# Workout Klassifizierung

Leonard Bardtke

[1] HAW-Hamburg
[2] Leonard.Bardtke@haw-hamburg.de

## 1 Abstract

Das Erkennen von Fitnesübungen wird mit einem "convolutional neural network" umgesetzt. In diesem sind fünf Klassen definiert, wodurch die Möglichkeit besteht Liegestützen und Sit-Ups zu erkennen. Um komplette Wiederholungen erkennen zu können, sind für jede Übung zwei unterschiedliche Positionen definiert. Außerdem ist eine Klasse definiert, welche für die Erkennung zuständig ist, falls keine Übung ausgeführt wird. Um die Wiederholungen der Übungen zählen zu können, werden csv-Datein verwendet, dessen Zähler im Falle einer kompletten Wiederholung inkrementiert wird. Die Ausgaben der Netzes werden in einer state-machine verarbeitet, wobei jede Ausgabe des Netzes als Zustand definiert ist. Um dem Benutzer über fehlerhafte Ausführungen einer Übung informieren zu können, wäre die Umsetzung einer seitlichen Ansicht auf die Fitness Übung erforderlich. Da aus in dieser Arbeit aus Platzgründen eine Frontalansicht der Fitness Übung umgesetzt wurde, ist die Erkennung von Fehlerhaften Ausführung nur bedingt möglich. Um die accuracy des neuronalen Netzes zu erhöhen ohne neue Trainingsdaten hinzuzufügen, ist zum Beispiel das Einfügen von Identity-Layern möglich. Außerdem ist es möglich das trainierte Modell durch das Verwenden von Farbfiltern unempfindelicher gegen Helligkeits- und Kontrastunterschieden zu machen.

## 2 Introduction

### 2.1 Motivation

Vor allem in der aktuellen Corona-Lage, in der ein Besuch im Fitness Studio nur unter Einschränkungen möglich ist, haben viele Leute das Home-Workout für sich eindeckt. Da man Zuhause jedoch oftmals keine Trainingsgeräte zur Verfügung hat, bieten sich Übungen mit dem eigenen Körpergewicht, wie zum Beispiel Liegestütze, an.

### 2.2 Aufgabe

Dieses Projekt beschäftigt sich damit, diese Übungen zu erkennen und deren Wiederholungen zu zählen. Zudem wird in dieser Arbeit behandelt, ob es möglich ist den Benutzer auf eine fehlerhafte Ausführung hinzuweisen, um möglichen

Verletzungen vorzubeugen oder somit die Effektivität des Trainings steigern zu können. Für die Umsetzung wurde ein selbst lernendes Neuronales Netz verwendet, da diese viele positive Eigenschaften gegenüber herkömmlichen Ansätzen besitzen, da zum Beispiel Analytische Lernverfahren bei hochdimensionalen Eingangsgrößen undurchführbar werden. Zum einen verfolgen neuronale Netze zudem das Prinzip des überwachten Lernens und sind in Lage ein System anhand von repräsentativen Beispielen zu trainieren und zum anderen sind neuronale Netz im Gegensatz zu fest programmierten Algorithmen in der Lage, ihr Verhalten in Bezug auf veränderte Eingaben anzupassen [3].

Für das in Folgenden Projekt wurde ein Guide verwendet, mit welchem ein eigenes Neuronales Netz zur Gesten Erkennung erstellt werden kann. [7]. Das hier verwendete neuronale Netz basiert auf "multi-gesture recognition", wobei die eintrainierten Gesten die zu erkennenden Übungen darstellen. Für jede Übung wurden zwei Klassen eintrainiert, um in der Lage zu sein, komplette Ausführungen zu erfassen. Mithilfe dieser Klassen können somit Liegestütze und Sit-ups gezählt werden.

### 2.3 Aufbau

Die Arbeit lässt sich in vier Phasen einteilen. Die erste Phase besteht darin zunächst Traingsdaten zu beschaffen, mit denen eine gute Generalisierung erreicht werden kann. Diese müssen gegebenenfalls vorverarbeitet werden. Anschließend muss eine Netzarchitektur festgelegt werden, welche für den Anwendungsfall geeignet ist. In der dritten Phase folgt das trainieren des neuronalen Netzes. Die Umsetzungs- und Generalisierungsfähigkeit muss daraufhin anhand von Testdaten oder mithilfe eines Praxistests überprüft werden. Anhand dieser Auswertung müssen gegebenenfalls neue Testdaten aufgenommen oder eine Anpassung des Netzes vorgenommen werden. Bei einer zuverlässigen Erkennung müssen die Ausgaben des Netzes anschließend in der vierten Phase korrekt verarbeitet werden, um in der Lage zu sein die Wiederholungen der Übungen zählen zu können.

## 3 Projekt Ablauf

### 3.1 Phase 1: Beschaffung der Trainingsdaten

Für eine optimale Erkennung von Liegestützen würde sich eine seitliche Aufnahme der Übung anbieten. Da dies aus Platzgründen jedoch nicht umsetzbar war, wurden die Trainingsdaten aus einer Frontalansicht aufgenommen. Für die Trainingsdaten wurden fünf verschiedene Klassen definiert. Da von dem neuronalen Netz immer die Klasse mit der höchsten Wahrscheinlichkeit Ausgegeben wird, wurde zunächst eine Klasse definiert, bei der keine Übung ausgeführt wird und sich zum Beispiel keine Person im Bild befindet (Fig.1) oder sich eine Person normal im Kameraausschnitt bewegt(Fig.2). Die zweite Klasse dient zur Erkennung von Liegestützen in der Ausgangsposition (Fig.3) und die dritte Klasse

zur Erkennung von Liegestützen in der unteren Position (Fig.4). Die Klassen vier und fünf enthalten Trainingsdaten von Sit-Ups in der Ausgangsposition (Fig.5) und Sit-Ups in der unteren Position (Fig.6). Jede Klasse enthält ca. einen Trainigsdatensatz von 1600 Bildern, womit der finale Trainingsdatensatz ca. 8000 Trainingsdaten umfasst. 10% dieser Daten werden beim als Validierungsdaten verwendet, um die Generalisierung des Netzes zu erhöhen.
Die Trainingsbilder werden als PNGs abgespeichert und mit verschiedenen Farbfiltern und unterschiedlichen Helligkeitsstufen versehen, um unempfindlicher gegen Helligkeits- und Kontrastunterschiede zu werden und somit für eine bessere Generalisierung zu sorgen.

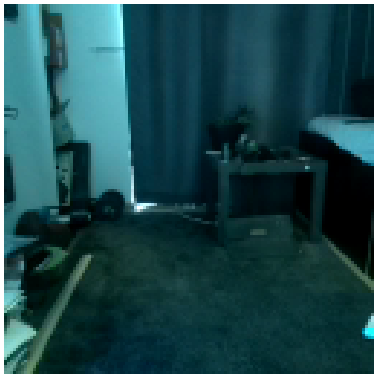

**Fig. 1.** Class0(empty)

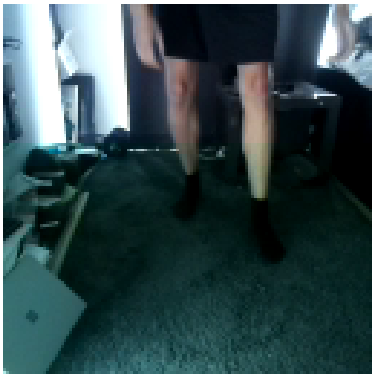

**Fig. 2.** Class0(walking around)

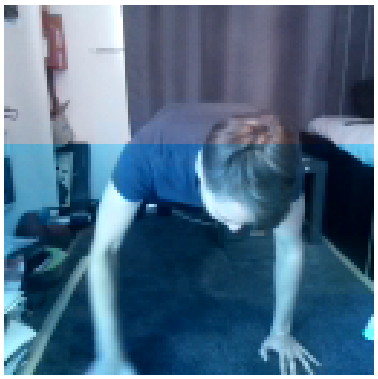

**Fig. 3.** Class1(PushUp-Top)

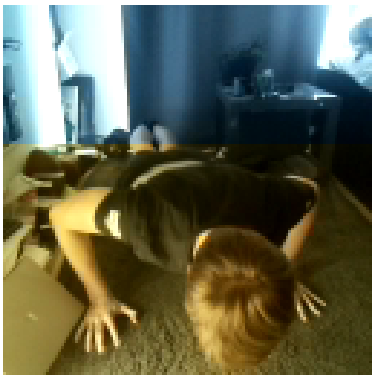

**Fig. 4.** Class2(PushUp-Down)

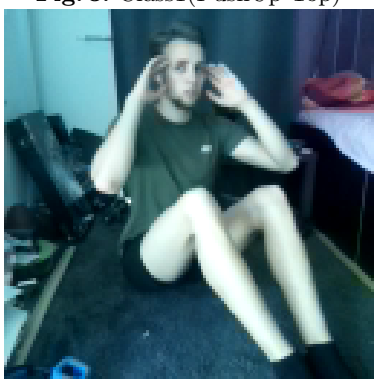

**Fig. 5.** Class3(SitUp-Top)

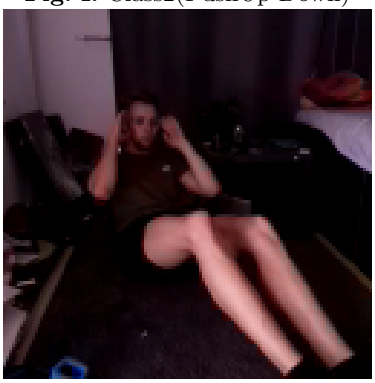

**Fig. 6.** Class4(SitUp-Down)

### 3.2 Phase 2: Netzarchitektur festgelegen

Bei dem trainierten Neuronalen Netz handelt es sich um ein Neuronales Faltungsnetz, da in diesen, im Gegensatz du Feed-Forward-Netzen, lokale Strukturen erhalten bleiben. Das Netz besteht aus einem Input-Layer, zwei darauffolgende Convolutional-Layer und einem Output-Layer. In allen Layern, bis auf den Output-Layer, wird die elu-Aktivierungsfunktion verwendet. Diese hat gegenüber logistischen Aktivierungsfunktionen den Vorteil, dass das Vanishing-gradient-Problem deutlich reduziert wird. Des Weiteren besteht der Vorteil, dass die elu-Aktivierungsfunktion(Fig.7) das Entstehen von toten Neuronen, welche durch das verwenden der relu-Aktivierungsfunktion auftreten können, verhindert wird.

Da es sich bei den Trainigsbildern um Farbbilder mit der Auflösung 128*128 handelt, besitzt der Input-Layer eine input-shape von 128*128*3 (Listing 1.1, Zeile 3-4). Zudem enthält der Input-Layer 32 Neuronen und einen Faltungskern der Größe 3*3. Da auf den Input-Featuremaps kein zero-padding angewandt wird, besitzen die Output-Featuremaps des Input-Layers eine Auslösung von 126*126. Der anschließende Max-polling-Layer (Listing 1.1, Zeile 6) sorgt mit einer 2*2 Faltungsmaske für eine Halbierung der Featuremap-Auflösung und somit für eine Datenkonzentration. Die Output-Featuremaps des Max-polling-Layers haben somit eine Auslösung von 63*63 (Fig.2, Z.29).
Der anschließende Layer verfügt ebenso über 32 Neuronen und einem Faltungskern der Größe 3*3(Listing 1.1, Zeile 8). Auch hier wird anschließend ein Max-pooling-Layer mit einer 2*2-Faltungsmaske verwendet, wodurch es zu Featuremaps mit der Auflösung 30*30 kommt.
Auch im daruffolgeden Layer wird ein Convolutional-Layer mit einer Faltungsmaske der Größe 3*3 verwendet, jedoch verfügt dieser Layer 64 Neuronen. Somit enthält der anschließende Max-pooling-Layer 64 Featuremaps mit der Auflösung 28*28 und führt eine Dimensionsreduktion durch.
Die Output-Featuremaps des Max-pooling-Layers mit der Auslösung 14*14, werden anschließend durch den Flatten-Layer (Listing 1.1, Zeile 16) zu einem Vektor umgeformt. Dieser besitzt daraufhin 12.544 Eintrage, welche in ein Dense-Layer mit 64 Neuronen gegeben wird (Listing 1.1, Zeile 17). Durch das verwenden eines Dropout-Layers (Listing 1.1, Zeile 19) wird zudem verhindert, dass es zum Overfitting des Netzes kommt und somit die Generalisierungsfähigkeit erhalten bleibt. Die Anzahl an Neuronen im Output-Dense-Layer entspricht der Anzahl der zu erkennenden Klassen. Da es sich um eine Klassifikation handelt und gewünscht ist, dass die Ausgabe des Output-Layers als Wahrscheinlichkeit der Ausgangsklassen interpretierbar ist, wird im Output-Layer ein Softmax-Aktivierungsfunktion verwendet(Listing 1.1, Zeile 21).

```
model = Sequential

model.add(Conv2D(filters=32, kernel_size=(3, 3),
                                    input_shape = (128, 128, 3)))
model.add(Activation('elu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(filters=32, kernel_size=(3, 3)))
model.add(Activation('elu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(filters=64, kernel_size=(3, 3)))
model.add(Activation('elu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(units=64))
model.add(Activation('elu'))
model.add(Dropout(0.5))
model.add(Dense(nb_classes))
model.add(Activation('softmax'))
```

**Listing 1.1.** Neuronale Netz

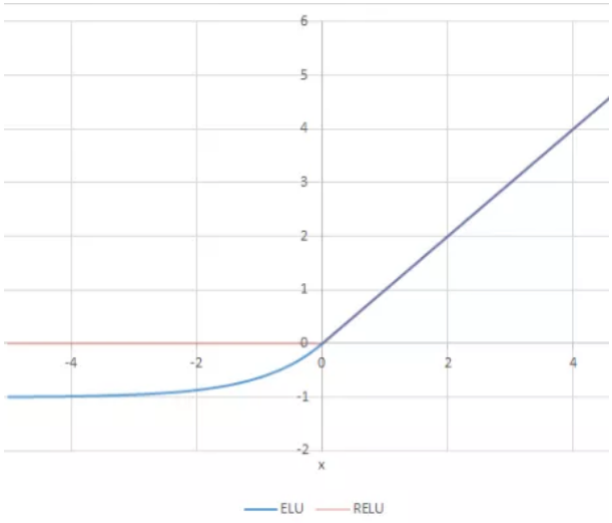


**Fig. 7.** elu-Aktivierungsfunktion [5]

### 3.3 Phase 3: Training und Testen

Das neuronale Netz wurde mithilfe des Minibatchverfahrens trainiert, da das Batchverfahren bei einer großen Trainingsdatenmenge sehr ineffiziert ist. Des Weiteren hat das Festlegen eines early-stoppings, dazu beigetragen, dass es beim trainieren des neuronalen Netzes nicht zum Overfitting kommt. Außerdem wird die Crossentropy-Fehlerfunktion verwendet(Listing 1.2, Zeile 1), um Lerning-Slowdowns zu verhindern, da die Lerngeschwindigkeit bei der Crossentropy-Fehlerfunktion größer wird, wenn auch der Fehler der Klassifikation groß ist. Zudem wird der Schrittweitenfaktor durch "adam" pro Gewicht individuell und adaptiv berechnet. Die adaptive Schrittweitenanpassung sorgt für ein besseres Lernverhalten im Backpropagation-Algorithmus, da der Schrittweitenfaktor größer wird, so lange sich die Richtung des Gradienten nicht ändert und kleiner wird, wenn sich die Richtung des Gradienten ändert. Zum testen des neuronalen Netzes wurde die 2-fache-Kreuzvalidierug verwendet, um das Netz mit Bildern zu testen, die nicht im Traings- und Validierungdatensatz enthalten waren. Zudem wurden bei den Testbildern andere Klamotten verwendet. Auf diese Weise wird die Generalisierungsfähigkeit des trainierten Modells überprüft. Des Weiteren wurde mit dem trainierten Modell ein Praxistest vollzogen, um die Anwendbarkeit des Netzes zu testen.

```
1       model.compile(loss='categorical_crossentropy',
2                     optimizer='adam',
3                     metrics=['accuracy'])
```

**Listing 1.2.** optimizer und Fehlerfunktion

### 3.4 Phase 4: Verarbeitung der Netzausgaben

Zum auswerten der Netzausgaben dient eine state-machine, welche für jede Ausgabe des Netzes einen Zustand besitzt(Fig.8). Um eine komplett Liegestütze auszuführen muss vor der Ausgangsposition (Fig.3) zuvor die untere Liegestütz Position(Fig.4) erfasst worden sein. Dies wird durch das verwenden eines "ready-flag" realisiert. Wenn somit die Ausgangsposition erkannt wird, bevor die untere Liegestütz Position detektiert wurde, wird dieser Zustand nicht betreten(Listing 1.3, Zeile 8-9). Erst bei der Erkennung der unteren Liegstütz Position wird das ready-flag auf True gesetzt (Listing 1.3, Zeile 5). Um bei einer Ausführung nicht zu viele Wiederholungen zu zählen, wird der ready-flag nach dem erkennen der Ausgangsposition erneut auf False gesetzt (Listing 1.3, Zeile 13).

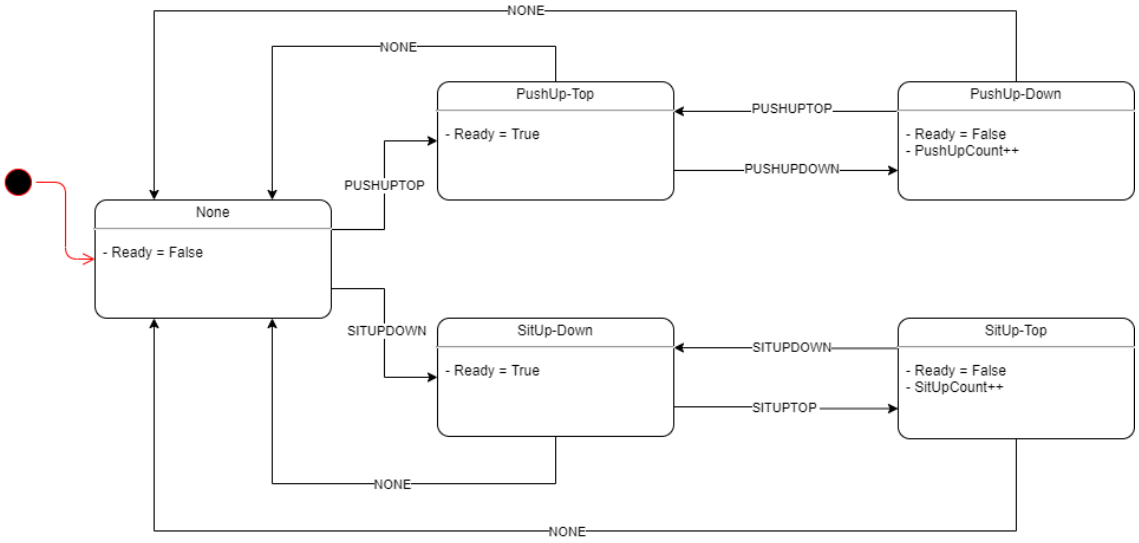


**Fig. 8.** state-machine

```
1           if action == PUSHUPDOWN:
2               if self.state == 'push-down':
3                   return
4               self.state= 'push-down'
5               self.ready = True
6               return
7
8           if not self.ready:
9               return
10
11          if action == PUSHUPTOP:
12              self.state = 'push-top'
13              self.ready = False
14              countPushUp()
15              return
```

**Listing 1.3.** state-machine am Beipsiel vom Liegestützen

Um in der Lage zu sein die Wiederholungen der Übungen zählen zu können, müssen diese abgespeichert werden. Dies erfolgt in einer csv-Datei. Wenn eine komplette Liegestütze erfasst wurde, wird die Funktion "countPushUp"(Listing 1.3,Zeile 14). Zu Beginn wird das csv-File "reps.csv" geöffnet(Listing 1.4,Zeile 2). Anschließend werden die gespeicherten Werte des csv-Files in Variablen gelanden (Listing 1.4,Zeile 4-6) und der entsprechende Zähler wird daraufhin inkrementiert (Listing 1.4,Zeile 8). Im Folgenden wird der inkrementierte Zähler in die entsprechende Stelle im csv-File geschrieben(Listing 1.4,Zeile 9-13). [4]

```
1         def countPushUp():
2           with open('reps.csv', newline='') as csvfile:
3               reader = csv.DictReader(csvfile)
4               for row in reader:
5                   pushUpREPS = row['Push-Ups:']
6                   sitUpREPS = row['Sit-Ups:']
7
8           newRepCount = int(pushUpREPS)+1
9           with open('reps.csv', 'w', newline='') as csvfile:
10              fieldnames =['Push-Ups:', 'Sit-Ups:']
11              writer = csv.DictWriter(csvfile , fieldnames=fieldnames)
12              writer.writeheader()
13              writer.writerow({'Push-Ups:': newRepCount, 'Sit-Ups:': sitUpREPS})
14          print("\nPush-ups: %d" % newRepCount)
```

**Listing 1.4.** countPushUp-Funktion

# 4 Auswertung

## 4.1 Trainingsanalyse

Der Ansatz bestand darin, dass Projekt mittels "pose-estimation" umsetzen, wobei jedoch kein funktionierendes Beispiel zu finden war, welches auf dem Raspberry-Pi 4 ausgeführt werden konnte. Nach dem Umstieg auf eine "multi-gesture recognition" wurde zunächst ein Modell zur Kniebeugen-Erkennung erstellt, welches zum Test der Funktionalität diente. Dies wurde mit ca. 550 Trainingsbildern pro Klasse traininert und zeigte die Umsetzbarkeit des Projektes im Praxistest. Im Anschluss wurde der Fokus auf Bodenübungen, wie Liegestütze und Sit-Ups, gelegt.
Das erste beiden Modellen enthalten lediglich drei Klassen, da sich zunächst auf das Erkennen von Liegestützen konzentriert wurde. Für das trainieren wurden ca. 1.500 Trainingsbilder verwendet. In der ersten Klasse des ersten Modells ist eine sehr hohe accuracy festzustellen (Model-1 von Fig.9), was auf eine ungleiche Verteilung von Trainingsdaten zurückzuführen ist. Da sich im Traingsdatensatz des ersten Modells viele Bilder der ersten Klasse befinden, hat sich das Modell auf das Erkennen dieser Klasse spezialisiert.
Um für eine gleichmäßigere Verteilung von Trainingsdaten zu sorgen, wurden für das zweite Modell mehr Daten der ersten und zweiten Klasse aufgenommen. Anschließend lagen ca. 1.000 Trainingsdaten pro Klasse vor. Zu dem wurde bei dem zweiten Modell Wert darauf gelegt, den Trainingsdatensatz um die Positionen jeweiligen Klassen zu erweitern, welche im Test für uneindeutige oder falsche Klassifizierungen gesorgt haben. Somit kam es im zweiten Modell zu einer sichtbaren Steigerung der accuracy(Model-2 von Fig.9).

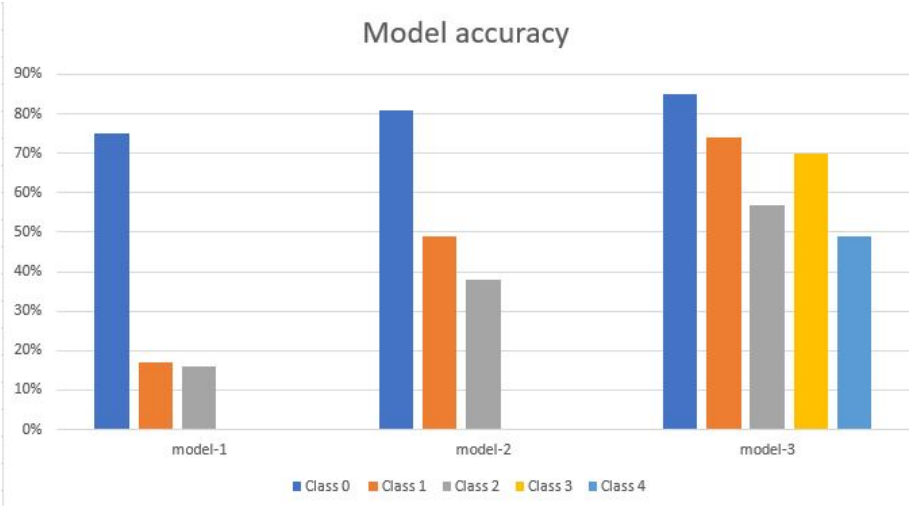


**Fig. 9.** "Model accuracy

Das dritte Modell wurde um zwei Klassen erweitert, um auch in der Lage zu sein Sit-Ups zu erkennen. Hier zeigte sich ein wesentlicher Nachteil der Frontalansicht. Wie in Fig.9-11 zu erkennen, besteht eine Ähnlichkeit dieser drei Positionen, weshalb es auch häuft zu Falscherkennung dieser Position kommt. Durch eine Erhöhung der Trainingsdaten oder eine Umpositionierung der der Pi-Kamera, kann dieses Problem beseitigt werden. Durch das Verwenden eines Droupout-Layers und dem Festlegen eines early-stoppings, kommt es bei dem Training nicht zu ein Overfitting des dritten Models, da sich die validation-accuracy über der trainings-accuracy befindet. (Fig.13)

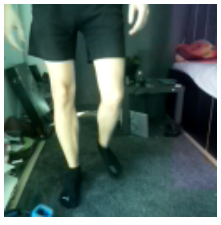

**Fig. 10.** Nachteil-der-Frontalansicht(Class0)

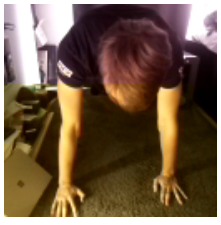

**Fig. 11.** Nachteil-der-Frontalansicht(Class1)

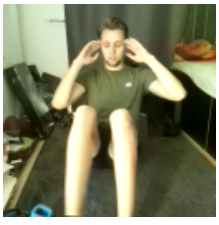

**Fig. 12.** Nachteil-der-Frontalansicht(Class3)

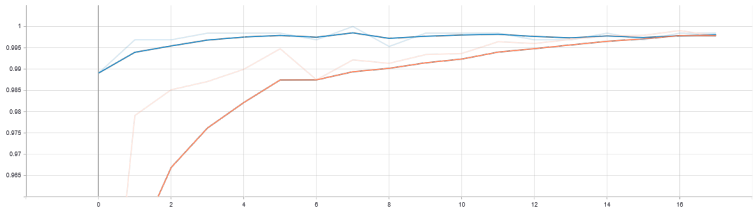

**Fig. 13.** epoch-accuracy von Model-3

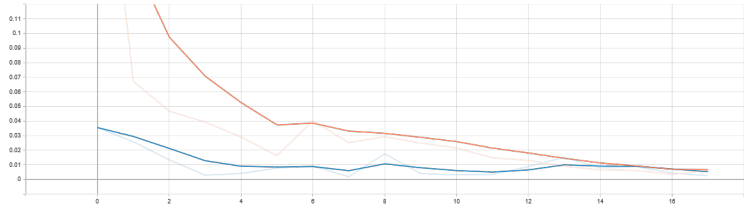

**Fig. 14.** epoch-loss von Model-3

### 4.2 Identity-Mapping

Im Folgenden wird der Frage nachgegangen, inwiefern sich die accuracy eines Netzes noch erhöhen lässt, ohne die Anzahl der Trainingsbilder erhöhen. Da durch das Vanishing-Gradient-Problem mehr als 2-3 Hidden-Layer nicht mehr sinnvoll trainierbar sind, werden dem neuronalen Netz Identity-Layer hinzugefügt(Listing 1.5, Zeile 8-17). Diese sorgen durch "shurtcut-connections" dafür, dass sie niemals schlechter werden können als ein äquivalentes flaches Netz. Mitunter ist es so möglich noch komplexere Funktionen zu approximieren. Mit gerade Mal einem Identity-Layer liegt die Gesamt-accuracy des vierten Modells ca. 20% höher als

im vorherigen flachen Netz (Fig.15,model-4). Auf diese Weise ist es möglich, eine höhere accuracy zu erreichen, ohne mehr Trainingsdaten zu verwenden.

```
    .
    .
    .
    x = Conv2D(filters=64, kernel_size=(3, 3))(x)
    x = Activation('elu')(x)
    x = MaxPooling2D(pool_size=(2, 2))(x)

    n = 1
    for i in range(0,n):
        x1 = Conv2D(filters=64, kernel_size=(3, 3), padding='same')(x)
        x1 = Activation('relu')(x1)
        x1 = BatchNormalization()(x1)

        x1 = Conv2D(filters=64, kernel_size=(3, 3), padding='same')(x1)
        x1 = Activation('relu')(x1)

        x = Add()([x1, x])

    x = Flatten()(x)
    .
    .
    .
```

**Listing 1.5.** Ausschnitt des Identity-Layer

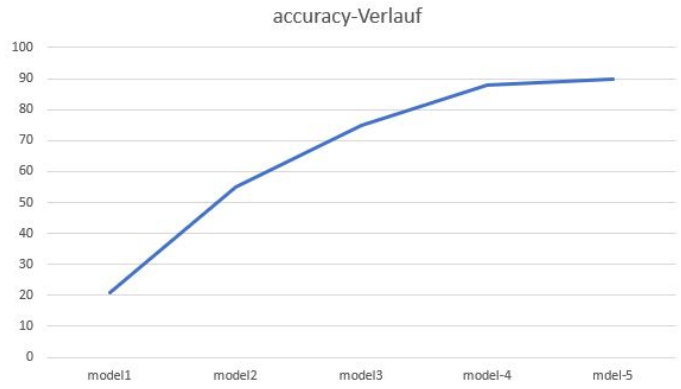


**Fig. 15.** Verlauf der accuracy

### 4.3 Erkennung falscher Ausführungen

Das Erkennen von fehlerhaften Ausführungen ist durch die Frontalansicht der Pi-Kamera nur bedingt möglich. Wenn zum Beispiel der Rücken bei der Ausführung einer Liegestütze nicht gerade ist, ist dies bei einer Frontalansicht nur schwer zu erkennen. Bei der Umsetzung mit einer seitlichen Ansicht könnte die fehlerhafter Ausführung einer Übung als eigene Klasse im neuronalen Netz definiert werden.

Bei einer Umsetzung mittels pose-estimation wäre es möglich, die Ausgabe des Netzes einer Klasse zuzuordnen. Die Bestimmten Anordnung der erkannten Körperteile, kann somit auch als fehlerhafte Ausführung einer Übung definiert werden.

## 5 Fazit

Bei der Erstellung eines Neuronalen Netzes ist, bei einer gleichmäßig verteilten Relevanz der Klassen, auf eine gleichmäßige Verteilung der Trainingsdaten auf die zu erkennenden Klassen zu achten. Ansonsten Besteht die Möglichkeit, dass sich das Netz auf die Erkennung dieser Klasse spezialisiert und die Genauigkeit der übrigen Klassen abnimmt. Des Weiteren ist zu beachten, dass es nicht zum Overfitting des Netzes kommt und die validation-accuracy über der trainings-accuracy liegt. Durch das Einfügen von Dropout-Layern oder dem Festlegen eines early-stoppings kann dies Vermieden werden. Außerdem spielt die Ausrichtung der Kamera eine große Rolle und kann bei dem Umsetzen des Projektes von entscheidender Relevanz sein. Die Datenvorverarbeitung kann großen Einfluss auf die Generalisierungsfähigkeit des neuronales Netzes haben. Mithilfe von Farbfiltern ist es möglich das Modell unempfindelicher gegen Helligkeits- und Kontrastunterschieden zu machen. Außerdem ist es möglich durch das Verwenden von Identity-Layern, Tiefe Netze zu erzeugen und komplexere Funktionen zu approximieren.
Um fehlerhafte Ausführung erkennen zu können, ist eine seitliche Ansicht auf die zu erkennende Übung notwendig. Des Weiteren kann es bei einer Frontalansicht auf die Fitness Übungen leicht falschen Klassifizierungen kommen, da die verschiedenen Übungen Teilweise eine Ähnlichkeit ausweisen.

# Bibliography

[3] 3. Eigenes neuronales netzwerk mit python. http://www.ai-united.de/eigenes-neuronales-netzwerk-mit-python. Eingesehen am 10.07.2020.

[4] 4. Csv file reading and writing. https://docs.python.org/3/library/csv.html/. Eingesehen am 10.07.2020.

[5] 5. Elu as a neural networks activation function. https://ml-cheatsheet.readthedocs.io/en/latest/$_{i}mages/elu.png.Eingesehenam$05.07.2020.

[7] 7. multi-gesture recognition. https://developer.arm.com/solutions/machine-learning-on-arm/developer-material/how-to-guides/teach-your-raspberry-pi-multi-gesture/single-page/. Eingesehen am 11.05.2020.