Einfache Gestensteuerung für einen Videoplayer auf dem Raspberry Pi

Michael Münstedt

10. Juli 2020

Zusammenfassung

Auf einem Raspberry Pi 4 Model B wird mittels eines neuronalen Netzes eine Gestensteuerung implementiert, mit der man zum Beispiel einen Videoplayer steuern kann. Es stellt sich heraus, dass einfache Gesten vor einer Wand, sogar ohne Bildvorverarbeitung, sehr einfach für ein Faltungsnetzwerk zu klassifizieren sind. Das neuronale Netz wird mit Tensorflow und Keras implementiert.

Inhaltsverzeichnis

1 Einleitung

Das Ziel ist es eine einfache Gestensteuerung für ein Multimedia Programm zu entwerfen. Das typische Szenario ist hier jemand, der auf dem Sofa sitzt und einen Film über seinen Raspberry Pi schaut. Um zu pausieren oder die Lautstärke zu verändern hebt er wie in Abbildung 1 seine Hand und macht Gesten anstatt aufzustehen und die Maus und Tastatur zu bedienen. Dafür hat der Raspberry Pi eine Kamera, die auf die Person gerichtet ist und mittels eines neuronalen Netzes die Gesten erkennt und das Programm entsprechend steuert. Dabei soll es drei Gesten geben: die flache Hand wird hoch gehalten, der Daumen wird bei geballter Faust ausgestreckt und der Daumen wird bei geballter Faust herunter gestreckt. Da man allerdings auch keine Geste zeigen kann, zum Beispiel wenn die Hand nicht im Bild ist, wird noch eine vierte "Geste"benötigt, die alles repräsentiert, was keine der drei Gesten ist.

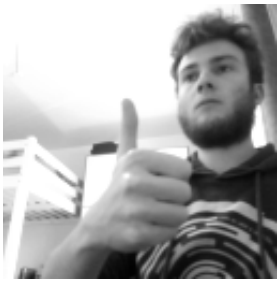

Abbildung 1: Beispiel einer Geste

2 Problem

Weil das Abspielen eines hochauflösenden Videos sehr rechenaufwendig ist und ein Raspberry Pi nur über sehr begrenzte Rechenleistung verfügt, muss die Gestenerkennung mit so wenig Ressourcen wie möglich auskommen. Um den Raspberry Pi hierbei zu entlasten wird ein Coral Edge TPU Stick benutzt, auf dem die Vorhersage des neuronalen Netzes berechnet wird. Häufig wird ein Bild jedoch, bevor es in das neuronale Netz zur Klassifizierung gespeist wird, vorher verarbeitet um eine höhere Genauigkeit der Vorhersage und eine höhere Flexibilität der Umgebungsvariablen(Helligkeit, Hintergrund, Rauschen...) zu erreichen. Da diese Bildvorverarbeitung allerdings ebenfalls viele Ressourcen benötigt, wäre im schlimmsten Fall nicht mehr genug Rechenleistung vorhanden um das Video mit ausreichend vielen Bildern pro Sekunde abzuspielen. Daher wird die Gestenerkennung ohne Bildvorverarbeitung stattfinden.

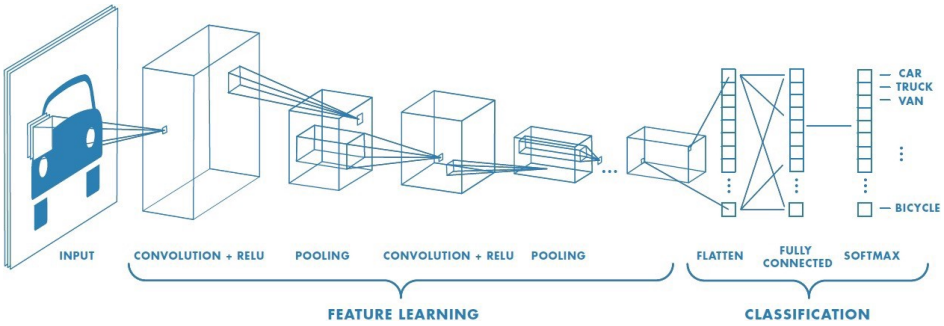

Abbildung 2: Typischer Aufbau eines Faltungsnetzwerks [2]

3 Trainings- und Testdaten erzeugen

Mittels eines Python Skriptes wird auf die Kamera des Raspberry Pis zugegriffen und es werden so schnell wie möglich Bilder gemacht. Pro Geste wird nun 1000 mal die Hand mit der entsprechenden Geste fotografiert. Dabei wird die Hand gedreht und bewegt, sodass möglichst viele verschiedene Blickwinkel entstehen. Gleichzeitig werden die Belichtungseinstellungen der Kamera randomisiert verändert um verschiedene Helligkeiten zu simulieren. Die so aufgenommenen Bilder werden anschließend noch einmal darauf geprüft ob die geforderte Geste tatsächlich darauf zu sehen ist. Von jeder Geste werden dann 20% der Bilder zufällig als Testdaten ausgewählt und 80% als Trainingsdaten verwendet.

4 Wahl des neuronalen Netzes

Da mit Bildern trainiert wird, eignet sich am besten ein Faltungsnetzwerk, weil der Input als Matrix eingegeben wird. Dadurch ist der Vorteil zum Beispiel gegenüber eines Muli layer Perceptrons, dass Objekte in einem Bild unabhängig von der Position des Objekts im Bild erkannt werden können [1]. Diese Eigenschaft ist natürlich bei der Gestenerkennung von hoher Bedeutung, insbesondere, da hier keine Bildvorverarbeitung, wie Zentrierung, stattfindet. Das Faltungsnetzwerk wird wie in Abbildung 2 gestaltet. Die Bilder werden im Grauwertmodus verwendet, da im Farbmodus drei verschiedene Kanäle verwendet werden und dies mehr Trainingsaufwand bedeutet. Außerdem spielt die Farbe beim Erkennen von Händen keine Rolle, da unabhängig von der Farbe der Hand die Gesten erkannt werden sollen.

Beim Training zeigt sich, dass die Validierungsgenauigkeit(im Folgenden Genauigkeit) mit dem ersten Netz, mit drei convolutional layern, nicht größer wird als 76%. Mit dem Hinzufügen eines vierten convolutional layers und der Erhöhung der Filtergröße kann die Genauigkeit auf 82% erhöht werden, was für eine zuverlässige Gestensteuerung aber deutlich zu wenig ist.

Auch mit weiteren Variationen der Anzahl der convolutional layer und der Filtergröße lässt sich die Genauigkeit nicht signifikant erhöhen. Dies liegt unter anderem daran, dass die Gesten oft einen sehr schlechten Kontrast zum Hintergrund haben, wenn sie sich zum Beispiel vor dem Gesicht befinden. Ein weiterer Punkt ist, dass zu viele Umgebungsvariablen verändert werden müssen um eine große Generalisierungssfähigkeit des neuronalen Netzes zu erreichen. Normalerweise wird dies durch entsprechende Bildvorverarbeitung vereinfacht.

Um die zuverlässige Erkennungsrate auch mit unterschiedlicher Kleidung, wechselnder Körperhaltung und anderer Kameraposition zu gewährleisten, bräuchte man zu viele Trainingsbilder. Wenn man pro Geste bei gleicher Körperhaltung, Kameraposition und Kleidung 1000 Bilder benötigt und man jeweils 5 verschiedene Körperhaltung, Kameraposition und Kleidung aufnehmen will benötigt man $1000 \cdot 5 \cdot 5 \cdot 5 = 125000$ Bilder pro Geste. Nicht nur die Datenaufnahme ist damit zu aufwendig, auch die Trainingszeit wäre deutlich zu lang. Der Computer mit dem trainiert wird, hat eine AMD fx6300 CPU mit sechs Kernen und 3,5 Gigahertz Takt und eine Grafikkarte ohne CUDA Unterstützung, womit diese nicht von Tensorflow und Keras benutzt werden kann. Beim Trainieren mit 1000 Bildern pro Geste und insgesamt vier Gesten dauert eine Trainingsepoche im Schnitt 120 Sekunden. Bei 125 mal so vielen Trainingsdaten würde das Training entsprechend zu lange dauern.

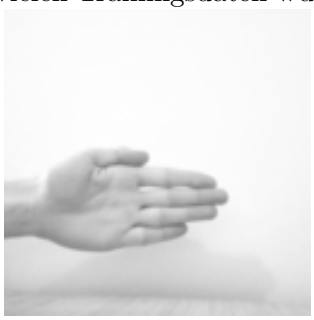

Abbildung 3: Flache Hand

Abbildung 4: Daumen hoch

Abbildung 5: Daumen runter

Wegen des zu großen Aufwandes wird deswegen die Kamera anders platziert. Sie ist nun nicht mehr auf die Person gerichtet, sondern auf die Wand hinter ihr. In den Trainingsdaten ist also nur eine Hand vor einer Wand und nicht die ganze Person inklusive Umgebung zu sehen(siehe Abbildungen 3, 4 und 5).

Mit den neuen Trainings- und Testbildern wird im Training sehr schnell eine Genauigkeit von 99% erreicht. Es werden nun verschiedene Variationen des, aus den Experimenten mit den alten Trainingsbildern entstandenen, neuronalen Netzes getestet um am schnellsten zu trainieren und die höchste Genauigkeit zu erreichen.

4.1 Auswirkung der Bildgröße

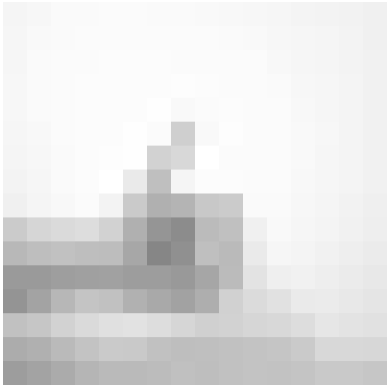

Abbildung 6: 16 mal 16 Pixel

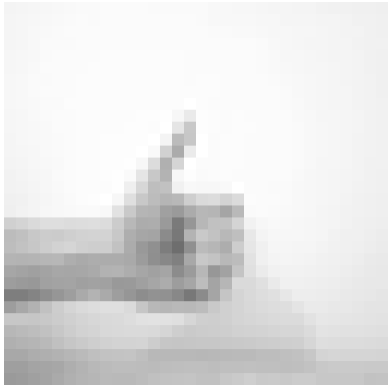

Abbildung 7: 32 mal 32 Pixel

Abbildung 8: 64 mal 64 Pixel

Abbildung 9: 128 mal 128 Pixel

In den Abbildungen 6, 7, 8 und 9 ist jeweils ein Trainingsbild in verschiedenen Größen für die Geste “Daumen Hoch” zu sehen. In der Abbildung 10 kann man erkennen, dass die Genauigkeit bei den vier verschiedenen Größen über 99% innerhalb der ersten drei Epochen erreicht wird.

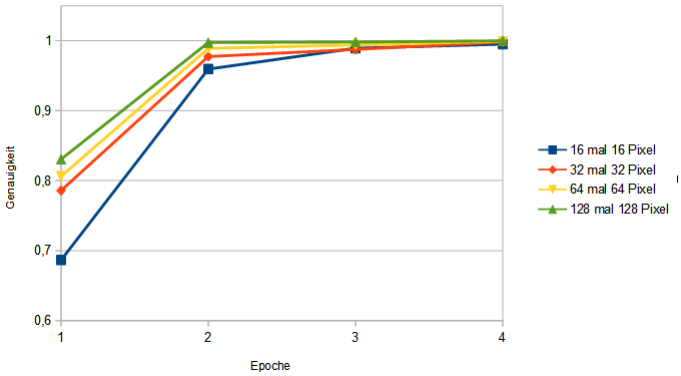

Abbildung 10: Auswirkung der Bildgröße auf die Genauigkeit

Da die Trainingszeit bei kleineren Bildern viel geringer ist(17 Sekunden bei 16 mal 16 Pixeln) als bei großen(269 Sekunden bei 128 mal 128 Pixeln) wird im ersten Versuch mit der Bildgröße 16 mal 16 gearbeitet. Hier stellt sich aber heraus, dass die 99% Genauigkeit in der Praxis nicht erreicht werden. Ungefähr jedes dritte Bild wird falsch erkannt. Besonders oft wird die flache Hand erkannt, wo eigentlich Daumen hoch oder Daumen runter gezeigt wird. Vermutlich liegt dies daran, dass der abstehende Daumen, der das charakteristische Merkmal gegenüber der flachen Hand ist, bei dieser kleinen Bildgröße teilweise nicht deutlich genug auf den Bilder zu erkennen ist. Der Genauigkeitsunterschied zwischen Praxis und Testdaten liegt vermutlich an der mangelnden Generalisierungsfähigkeit des Netzes. Da die Bilder in der Praxis zum Beispiel andere Schatten haben als die Testdaten, ist die Generalisierungsfähigkeit eine wichtige Eigenschaft des Netzes. Bei 32 mal 32 Pixeln ist die Genauigkeit in der Praxis schon deutlich besser, nur jedes zehnte Bild wird falsch erkannt. Bei 64 mal 64 Pixeln kommt es dann zu keinen falschen Vorhersagen mehr. Auch die Gesten einer anderen Person als aus den Trainingsdaten werden richtig erkannt und obwohl auf den Traingsbildern keine Ärmel zu sehen sind, erkennt das Netz auch zuverlässig alle Gesten bei langärmeliger Kleidung. Somit wird als Bildgröße 64 mal 64 Pixel verwendet, da die Genauigkeit nicht mehr verbessert werden kann und bei mehr Pixeln nur die Trainings- und Inferenzzeit steigen würde.

4.2 Overfitting

Beim Training fällt auf, dass die Trainingsgenauigkeit höher ist als die Validierungsgenauigkeit, was auf overfitting, also die Überanpassung an die Trainingsdaten, schließen lässt. Daher wird ein Dropout layer eingesetzt welches in jedem Trainingszyklus 30% der Parameter deaktiviert und so hilft overfitting zu verhindern [5].

4.3 Optimizer und loss function

Es wird der Optimizer Adam mit einer Lernrate von 0,001 verwendet, da dieser sehr effizient ist und im allgemeinen gute Ergebnisse liefert [6]. Als loss function wird categorical crossentropy verwendet, weil mehr als zwei Klassen erkannt werden müssen [4].

4.4 Anzahl der convolutional layer

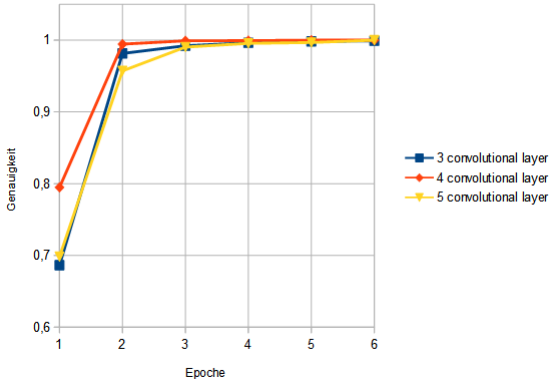

Abbildung 11: Auswirkung der Anzahl der convolutional layer auf die Genauigkeit

Es stellt sich die Frage bei wie vielen convolutional layern am schnellsten die höchste Genauigkeit erreicht wird. In einer Messreihe zeigt sich wie in Abbildung 11 zu sehen, dass mit drei, vier und fünf convolutional layern eine sehr hohe Genauigkeit erzielt wird. Mit dem Ziel eine Genauigkeit von 99% zu erreichen, eignet sich das Netz mit vier convolutional layern am besten, da dieses schon nach zwei Epochen und nach 278 Sekunden über 99% Genauigkeit erreicht. Die anderen beiden Netze mit drei und fünf convolutional layern benötigen mindestens drei Epochen und brauchen damit länger(siehe Abbildung 12.

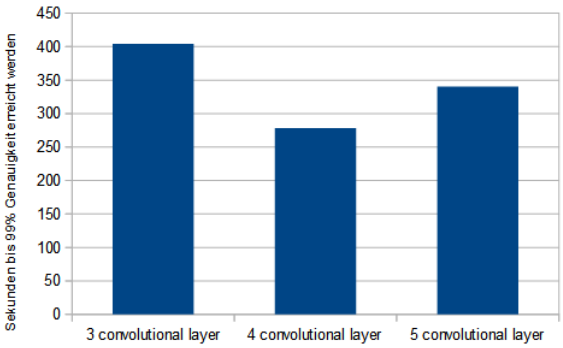

Abbildung 12: Dauer bis 99% Genauigkeit erreicht werden

4.5 Batch Größe

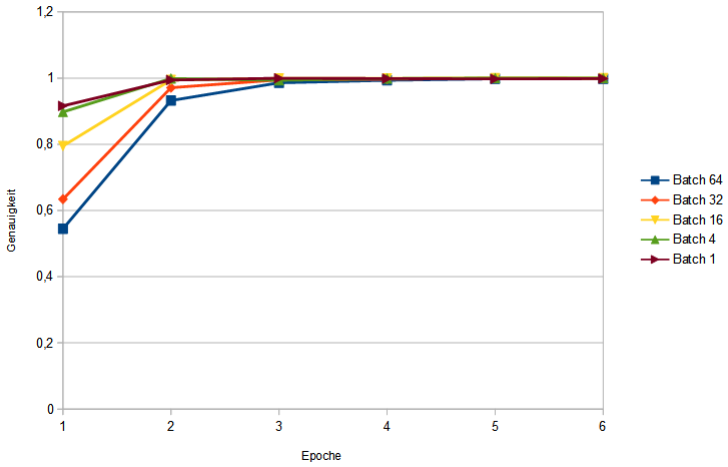

Abbildung 13: Auswirkung der Batch Größe auf die Genauigkeit

In einem Experiment stellt sich wie in Abbildung 13 zu sehen heraus, dass bei kleinerer Batch Größe auch die Genauigkeit steigt.

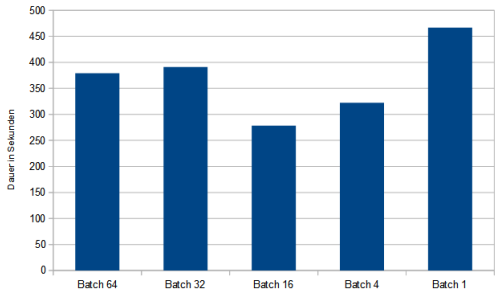

Abbildung 14: Dauer bis 99% Genauigkeit erreicht werden

Mit dem Ziel 99% Genauigkeit zu erreichen, eignet sich die Batch Größe 16 am besten, da 99% Genauigkeit im Gegensatz zum Training mit Batch Größe 32 und 64 schon nach zwei Epochen erreicht wird, was die Zeit verringert. Bei noch kleinerer Batch Größe steigt die Zeit dann wieder an(Abbildung 14).

```
model = Sequential()
model.add(InputLayer(input_shape=[SIZE,SIZE,1]))

model.add(Conv2D(filters=32,kernel_size=3,activation='relu'))
model.add(MaxPool2D(pool_size=2))

model.add(Conv2D(filters=50,kernel_size=3,activation='relu'))
model.add(MaxPool2D(pool_size=2))

model.add(Conv2D(filters=80,kernel_size=3,activation='relu'))
model.add(MaxPool2D(pool_size=2))

model.add(Conv2D(filters=100,kernel_size=3,activation='relu'))
model.add(MaxPool2D(pool_size=2))

model.add(Flatten())
model.add(Dropout(rate=0.3))
model.add(Dense(len(labels),activation='softmax'))
optimizer = Adam(lr=1e-3)
model.compile(optimizer=optimizer, loss='categorical_crossentropy',metrics=['accuracy'])
```

Abbildung 15: Architektur des verwendeten Netzes

Das schließlich verwendete Netz(Abbildung 15) hat 124554 Parameter.

4.6 Vereinfachtes Netz

Da die Gestenerkennung vor einer Wand mit den gewählten Gesten, wie an der schnellen Trainingsgeschwindigkeit und der sehr hohen Genauigkeit zu erkennen, nicht sehr komplex ist, reicht es ein einfacheres neuronales Netz zu benutzen. Durch Reduzierung der Filtergröße und convolutional layer wird die Anzahl der zu trainierenden Parameter verringert und somit auch die Trainingszeit.

```
model = Sequential()
model.add(InputLayer(input_shape=[SIZE,SIZE,1]))

model.add(Conv2D(filters=8, kernel_size=3, activation='relu'))
model.add(MaxPool2D(pool_size=2))

model.add(Conv2D(filters=16, kernel_size=3, activation='relu'))
model.add(MaxPool2D(pool_size=2))

model.add(Conv2D(filters=32, kernel_size=3, activation='relu'))
model.add(MaxPool2D(pool_size=2))

model.add(Flatten())
model.add(Dropout(rate=0.3))
model.add(Dense(len(labels),activation='softmax'))
optimizer = Adam(lr=1e-3)
model.compile(optimizer=optimizer, loss='categorical_crossentropy',metrics=['accuracy'])
```

Abbildung 16: Einfaches Netz mit weniger Filtern

In Abbildung 16 ist das einfachere Netz zu sehen, welches nur noch 10500 Parameter hat. Trotz fast zwölffach weniger Parametern als in Abbildung 15 wird ebenfalls eine Genauigkeit von über 99% erreicht. Es werden im Vergleich zum komplexeren Netz mehr Epochen benötigt(sieheAbbildung 17),

aber durch die verringerte Komplexität dauert das Training nur 108 Sekunden bis 99% Genauigkeit erreicht werden.

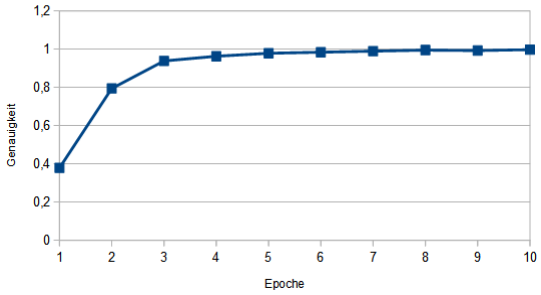

Abbildung 17: Genauigkeit beim einfachen Netz

5 Auswertung der Bilder

Das trainierte Netz wird auf dem Raspberry Pi geladen und der Videoplayer wird gestartet. Solange das Video läuft werden Bilder aufgenommen. Die Bilder werden zur Klassifikation an das Model übergeben. Je nach erkannter Geste wird dann der Videoplayer gesteuert.

Durch den Coral TPU Stick wird die Inferenzzeit dabei deutlich verbessert, sodass 20 bis 25 Bilder pro Sekunde verarbeitet werden können statt 5 bis 6 ohne die TPU.

Als Videoplayer wird der OMXPlayer verwendet, da dieser wenig Ressourcen benötigt und den Vorteil gegenüber zum Beispiel dem VLC player hat, den ganzen Bildschirm einzunehmen. Dadurch entfällt ein gewohntes Interface zum Steuern per Maus. Der OMXPlayer bietet verschiedene Steuerungsmöglichkeiten über Tastatureingabe, so kann zum Beispiel mit der Taste “p” pausiert werden [7]. Auf dem Raspberry Pi wird der OMXPlayer in einem Python Skript mittels des Python Moduls “Subprocess” gestartet. Um den OMXPlayer per Geste zu steuern, wird in dem Python Skript auf die Standard Eingabe des OMXPlayers geschrieben. Die Gesten werden also direkt auf die Tastaturkommandos übertragen.

Bei der Gestensteuerung gibt es zwei Zustände, um mit den drei Gesten Hand, Daumen hoch und Daumen runter möglichst viele Funktionen zu bieten. Im “play” Zustand wird das Video wiedergegeben und mit Daumen hoch und Daumen runter kann die Lautstärke verändert werden. Mit der flachen Hand kommt man dann in den “pause” Zustand und das Video wird pausiert. Hier kann mit dem Daumen hoch wieder in den “play” Zustand gewechselt werden und mit Daumen runter kann das Programm beendet werden. Mit mehr Gesten könnte dann noch mehr Funktionalität, wie Veränderung der Abspielgeschwindigkeit, eingebaut werden.

6 Ausblick

Alternativ zum OMXPlayer eignet sich der VLC Player, der mittels D-Bus gesteuert werden kann [3]. Die Steuerung über D-Bus ist dabei deutlich eleganter und mächtiger als über die Standardeingabe.

Die externe TPU wird auf dem verwendeten Raspberry Pi 4 Model B mit einem Gigabyte RAM für diese Anwendung nicht benötigt, da ohne TPU 5 bis 6 Gesten pro Sekunde klassifiziert werden können. Dies ist für eine Videoplayer Steuerung mehr als ausreichend.

Abschließend zeigt sich, dass die gewählte Form der Gestenerkennung kein sehr komplexes neuronales Netz erfordert. Schon mit sehr geringer Trainingszeit werden Genauigkeiten von über 99% erreicht und das obwohl keine Bildvorverarbeitung stattfindet.

Zu prüfen ist, ob ein neuronales Netz überhaupt notwendig ist um die verwendeten einfachen Gesten zu unterscheiden oder ob man auch zum Beispiel mit Nächste-Nachbarn-Klassifikation gute Ergebnisse erzielt. Des Weiteren muss getestet werden, ob auch komplexere Gesten mit den hier gezeigten Netzen zuverlässig klassifiziert werden können.

Literatur

[1] *A Comprehensive Guide to Convolutional Neural Networks*. https://jaai.de/convolutional-neural-networks-cnn-aufbau-funktion-und-anwendungsgebiete-1691/. 2014.

[2] *A Comprehensive Guide to Convolutional Neural Networks*. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. eingesehen am 07.07.2020.

[3] E. Joy u. a. "Gesture Controlled Video Player A Non-Tangible Approach to Develop a Video Player Based on Human Hand Gestures Using Convolutional Neural Networks". In: *2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS)*. 2018, S. 56–60.

[4] *Keras API - categorical crossentropy*. https://keras.io/api/losses/probabilistic_losses/. eingesehen am 10.07.2020.

[5] *Keras API - Dropout layer*. https://keras.io/api/layers/regularization_layers/dropout/. eingesehen am 05.07.2020.

[6] Diederik P. Kingma und Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. arXiv: 1412.6980 [cs.LG].

[7] *OMXPlayer: An accelerated command line media player*. https://www.raspberrypi.org/documentation/raspbian/applications/omxplayer.md. eingesehen am 03.07.2020.

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: Münstedt

Vorname: Michael

dass ich die vorliegende Hausarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Einfache Gestensteuerung für einen Videoplayer auf dem Raspberry Pi

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Hamburg	10.07.2020	M. Münstedt
Ort	Datum	Unterschrift im Original