

# **Maschinelles Lernen**

## **Studienarbeit**

### **Die Programmierung einer Bilderkennung.**

Verfasser: Rashid Goulebe

Matrikelnummer: 3417063

Professor: Prof. Dr. Boris Bittner

## Inhaltsverzeichnis

<b>Einleitung.....</b>	<b>2</b>
<b>Vorbereitung.....</b>	<b>3</b>
<b>Programmierung des neuronalen Netzes .....</b>	<b>4</b>
<i>Daten Aufbereiten und umformatieren. ....</i>	<i>4</i>
<i>Neuronales Netz Erstellen und Konfigurieren. ....</i>	<i>5</i>
<i>Neuronales Netz trainieren und testen.....</i>	<i>6</i>
<i>Anmerkungen und Optimierungsvorschläge. ....</i>	<i>7</i>
<b>Abbildungsverzeichnis .....</b>	<b>8</b>
<b>Quellenverzeichnis: .....</b>	<b>8</b>

## Einleitung

*„Unsere Intelligenz macht uns menschlich und KI ist eine Ergänzung dessen.“<sup>1</sup>*

Ein Zitat von Yann LeCun, Chief Artificial Scientist bei Facebook AI Research.

Yann LeCun betont hier, wie nützlich Künstliche Intelligenz (KI) ist und wie sie uns in verschiedenen Bereichen unterstützen und ergänzen kann.

KI erleichtern unseren Alltag. Sie ist kaum wegzudenken und ergänzt uns schon heute wie unsere rechte Hand. Anwendungen wie die Regelung der Ampelsteuerung zur Optimierung des Pendelverkehrs, Suchalgorithmen in Google oder die Erkennung eines Brusttumors sind durch maschinelles Lernen verbessert oder sogar erst möglich geworden.

Im Bereich der Bilderkennung hat maschinelles Lernen in den letzten Jahren enorme Fortschritte erzielt. Neuronale Netze die nicht nur zu 99,9% unterscheiden können ob auf einem Bild eine Katze zu sehen ist oder nicht, sondern auch präzise bestimmen können wo sich diese befindet, sind heute schon mit verhältnismäßig geringem Aufwand programmierbar.

Diese Seminararbeit erläutert, wie ein neuronales Netz programmiert und trainiert werden kann, welches zu 93% genau erkennt welche Zahl eine Hand zeigt.

---

<sup>1</sup> „Artificial Intelligence, Revealed“.

Ein nützliches Anwendungsbeispiel für solch ein neuronales Netz wäre Beispielsweise die Ansteuerung einer Lampe oder eines Rollladens per Handzeichen.

Da unsere Hand 5 Finger hat, kann damit im Dezimalzahlensystem ein Zahlenbereich von 0-5 abgedeckt werden. *Abbildung 1* wäre im Dezimalsystem dem entsprechend eine 2.

Um einen größeren Zahlenbereich abdecken zu können, betrachten wir unsere Hand als eine 5 Bit lange Binärzahl. Dadurch entstehen statt 6 Zuständen insgesamt 32 Zustände. *Abbildung 1* ist in diesem Fall eine 0.0.1.1.0. und somit eine 6.



*Abbildung 1: Hand*

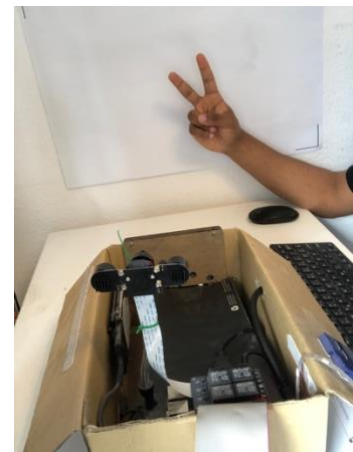
## Vorbereitung

Um ein neuronales Netz zu trainieren, werden viele Daten benötigt. Je mehr desto besser. Für die Bilderkennung wurden insgesamt 3200 Bilder aufgenommen. Um die Arbeit zu automatisieren wurde hierfür einen Raspberry-Pi 3b mit einem Kameramodul benutzt. Der Raspberry liest



*Abbildung 5: Raspberry-Pi Box*

über ein Python Skript eine Zahl von 0 bis 31 von der Tastatur ein. Nach bestätigen mit „Enter“ nimmt das Skript 100 Bilder in



*Abbildung 3: Aufbau*

einem 3 Sekunden Takt auf und speichert diese unter dem dazugehörigen Ordner im Verzeichnis ab.

Auf diese Weise war es möglich in 3 Stunden 3200 Bilder aufnehmen.

Abbildung 3 zeigt den Aufbau des Projekts. In der unteren Hälfte des Bildes befindet sich der Raspberry-Pi. Aus einem vorherigen Projekt ist der Pi noch in einem Schuhkarton, umwickelt von einer RGB-Lichterkette.

Abbildung 4 zeigt die gesamte Raspberry-Pi Box von vorne. Das Kameramodul ist auf einem ehemaligen verstellbaren Feuerzeughals befestigt, was die Einstellung des Winkels vereinfacht.

Um das neuronale Netz zu programmieren, wurde eine Virtuelle Maschine in der Google Cloud angemietet.

Virtuelle Maschinen haben den Vorteil, dass man für verhältnismäßig wenig Geld Zugriff auf Leistungsstarke Maschinen erhält, die gegebenenfalls auf das Trainieren von künstlichen Intelligenzen ausgelegt und spezialisiert sind.

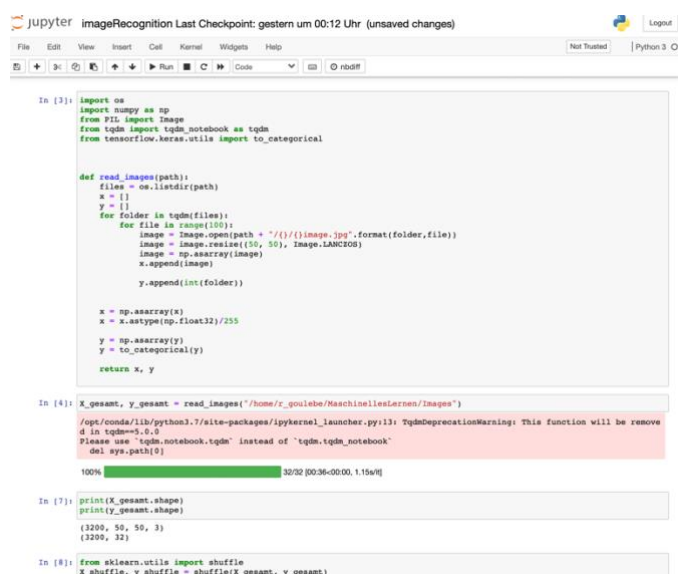
## Programmierung des neuronalen Netzes

Der Quellcode des neuronalen Netzes lässt sich in die drei Bereiche „**Daten Aufbereiten und umformatieren**“, „**Neuronales Netz Erstellen und Konfigurieren**“ und „**Neuronales Netz trainieren und testen**“ untergliedern.

### Daten Aufbereiten und umformatieren.

Ein neuronales Netz ist sehr Fehleranfällig, wenn die Daten nicht exakt in dem Format übergeben werden, indem der Input-Layer die Daten erwartet. Wenn ein Bild beispielsweise 128x128 Pixel hat, das neuronale Netz allerdings 64x64 Pixel im Input-Layer erwartet, müssen die Daten zunächst runter skaliert werden.

Abbildung 4 zeigt im oberen Abschnitt eine Funktion namens **read\_images**. Dieser Funktion kann ein Zielverzeichnis übergeben werden. Aus diesem Zielverzeichnis werden alle Ordner durchsucht. Die Bilddateien in den einzelnen Ordnern werden eingelesen und auf die Größe von 50x50x3 Pixel skaliert. Die x3 am Ende steht für den RGB-Farbwert des Pixels.



```

In [3]: import os
import numpy as np
from PIL import Image
from tqdm import tqdm_notebook as tqdm
from tensorflow.keras.utils import to_categorical

def read_images(path):
    files = os.listdir(path)
    x = []
    y = []
    for folder in tqdm(files):
        for file in Image(100):
            image = Image.open(path + "/" + folder + "/" + file)
            image = image.resize((50, 50), Image.LANCZOS)
            x.append(image)
            y.append(int(folder))

    x = np.asarray(x)
    x = x.astype(np.float32)/255

    y = np.asarray(y)
    y = to_categorical(y)

    return x, y

In [4]: X_gesamt, y_gesamt = read_images("/home/r_goulebe/MaschinellesLernen/Images")

/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:13: TqdmDeprecationWarning: This function will be removed in tqdm==5.0.0
Please use "tqdm_notebook.tqdm" instead of "tqdm.tqdm_notebook"
  del sys.path[0]

100%|#####| 32/32 [00:36<00:00, 1.15w/s]

In [7]: print(X_gesamt.shape)
print(y_gesamt.shape)
(3200, 50, 50, 3)
(3200, 32)

In [8]: from sklearn.utils import shuffle
X_shuffle, y_shuffle = shuffle(X_gesamt, y_gesamt)
    
```

Abbildung 7: Daten Aufbereiten & umformatieren

Anschließend werden die 50x50x3 großen Bilder einem Numpy-Array zugefügt. Das Array hat nach Einlesen der 3200 Bilder ein Format von (3200x50x50x3). Dieses Array bildet unsere X-Werte.

Die dazugehörigen Y-Werte sagen aus, welche Nummer auf dem jeweiligen Bild zu sehen ist. Das Array mit den Y-Werten wird parallel zum Einlesen der Bilder erstellt.

Anschließend werden die Daten mit einem **shuffle** Befehl vermischt. Das Vermischen der Daten beugt einem einseitig trainierten Netz vor.

Abschließend werden die Daten in 80% Trainings- und 20% Test-Daten aufgeteilt.

## Neuronales Netz Erstellen und Konfigurieren.

Ein neuronales Netz besteht grundlegend aus einem **Input-Layer**, null bis mehreren **Hidden-Layern** und einem **Output-Layer**.

```
In [16]: from keras.models import Sequential
from keras.layers import Dense, Conv2D

model = Sequential()
# Input-Layer
model.add(Conv2D(8, kernel_size=(3, 3), input_shape=(50,50, 3), activation="relu"))

# Hidden-Layer
model.add(Conv2D(16, kernel_size=(3, 3), activation="relu"))
model.add(Conv2D(32, kernel_size=(3, 3), activation="relu"))
model.add(Flatten())

# Output-Layer
model.add(Dense(32, activation="softmax"))

model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
```

Abbildung 9: Neuronales Netz erstellen und Konfigurieren

### Der Input-Layer:

In diesem Beispiel ist der Input-Layer ein **Convolutional Neuronal Network** (CNN).

Ein CNN-Layer ist in der Lage 2-dimensionale Daten einzulesen und darüber hinaus über mehrere Ebenen hinweg Kanten und Formen auf den Bildern zu finden. Das funktioniert, indem wir dem CNN eine **kernel\_size** übergeben. In diesem Fall eine *kernel\_size* von (3,3) was einer 3x3 Matrix entspricht. Diese Matrix wird mathematisch über das Bild hinweg geschoben und errechnet mit unterschiedlichen Gewichten einen Farbübergang oder eine Veränderung der Helligkeit. Durch einen CNN-Layer wird dem neuronalen Netz zusätzliche Intuition über die Daten mitgegeben.<sup>2</sup>

Ebenso definieren wir den CNN-Layer über den Parameter **input\_shape** als einen Input-Layer und bestimmen, dass die Daten in Form einer 50x50x3 großen Matrix übergeben werden.

Der Parameter **activation** steht auf **relu**. Relu steht für Rectifier was auf deutsch Gleichrichter bedeutet.<sup>3</sup> Diese Funktion sorgt dafür, dass die Eingabe direkt ausgegeben wird, wenn sie positiv ist. Sie bestimmt wie die Gewichte trainiert werden.

<sup>2</sup> „Deep Learning, Neuronale Netze & AI“.

<sup>3</sup> „Rectifier (neuronale Netzwerke)“.

## Die Hidden-Layer:

Bei diesem neuronalen Netz bestehen die Hidden-Layer aus zwei weiteren CNN-Layer mit 16 und 32 3x3 Matrizen und einem Flatten-Layer. Auf dieser Ebene sind die CNN-Layer bereits in der Lage Kanten und Formen zu erkennen.

Diese Kantenerkennung lässt sich visualisieren, indem die Gewichte des Modells ausgelesen und mit Matplotlib geplottet werden. Auf [Abbildung 6](#) ist zu sehen, dass der CNN-Layer an der gelb markierten Stelle einen Farbübergang feststellen konnte. Diese Erkenntnis, zusammengesetzt mit vielen weiteren Interpretationen, lassen das neuronale Netz schlussfolgern, dass es sich auf dem Bild zu 93% um eine 0.1.1.1.0. also um eine 14 handeln muss. <sup>4</sup>

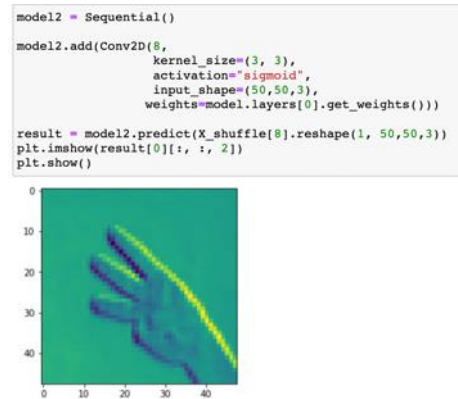


Abbildung 11: CNN-Layer visualisieren

Nach den Zwei CNN-Layern kommt abschließend ein **Flatten-Layer**. Dieser Flatten-Layer formatiert die Daten lediglich von einer zwei-dimensionalen in eine ein-dimensionale Matrix um. Das ist notwendig, da der darauffolgende Dense-Layer als Input eine ein-dimensionale Matrix erwartet.

## Der Output-Layer:

Der Output-Layer gibt die ermittelte Entscheidung des neuronalen Netzes an. Besitzt der Output-Layer nur Ein einziges Neuron, so ist die Antwort Binär wie Ja/Nein bzw. True/False. Wenn ein neuronales Netz beispielsweise erkennen soll ob auf einem Bild eine Katze zu sehen ist oder nicht, so lautet die Antwort entweder Ja oder Nein.

Da es in der Studienarbeit allerdings 32 mögliche Antworten gibt, benötigt der Output-Layer auch 32 Neuronen.

## Neuronales Netz trainieren und testen

Abschließend muss das Modell trainiert und getestet werden.

Um das Modell zu trainieren, werden ihm die **Trainingsdaten** von X und Y übergeben. Ebenso müssen die **Epochen** und eine **Batch-Größe** angegeben werden. Die Batch-Größe bestimmt wie viele Daten für ein Training genutzt werden. Die Epochen geben an, wie oft die gesamten Trainingsdaten durchlaufen werden sollen. Beim Ausführen des Trainings wird nach jedem Durchgang die Genauigkeit als **accuracy**

```
In [45]: model.fit(
X_train,
y_train,
epochs=10,
batch_size=32,
shuffle=True)

Epoch 1/10
80/80 [=====] - 7s 80ms/step - loss: 3.3728 - accuracy: 0.1681
Epoch 2/10
80/80 [=====] - 6s 78ms/step - loss: 0.6502 - accuracy: 0.8119
Epoch 3/10
80/80 [=====] - 6s 78ms/step - loss: 0.2343 - accuracy: 0.9355
Epoch 4/10
80/80 [=====] - 6s 78ms/step - loss: 0.0925 - accuracy: 0.9759
Epoch 5/10
80/80 [=====] - 6s 79ms/step - loss: 0.0458 - accuracy: 0.9897
Epoch 6/10
80/80 [=====] - 6s 79ms/step - loss: 0.0267 - accuracy: 0.9935
Epoch 7/10
80/80 [=====] - 6s 78ms/step - loss: 0.0087 - accuracy: 0.9966
Epoch 8/10
80/80 [=====] - 6s 79ms/step - loss: 0.0168 - accuracy: 0.9962
Epoch 9/10
80/80 [=====] - 6s 79ms/step - loss: 0.0082 - accuracy: 0.9960
Epoch 10/10
80/80 [=====] - 6s 79ms/step - loss: 0.0046 - accuracy: 0.9990

Out[45]: <keras.callbacks.History at 0x7f9a572a50>

In [47]: model.evaluate(X_test, y_test)

20/20 [=====] - 0s 18ms/step - loss: 0.5743 - accuracy: 0.9266

Out[47]: [0.5743206739425659, 0.926562488079071]
```

Abbildung 13: Training und Test

<sup>4</sup> „Deep Learning, Neuronale Netze & AI“.

angegeben. Nach der 10. Epoche hat das Netz eine Genauigkeit von 99,9% auf die Trainingsdaten.

Wichtig ist zu beachten, dass sich die Genauigkeit nur auf unsere Trainingsdaten und nicht auf unsere Testdaten bezieht.

Um unser Modell auf Bilder zu testen die es zuvor noch nie gesehen hat, nutzen wir den **model.evaluate** Befehl zusammen mit unseren Test-Daten.

In *Abbildung 7* ist zu sehen, dass wir auf unsere Trainingsdaten eine Genauigkeit von 99.9% haben, unsere Testdaten allerdings nur zu 92,6% genau deuten können. Für den professionellen Einsatz muss das Modell weiter verbessert werden.

Insgesamt ist das Ergebnis allerdings ein guter Anfang, um darauf aufzubauen.

Im folgenden Abschnitt werden die nächsten Schritte erläutert die zu erwägen sind, um eine Steigerung der Genauigkeit zu erzielen.

## Anmerkungen und Optimierungsvorschläge.

### Batchsize:

Auffällig ist, dass das Modell bereits nach 6 Epochen eine Genauigkeit von 99,3 % aufweist. In den darauffolgenden Epochen verbessert sich das neuronale Netz nur um hundertstel. Das Spricht dafür, dass das Modell in den letzten 4 Epochen wenig, bis keine Veränderung an den Gewichten vornimmt. Um Zeit beim Trainieren des neuronalen Netzes einzusparen, können die Epochen von 10 auf 6 reduziert werden, sofern die Genauigkeit für den Anwendungsfall hinreichend präzise ist.

### Pixel:

In diesem Beispiel hat das Originalbild eine Größe von  $640 \times 480$ . Um Rechenleistung zu sparen und die Trainingsdauer zu verkürzen, wurde die Bildgröße auf  $50 \times 50$  komprimiert. Bei diesem Vorgang gingen viele Informationen verloren, die das neuronale Netz eventuell genauer machen. Ein komplexeres Modell mit einem größeren Input-Layer und mehreren Hidden-Layern würde eventuell mehr rechen Leistung in Anspruch nehmen, könnte allerdings die Genauigkeit weiter verbessern.

### Datenmenge:

Für diesen Versuch wurden 2.560 Bilder für das Training und 640 Bilder für Genauigkeitsmessungen des Netzes benutzt. Verschiedene Bibliotheken in Python bieten die Möglichkeit durch Spiegelung oder geringer Verschiebung des Bildes die Datenmengen künstlich zu vergrößern. Eine größere Datenmenge für das Training, können das Modell weiter verbessern.

### Layer

In diesem Projekt wurden nur CNN-Layer, ein Flatten-Layer und ein Dense-Layer benutzt. Die Bibliothek von Keras bietet viele weitere Layer-Arten an. Ein komplexeres Netz mit verschiedenen Layern und anderen Parametern kann zu einem noch präziseren Ergebnis führen.



## Abbildungsverzeichnis

Abbildung 1: Hand .....	3
Abbildung 1: Hand .....	3
Abbildung 3: Aufbau .....	3
Abbildung 3: Aufbau .....	3
Abbildung 2: Raspberry-Pi Box .....	3
Abbildung 2: Raspberry-Pi Box .....	3
Abbildung 4: Daten Aufbereiten & umformatieren .....	4
Abbildung 4: Daten Aufbereiten & umformatieren .....	4
Abbildung 5: Neuronales Netz erstellen und Konfigurieren .....	5
Abbildung 5: Neuronales Netz erstellen und Konfigurieren .....	5
Abbildung 6: CNN-Layer visualisieren .....	6
Abbildung 6: CNN-Layer visualisieren .....	6
Abbildung 7: Training und Test .....	6

## Quellenverzeichnis:

- Facebook Engineering. „Artificial Intelligence, Revealed“, 1. Dezember 2016.  
<https://engineering.fb.com/2016/12/01/ml-applications/artificial-intelligence-revealed/>.
- Udemy. „Deep Learning, Neuronale Netze & AI: Der Komplettkurs“. Zugriffen 31. Juli 2021. <https://www.udemy.com/course/deep-learning-und-ai/>.
- „Rectifier (neuronale Netzwerke)“. In *Wikipedia*, 29. Dezember 2020.  
[https://de.wikipedia.org/w/index.php?title=Rectifier\\_\(neuronale\\_Netzwerke\)&oldid=207030353](https://de.wikipedia.org/w/index.php?title=Rectifier_(neuronale_Netzwerke)&oldid=207030353).