft über die Vertrauenswürdigkeit und generelle Aussage des Labels gibt. Die Idee habe ich in meinem Erdkunde Leistungskurs bekommen als wir über die Problematik gestoßen sind, dass die Be-

<sup>35</sup>Bei CNN's zählen Aktivierungsfunktionen als eigene Layer; nicht zwangsläufig Teil eines Layers

<sup>36</sup>Normale Layer aus normalen NNs; Auch Dense Layer genannt

<sup>37</sup>Rohrer 2016

<sup>38</sup>Rohrer 2016

deutung der verschiedenen Label eher untransparent gegenüber dem Verbraucher ist.

## 3.2 Die Label/Siegel

## 3.3 Erstellen des Modells

Zum erstellen und trainieren des Modells werde ich die Sprache Python und das Framework TensorFlow verwenden.

### 3.3.1 Trainieren des Modells mit TensorFlow und Python

*Vollständiger Code in meinem Colab Notebook: <https://bit.ly/34Ggfuh><sup>39</sup> und im Anhang A.3*

## 3.4 Entwickeln der App

Zum entwickeln der App verwende ich die Sprache Dart und das zugehörige Framework Flutter. Im Gegensatz zu nativ geschriebenen Apps bietet Flutter die Möglichkeit nur einmal den Code in Dart zu schreiben und anschließend kann die App für alle großen Plattformen kompiliert werden, dazu zählen iOS, Android, aber auch Linux, Windows, MacOS und Web/Javascript. Nun muss die App ja Zugriff auf die Kamera haben und auch in die Supermärkte „mitgebracht“ werden, weshalb nur iOS und Android relevant sind.

### 3.4.1 Das Framework: Flutter

Flutter Apps funktionieren anders als nativ entwickelte Apps. Herkömmliche native Apps verwenden die UI<sup>40</sup> Komponenten des Betriebssystems und sehen daher auf jedem Gerät mit unterschiedlichen Betriebssystemversionen leicht unterschiedlich aus. Flutter hingegen stellt ein „Canvas“ Element bereit, welches als unterliegende Grafik-Engine Google's Skia nutzt.<sup>41</sup> In Flutter stehen eine Menge UI Komponenten zur Verfügung die entweder Googles Material Design guidelines oder Apples Human interface guidelines folgen. Der Dart Code stellt dann als Einzigen Eintrittspunkt die `main()` Methode bereit, aus welchem dann die App gestartet wird. Das Framework wird mit der Methode `runApp(Widget)` initialisiert. In Flutter ist jedes UI Element ein „Widget“, wodurch sich dann in Kombination in einer App große Widgethierarchien erstellen lassen. Der Code einer simplen App, welche nur den Text „Hello World!“ in der Mitte des Bildschirms anzeigen würde, sehe demnach so aus:

---

<sup>39</sup>Ungekürzter Link: [https://colab.research.google.com/drive/1ty\\_QQLL038YT6KpBjSdqGvIGyH0YXwxW](https://colab.research.google.com/drive/1ty_QQLL038YT6KpBjSdqGvIGyH0YXwxW)

<sup>40</sup>User Interface; deutsch: Benutzeroberfläche

<sup>41</sup>Flutter-Team 2020; mehr dazu im Anhang A.4

```

1 import 'package:flutter/widgets.dart';
2
3 void main() ⇒ runApp(MyApp());
4
5 class MyApp extends StatelessWidget {
6   @override
7   Widget build(BuildContext context) {
8     return Center(
9       child: Text('Hello World!'),
10    );
11  }
12 }

```

Zeile 1: Importieren der Widgets aus dem Framework zum bereitstellen der Klassen, wie Stateless- und StatefulWidget und den Methoden, wie runApp().

Zeile 3: Die main() Methode mit dem einzigen Aufruf runApp(), was die Klasse MyApp als Flutter App initialisiert.

Zeile 5f.: MyApp erbt die Klasse StatelessWidget, überschreibt die build() Methode und gibt eine Widgethierarchie zurück.

Zeile 8f.: Das Center Widget nimmt als einzigen benannten Parameter ein weiteres (child) Widget an, welches in diesem Fall ein Text Widget ist. Stateless bedeutet hier, dass sich der Zustand des Widgets nicht während der Laufzeit verändern kann. Im Gegensatz dazu gibt es auch noch StatefulWidget's welche die Möglichkeit haben bei Bedarf das UI zu „rebuilden“.

Um der App das Aussehen von nativen Apps zu verleihen, verwendet man üblicherweise ein MaterialApp (Material Design / Android) oder CupertinoApp (Human Interface / iOS) Widget, was zudem noch wichtige Variablen, wie ThemeData für verschiedene Farben, die in der App einfach verwendet werden können oder LocalizationsDelegate welche für die Bereitstellung verschiedener Sprachen gebraucht werden, beinhaltet.

### 3.4.2 Importieren des Models

Es gibt eine speziell für mobile Geräte angepasste Version von TensorFlow mit dem Namen TensorFlow Lite. TFLite hat einen geringeren Speicherbedarf kann dafür aber auch weniger als das herkömmliche Framework.<sup>42</sup> Implementationen dafür gibt es allerdings nicht in Dart, stattdessen verwendet man PlatformChannel's in Flutter, welche die Möglichkeit bieten platformspezifischen Code aus einer Flutter App auszuführen (Java/Kotlin für Android und Objective-C/Swift für iOS).<sup>43</sup>

Desweiteren muss das zuvor erstellte TensorFlow Model in ein TensorFlow Lite Model umgewandelt werden.

### 3.4.3 Veröffentlichen der App

Ich habe die App im Google PlayStore veröffentlicht, theoretisch wäre es auch möglich die App für iOS im Appstore anzubieten, leider ist ein Apple Entwicklerkonto

<sup>42</sup>TensorFlow-Team 2020

<sup>43</sup>Flutter-Team unbekannt

mit höheren Kosten verbunden.

### **3.5 Testen der App**

### **3.6 Fazit**

Ich habe in den letzten Monaten viel über die App entwicklung mit Flutter gelernt, leider habe ich mit dieser App schon etwas früher begonnen, weswegen ich heute wahrscheinlich viele Dinge anders gemacht hätte. Ich habe nicht viel darauf geachtet den UI Code von dem Logik Code zu trennen und jetzt kommt es häufig vor, dass ich lange nach etwas suchen muss. Ich plane allerdings dies noch zu beheben. Auch habe ich viele Variablen als dynamisch deklariert, was ich auch noch verbessern sollte. Das beeinträchtigt allerdings nicht die Funktionalität, beziehungsweise den Nutzer sondern nur mich, solange ich noch weiter an der App arbeiten möchte.



## A Anhang

### A.1 Weitere Aktivierungsfunktionen

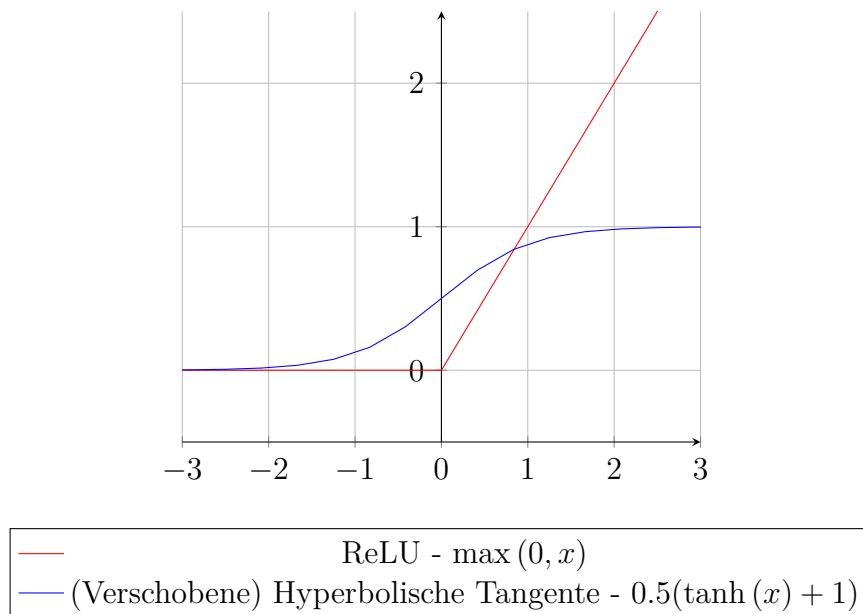


Abbildung 12: Weitere Aktivierungsfunktionen, ergänzend zu der Sigmoid Funktion aus Kapitel 2.3

Die ReLU (Rectified linear Unit) Funktion ist im Vergleich zu anderen Aktivierungsfunktionen, wie der Sigmoidfunktion oder der Hyperbolischen Tangente, deutlich simpler, was sich in Leistungsansprüchen des Trainingsprozesses wieder spiegelt.<sup>44</sup>

<sup>44</sup>Harrison Kinsley 2020

## A.2 Code für das Beispiel aus 2.3

```

1  import tensorflow as tf
2  import tensorflow.keras.layers as layers
3
4  numberOfNeuronsInFirstLayer = 16
5  numberOfNeuronsInSecondLayer = 16
6  numOfEpochs = 5
7
8  mnist = tf.keras.datasets.mnist
9
10 (x_train, y_train), (x_test, y_test) = mnist.load_data() # Laden des MNIST Datasets
11 # Und aufteilen in Trainingsdaten und Testdaten
12
13 x_train = tf.keras.utils.normalize(x_train, axis=1) # Normalisieren des Datasets
14 x_test = tf.keras.utils.normalize(x_test, axis=1)
15
16 model = tf.keras.models.Sequential() # Erstellen des Neuronalen Netzwerks
17 model.add(layers.Flatten())
18 model.add(layers.Dense(numberOfNeuronsInFirstLayer, activation=tf.nn.sigmoid))
19 # Hinzufügen der Layer
20 model.add(layers.Dense(numberOfNeuronsInSecondLayer, activation=tf.nn.sigmoid))
21 model.add(layers.Dense(10, activation=tf.nn.softmax))
22
23 model.compile(optimizer='adam',
24               loss='sparse_categorical_crossentropy',
25               metrics=['accuracy']) # Kompilieren der Layer zu einem trainierfähigen Model
26
27 model.fit(x_train, y_train, epochs=numOfEpochs)
28 # Trainieren des Modells mit den Trainingsdaten und x Epochen
29
30 model.save('beispielModel_MNIST') # Speichern des Modells

```

Listing 1: Umsetzung mit Python und Tensorflow

Ein interaktives Beispiel gibt es zusätzlich hier in meinem Colab Notebook:  
<https://bit.ly/34Ggfuh><sup>45</sup>

## A.3 Labelcheck Code

asdf

---

<sup>45</sup>Ungekürzter      Link:  
 QQLL038YT6KpBjSdqGvIGyH0YXwxW

[https://colab.research.google.com/drive/1ty\\_](https://colab.research.google.com/drive/1ty_)

## A.4 Die Flutter Architektur

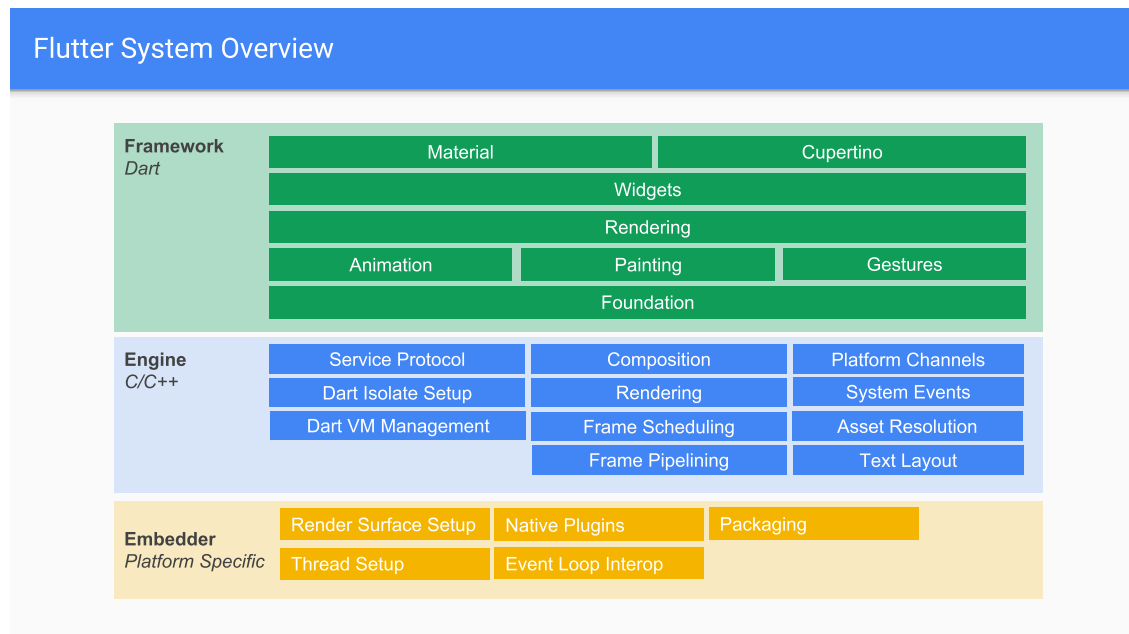


Abbildung 13: Flutter's Architektur (Flutter-Team 2020)

Flutter's Architektur ist in drei Ebenen unterteilt. Als Basis die „Embedder Ebene“, welche für jede Plattform angepasst werden muss und zum Beispiel für das Thread Management zuständig ist. Dadrüber liegt die „Engine Ebene“, welche zu einem Großteil in C++ geschrieben ist und zu welcher auch die Grafikengine Skia gehört. Dadrüber liegt die „Framework Ebene“, welche zum Beispiel die UI Komponenten beinhaltet und komplett in Dart entwickelt wird.<sup>46</sup>

<sup>46</sup>Flutter-Team 2020

## Literaturverzeichnis

- Byrnes, Nanette (2017). *Tim Cook: Apple Isn't Falling Behind, It's Just Not Ready to Talk About the Future*. URL: <https://www.technologyreview.com/2017/06/14/4525/tim-cook-apple-isnt-falling-behind-its-just-not-ready-to-talk-about-the-future/> (besucht am 05.12.2020).
- Chen, Qiang (2018). *Deep learning and chain rule of calculus*. URL: <https://medium.com/machine-learning-and-math/deep-learning-and-chain-rule-of-calculus-80896a1e91f9> (besucht am 07.01.2020).
- cleanpng (2020). *Deep learning maschinelles lernen, Künstliche neuronale Netzwerk der informatik Convolutional neural network - Vernetzung*. URL: <https://de.cleanpng.com/png-s07s8u> (besucht am 17.11.2020).
- Flutter-Team (2020). *The Engine architecture*. URL: <https://github.com/flutter/flutter/wiki/The-Engine-architecture> (besucht am 03.01.2021).
- (unbekannt). *Architectural overview: platform channels*. URL: <https://flutter.dev/docs/development/platform-integration/platform-channels?tab=android-channel-java-tab#architecture> (besucht am 03.01.2021).
- Fortuner, Brendan (2017). *ML Glossary - Gradient Descent*. URL: [https://ml-cheatsheet.readthedocs.io/en/latest/gradient\\_descent.html](https://ml-cheatsheet.readthedocs.io/en/latest/gradient_descent.html) (besucht am 28.12.2020).
- Geoffrey E. Hinton Alex Krizhevsky, Ilya Sutskever (2012). *ImageNet Classification with Deep Convolutional Neural Networks*. URL: <https://papers.nips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf> (besucht am 17.12.2020).
- Google (2020). *Machine Learning Glossary*. URL: <https://developers.google.com/machine-learning/glossary> (besucht am 26.12.2020).
- Harrison Kinsley, Daniel Kukiela (2020). *Neural Networks from Scratch (NNFS)*. URL: <https://nnfs.io> (besucht am 20.11.2020).
- Heaton, Jeff (2017). *Heaton Research, The Number of Neurons in the Hidden Layers*. URL: <https://www.heatonresearch.com/2017/06/01/hidden-layers.html> (besucht am 17.12.2020).
- Hebert, Chris (2020). *Accelerating WinML and NVIDIA Tensor Cores*. URL: <https://developer.nvidia.com/blog/accelerating-winml-and-nvidia-tensor-cores/> (besucht am 05.12.2020).
- Krzyk, Kamil (2018). *Coding Deep Learning for Beginners — Linear Regression (Part 2): Cost Function*. URL: <https://towardsdatascience.com/coding-deep-learning-for-beginners-linear-regression-part-2-cost-function-49545303d29f> (besucht am 27.12.2020).
- Rettner, Rachael (2020). *AI system solves 50-year-old protein folding problem in hours*. URL: <https://www.livescience.com/artificial-intelligence-protein-folding-deepmind.html> (besucht am 05.12.2020).

- Roberts, Eric S. (unbekanntes Jahr). *History: The 1940's to the 1970's*. URL: <https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history1.html> (besucht am 14.12.2020).
- Rohrer, Brandon (2016). *How do Convolutional Neural Networks work?* URL: [https://e2eml.school/how\\_convolutional\\_neural\\_networks\\_work.html](https://e2eml.school/how_convolutional_neural_networks_work.html) (besucht am 07.01.2020).
- Rudolph, Dennis (2019). *Kettenregel Ableitung*. URL: <https://www.gut-erklaert.de/mathematik/kettenregel-ableitung.html> (besucht am 02.01.2021).
- Sanderson, Grant (2017). *But what is a Neural Network?* URL: <https://www.3blue1brown.com/neural-networks> (besucht am 18.12.2020).
- Sra, Suvrit (2019). *25. Stochastic Gradient Descent*. URL: <https://www.youtube.com/watch?v=k3AiUhwHQ28> (besucht am 30.12.2020).
- TensorFlow-Team (2020). *TensorFlow Lite guide*. URL: <https://www.tensorflow.org/lite/guide> (besucht am 03.01.2021).
- Warren McCulloch, Walter Pitts (1943). *A logical calculus of the ideas immanent in nervous activity*.
- Werbos, Paul (1975). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*.
- Yann LeCun Léon Bottou, Yoshua Bengio und Patrick Haffner (1998). *Gradient-Based Learning Applied to Document Recognition*. URL: <http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf> (besucht am 17.12.2020).