Gestenbasierte Steuerung von Philips Hue Lampen

Steffen Kunz

Steffen.Kunz@haw-hamburg.de

25. Februar 2021

Zusammenfassung

Mittels Gesten lassen sich die Lampen wesentlich besser steuern als über eine App oder Sprachsteuerung. Das größte Problem ist der Aufwand die Lampen zu steuern. Entweder über die App oder über Sprachsteuerung mittels Alexa. Diese hat allerdings häufig Verbindungsprobleme bzw. erkennt die Lampen nicht. Daher wurde eine gestenbasierte Steuerung von Philips Hue Lampen realisiert.

1 Einleitung

Das Ziel ist es eine gestenbasierte Steuerung von Philips Hue Lampen zu realisieren. Dabei können diese Lampen auch von einer App oder einem Gadget, wie Alexa, gesteuert werden, allerdings umständlicher. Daher ist die Motivation eine intuitive Steuerung der Lampen zu entwerfen, die jederzeit funktioniert. Dies ist eine große Abgrenzung zur Sprachsteuerung über Alexa, da diese häufig Verbindungsschwierigkeiten hat und für den internen Zugriff auf die Lampen eine externe Kommunikation benötigt.
Dabei ist das typische Szenario, dass jemand auf dem Sofa sitzt und den Arm hebt und anschließend nach links oder rechts streckt. Daraufhin soll das Licht ein- bzw. ausgeschaltet werden.
Dafür hat der Raspberry Pi eine Nachtsichtkamera, welche auf das Sofa gerichtet ist, wie in Abbildung 1 zu sehen und mittels eines neuronalen Netzes die Gesten erkennt und das Licht entsprechend steuert.
Hier kommt ein selbst lernendes Netz zum Einsatz, welches in der Lage ist, Regeln für das Lösen bestimmten Aufgaben selbst zu erlernen und in großen Daten Zusammenhänge und Muster zu erkennen.
Dafür soll es drei Gesten geben: zum einen den Arm nach oben gestreckt und zum anderen zu den jeweiligen Seiten, exemplarisch in Abbildung 3 zu erkennen.

Da es allerdings auch vorkommen kann, dass keine Geste gezeigt wird, dient die Arm nach oben strecken Geste als Aktivierung.

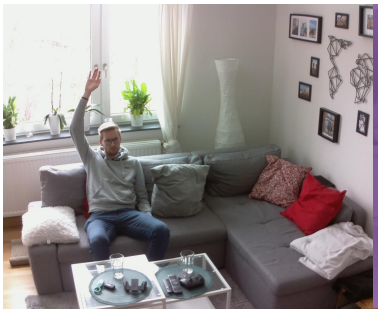

Abbildung 1: Tag Aktivierung

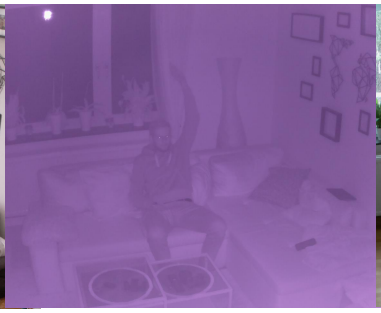

Abbildung 2: Nacht Aktivierung

Abbildung 3: Tag Ausschalten

2 Aufbau und Umsetzung

Um weniger Probleme mit den Lichtverhältnissen zu bekommen, wurde der Raspberry Pi 3 B+ auf einen Schrank in 2 Meter Höhe gestellt und mit einer Electreeks® Raspberry Pi Kamera Modul mit Automatik Infrarot-Sperrfilter – Full HD mit Infrarot LEDs ausgestattet. Diese ermöglicht Aufnahmen bei Nacht, wie in Abbildung 2 zu sehen ist. Um die verschiedenen Gesten zu erkennen, wird ein Bild-Klassifikator verwendet. Da die Inferenz Zeit des Raspberry Pi bereits genügt, wird von der Verwendung weiterer Beschleunigungs-Hardware abgesehen. Dies wäre z.B. möglich mit einem Google Coral Stick.
Weitere Standorte des Raspberry Pis wären denkbar, allerdings sind dort zu große Störfaktoren durch ein großes Fenster aufgetreten, wie in Abbildung 4 oder Abbildung 5 zu sehen ist.

Abbildung 4: Winkel unten Tag

Abbildung 5: Winkel unten Nacht

Zum weiteren Aufbau gehört eine Philips Bridge V2, sowie TRÅDFRI LED-Leuchtmittel von IKEA. Diese werden über die Bridge gesteuert, welche vom Raspberry Pi via REST-Schnittstelle Anfragen bewerkstelligt. Die Leuchtmittel dienen lediglich zur Demonstration der erkannten Gesten.
Der Klassifikator soll grundlegend erkennen, ob die Lampen ein- oder ausgeschaltet werden sollen.
Um diese Funktionalität zu erreichen wurde ein Convolutional Neural Network, kurz CNN, verwendet. Abbildung 6 zeigt den Aufbau eines CNNs.

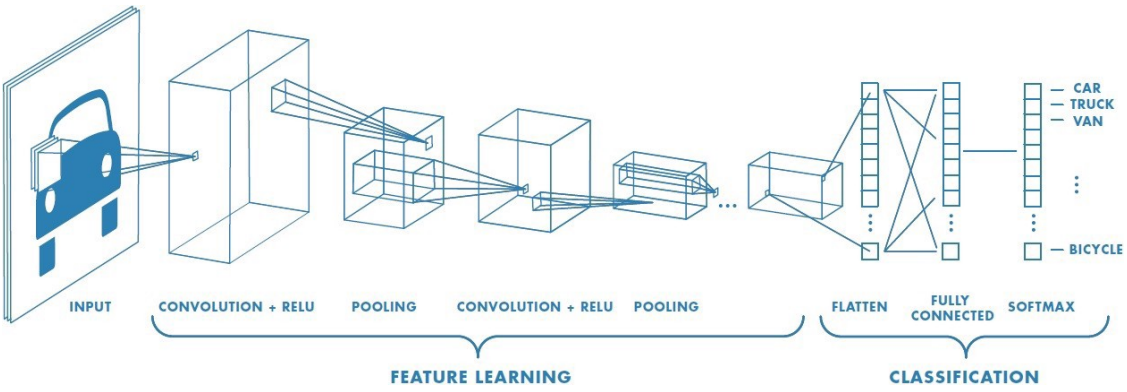

Abbildung 6: Aufbau Convolutional Neural Network [CNN]

2.1 Raspberry Pi Aufbau

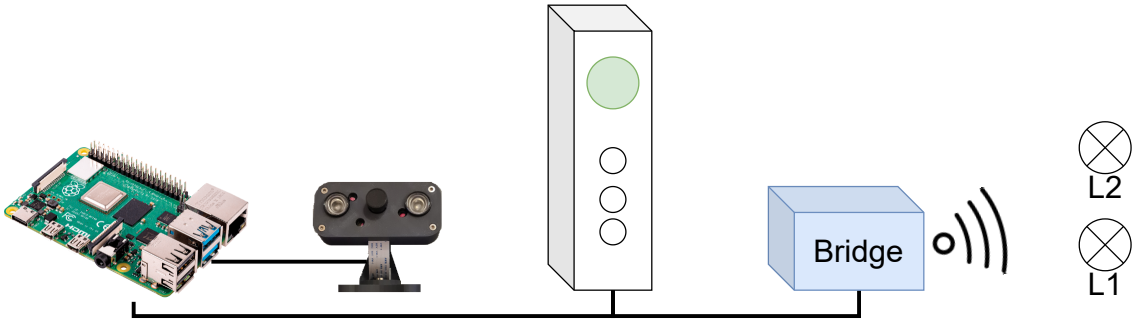

Abbildung 7: Raspberry Pi Aufbau

In Abbildung 7 ist der Aufbau des Raspberry Pis mit Kamera, sowie der Philips Bridge gezeigt. Dabei ist die Kamera mit einem Flachbandkabel an den Raspberry Pi angeschlossen und dieser ist mit einem Ethernet-Kabel an den Router. Die Bridge ist ebenfalls mit einem Ethernet-Kabel an den Router angeschlossen und kommuniziert per Funk

(ZigBee) mit den einzelnen Lampen (Hier L1 und L2). Dies ist ein gewöhnlicher Aufbau, da die Bridge ausschließlich mit Ethernet-Kabel verbunden werden kann und überall anwendbar je nach Klassifizierung des Models.

2.2 Trainings- und Testdaten

Es gibt verschiedene Arten und Wege um Trainingsdaten zu beschaffen. Ein sehr einfacher und schneller Weg an viele Trainingsdaten heranzukommen ist der, dass bereits gelabelte Trainingsdaten verwendet werden. Da diese Trainingsdaten aber auf den einen Raum spezialisiert sein sollen, in dem das System verwendet wird, können hier keine vorgelabelten Trainingsdaten eingesetzt werden.
Daher wurde mittels eines Python-Skripts auf die Kamera des Raspberry Pis zugegriffen und Bilder gemacht.
Pro Geste wurden ca. 500 Bilder gemacht, die anschließend selbst gelabelt werden müssen, also klassifiziert. Beim Erstellen der Bilder wird die Geste leicht bewegt, so dass nicht nur die perfekte Ausführung zum Erfolg führt. Zudem werden die Belichtungseinstellungen verändert, um verschiedene Helligkeiten und Unschärfen zu simulieren und somit für später bessere Ergebnisse zu erzielen.
Von jeder Geste werden dann 20% der Bilder zufällig als Testdaten ausgewählt und 80% als Trainingsdaten verwendet.
Um eine bessere Genauigkeit zu erzielen, wurden die Trainingsdaten an verschiedenen Tagen zu unterschiedlichen Tageszeiten aufgezeichnet. Dies führte von einer 90% Genauigkeit zu einer 99% Genauigkeit.

2.3 Lampensteuerung

Code-Snippet 1 zeigt die gekapselte Implementierung der Lampensteuerung.
Dafür wird die Python Libary 'phue'([phu]) verwendet. Diese kommuniziert über die REST-Schnittstelle der Bridge und liefert nützliche Methoden. Das in Zeile 8 auskommentierte 'self.b.connect()' muss nur einmal initial ausgeführt werden, damit das neue Endgerät und die Bridge einen 'Handshake' durchführen können. Anschließend bleibt die Funktion auskommentiert.
In Zeile 10 wird das Wohnzimmer definiert, da es bei der Anwendung aufwändig ist, ständig 'self.b.set_light(3, 'on', True/False) auszuführen, wurde dafür eine Methode in der gekapselten 'phuecontrol.py' erstellt. Die Zahl als erster Parameter ist die ID, die die Bridge der dazugehörigen Lampe einmalig vergibt. In diesem Fall ist Lampe 3 und 4 im Wohnzimmer. Hingegen Lampe 1 und 2 im Schlafzimmer wären. Der zweite Parameter ist der Status der Lampe, dieser heißt 'on' und kann True oder False sein. In diesem Fall ist die dazugehörige Lampe entweder ein- oder ausgeschaltet.

```
1 from phue import Bridge
2 import logging
3
4 class pHue:
```

```
5     def __init__(self, ip):
6         logging.basicConfig()
7         self.b = Bridge(ip)
8         #self.b.connect()
9
10    def livingroom(self, State):
11        # Change the light state
12        self.b.set_light(3, 'on', State)
13        self.b.set_light(4, 'on', State)
```

Code-Snippet 1: phuecontrol.py

2.4 Netz-Architektur

Code-Snippet 2 zeigt ein einfaches Convolutional Network, welches aus einer einfachen LeNet-Architektur besteht. Diese umfasst Blöcke mit Convolution, Activation und Max-Pooling gefolgt von einer oder mehreren Dense layers. Diese Architektur funktioniert gut für den Einsatz im Raspberry Pi 3, da sie recht klein ist und etwa 10 FPS ermöglicht.

```
1     model = Sequential()
2     model.add(Conv2D(32, kernel_size=(3, 3),
3         activation='elu',
4         input_shape=(128, 128, 3)))
5     model.add(MaxPooling2D(pool_size=(2, 2)))
6
7     model.add(Conv2D(32, kernel_size=(3, 3),
8         activation='elu'))
9     model.add(MaxPooling2D(pool_size=(2, 2)))
10
11    model.add(Conv2D(64, kernel_size=(3, 3),
12        activation='elu'))
13    model.add(MaxPooling2D(pool_size=(2, 2)))
14
15    model.add(Flatten())
16    model.add(Dense(64,
17        activation='elu'))
18    model.add(Dropout(0.5)) # reduziert Overfitting
19    model.add(Dense(16,
20        activation='softmax'))
21
22    model.compile(loss='categorical_crossentropy',
23                  optimizer='adam',
24                  metrics=['accuracy'])
25
26    model.summary() # shows NN
```

Code-Snippet 2: Netz-Architektur

Das Netz besteht aus drei Convolutional-Layern und zwei Dense Layer. Als Aktivierungsfunktion wurde 'elu' verwendet und im Output Layer die 'Softmax' Aktivierungsfunktion. Die ersten beiden Convolutional-Layer bestehen aus jeweils 32 Neuronen (Z. 2,9) und haben einen 3*3 Kernel. Auf jedem Convolutional-Layer folgt ein Pooling-Layer, um die Dimensionen zu reduzieren und somit die Daten zu konzentrieren (Z. 5,9,13).

Damit die Daten in einem eindimensionalen Output umgewandelt werden können, wird zunächst ein Flatten-Layer verwendet (Z. 15), dieser wandelt die Feature-Maps in einen eindimensionalem Vektor um. Somit können die folgenden Dense Layer die Daten verarbeiten(Z. 16f). Nach dem ersten Dense Layer kommt ein Dropout-Layer zum Einsatz, um 'Overtting' zu verhindern.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_19 (Conv2D)           (None, 126, 126, 32)      896
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 63, 63, 32)        0
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 61, 61, 32)        9248
_________________________________________________________________
max_pooling2d_19 (MaxPooling (None, 30, 30, 32)        0
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 28, 28, 64)        18496
_________________________________________________________________
max_pooling2d_20 (MaxPooling (None, 14, 14, 64)        0
_________________________________________________________________
flatten_6 (Flatten)          (None, 12544)             0
_________________________________________________________________
dense_10 (Dense)             (None, 64)                802880
_________________________________________________________________
dropout_5 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_11 (Dense)             (None, 16)                1040
=================================================================
Total params: 832,560
Trainable params: 832,560
Non-trainable params: 0
_________________________________________________________________
```

Abbildung 8: Model Zusammenfassung [ksm]

Abbildung 8 zeigt die Zusammenfassung des Models. Dabei besteht dieses aus 832.560 zu berechnenden Parametern und 208 Neuronen.

2.5 Testaufbau

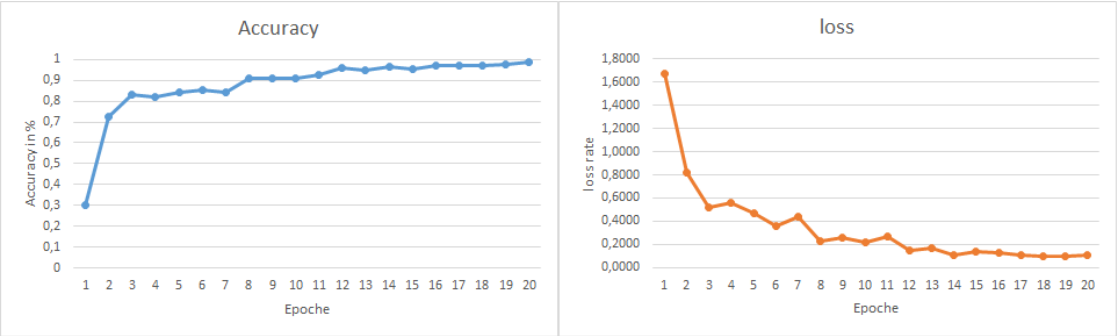

Abbildung 9: Accuracy/Loss

Abbildung 9 zeigt im linken Teil die Genauigkeit des Netzes und im rechten Teil die 'loss rate'.
Dabei ist bei der Genauigkeit (Accuracy) zu sehen, dass diese bereits nach ungefähr 14 Epochen bei 98% ist und dabei die 'loss rate' bei 10% liegt.
Allgemein gilt, dass je kleiner die 'loss rate' ist, desto besser und genauer ist ein Model. Da die Abbildung 9 nur die ersten 20 Epochen zeigt und das Training über 100 Epochen auf dem Raspberry Pi ausgeführt wurde, ist das erstellte Model am Ende sehr genau und gut trainiert. Am Ende schaffte das Model eine Genauigkeit von 99% und eine 'loss rate' von 0,05.
Jede Epoche hat ungefähr 87 Sekunden gedauert und somit ergibt sich eine Gesamt Trainingszeit von:

$$\sim 87 * 100 \;=\; \sim 8700 \text{ Sekunden} \;=\; \sim 145 \text{ Minuten}$$

2.6 Training und Testen

Beim Training des Netzes auf dem Raspberry Pi wurde eine 'Batch-Size' von 16 gewählt, da es besonders effizient bei großen Datenmengen ist.
Um 'Overfitting' zu reduzieren wurde 'EarlyStopping' eingesetzt, wie in Code-Snippet 3 Zeile 10 zu erkennen ist. Zudem wurde nur das Model gespeichert, welches die beste Genauigkeit und die niedrigste 'loss rate' hat (Zeile 9).

```
model.fit_generator(
        train_generator,
        steps_per_epoch=16 // batch_size,
        epochs=100,
        validation_data=val_generator,
        validation_steps=10,
        callbacks=[
            ModelCheckpoint('%s/model.h5' % train_data_dir,
            save_best_only=True),
            EarlyStopping(patience=10),
            ReduceLROnPlateau(factor=0.2, patience=5,verbose=1)
            ]
        )
```

Code-Snippet 3: Training des Netzes

Final getestet wurde das Model mit mehreren Livetests, um zu kontrollieren, ob die erkannten Gesten passen und die Lampen ein- bzw. ausgeschaltet werden.

2.7 Umsetzung

Für die Umsetzung wurde ein einfacher Automat entworfen. Dieser ist in Abbildung 10 zu sehen. Da der zu beobachtende Raum von dem Raspberry Pi permanent observiert wird, wurde eine Aktivierungs-Geste eingeführt. Sobald erkannt wurde, dass ein Arm gehoben wurde, also der Zustand von 'q0' auf 'q1' wechselt, kann das Licht mittels Arm nach rechts bzw. links an-/ausgeschaltet werden. Danach ist das System wieder im Ursprungsstatus, wie am Automaten zu erkennen ist.

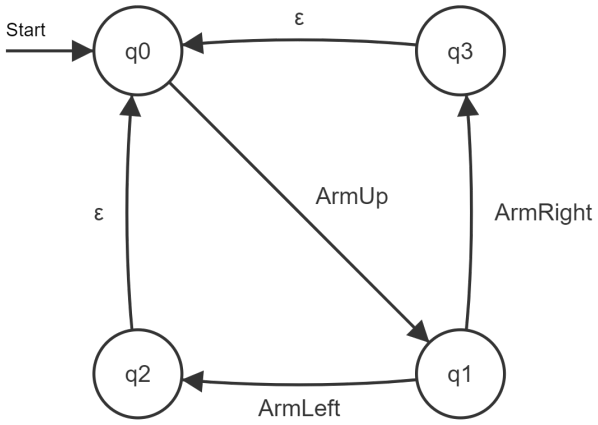

Abbildung 10: Automat Gestenerkennung.

Abbildung 10 zeigt den Automat der Gestenerkennung. Es gibt drei Gesten: ArmUp, ArmLeft und ArmRight. ArmUp ist die Aktivierungs-Geste und ArmLeft bzw. ArmRight steuert das Licht. Dabei muss zuerst der Arm nach oben gestreckt werden und anschließend zu einer Seite. Bei Arm nach links wird das Licht ausgeschaltet und bei Arm nach rechts wird das Licht angeschaltet. Dabei spielt es keine Rolle ob der linke oder Rechte Arm zur Aktivierung nach oben gestreckt wird.

3 Fazit

Die gestenbasierte Steuerung von Philips Hue Lampen ist erstaunlich einfach und intuitiv. Hinzu kommt eine sehr einfache Bedienung und sogar Personen außerhalb der Test- und Trainingsdaten werden problemlos erkannt und können das Licht steuern.
Durch die Aktivierungs-Geste treten plötzlich erkannte Gesten gar nicht erst auf und das System funktioniert wie geplant.
Da sich der Anwendungsbereich auf das Sofa beschränkt, ist es aktuell nicht möglich die Lampen im Kamera-Bereich mittels der Geste zu steuern. Dies wäre allerdings problemlos möglich, indem andere Positionen außerhalb des Sofas zur Aktivierung zu den Trainingsdaten hinzugefügt werden und anschließend gelernt werden.
Abschließend lässt sich festhalten, dass die Wahl einer Bild-Klassifizierung für das Steuern von Philips Hue Lampen sehr gut eignet ist. Hier wären auch andere Smart-Gagdets denkbar, wie Funksteckdosen etc. Da es sich dabei lediglich um die Implementierung der Umsetzung in der Verarbeitung der erkannten Geste handelt und diese austauschbar ist.

4 Ausblick

Da dieses Projekt mit Tensorflow 2 umgesetzt wurde und damit etwa 1 FPS erreicht, wären zwei denkbare Erweiterungen:

1. TensorFlow 1 verwenden, um die Performance zu verbessern. Also mehr FPS als bisher zu erzielen, um eine schnellere Erkennung zu gewährleisten.
2. den Google Coral Stick verwenden, um die Berechnungen auf der externen TPU durchzuführen und damit ebenfalls mehr FPS zu erreichen und somit eine bessere Performance.

Des Weiteren wäre eine mögliche Verbesserung mehr Gesten zu klassifizieren für andere Optionen. So wäre denkbar, dass eine weitere Geste das Licht dimmen könnte oder einen anderen Farbton einschaltet. Dies unterstützt die 'phue'-Libary bereits und müsste nur über eine neu gelernte Geste angewendet werden.

Zuletzt könnte das Model um eine automatische 'Person betritt den Raum'-Geste erweitert werden, die zwischen 0 Uhr und 6 Uhr morgens, bzw. frei konfigurierbare Zeiträume, das Licht automatisch auf 10% gedimmt einschaltet. Dies würde dann wie in Abbildung 11 gezeigt eine weitere Klasse in dem Model bedeuten, die dann nur zeitgesteuert funktioniert.

Abbildung 11: Automatische Personen Erkennung und gedimmtes Licht eingeschaltet.

Literatur

[CNN] CNN. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional/neural-networks-the-eli5-way-3bd2b1164a53, . – aufgerufen: 17.01.2021

[ksm] The sequential model. https://keras.io/guides/sequential_model/, . – aufgerufen: 10.12.2020

[nnv] C231n Convolutional Neural Networks for Visual Recognition. https://cs231n.github.io/convolutional-networks/, . – aufgerufen: 05.12.2020

[phd] Philips Hue Developer. https://developers.meethue.com/, . – aufgerufen: 19.11.2020

[phu] phue Libary. https://github.com/studioimaginaire/phue, . – aufgerufen: 19.11.2020

[rpi] Raspberry Pi Kamera V2 Nightvision. https://electreeks.de/shop/raspberry-pi-kamera-nachtsicht-mit-infrarot-sperrfilter-full-hd-v2/, . – aufgerufen: 04.12.2020

[tfl] TensorFlow Lite für Raspberry Pi. https://www.tensorflow.org/lite/guide/build_rpi/, . – aufgerufen: 08.12.2020