Machine Learning basierter Helfer für gesundes Arbeiten mit dem Computer

Tobias Westphal
tobias.westphal@haw-hamburg.de
Betreut durch: Prof. Dr. Stefan Pareigis and Maximilian Mang
Department of Computer Science, HAW Hamburg

Abstract In dieser Ausarbeitung wird eine Anwendung beschrieben, die die Vermeidung einer krummen Sitzposition und einer Ablenkung durch das Smartphone verbessert. Des Weiteren strebt die Anwendung eine Verbesserung des regelmäßigen Lüftens und der regelmäßigen Flüssigkeitszunahme beim Arbeiten mit dem Computer an. Das Ziel war es die Anwendung auf einem Raspberry Pi laufen zu lassen, der dabei als Sensor eine Kamera und als Aktor einen Lautsprecher besitzt. Das, für die Erkennung der krummen Sitzposition und des Smartphones, verwendete Convolutional Neuronal Net (CNN) wird dabei auf einem Coral USB Accelerator inferiert. Das CNN, welches 312.712 trainierbare Parameter besitzt, wurde von Grund auf selbst trainiert. Für das Training des Neuronalen Netzes und für das Funktionieren der Anwendung bei vielen verschiedenen Kleidungen ist lediglich ein Bilderdatensatz nötig, der mit nur einer Kleidungsmontur aufgenommen und zu Grauwertbildern konvertiert wurde.

1 Einleitung

Besonders als Informatik-Student verbringt man viel Zeit vor dem Computer. Dabei neigt man schnell dazu, sich zum Bildschirm zu beugen und krumm zu sitzen. Krummes sitzen auf Dauer ist für eine gesunde Körperhaltung nicht förderlich und kann zu Muskel- und Bänderstörungen führen [0]. Gerade/Aufrecht sitzen ist nicht die ultimative Lösung für dieses Problem, sondern Bewegung. Eine aufrechtere Sitzposition sei für eine gesunde Körperhaltung allerdings wichtig und wird durch die richtige Einstellung des Arbeitsplatzes gefördert[1].
In Lernphasen neigt man des Weiteren oft dazu, durch das Schauen auf das Smartphone, zu prokrastinieren. Sofern man dann sehr in ein Thema vertieft ist, ist es persönlich schwer darauf zu achten genügend zu trinken, als auch darauf zu achten für den richtigen Sauerstoff Gehalt im Zimmer zu sorgen und dementsprechend zu lüften.
Im Rahmen des Wahlpflicht Moduls "Embedded Machine Learning" ist zu untersuchen, inwieweit ein mit Kamera ausgestatteter Mikrocontroller in der Lage ist, die genannten Probleme unter der Verwendung von Machine Learning durch Klassifikation zu erkennen. Des weiteren soll somit ein gesünderes Arbeiten mit dem Computer gesteigert werden, indem der Benutzer über akustische Signale über sein Verhalten informiert wird.
Das zu untersuchende System soll die folgenden Anforderungen erfüllen:

1. einen warnen, sofern man krumm sitzt, bis man wieder aufrecht sitzt,
2. einen warnen, sofern man sein Smartphone benutzt,
3. einem Bescheid geben, sofern man nicht genügend getrunken hat und
4. einem Bescheid geben, dass man das Fenster öffnen soll.

2 System

2.1 Verwendete Hardware

1. Raspberry Pi 4 Model B mit 4GByte Arbeitsspeicher [2]
2. Kamera Modul V2 [3]
3. Coral USB Accelerator [4]
4. Aux fähiger Lautsprecher

2.2 Hardware Aufbau des Systems

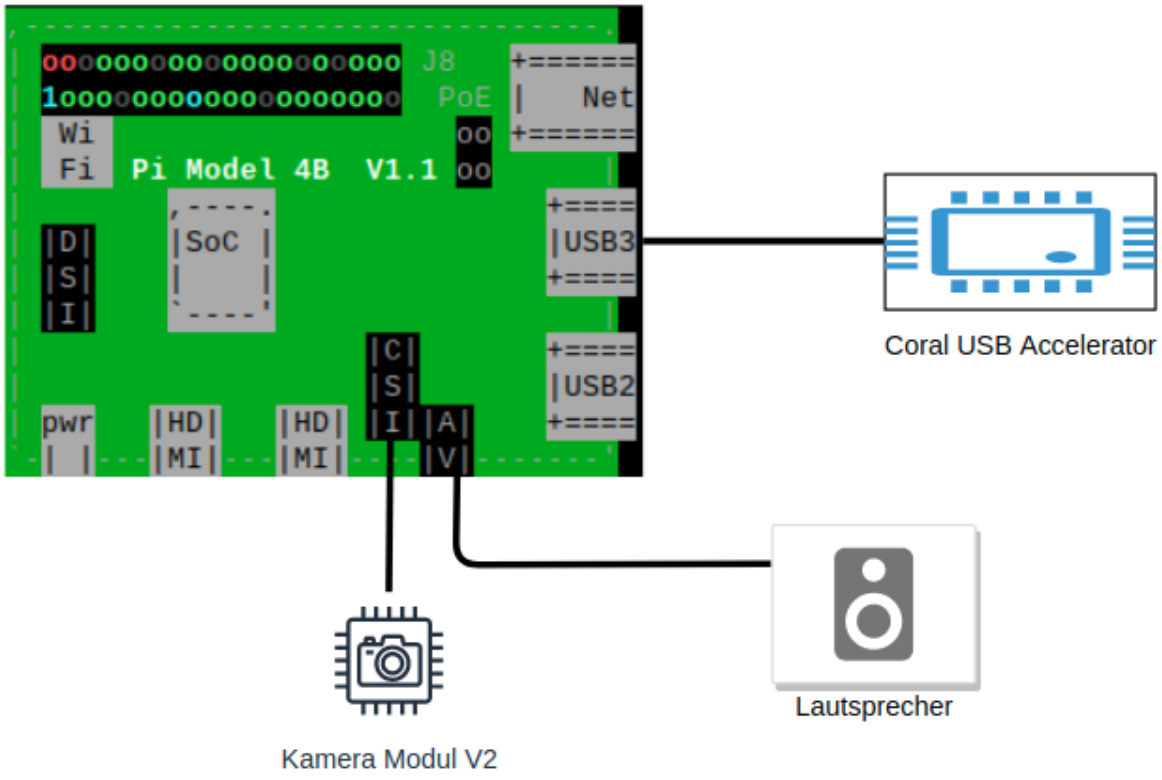


Figure 1: Verbindung der Hardware Komponenten mit dem Raspberry Pi

Sicherlich sollte der Raspberry Pi mit Strom versorgt werden. Des Weiteren um die Software Anwendung auf dem Rasperry Pi OS starten zu können, sollte über der Pi an das Heimnetz über WiFi/Ethernet Kabel angeschlossen werden, oder direkt per HDMI mit Bildschirm und per USB mit Tastatur verbunden werden.

2.3 Software System Aufbau

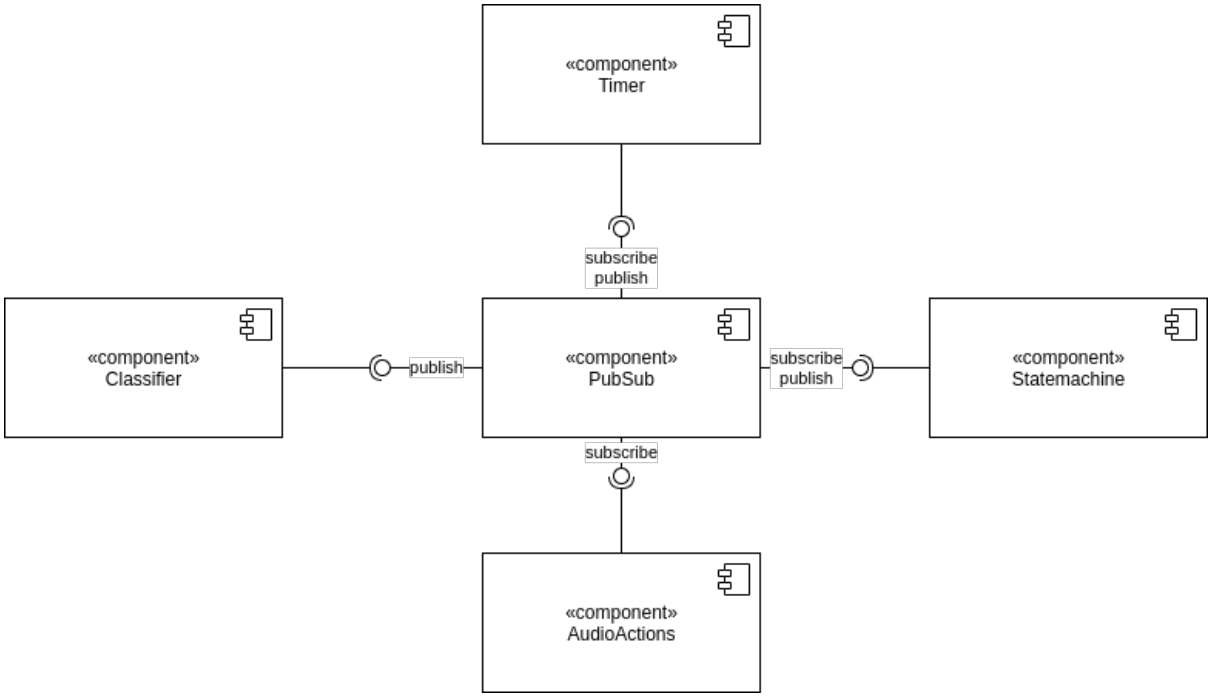


Figure 2: Komponenten Diagramm: System Komponenten

Das hier dargestellte Komponenten Diagramm zeigt den Software Aufbau der Anwendung, die in Python geschrieben ist.

1. **PubSub** - Eine Python API [5], die eine Event-basierte System Architektur ermöglicht. Consumer melden sich bei Topics an. Producer können auf Topics Events/Nachrichten veröffentlichen, die an alle registrierten Consumer gesendet werden. Die Benutzung dieser API hat den Vorteil, dass die Komponenten voneinander entkoppelt sind und diese nebenläufig in verschiedenen Threads ausgeführt werden.

2. **Statemachine** - Beinhaltet das System Verhalten durch die Implementierung eines Zustandsautomaten (siehe S.6). Zur Vereinfachung wurde sich an einer bereits existierenden Library bedient, die das Statepattern implementiert. [6]

3. **Timer** - Eine eigene Timer Klasse, die spezielle anwendungsspezifische Zeitzähler bereitstellt.

4. **AudioActions** - Kümmert sich um den Audio Output der Anwendung. Benutzt Funktionalitäten der PyAudio library [7]. Die Audio Dateien sind Audiospuren

bestimmter Aussagen wie "Please Drink", die mit der Text-To-Speech Funktionalität von Google Übersetzer [8] erzeugt und mit AudaCity [9] aufgenommen wurden.

5. **Classifier** - Liest Bilder ein unter Verwendung der PiCamera Library [10], die auf die, mit dem Raspberry Pi verbundenen, Kamera zugreift und zeitnahe Bilder aufnimmt. Diese Bilder werden jeweils durch ein Convolutional Neuronales Netz klassifiziert. Das Neuronale Netz ist ein TensorFlow Lite Model [11], welches auf dem Coral USB Accelerator inferiert wird. Der USB Accelerator sorgt für kürzere Inferenz Zeiten als das Inferieren auf der CPU des Raspberry Pi's.

2.4 Software System Verhalten

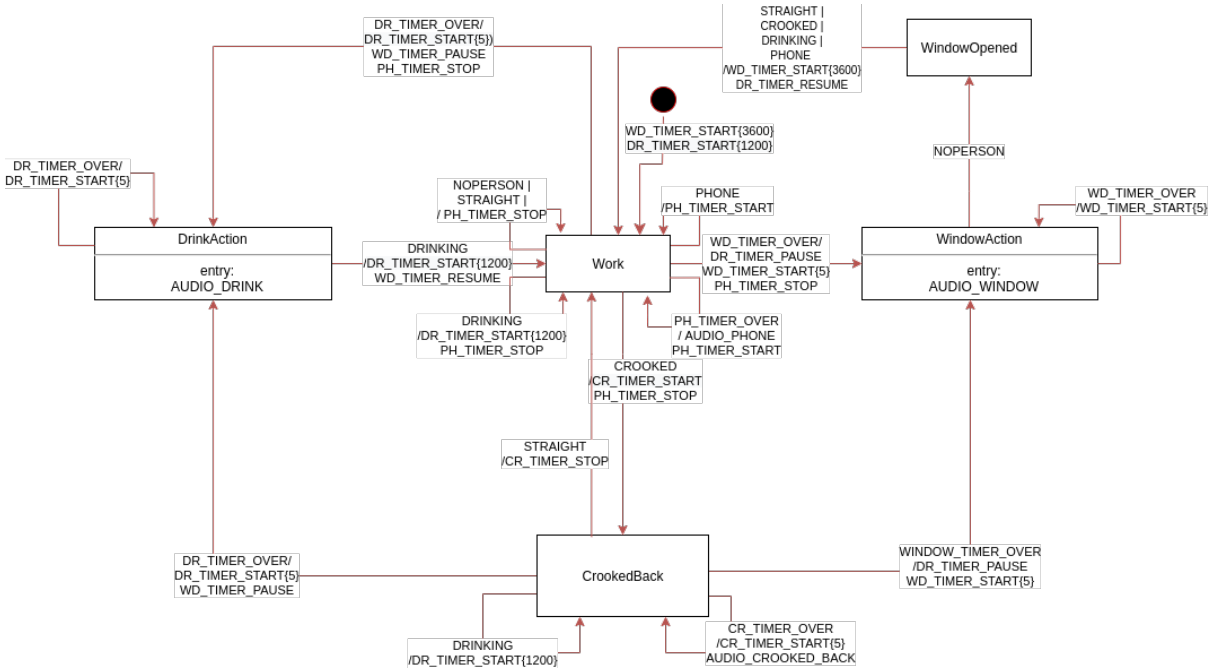


Figure 3: Zustandsautomat der Anwendung. Beinhaltet in der Statemachine Komponente

Zum Verständnis: Startzustand ist "Work". Um in andere Zustände zu wechseln muss die Statemachine Komponente Events erhalten. Die benötigten Events für bestimmte Transitionen in andere Zustände sind auf den Transitions-Pfeilen in Großbuchstaben dargestellt. Die Beschriftungen verfolgen das Schema TRIGGER_EVENT/AUSLÖSENDES_EVENT{PAYLOAD}. Als Payload kann bei auslösenden Timer Events die Zeit mitgegeben werden, in wie vielen Sekunden der jeweilige Timer ablaufen soll. CROOKED, DRINKING, NOPERSON, PHONE, STRAIGHT sind Events, die von der Classifier Komponente published werden. Diese entsprechen den erkannten Klassen. (Siehe S. 10)

Grobes Verhalten aus Benutzer Sicht: Zu Beginn wird ein Timer für die Trink-Erinnerung und für die Fenster-Öffnung- Erinnerung gestartet. Sofern der erste Timer abläuft wird der Benutzer dazu aufgefordert etwas zu trinken, erst sobald er etwas trinkt hört die Aufforderung auf und der Timer wird neugestartet. Das Ablaufen des Timers für die Fenster-Öffnungs-Erinnerung sorgt dafür, dass der Benutzer zyklisch dazu aufgefordert wird aufzustehen und sodass das Fenster geöffnet wird. Erst sofern er dies tut beendet das System die zyklische Aufforderung und der Timer fängt wieder an herunter zu zählen. Sofern erkannt wird, dass der Benutzer krumm sitzt, wird dieser zyklisch dazu aufgefordert

sich wieder gerade hinzusetzen. Erst sofern eine gerade Sitzposition erkannt wird, stoppt die zyklische Erinnerung.
Sofern erkannt wird dass der Benutzer sein Smartphone benutzt, wird dieser daran erinnert, dass die Anwendung dies sieht. Erst sobald der Benutzer sein Smartphone weglegt, stoppt die zyklische Erinnerung.

Event	**Audio-Nachricht**	**auslösender Zustand/ Event**
AUDIO_DRINK	"Please Drink."	DrinkAction
AUDIO_PHONE	"Phone."	Work/ PH_TIMER_OVER
AUDIO_CROOKED_BACK	"Straight back please."	CrookedBack/ CR_TIMER_OVER
AUDIO_WINDOW	"Open the window to get fresh air."	WindowAction

Table 1: Abbildung der AUDIO-Events auf Inhalte der durch die AudioActions-Komponente gestarteten Audio-Dateien

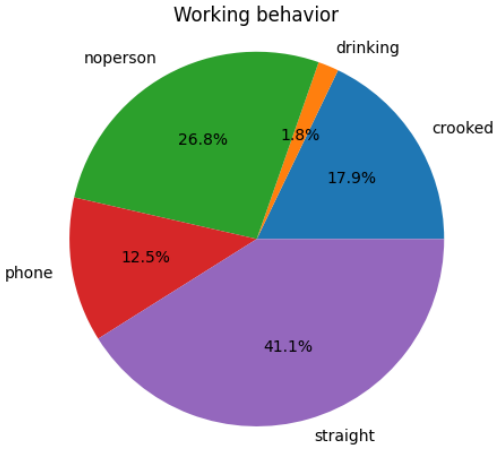


Figure 4: Beispiel einer erzeugte Auswertung der Classifier Kompononente bei Beendigung der Anwendung. Die Prozentzahlen sind abhängig von der Dauer, die die Klassen erkannt wurden.

2.4.1 Laufzeitsicht an zwei Beispielen

Figure 5 und 6 zeigen den Event-basierten Nachrichtenaustausch der System Komponenten über die PubSub Komponente. Es wird gezeigt, dass sich die jeweiligen Komponenten zunächst bei der PubSub Komponente für Topics anmelden.

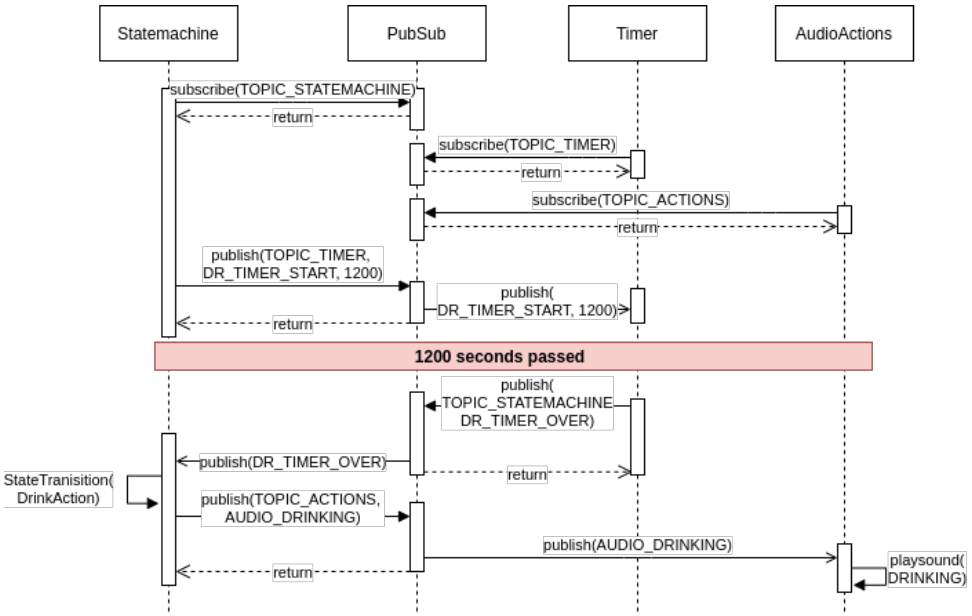


Figure 5: Sequenzendiagramm - Timer für die Trink-Erinnerung läuft ab

Figure 5 stellt dar, dass sofern ein Timer der Timer Komponente abläuft, dass dieser ein entsprechendes Events erzeugt. Dieses Event wird dann von der State Machine Komponente ausgewertet. Im dargestellten Fall wird als Reaktion das AUDIO_DRINK Event ausgelöst und an die AudioActions Komponente weitergeleitet, die über angeschlossene Lautprecher eine Audio-Datei abspielt.

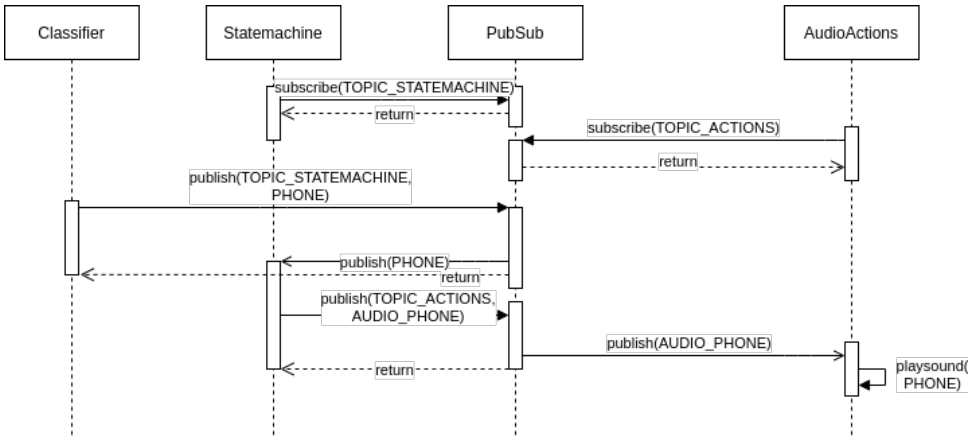


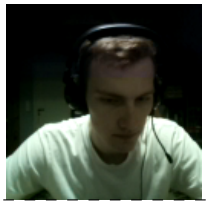
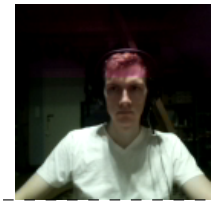
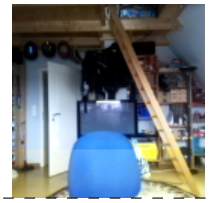
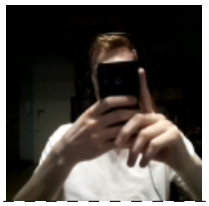
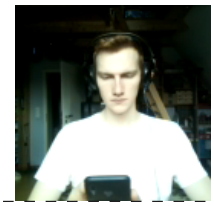
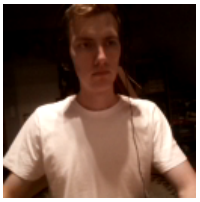
Figure 6: Sequenzendiagramm - Classifier erkennt, dass der Benutzer sein Smartphone benutzt

Figure 6 stellt beispielhaft dar, dass die erkannten Klassen des Classifiers zum Publishen von gleichnamigen Events führt. Die Klassifikations Events sorgen für individuelles Systemverhalten. Hier sorgt das PHONE-Event für das Abspielen der Phone-Audio Datei.

3 Klassifikation mit Machine Learning

Ein Großteil des Projekts wurde sich damit beschäftigt, ein Convolutional Neuronales Netz für die Erkennung der folgenden Klassen zu trainieren.

3.1 Zu erkennende Klassen der Classifier Komponente

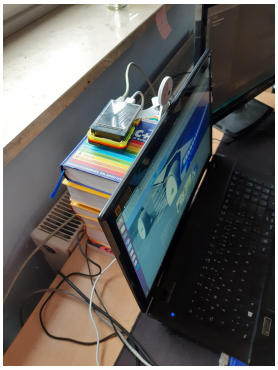
ID	Klassenname	Beschreibung	Beispiel Bilder
0	*Crooked*	*Die vorm Rechner sitzende Person ist nach vorne gebeugt, oder ist sehr weit zurück gelehnt*	 
1	*Drinking*	*Die vorm Rechner sitzende Person trinkt etwas aus einer Flasche*	
2	*NoPerson*	*Es ist keine vorm Rechner sitzende Person zu erkennen, lediglich der Arbeitsraum.*	
3	*Phone*	*Die vorm Rechner sitzende Person benutzt ihr Smartphone so, dass dieses auf Kopf- oder auf Bauchhöhe gehalten wird.*	 
4	*Straight*	*Die vorm Rechner sitzende Person sitzt aufrecht und gerade vor dem Rechner*	

3.2 Wahl der Kamera Position

Zunächst musste die Positionierung der Kamera diskutiert werden. Die Kamera Position sollte so gewählt werden, dass sie die entscheidenden Anforderungen an das System wahrnehmen kann. Zu Beginn des Projekts standen zwei Arten der Lokation für das System zur Auswahl.

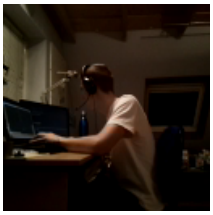
Figure 9: Arten der Lokation des Systems

(a) Positionierung A neben der arbeitenden Person auf einem Karton Turm

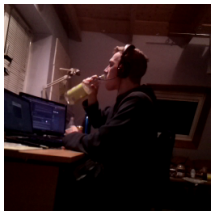


(b) Positionierung B vor der arbeitenden Person auf dem Bildschirm

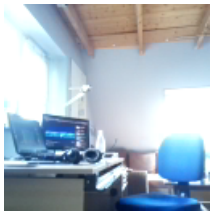
Figure 10: Sicht der Kamera mit Positionierung A bei den zu erkennenden Klassen



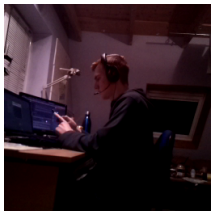
(a) Crooked



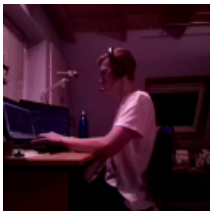
(b) Drinking



(c) NoPerson

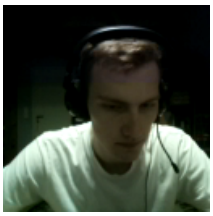


(d) Phone



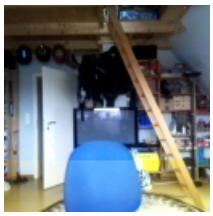
(e) Straight

Figure 11: Sicht der Kamera mit Positionierung B bei den zu erkennenden Klassen

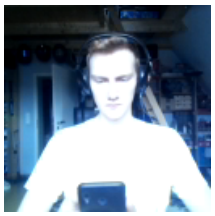


(a) Crooked

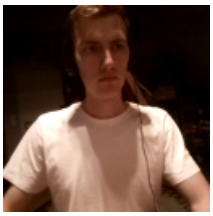
(b) Drinking



(c) NoPerson



(d) Phone



(e) Straight

3.2.0.1 Auswertung Kamera Positionierung: Wohingegen bei Positionierung A, besonders leicht auszumachen ist, ob eine Person krumm oder gerade sitzt, ist es schwer zu erkennen, ob die Person gerade ein Smartphone benutzt(siehe Fig. 10(d)), da dies bei der Position sehr klein erscheint.
Durch Verwendung von Positionierung B ist deutlich zu erkennen, ob die Person ein Smartphone in der Hand hält. Der zu sehende Unterschied zwischen Crooked und Straight ist nicht so deutlich wie bei Positionierung A und könnte je nach Kameraposition und Ausrichtung für Probleme sorgen.

In Bezug auf den Aufwand des Aufbaus der Positionierungen wird deutlich, dass A nicht wirklich praktikabel und platzsparend im Gegensatz zu Position B ist.
Trotz der Verwechslungsgefahr der Klassen bei B wurde sich für diese Positionierung entschieden, da diese einen leichteren Aufbau ermöglicht und gezeigt werden soll, ob Klassifikation mit der Erkennung auch kleinerer Unterschiede zurecht kommt.

3.3 Datensatz

Für die Aufnahme eines eigenen Datensatzes wurde sich an einem Beispiel Python Skript inspiriert [12]. Hier durch konnte schnell über die Konsole pro Klasse ein gelabelter Datensatz aufgenommen werden. Das Labeling ist durch die erstellte Ordnerstruktur gegeben. Ein weiteres Skript erstellt anhand des "data" Ordners ein Validierungs Datensatz, sodass 10% der Bilder in val_data verschoben werden.

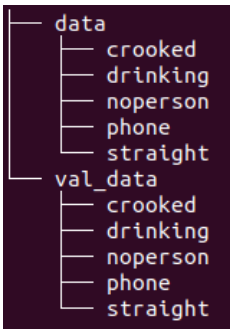


Figure 12: Ordnerstruktur eines Datensatzes

Das Skript für die Aufnahme wurde allerdings dahingehend modifiziert, dass Bilder nicht für eine bestimmte Zeit, sondern eine bestimmte Anzahl an Bilder aufgenommen werden. Sodass gleich große Datensätze aufnehmbar sind.

In den Beispiel Bildern (Fig. 11) bei der Auswahl der Kamera Position ist zu sehen, dass das weiße T-Shirt etwas lila ist. Dies ist eine Farbverschiebung des Aufnahme Skriptes um unterschiedliche Beleuchtungsverhältnisse darzustellen, die im Verlauf des Projektes weiter benutzt wurde. Dies ist also schon ein leichtes augmentieren der Daten (Siehe S. 23).

Für das Verständnis der folgenden Sektionen werden hier nun die benutzten Datensätze definiert.

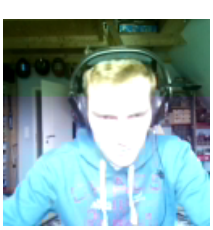
3.3.1 Datensätze für das Taining und zur Evaluierung

Jede aufgenommene Kleidungsfarbe wurde sowohl im hellen, als auch bei Beleuchtung am Abend aufgenommen. Alle Datensätze haben für jede der 5 Klassen eine ähnliche Aufteilung der Bilderanzahl. Bsp.: Train1 hat für jede Klasse ca. 2500 Trainingsbilder.

Name	Kleidungsfarben	Bilder Trainings-set	Bilder Vali-dierungsset	Enthalten in
Train1	grün, weiß, rot, camo, blau, grau, hellgrau	12439	1398	X
Train2	grün	1773	212	Train1
TrainRotG	rot, grün	3566	404	Train1
TrainWeißG	weiß, grün	3521	400	Train1
Eval1	schwarz	X	1996	X
EvalWeiß	rot	X	1748	TrainWeißG, Train1
EvalRot	weiß	X	1792	TrainRotG, Train1

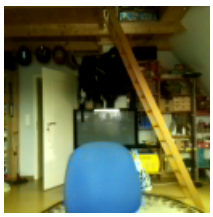
Table 3: Definition der verwendeten Bilder-Datensätze

Figure 13: Beispiel Bilder der Klassen mit verschiedener Kleidung

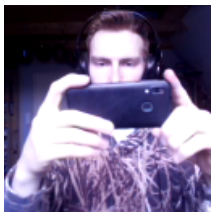


(a) Crooked - Blaue Kleidungsfarbe

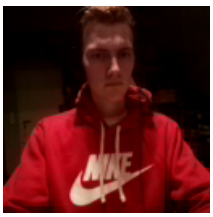
(b) Drinking - Graue Kleidungs-farbe



(c) NoPerson



(d) Phone - Camoflage Klei-dungsfarbe



(e) Straight - rote Kleidungsfarbe

Figure 13 (a), (c) und (d) zeigen, dass das Aufnahme Skript für leichtes verfärben der Bilder sorgt.

3.4 Convolutional Neural Network

Das Convolutional Neuronale Netz (CNN) für die Erkennung der 5 Klassen wird von Grund auf selbst trainiert. Es besteht zunächst aus 3 "Convolutional" Blöcken, in denen die Feature Maps mit den Faltungskernen matrix-multipliziert werden. Der Dropout-Layer (Listing 1, Zeile 11 & 16) sorgt dafür, dass gewisse Faltungskerne in Trainingsschritten mit einer bestimmten Frequenz nicht trainiert werden[13].
Der abschließende "Fully-Connected block" reshaped die, aus dem "Convolutional block 3", Output Feature Maps zu einem 1-dimensionalen Vektor (L. 1, Z. 19). Der Block endet mit einem 5 Neuronen großen Layer. Dieser letzte Layer gibt durch die Softmax-Aktivierungsfunktion einen Outputvektor mit der Größe 5 aus, sodass man eine Wahrscheinlichkeit für jede der 5 verschiedenen Klassen erhält. Als "Optimizer" für die Gewichtskalkulationen wird "Adam", oder auch adaptive momentum estimation verwendet. Für die Fehlerfunktion wird die "categorical_crossentropy" verwendet. (L. 1, Z. 21 & 24)

```
1  model = Sequential()
2
3  # Convolutional block 1
4  model.add(Conv2D(32, kernel_size=(2,2), activation="relu",
5      input_shape=(128, 128, 3)))
6  model.add(MaxPooling2D(pool_size= (2,2)))
7
8  # Convolutional block 2
9  model.add(Conv2D(64, kernel_size=(4,4), activation="relu"))
10 model.add(MaxPooling2D(pool_size= (4,4)))
11 model.add(Dropout(0.5))
12
13 # Convolutional block 3
14 model.add(Conv2D(128, kernel_size=(4,4), activation="relu"))
15 model.add(MaxPooling2D(pool_size= (4,4)))
16 model.add(Dropout(0.5))
17
18 # Fully-Connected block
19 model.add(Flatten())
20 model.add(Dense(128, activation="relu"))
21 model.add(Dense(5, activation="softmax")) # 5 Classes
22
23 model.compile(loss="categorical_crossentropy",
24               optimizer="Adam",
25               metrics=["accuracy"])
```

Listing 1: Netzwerk mit 312,677 trainierbaren Parametern

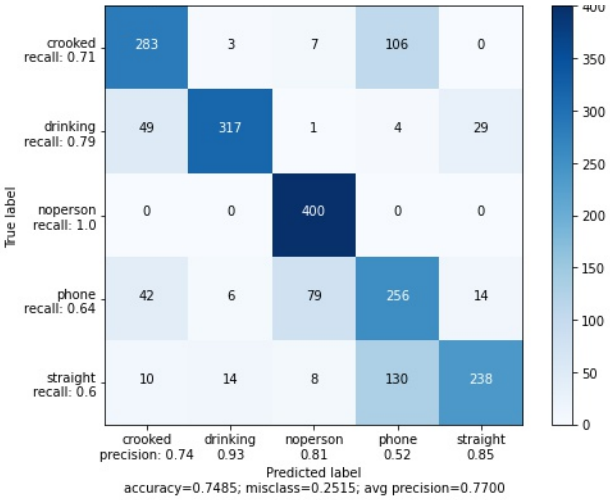
3.5 Evaluierung der Benutzung verschiedener Parameter für das Neuronale Netz

Für den Vergleich der verschiedenen Parametern wurde das Neuronale Netz jeweils maximal 30 Epochen â 50 Steps trainiert. Für das Training wurde hier der Datensatz Train2 verwendet, der für jede Klasse 400 Bilder verwendet. (Vor dem Training werden die Bilder mit dem Augmentation Set 1 modifiziert (Siehe S. 23)).
Für den Vergleich werden die jeweils trainierten Netze mit dem Eval1 Datensatz evaluiert, den sie beim Training nicht zu Gesicht bekommen haben. Zur Darstellung der Evaluierung wird die Confusion Matrix verwendet, die zeigt, wie viele Bilder richtig und falsch klassifiziert wurden. Die in der Confusion Matrix zu sehende Accuracy (links unter der Matrix) wird berechnet, indem die Summe aller richtig erkannten Bilder durch die Summe aller Bilder geteilt werden. Die Accuracy ist Hauptmetrik für den Vergleich und ist gleich dem durchnittlichen Recall. Die richtig erkannten Bilder spannen die Kästchen-Diagonale von oben links nach unten rechts der Matrix.
Figure 14, 17 und 19 sind gleich, da dies die Grundeinstellung ist, die im Code Aufbau des Neuronalen Netzes in Listing 1 gezeigt wird. Für die einzelnen Parameter wird immer von diesem Netz ausgegangen und lediglich der einzelne Parameter ausgetauscht.

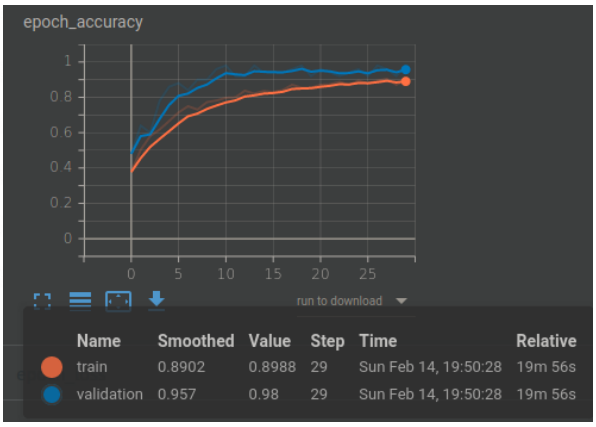
3.5.1 Vergleich Fehlerfunktionen

Für die Fehlerfunktion wurde ausgehend von Listing 1 Zeile 23 augetauscht.

Figure 14: Evaluierung und Trainingsverlauf bei Benutzung der **categorical_crossentropy** Fehlerfunktion

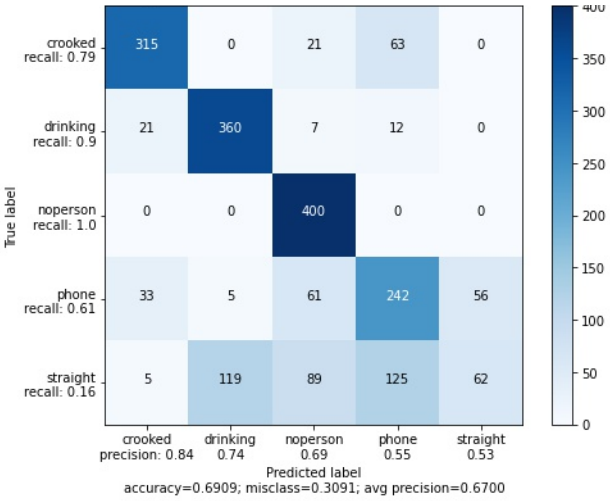


(a) Confusion Matrix auf Eval1

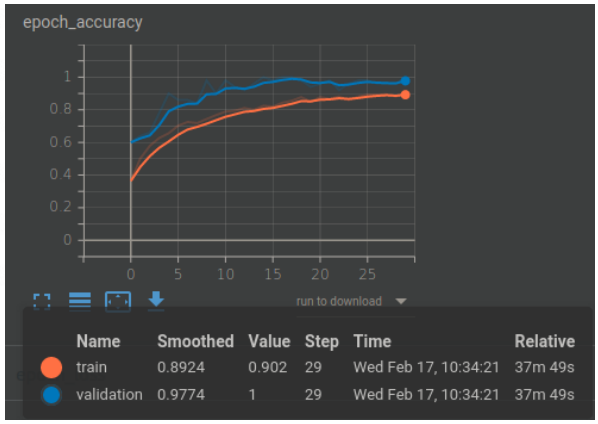


(b) Acurracy der Trainings- und Validierungsdaten

Figure 15: Evaluierung und Trainingsverlauf bei Benutzung der **kullback_leibler_divergence** Fehlerfunktion

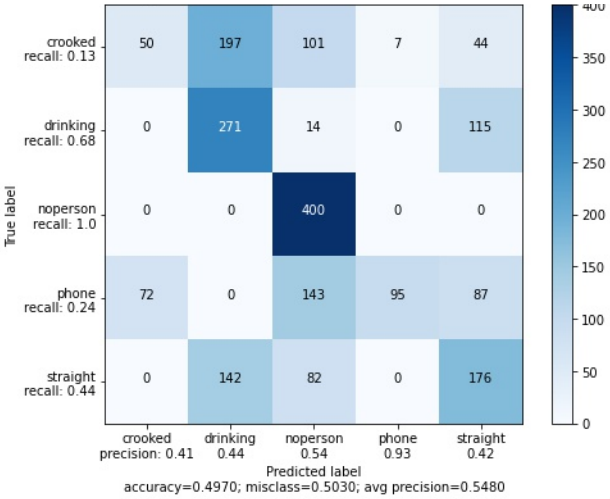


(a) Confusion Matrix auf Eval1

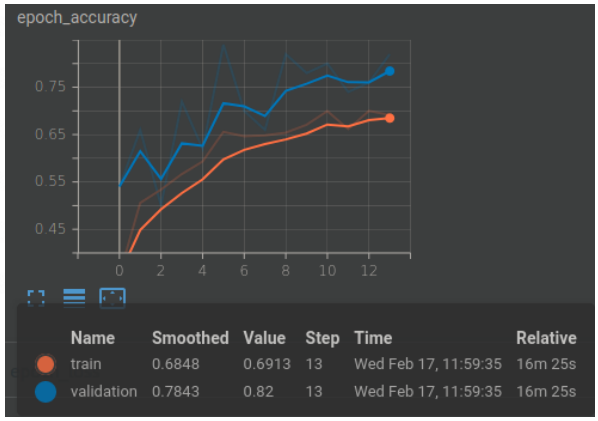


(b) Acurracy der Trainings- und Validierungsdaten

Figure 16: Evaluierung und Trainingsverlauf bei Benutzung der **categorical_hinge** Fehlerfunktion



(a) Confusion Matrix auf Eval1



(b) Acurracy der Trainings- und Validierungsdaten (Durch Early Stopping früher gestoppt)

Wohingegen das Training mit Categorical_Hinge Fehlerfunktion aufgrund von nicht weiter steigender loss_accuracy (nicht dargestellt) abbricht und demnach sehr schlecht bei der Evaluierung abschneidet, unterscheiden sich die Categorical_Crossentropy und Kullback_Leibler_Divergence Fehlerfunktionen kaum. In dem hier gezeigten Beispiel schneidet die Categorical_CrossEntropy Funktion von der Accuracy der Evaluierung

höher ab und ordnet pro Klasse die meisten Bilden dem richtigen label zu. Die Kullback_leibler_divergence Funktion besitzt zwar eine etwas geringere Accuracy der Evaluierungsdaten, hat im Gegensatz zur Crossentropy Fehlerfunktion allein Probleme mit der Klasse "straight". Laut dieser Referenz [14] sind die beiden Fehlerfunktionen funktional equivalent. Demnach wurde sich für die Categorical_Crossentropy Fehlerfunktion entschieden.

3.5.2 Vergleich Aktivierungsfunktionen

Für den Vergleich der Aktivierungsfunktionen, werden alle Aktivierungsfunktionen in Listing 1, also Zeile 4, 9 und 14 außer der "softmax" Funktion in Zeile 21 ausgetauscht.

Figure 17: Darstellung der ReLU Aktivierungsfunktion im Vergleich zur Swish Aktivierungsfunktion

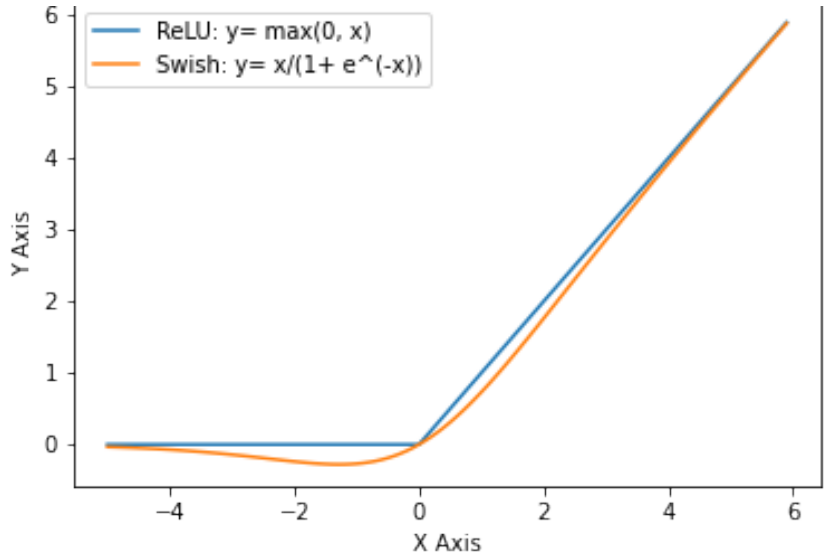
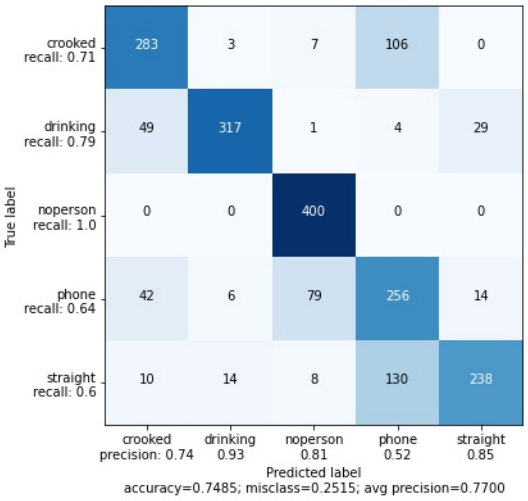
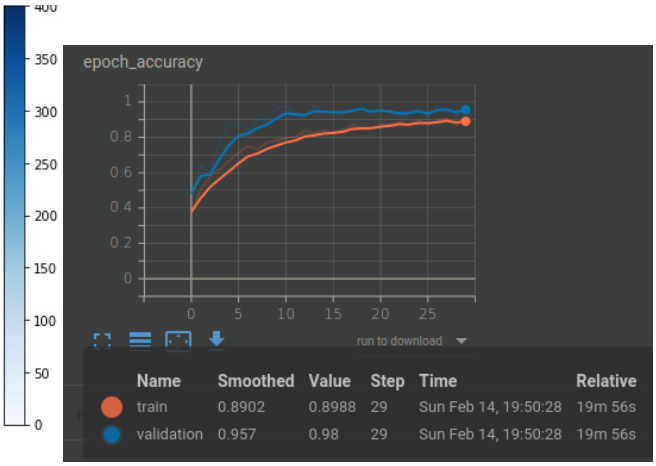


Figure 18: Evaluierung und Trainingsverlauf bei Benutzung der Rectified Linear Unit (ReLU) Aktivierungsfunktion

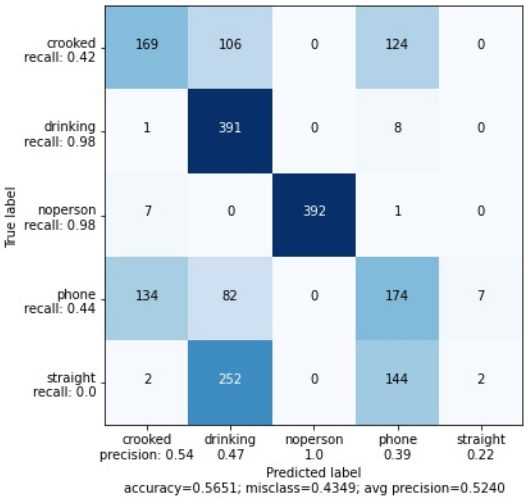


(a) Confusion Matrix auf Eval1

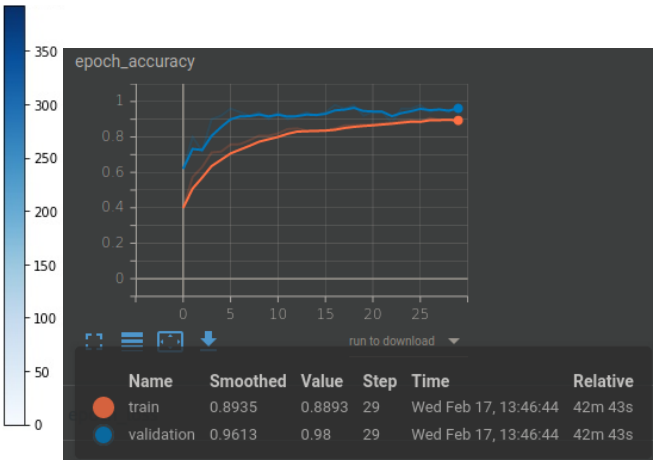


(b) Acurracy der Trainings- und Validierungsdaten

Figure 19: Evaluierung und Trainingsverlauf bei Benutzung der Swish Aktivierungsfunktion



(a) Confusion Matrix auf Eval1



(b) Acurracy der Trainings- und Validierungsdaten

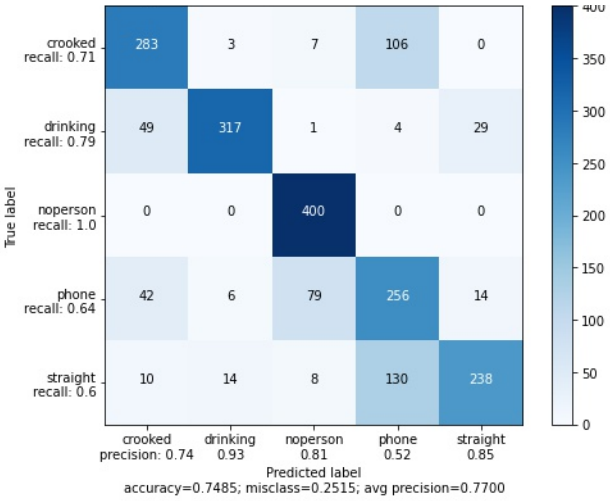
Wohingegen in Referenz [15] davon gesprochen wird, dass die von Google Forschern entdeckte Swish Aktivierungsfunktion bei der Klassifikation auf ImageNet Datensatz eine um 0.9% höhrere Accuracy als die ReLU Funktion erzielt, schneidet das Netz auf den Evaluationsdaten 20% schlechter ab. Trotzdessen sind die Accuracy Werte der Trainings-/ und Validierungsdaten nach 30 Epochen, hier jeweils (b) der Figures 18 und 19, sehr

ähnlich. Die Accuracy Kurve der Swish-Funktion scheint allerdings etwas schneller zu steigen. Demnach wird im Projekt weiter die ReLU Aktivierungsfunktion verwendet.

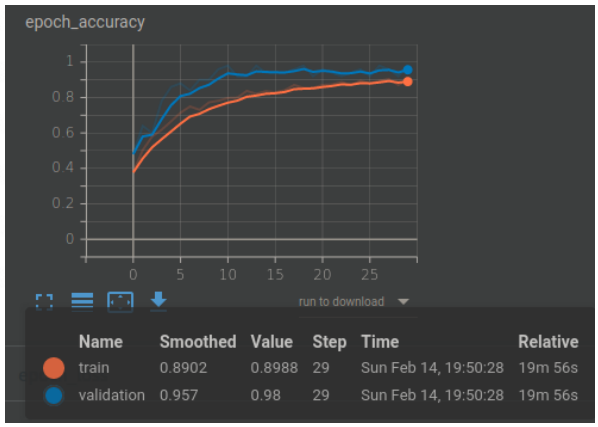
3.5.3 Vergleich Optimierer

Für den Vergleich der Optimierer wurde in Listing 1 Zeile 24 ausgetauscht.

Figure 20: Evaluierung und Trainingsverlauf bei Benutzung des Adam Optimierers

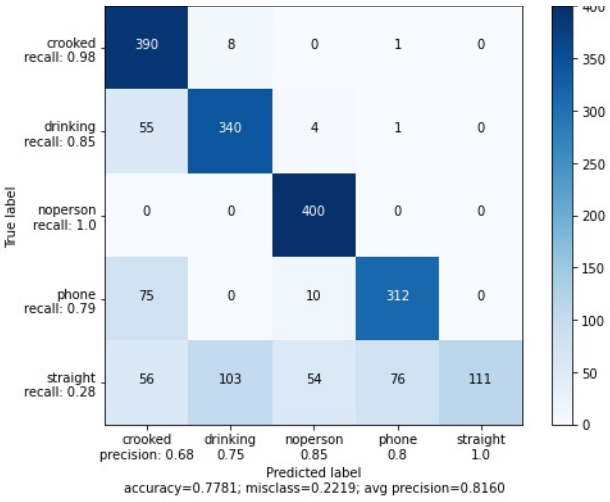


(a) Confusion Matrix auf Eval1

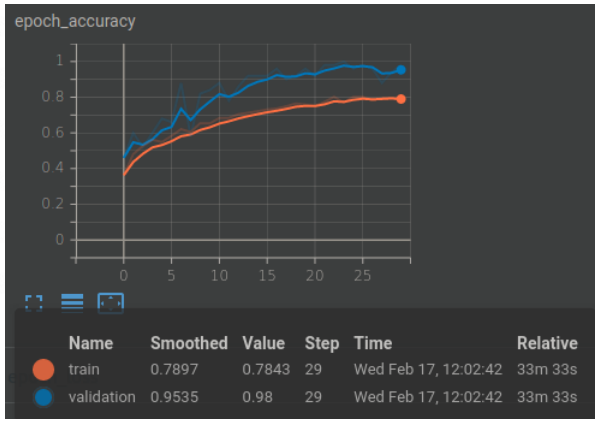


(b) Accuracy der Trainings- und Validierungsdaten

Figure 21: Evaluierung und Trainingsverlauf bei Benutzung des AdaMax Optimierers



(a) Confusion Matrix auf Eval1



(b) Trainingsverlauf (Acurracy) der Trainings- und Validierungsdaten

In der Referenz [16] ist die Rede davon, dass der AdaMax Optimierer in gewissen Situation besser abschneidet als der Adam Optimierer, dies ist anscheinend solch eine Situation. Testweise wurde der Vergleich nochmal mit dem Datensatz Train1 evaluiert. Dort schneiden

Adam mit 79.53% zu AdaMax mit 69.88% ab.
Des weiteren sieht man, dass der Adam Optimierer etwas schneller höhere Epochen accuracy erreicht. In Figure 20(b) ist Adam bei Epoche 5 schon bei 65% wohingegen AdaMax (Fig. 21(b)) bei Epoche 5 erst bei 55% Accuracy ist. Somit wurde sich bei der Weiteren Nutzung für den Adam Optimierer entschieden.

3.5.4 Fazit Auswertung der Parameter des Neuronalen Netzes

Da nicht alle Verbesserungen der Referenzen bewiesen werden konnten, ist davon auszugehen, dass die Kombination aus Adam Optimierer, ReLU Aktivierungsfunktion und Crossentropy Fehlerfunktion in diesem Fall sehr optimal gewählt ist. Andernfalls bleibt nur zu sagen, dass die gezeigte Kombination wahrscheinlich zufälligerweise höhere Accuracy Werte mit dem Evaluierungsdatensatz erreicht und man erst durch sehr häufige Wiederholungen des Trainings prüfen kann, ob meine Ergebnisse aussagekräftig sind.

3.6 Evaluation - Lernverbesserung in Bezug auf die verwendeten Trainingsdaten

3.6.1 Vergleich Augmentation

Grundsätzlichen gibt es bereits leichte Augmentation durch die Farbverschiebung des Aufnahmeskript. Zu Beginn des Projekts gab es Probleme ein Netz zu trainieren, dass nicht overfittet. Overfitting liegt vor, sofern das Netz an die Trainingsdaten überangepasst ist und nicht mehr auf fremde Daten abstrahieren kann. Für die Verbesserung des Abstahierens wurde zum einem versucht mit verschiedener Kleidung zu trainieren (Train1 Datensatz), des Weiteren aber auch mit verzerren, verschieben, spiegeln der Bilder eine bessere Abstraktion hinzubekommen (Data Augmentation)
Die Data Augmentation auf den Trainingsdaten wird durch die ImageDataGenerator Klasse [17] umgesetzt (Listing 2, Zeile 1-8 & Listing 3, Zeile 1-9). Die Validierungsdaten werden nicht verändert. Über die ImageDataGenerator.flow_from_directory Funktion werden die Daten bequem über die Ordner Struktur des Datensatzes eingelesen und gelabelt.

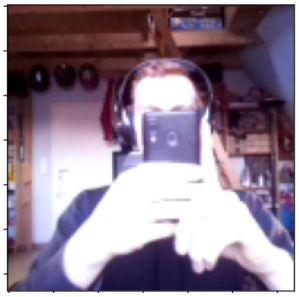
```
1 train_datagen = ImageDataGenerator(
2     rescale=1./255,
3     width_shift_range= int(BILDDIMENSION[0]*0.3), #Verschiebung in der Bildbreite
4     rotation_range=10, #Drehung in Grad
5     zoom_range=[0.8, 1.2], #Bereich für zufälliges zoomen (1 = kein zoom)
6     shear_range=5, #Intensität für Verzerrung
7     horizontal_flip=True, #Horizontal Spiegel (an der Vertikalen-Achse)
8     brightness_range=0.1, 1.5) #Helligkeit
```

Listing 2: Data Augmentation Set1

```
1 train_datagen = ImageDataGenerator(
2     rescale=1./255,
3     width_shift_range= int(BILDDIMENSION[0]*0.3), #Verschiebung in der Bildbreite
4     height_shift_range= int(BILDDIMENSION[0]*0.2), #Verschiebung in der Bildhöhe
5     rotation_range=15, #Drehung in Grad
6     zoom_range=[0.8, 1.2], #Bereich für zufälliges zoomen (1 = kein zoom)
7     shear_range=15, #Intensität für Verzerrung
8     horizontal_flip=True, #Horizontal Spiegel (an der Vertikalen-Achse)
9     brightness_range=0.1, 1.5) #Helligkeit
```

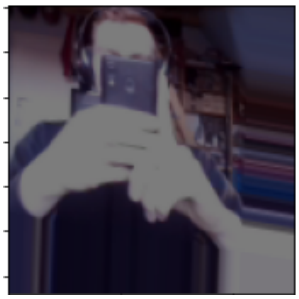
Listing 3: Data Augmentation Set2

Figure 22: Bilder Repräsentation der folgenden Augmentation Methodiken



(a) Beispiel: Keine Augmentation
Verwendet in Fig. 23

(b) Beispiel: Augmentation Set 1
Verwendet in Fig. 24



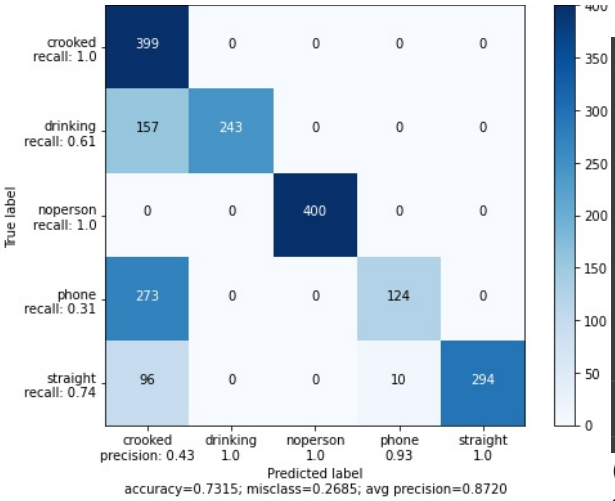
(c) Beispiel: Augmentation Set 2
Verwendet in Fig. 25

(d) Beispiel: Filterung zu Graustufen
Verwendet in Fig. 26

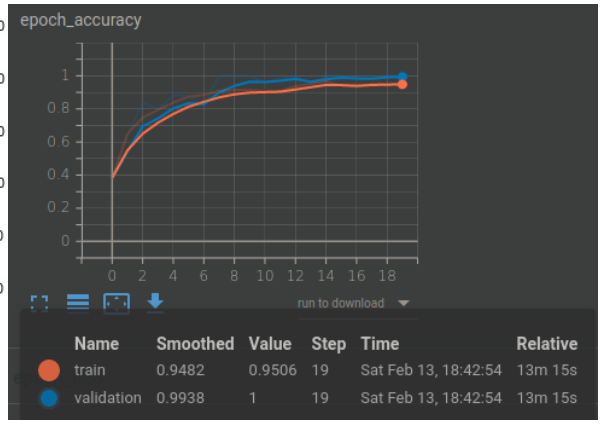
(e) Beispiel: Sobelfilterung
Verwendet in Fig. 27

Die Netze für die folgenden Vergleiche wurden auf dem Datensatz Train1, dem großen Datensatz, oder Train2, dem kleinen Datensatz, trainiert(Siehe S.14). Evaluiert wird auf dem Datensatz Eval1, der nicht in Train1 oder Train2 enthalten ist. Die Netze wurden maximal 20 Epochen â 50 Steps trainiert, um sie hinreichend vergleichen zu können.

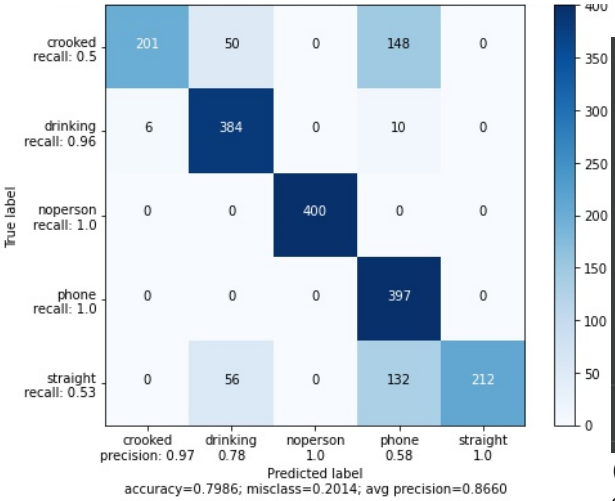
Figure 23: Training ohne Data Augmentation



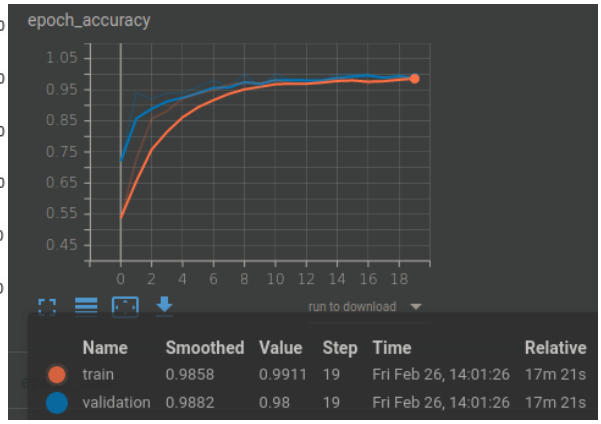
(a) Confusion Matrix auf Eval1 mit Train1 CNN



(b) Accuracy der Trainings- und Validierungsdaten des zu (a) gehörigen Netzes



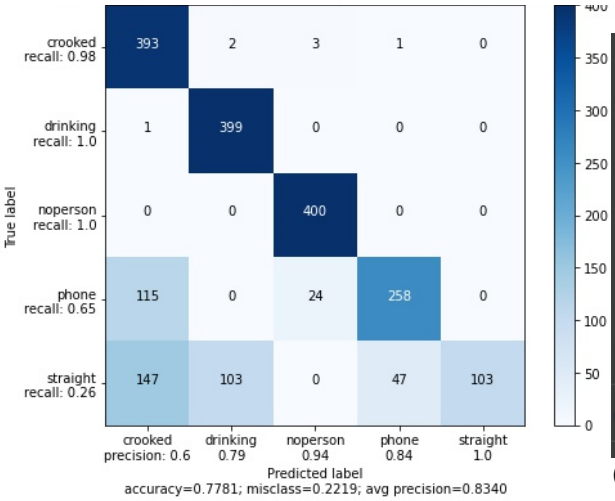
(c) Confusion Matrix auf Eval1 mit Train2 CNN



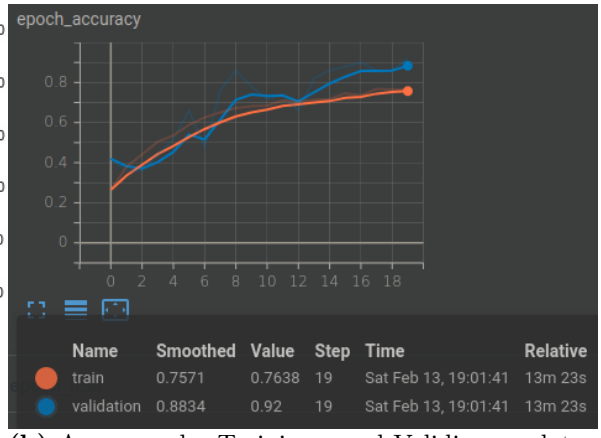
(d) Accuracy der Trainings- und Validierungsdaten des zu (c) gehörigen Netzes

Figure 23 ist ein Test der zeigen sollte, dass Netze, dessen Trainingsbilder nicht mit Preprocessing Funktionen (verzerren, verschieben, etc.) oder Ähnliches bearbeitet wurden, für geringen Accuracy Werten der Evaluierungsdaten sorgt. In Figure 23 (a)&(c) zeigt sich jedoch, dass sowohl das Training auf Train1 als auch auf Train2 in der Lage ist zu generalisieren und in der Lage ist viele fremde Bilder richtig zu erkennen.

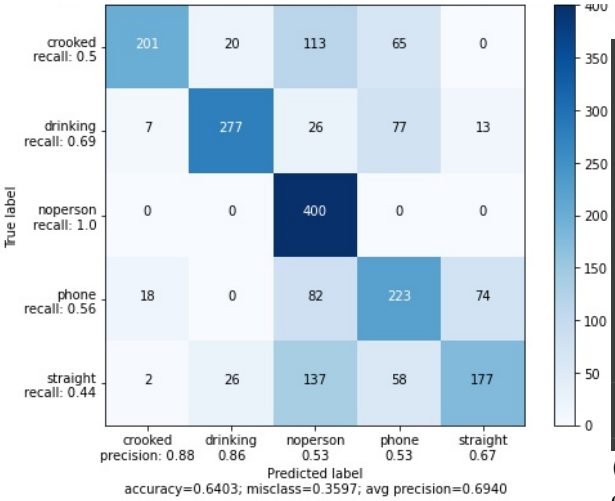
Figure 24: Training mit Data Augmentation Set1



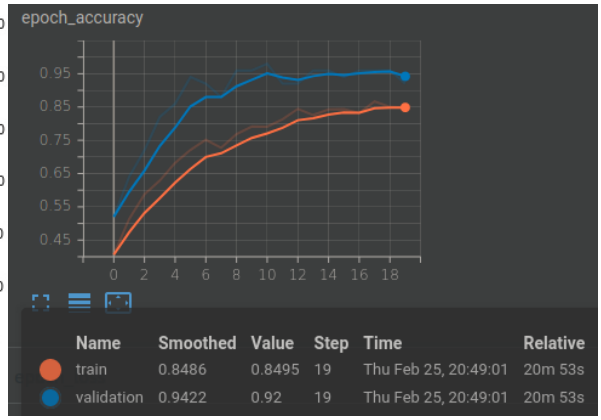
(a) Confusion Matrix auf Eval1 mit Train1 CNN



(b) Accuracy der Trainings- und Validierungsdaten des (a) Netzes



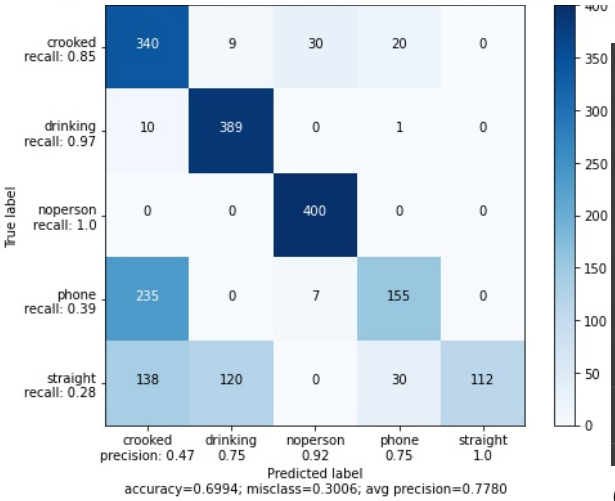
(c) Confusion Matrix auf Eval1 mit Train2 CNN



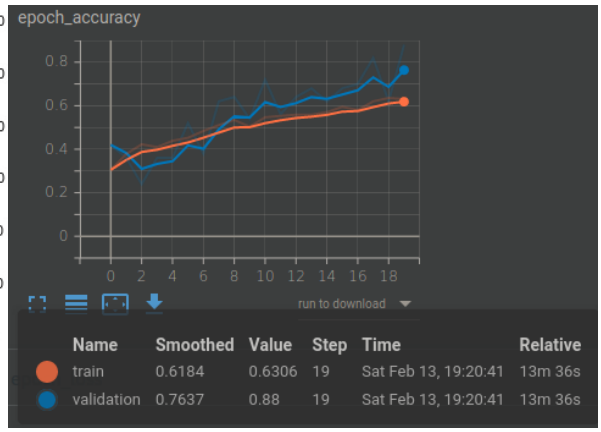
(d) Accuracy der Trainings- und Validierungsdaten des (c) Netzes

Sofern mit Data Augmentation gearbeitet wird, braucht es mehr Trainingsepochen hohe Accuracy Werte auf Eval1 zu erreichen als ohne Augmentation. Testweise wurde das Netz aus Fig. 24(c) noch bis 30 Epochen trainiert und konnte dort auf dem Eval1 Datensatz eine Accuracy von 74.85% erreichen. Das Netz aus Fig. 24(a) hat bei 30 Epochen Training auf Eval1 nur eine Accuracy von 65% erreicht, dabei ist die Accuracy der Trainingsdaten nur um 2% gestiegen.

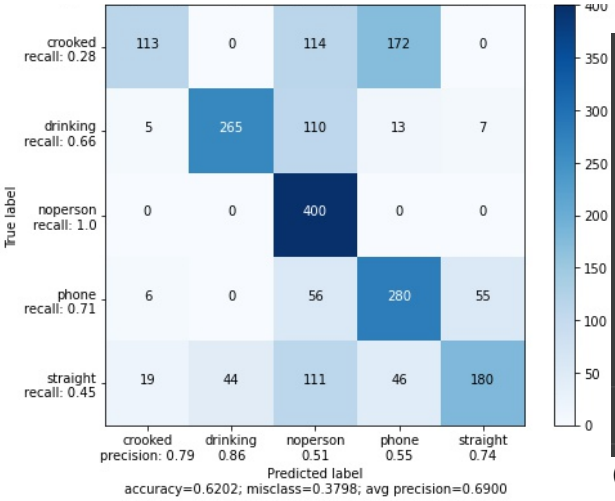
Figure 25: Training mit Data Augmentation Set2



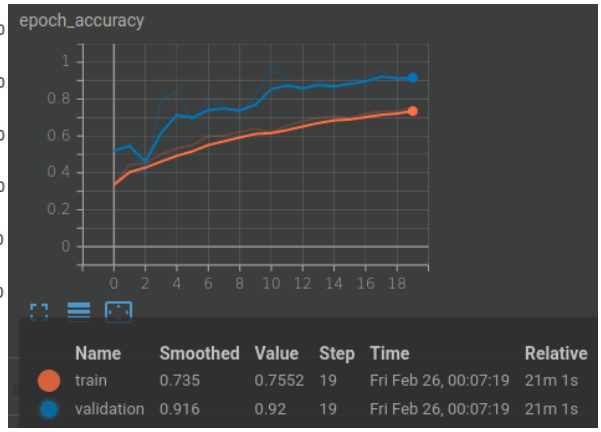
(a) Confusion Matrix auf Eval1 mit Train1 CNN



(b) Accuracy der Trainings- und Validierungsdaten



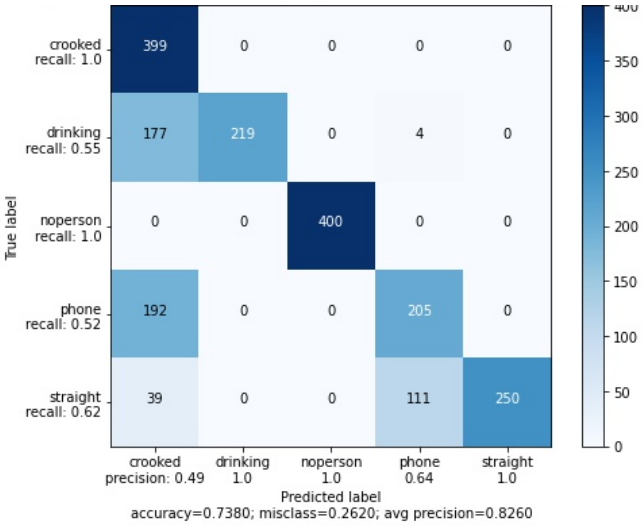
(c) Confusion Matrix auf Eval1 mit Train2 CNN



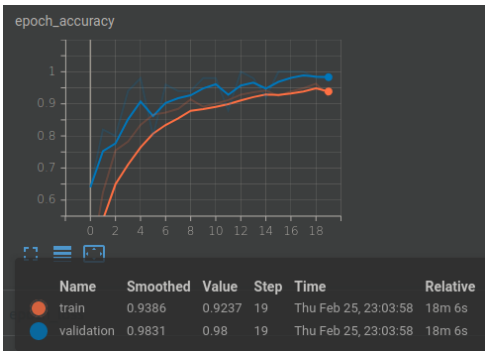
(d) Accuracy der Trainings- und Validierungsdaten des (c) Netzes

Sofern mit stärkerer Data Augmentation gearbeitet wird, sind noch mehr Trainingsepochen nötig, um hohe Accuracy Werte auf Eval1 zu erreichen als mit Data Augmentation Set 1. Testweise wurde das Netz aus Fig. 25(c) noch bis 30 Epochen trainiert und konnte dort auf dem Eval1 Datensatz nur eine geringere Accuracy von 56.16% erreichen. Das Netz aus Fig. 24(a) hat bei 30 Epochen Training auf Eval1 nur eine Accuracy von 66.78% erreicht, dabei ist die Accuracy der Trainingsdaten nur um 6% gestiegen.

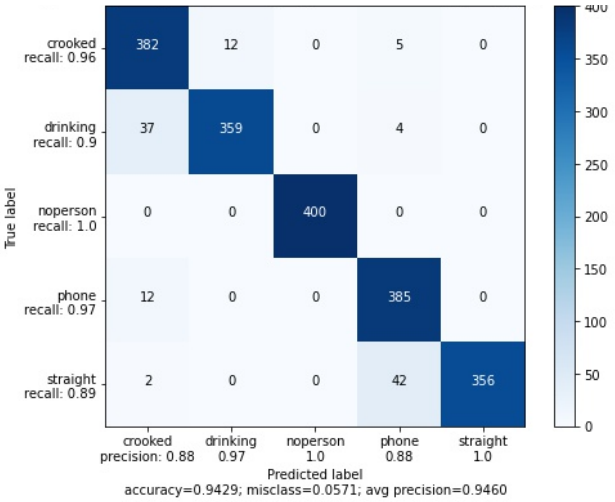
Figure 26: Training mit vorheriger Grayscale-Filterung (Grauwertreduktion)



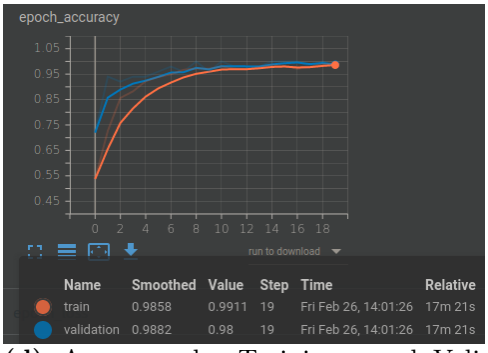
(a) Confusion Matrix auf Eval1 mit Train1 CNN



(b) Accuracy der Trainings- und Validierungsdaten des (a) Netzes



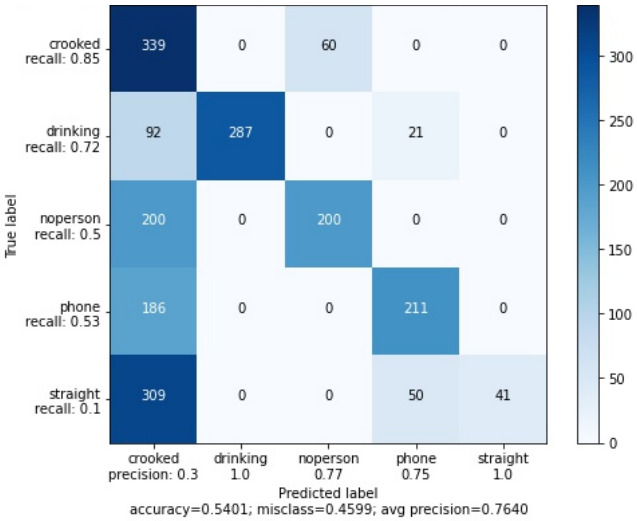
(c) Confusion Matrix auf Eval1 mit Train2 CNN



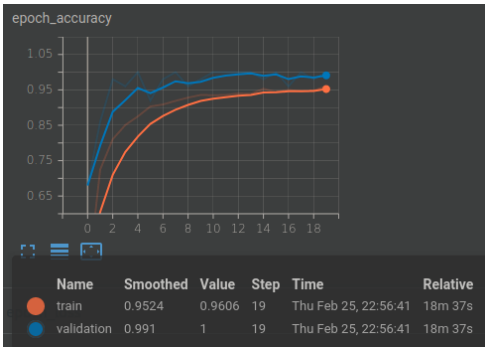
(d) Accuracy der Trainings- und Validierungsdaten des (c) Netzes

In Figure 26 wurde nun anstelle der Data Augmentation aus Listing 2 und 3 eigenes Preprocessing betrieben. Die Farbbild-Trainingsdaten werden hier auf Grauwertbilder konvertiert, wodurch versucht wird, von den verschiedenen Kleidungsfarben unabhängig zu werden. In Fig. 26(a) sieht man, dass das mit großem Datensatz tranierte CNN sehr ähnlich zu Fig. 23 (a) (ohne Augmentation) abschneidet, aber eine geringere Evaluierungs Accuracy als Fig. 24(a) (Augmentation Set 1) besitzt. Das auf dem kleinen Datensatz trainierte CNN in Fig.26(c) schneidet mit der höchsten Accuracy unter den mit Train2 trainierten Netzen ab.

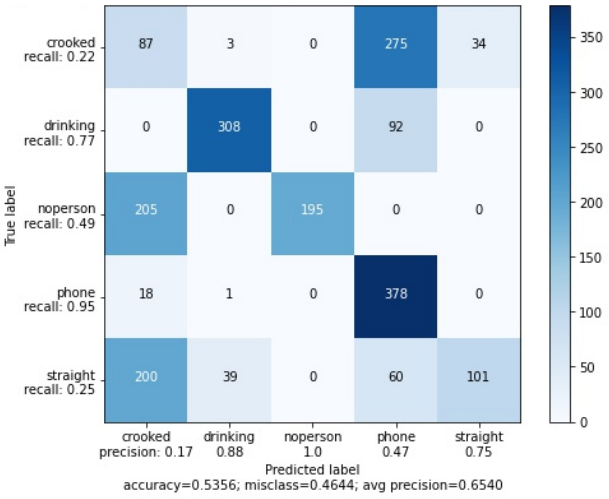
Figure 27: Training mit vorheriger Sobel-Filterung (Kantendetektor)



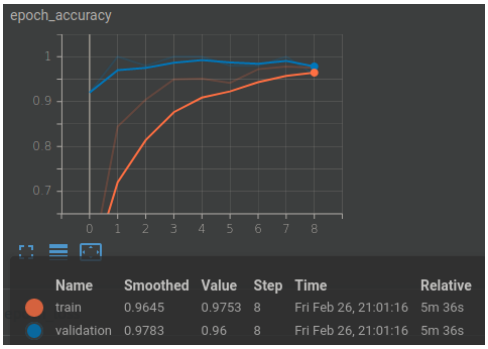
(a) Confusion Matrix auf Eval1 mit Train1 CNN



(b) Accuracy der Trainings- und Validierungsdaten des (a) Netzes



(c) Confusion Matrix auf Eval1 mit Train2 CNN



(d) Accuracy der Trainings- und Validierungsdaten des (c) Netzes

In Figure 27 wurde nun noch einen Schritt weiter gegangen mit der eigenen Data Augmentation. Hier wurden über die Bilddatensätze vor dem Training der Sobel-Filter laufen gelassen, wodurch Bilder entstehen, in denen nur die Kanten von Objekten angezeigt werden.

Wohingegen die Accuracy Verläufe auf den Trainingsdaten der beiden Netze in Fig. 27(b)&(d) zeigen, dass die Netze ausgelernt wurden, zeigt die Evaluierung in Fig. 27(a)&(c) die geringsten Accuracy Werte aller Augmentation Methodiken.

3.6.1.1 Auswertung des Augmentation Vergleichs

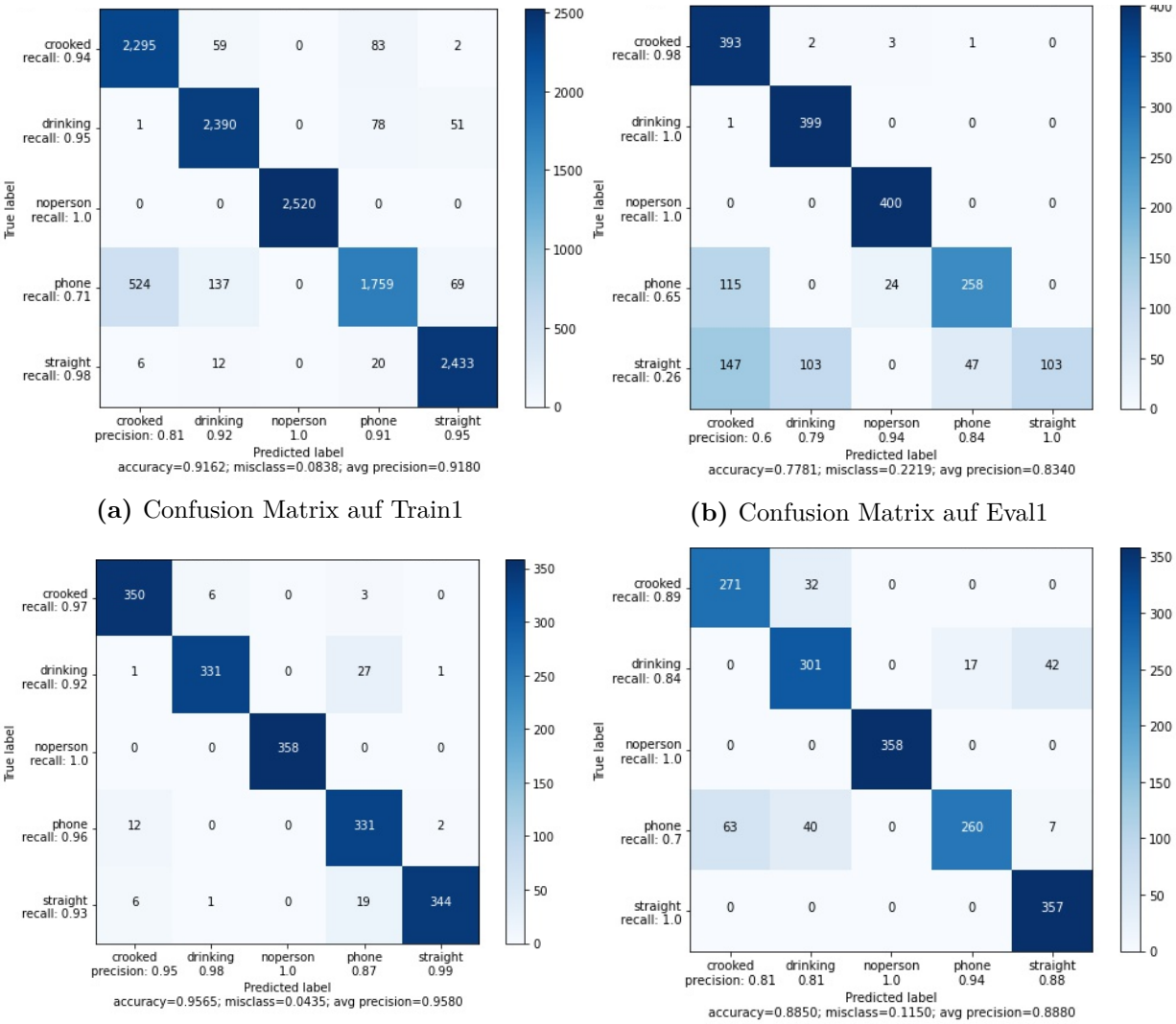
Das auf Seite 23 erwähnte Overfitting Problem konnte durch Fig. 23 widerlegt werden. Das Overfitting Problem hatte vermutlich andere Gründe, die nicht wieder reprodziert werden konnten.
Figure 24 zeigt, dass das mit Train1 trainierte Netz mit höheren Accuracy Werten bei der Evaluierung mit Data Augmentation Set 1 abschneidet, als ohne Data Augmentation. Für das mit Train2 trainierte Netz, schneidet das Training mit Data Augmentation Set 1 und 2 allerdings mit geringeren Accuracy Werten auf Eval1 ab, also ohne Augmentation.
Figure 25 zeigt, dass stärkere Augmentation nicht unbedingt vorteilhaft ist, so schneiden beide Netze mit geringer Accuracy auf den Evaluierungsdaten ab, als die korrespondierenden Data Augmentation Set 1 (Fig. 24) Netze.
Figure 26 zeigt, dass die Filterung der Farbbilder auf Grauwertbilder dafür sorgt, dass es keine Unterscheidung verschiedener Kleidungsfarben braucht und bereits ein Netz, welches mit nur einer Kleidungsfarbe trainiert wurde, bereits gut abstrahieren kann.
Figure 25 zeigt, dass die Filterung der Farbbilder mit dem Sobel-Kantendektor für geringere Evaluierungs Accuracy sorgt.
Zum Teil zeigt dieser Abschnitt 3.6.1, dass ein großer Datensatz vieler verschiedener Kleidungsfarben nicht nötig ist, da das mit Train2 trainierte Netz sehr hohe Accuracy auf Eval1 erreicht. Jedoch sei hier zu sagen, dass nur auf einem Datensatz evaluiert wird, auf dem zwar nicht trainiert wurde, der allerdings nur eine Farbe abbildet. Aus diesem Grund werden in der Sektion 3.6.2, die besten Netze für Train1 und Train2 aus dieser Sektion auf allen Kleidungsfarben evaluiert.

3.6.2 Großer oder kleiner Trainingsdatensatz?

Das Aufnehmen der verschiedenen Kleidungsfarben/Kleidungsstücken des Train1 Datensatzes ist sehr zeitaufwendig. Erstrebenswert ist ein kleiner Trainingsdatensatzes mit nur wenigen aufgenommenen Kleidungsstücken, der alle Klassen bei vielen verschiedenen nicht trainierten Kleidungsstücken erkennt. Zum Teil wurde das Können eines kleinen Datensatzes schon in Sektion 3.6.1 bewiesen, dort allerdings nur auf Evaluierungsdatensatz Eval1. Um noch genauer die Probleme und Chancen eines kleines Datensatzes deutlich zu machen hier letztlich der Vergleich auf einem größeren Evaluierungsdatensatz.
Dazu wird im folgenden aus Sektion 3.6.1 das Netz mit der höchsten Evaluierungs-Accuracy, welches mit Train1 trainiert wurde, mit dem Netz mit der höchsten Evaluierungs-Accuracy verglichen, welches mit Train2 trainiert wurde.
Die folgenden Netze wurden maximal 20 Epochen â 50 Steps trainiert.

3.6.2.1 Ausgangslage des Netzes trainiert mit Train1 und Data Augmentation Set 1

Figure 28: Evaluierung auf allen Bilderdaten des Train1 Netzes



(a) Confusion Matrix auf Train1

(b) Confusion Matrix auf Eval1

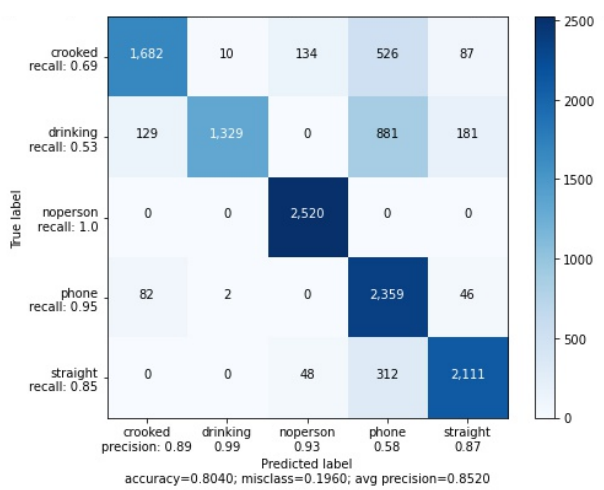
(c) Confusion Matrix auf EvalRot

(d) Confusion Matrix auf EvalWeiß

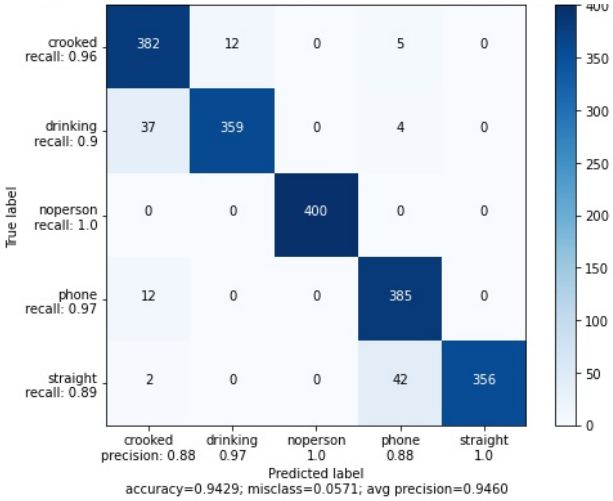
Figure 28 repräsentiert das Neuronale Netz aus Fig. 24(a). Fig. 28(a), (c) und (d) zeigen, dass die Klassen bei allen Kleidungsstücken sehr gut zwischen den Klassen unterscheiden kann. Dies sind allerdings alles Trainingsbilder auf denen Evaluiert wird. (b) ist trainingsfremd und zeigt, dass das Netz hier größere Probleme mit straight hat, trotzdessen wurden die meisten Bilder richtig klassifiziert.

3.6.2.2 Netz trainiert auf mit dem kleinen Datensatz Train2

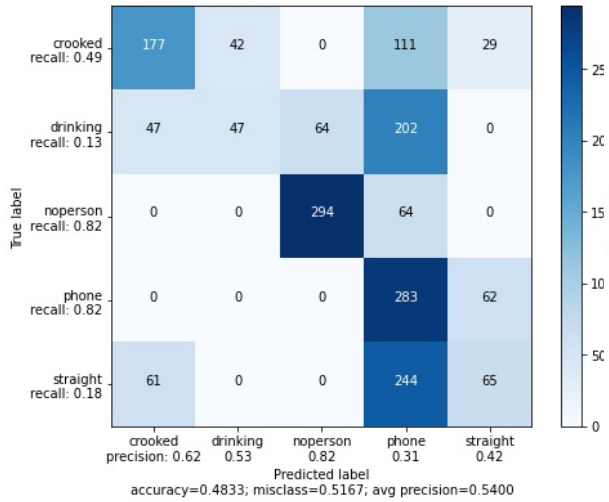
Figure 29: Evaluierung auf allen Bilderdaten durch das Train2 Netzes mit Grauwertbildern



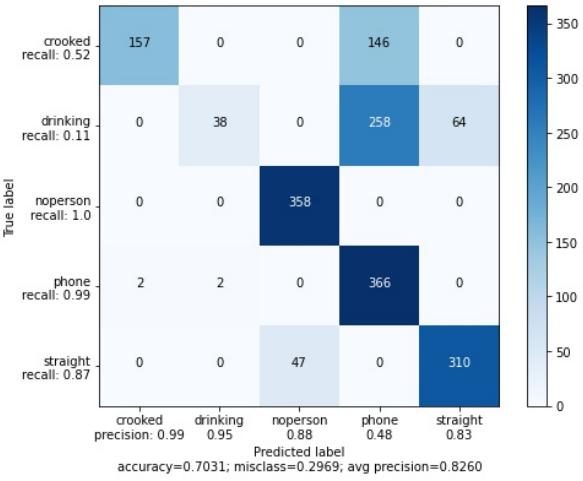
(a) Confusion Matrix auf Train1



(b) Confusion Matrix auf Eval1

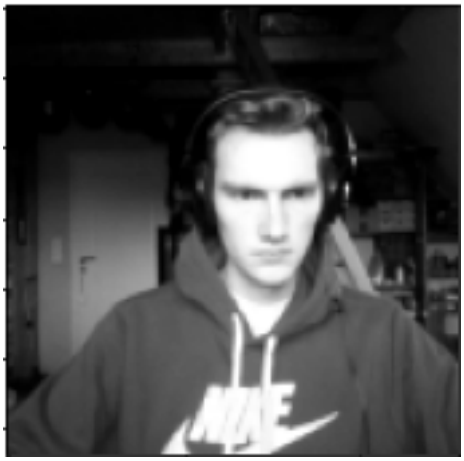


(c) Confusion Matrix auf EvalRot



(d) Confusion Matrix auf EvalWeiß

Figure 30: Beispiel Bilder aus EvalRot und EvalWeiß mit Reduktion auf Grauwerte



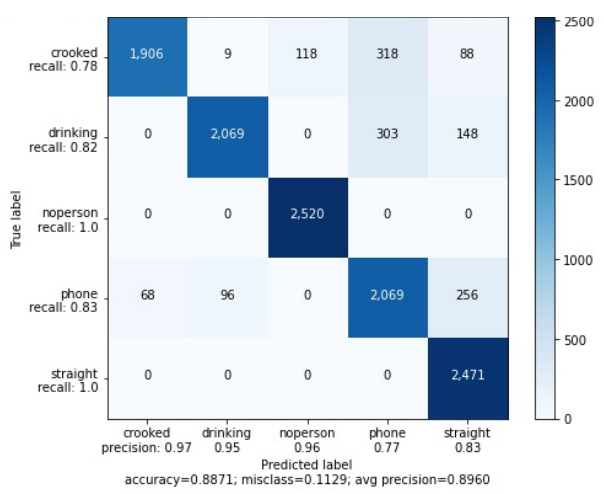
(a) EvalRot zugehörig - Crooked

(b) EvalWeiß zugehörig - Drinking

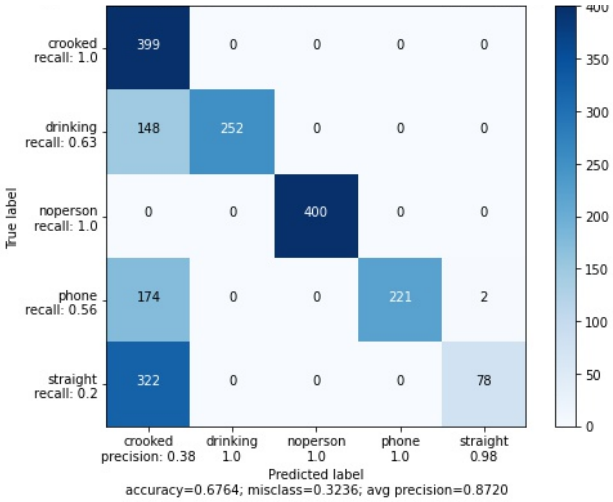
Figure 29 repräsentiert das Neuronale Netz aus Fig. 26(c)
Fig. 29(a) und (b) evaluiert auf fast komplett trainingsfremden Daten (lediglich grün ist in Train1 enthalten) und zeigt, wie gut der kleine Datensatz mit nur einer Kleidungsfarbe den großen Datensatz mit fast allen Kleidungsfarben klassifizieren kann.
Trotzdessen werden einige Bilder fälschlicherweise der Phone Klasse zugeordnet. Der Grund dafür wird deutlich durch Blick auf Figure 30(a). Der "Nike" Druck auf dem Pullover wird höchstwahrscheinlich für das Smartphone gehalten, da auf dieser Höhe auch viele Trainingsbilder das Handy zeigen (Siehe Fig.11(d)).
Fig. 27(d) zeigt, dass es auch hier zu Verwechslungen bei der Drinking Klasse kommt, oft mit der Phone Klasse. Fig. 30(b) im Vergleich zu Fig.22(a) lässt darauf schließen, dass entweder die Flasche für ein Smartphone gehalten wird oder dass der in Figure 30(b) entstehende Schatten auf Brusthöhe für die Phone Klassifizierung sorgt.

3.6.2.3 Evaluierung des Training auf Train1 mit Grauwertbildern zuzüglich einer weiteren Farbe

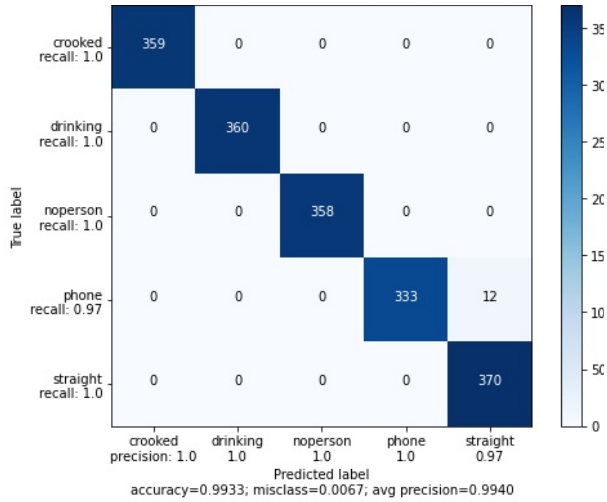
Figure 31: Evaluierung auf allen Bilderdaten durch das TrainRotG Netz mit Grauwertbildern



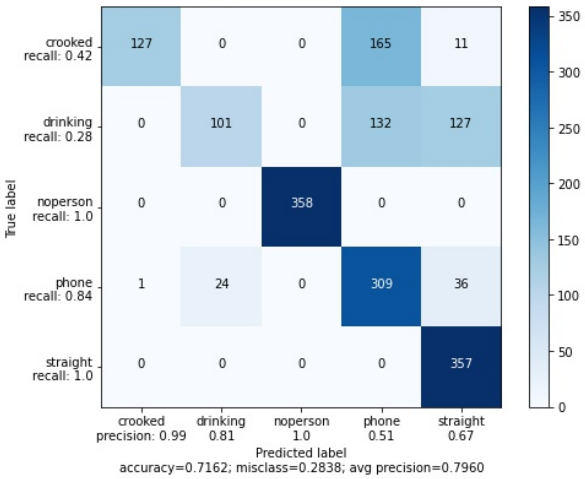
(a) Confusion Matrix auf Train1



(b) Confusion Matrix auf Eval1



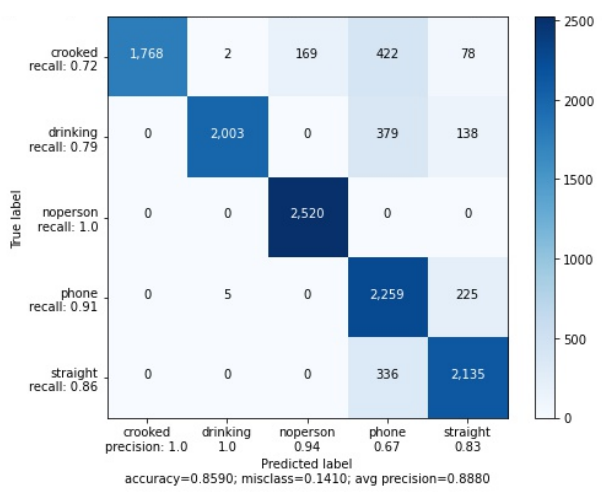
(c) Confusion Matrix auf EvalRot



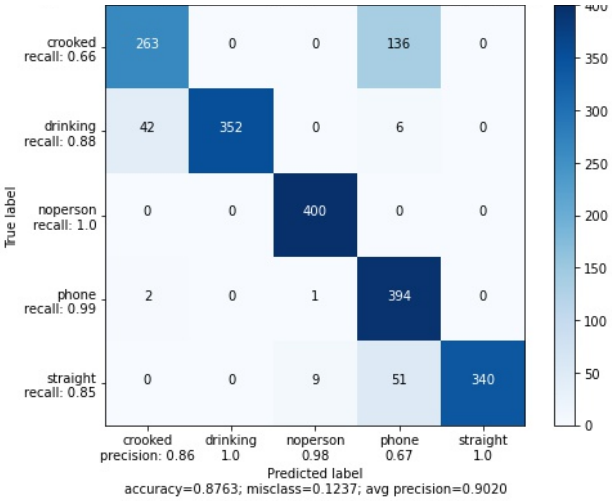
(d) Confusion Matrix auf EvalWeiß

Das Netz aus Figure 31 hat keine Probleme mehr mit dem "Nike" Schriftzug, siehe Fig. 31(c), da diese Bilder nun im Trainingsdatensatz inkludiert sind. Figure 31(a) zeigt größere Accuracy Werte als 29(a) und 31(c) bessere Werte als 29(c). Bei dem eintönigen Kleidungsstücken wie schwarz (Fig. 31(b)) kommt es nun allerdings zu kleineren Accuracy Werten als bei Fig. 29(b).

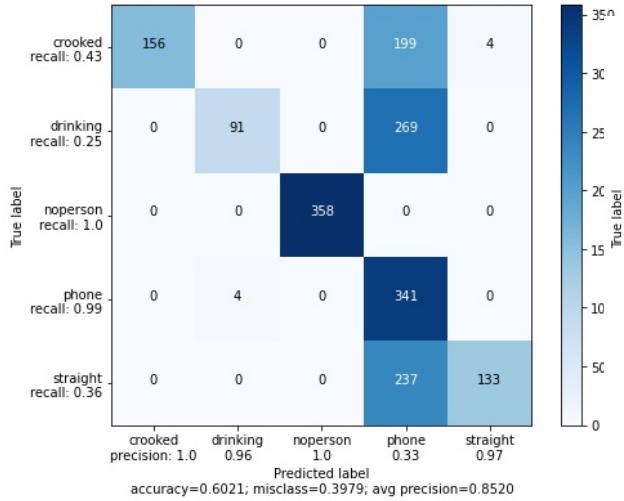
Figure 32: Evaluierung auf allen Bilderdaten durch das TrainWeißG Netz mit Grauwertbildern



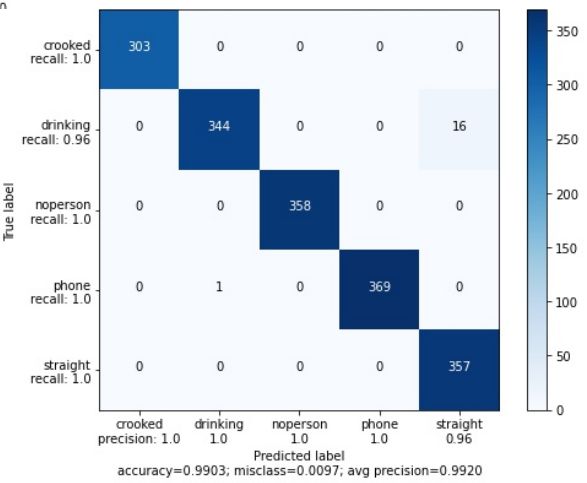
(a) Confusion Matrix auf Train1



(b) Confusion Matrix auf Eval1



(c) Confusion Matrix auf EvalRot



(d) Confusion Matrix auf EvalWeiß

Das Netz aus Figure 32 hat wie das Netz aus Figure 29 wieder Probleme mit dem Schriftzug des "Nike" Pullovers nun aber weniger Probleme mit der schwarzen als auch weißen Kleidung.

3.6.2.4 Auswertung: Großer oder kleiner Trainingsdatensatz?

Figure 28 zeigt, dass das Training auf einem Datensatz mit vielen verschiedene Bilder Ergebnisse mit hohen Accuracy Werten liefert. Dieses Training ist allerdings in Bezug auf die Datenbeschaffung sehr zeitaufwendig.

Das Netz aus Figure 29 ist zwar optimal in Bezug auf den Zeitaufwand der Datenbeschaf-

fung und hat auf dem Datensatz Train1 eine hohe Evaluations Accuracy, es ist allerdings zu sehen, dass das Netz bei bestimmten Kleidungsstücken (bestimmte Muster auf der Kleidung) weniger als 50% der Bilder richtig klassifiziert. Trotzdessen erzielt das Netz bei einem Live Test der Anwendung gute Ergebnisse, sofern nicht der rote Pullover verwendet wird.
Figure 31 und 32 zeigen, dass mit leicht mehr Zeitaufwand bei der Datenbeschaffung, durch Hinzunahme eines weiteren Kleidungsstückes, Klassifikationsergebnisse zu erzielen sind, die vergleichbar mit Figure 28 sind.

4 Fazit und weitere Verbesserungen

Diese Arbeit stellt eine Anwendung vor, die unter Verwendung von Maschinellen Lernen dabei hilft, Benutzern auf krumme Körperhaltung, regelmäßige Flüssigkeitszunahme, genug Sauerstoff in der Luft zu achten und sie davon abzuhalten in produktiven Phasen durch das Smartphone abgelenkt zu werden.
Dabei konnte durch Sektion 3.6 gezeigt werden, dass ein Netz, welches nur mit einem Kleidungsstück trainiert wurde, auf bereits sehr viele verschiedene Kleidungsstücke abstrahieren kann und dabei die richtigen Klassen erkennt. Dies ist ein großer Vorteil im Gegensatz zum Training mit sehr vielen verschiedenen Kleidungsstücken, da man als Benutzer nicht seinen gesamten Kleiderschrank in die Anwendung einpflegen möchte. Lediglich auf Kleidungsstücke mit bestimmten Mustern ist schwer zu abstrahieren, ohne darauf trainiert zu haben.
Als weitere Reduktion des Aufwandes der Datenbeschaffung hätte noch geprüft werden können, wie klein die Anzahl der Bilder des z.B.. TrainWeißG Datensatzes sein hätte können, sodass immer noch hinreichend hohe Evaluierung-Accuracy Werte zu erreichen sind.

5 Referenzen

[0] - Die richtige Arbeitspostion. http://www.ifh.uni-karlsruhe.de/misc/Position.htm. Eingesehen 28.01.2021.
[1] - Ergonomisch Arbeiten. https://www.konstruktionspraxis.vogel.de/acht-tipps-fuer-das-perfekte-ergonomische-sitzen-a-677049/ Eingesehen am 20.02.2021
[2] - Raspberry Pi 4. https://www.raspberrypi.org/products/raspberry-pi-4-model-b/. Eingesehen 15.02.21.
[3] - Camera Module V2. https://www.raspberrypi.org/products/camera-module-v2/. Eingesehen 28.02.2021
[4] - Coral USB Accelerator. https://coral.ai/products/accelerator Eingesehen 28.02.2021.
[5] - PyPubHomepage. https://pypubsub.readthedocs.io/en/v4.0.3/ Eingesehen 25.01.2021.
[6] - Python Statemachine Bibliothek. https://python-statemachine.readthedocs.io/en/latest/readme.html. Eingesehen 25.01.2021
[7] - PyAudio Bilbiothek. https://pypi.org/project/PyAudio/. Eingesehen 21.01.2021
[8] - Google Übersetzer Text-To-Speech. https://translate.google.com/?hl=de&sl=en&tl=de&text=Please%20Drink&op=translate Eingesehen am 28.02.2021
[9] - Aufnahmeprogramm Audacity. https://www.audacityteam.org/ Eingesehen am 28.02.2021
[10] - PiCamera Bibliothek. https://picamera.readthedocs.io/en/release-1.13/. Eingesehen 21.02.2021
[11] - Tensorflow Lite Inference. https://www.tensorflow.org/lite/guide/inference#load_and_run_a_model_in_python Eingesehen 13.01.2021
[12] - 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#rec Eingesehen 10.11.2020
[13] - Dropout Layer. https://keras.io/api/layers/regularization_layers/dropout/ Eingesehen am 10.02.2021
[14] - How to choose lossfunction. https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/ Eingesehen am 10.02.2021
[15] - Prajit Ramachandran, Barret Zoph, Quoc V. Le. Swish: a Self-Gated Activation Function. https://arxiv.org/abs/1710.05941v1. Eingesehen am 10.02.2021
[16] - Keras Optimizers Classes. https://analyticsindiamag.com/guide-to-tensorflow-keras-optimizers/ Eingesehen am 10.02.2021

[17] - Image data preprociessing. https://keras.io/api/preprocessing/image/ Eingesehen am 10.2.2021