



Hochschule
Albstadt-Sigmaringen
Albstadt-Sigmaringen University

Software-Entwicklung einer Smart-Kamera

Security und Internet der Dinge

Name:	James Edonnard Kibii
Matr. Nr.:	102672
Studiengang:	Systems Engineering
Betreuender Professor:	Prof. Dr. Thomas Eppler
Semester:	Sommersemester 2022

Inhaltsverzeichnis

Formel- und Abkürzungsverzeichnis	III
Abbildungsverzeichnis.....	IV
Tabellenverzeichnis.....	VI
1 Einleitung	1
2 Systemdesign.....	2
2.1 Anforderungen	2
2.1.1 Hardware	2
2.1.2 Software	2
2.1.3 Bibliotheken	2
2.2 Datenfluss	12
3 Systemarchitektur	13
3.1 MQTT	13
3.2 OpenCV	16
3.3 Maschinelles Lernen	17
3.3.1 Design des Trainingsprogramms	18
3.3.2 Durchführung des Modell-Trainingsprogramm.....	24
3.3.3 Implementierung des ML-Modells in das System	26
3.4 Verschlüsselung.....	27
4 Diskussion.....	31
4.1 Ergebnis	31
4.2 Anwendung in der Praxis	35
5 Zusammenfassung.....	36
6 Literaturquellen	VI
Anhang.....	VII
Anhang A: Verfahren zur Prädiktion von Gesicht und Maske	VII
Anhang B: Stream-Funktion.....	VIII
Anhang C: Dataset mit Bildern mit und ohne Maske	IX

Formel- und Abkürzungsverzeichnis

Formelzeichen

Zeit: Sekunde - [s]

Abkürzungen

MQTT	-	„ <i>Message Queue Telemetry Transport</i> “
OpenCV	-	„ <i>Open Source Computer Vision Library</i> “
KI	-	Künstliche Intelligenz
ML	-	Maschinelles Lernen
IoT	-	„ <i>Internet of Things</i> “ (Internet der Dinge)
CUDA	-	„ <i>Compute Unified Device Architecture</i> “
cuDNN	-	„ <i>CUDA Deep Neural Network</i> “
DNN	-	„ <i>Deep Neural Network</i> “
GPU	-	„ <i>Graphics Processing Unit</i> “ (Grafikprozessor)
E2EE	-	„ <i>End-to-End Encryption</i> “ (Ende-zu-Ende-Verschlüsselung)
IANA	-	„ <i>Internet Assigned Numbers Authority</i> “

Abbildungsverzeichnis

Abbildung 1: Sender-Programm-Bibliotheken	2
Abbildung 2: Empfänger-Programm-Bibliotheken	3
Abbildung 3: Bibliotheksinstallation über requirements.txt	3
Abbildung 4: Verschlüsselungs-Programm-Bibliotheken	3
Abbildung 5: ML Trainingsprogramm-Bibliotheken	4
Abbildung 6: Registry Editor (1)	5
Abbildung 7: Registry Editor (2)	5
Abbildung 8: LongPathsEnabled Dialogfenster	5
Abbildung 9: NVIDIA-Treiber	6
Abbildung 10: Task-Manager-Fenster	6
Abbildung 11: Visual-Studio-Installer	7
Abbildung 12: CUDA-Toolkit herunterladen	8
Abbildung 13: CUDA-Installation	8
Abbildung 14: Umgebungsvariablen	9
Abbildung 15: cuDNN-Download	9
Abbildung 16: cuDNN-Ordner	10
Abbildung 17: CUDA-Ordner (v11.7)	10
Abbildung 18: CUDNN-Pfad	11
Abbildung 19: Tensorflow importieren (mit CUDA & cuDNN)	11
Abbildung 20: Tensorflow importieren (ohne CUDA noch cuDNN)	11
Abbildung 21: Flussdiagramm	12
Abbildung 22: MQTT-Kommunikation	14
Abbildung 23: VideoStream-Konstruktor (Publisher)	14
Abbildung 24: connect-Funktion	15
Abbildung 25: publish-Funktion	15
Abbildung 26: VideoStream-Konstruktor (Subscriber)	15
Abbildung 27: connect-Funktion	15
Abbildung 28: read_message-Funktion	16
Abbildung 29: send-Funktion	17
Abbildung 30: Argument-Parser	18
Abbildung 31: Initialisierung der Bilddaten	19
Abbildung 32: One-Hot-Kodierung	20
Abbildung 33: Train-Test-Split-Funktion	20
Abbildung 34: Convolutional Neural Networks	21
Abbildung 35: Initialisierung der Trainings-Parameter	21
Abbildung 36: Der Image-Data-Generator	22
Abbildung 37: Erstellung des Modells	22
Abbildung 38: Alle Layers einfrieren, um Aktualisierungen zu verhindern	23
Abbildung 39: Modell kompilieren	23
Abbildung 40: Model-Fitting	24

Abbildung 41: Netzwerk-Evaluierung	24
Abbildung 42: Modell speichern und Plot erstellen.....	24
Abbildung 43: Ausführung des Trainingsprogramms.....	25
Abbildung 44: Klassifizierungsbericht	25
Abbildung 45: Trainings-Loss und -Accuracy	25
Abbildung 46: face_detector-Ordner	26
Abbildung 47: Modelle laden	26
Abbildung 48: detect_mask-Funktion	27
Abbildung 49: Informationsleck	28
Abbildung 50: Generierung eines geheimen Schlüssels	29
Abbildung 51: Geheimer Schlüssel	29
Abbildung 52:create_cipher-Funktion.....	30
Abbildung 53: encrypt-Funktion.....	30
Abbildung 54: decrypt-Funktion.....	30
Abbildung 55: Live-Stream-Benachrichtigung	31
Abbildung 56: Ausführen des Empfängerprogramms.....	31
Abbildung 57: Ohne Maske	32
Abbildung 58: Mit einer Maske (rechts: FFP2-Maske, links: OP-Maske).....	32
Abbildung 59: Maskenerkennung für mehrere Personen	33
Abbildung 60: Bedecken von Mund und Nase.....	33
Abbildung 61: Unsachgemäßes Tragen der Maske	34
Abbildung 62: security.org-Webseite	34

Tabellenverzeichnis

Tabelle 1: Standard-MQTT-Ports	15
Tabelle 2: 1-aus-n mit $n=5$	20

1 Einleitung

„Werden Roboter uns irgendwann ersetzen?“ Dies ist eine Frage, die im 21. Jahrhundert heftig und mit sehr unterschiedlichen Bewertungen diskutiert wird.

Beim Internet der Dinge geht es nicht nur um vernetzte Geräte, sondern auch um die Informationen, die diese Geräte sammeln, und um die aussagekräftigen, unmittelbaren Erkenntnisse, die aus diesen Informationen gewonnen werden können. Die über das IoT gesammelten Informationen können für die Fernüberwachung genutzt werden, aber es muss auch sichergestellt werden, dass die Geräte ihre Aufgaben mit minimaler menschlicher Interaktion sehr effizient erfüllen können. Maschinelles Lernen bietet die Möglichkeit, ein elektronisches Gerät dazu zu bringen, etwas zu tun, ohne dass es ausdrücklich aufgefordert wird. Standardmäßig macht ein elektronisches Gerät genau das, was ihm gesagt wird – nicht mehr und nicht weniger. Beim ML werden Algorithmen und Datensätze verwendet, damit ein Gerät diese Daten analysieren und auf bestimmte Aufgaben anwenden kann. Für IoT-Geräte hat dies den großen Vorteil, dass die Geräte lernen, Prädiktionen treffen und sich besser an die sich ständig verändernde Umgebung anpassen können.

Der folgende Bericht beschreibt einen Software-Prototyp, der aus der Ferne auf eine Kamera zugreift und in der Lage ist, menschliche Gesichter zu erkennen und festzustellen, ob der Mensch eine Gesichtsmaske trägt oder nicht. Der Bericht befasst sich mit dem Training eines ML-Modells und der Implementierung des Modells in das System.

Da der Zugriff auf das Kamerabild aus der Ferne erfolgt, werden in dem Bericht das verwendete Transportprotokoll und die Maßnahmen zur Sicherung der Übertragung des Kamerabildes erwähnt. Das Projekt deckt nur den Software-Teil des Projekts ab und verwendet bereits gebaute Hardware. Die Umsetzung dieses Projekts ist jedoch theoretisch nicht auf die in diesem Projekt verwendeten Hardware beschränkt.

2 Systemdesign

2.1 Anforderungen

2.1.1 Hardware

Für dieses Projekt werden 2 Computer als Sende- bzw. Empfangsgerät benötigt. Obwohl es möglich ist, dasselbe Gerät zum Senden und Empfangen von Daten zu verwenden, ist es ratsam, getrennte Geräte zu verwenden, um zu vermeiden, dass der Computer aufgrund der leistungsstarken importierten Bibliotheken hängen bleibt (siehe 2.1.2).

Beide Geräte sollten in der Lage sein, mit dem Internet verbunden sein, und zumindest das Sendergerät muss mit einer Kamera ausgestattet sein.

2.1.2 Software

Das Projekt wird auf Geräten durchgeführt, die die folgenden Software unterstützen:

1. Windows 10/11 Betriebssystem
2. Python 3.9.13

Die Beschreibungen in diesem Projekt werden daher unter Berücksichtigung dieser Softwareanforderungen erstellt.

2.1.3 Bibliotheken

Um die Programme auszuführen, müssen diese Bibliotheken installiert sein¹:

Sender-Programm:

```
8
9  import cv2
10 import paho.mqtt.client as mqtt
11 import base64
12 import time
13 import configparser
14 import subscriber
15 from cryptography.fernet import Fernet
16
```

Abbildung 1: Sender-Programm-Bibliotheken

¹ „import subscriber“ ist keine mit pip installierbare Bibliothek. Details dazu sind unter MQTT erwähnt.

Empfänger-Programm:

```

8
9     import base64
10    import cv2
11    import numpy as np
12    import paho.mqtt.client as mqtt
13    import configparser
14    import subscriber
15    import argparse
16    import imutils
17    import time
18    import os
19    from cryptography.fernet import Fernet
20
21    from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
22    from tensorflow.keras.preprocessing.image import img_to_array
23    from tensorflow.keras.models import load_model
24

```

Abbildung 2: Empfänger-Programm-Bibliotheken

Es gibt 2 requirements.txt-Dateien (requirements_sender.txt und requirements_receiver.txt) für die Installation der erforderlichen Bibliotheken in jedem Gerät. Dies kann durch Eingabe in das Kommandoterminal `pip install -r requirements_receiver.txt` ausgeführt werden.

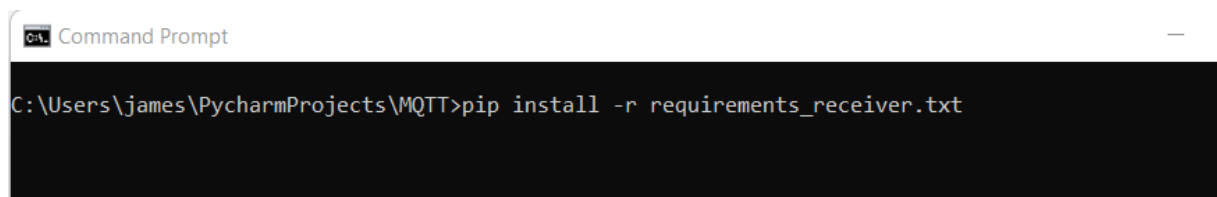


Abbildung 3: Bibliotheksinstallation über requirements.txt

Verschlüsselungs-Programm:

```

8
9     from configparser import ConfigParser
10    import base64
11    from cryptography.fernet import Fernet
12    import time
13

```

Abbildung 4: Verschlüsselungs-Programm-Bibliotheken

ML-Modell Trainingsprogramm:

```

6
7  from tensorflow.keras.preprocessing.image import ImageDataGenerator
8  from tensorflow.keras.applications import MobileNetV2
9  from tensorflow.keras.layers import AveragePooling2D
10 from tensorflow.keras.layers import Dropout
11 from tensorflow.keras.layers import Flatten
12 from tensorflow.keras.layers import Dense
13 from tensorflow.keras.layers import Input
14 from tensorflow.keras.models import Model
15 from tensorflow.keras.optimizers import Adam
16 from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
17 from tensorflow.keras.preprocessing.image import img_to_array
18 from tensorflow.keras.preprocessing.image import load_img
19 from tensorflow.keras.utils import to_categorical
20 from sklearn.preprocessing import LabelBinarizer
21 from sklearn.model_selection import train_test_split
22 from sklearn.metrics import classification_report
23 from imutils import paths
24 import matplotlib.pyplot as plt
25 import numpy as np
26 import argparse
27 import os
28

```

Abbildung 5: ML Trainingsprogramm-Bibliotheken

Um das ML-Modell zu trainieren, zu testen und einzusetzen, werden 2 Bibliotheken benötigt: tensorflow und scikit-learn. Scikit-learn wird durch Eingabe von `pip install -U scikit-learn` im Kommandofenster installiert, während Tensorflow durch Eingabe von `pip install tensorflow` installiert wird.

Windows Home Edition: Aufhebung der Pfadbegrenzung von 260 Zeichen

Für Dateipfade in Windows Home Edition gilt eine Beschränkung auf 260 Zeichen, die eine vollständige Installation von Tensorflow verhindern kann. Funktionen wie z.B. `tensorflow.keras.preprocessing.image` können möglicherweise nicht funktionieren, auch wenn man Tensorflow im Programm importieren kann. Dies kann im Registry Editor geändert werden.

Hinweis: Der Registrierungs-Editor ist ein mächtiges Werkzeug, und sein Missbrauch kann ein System instabil oder sogar funktionsunfähig machen. Dies ist ein ziemlich einfachen Trick und solange man sich an die Anweisungen unten hält, sollte keine Probleme auftauchen. Wenn bei der Installation von Tensorflow kein Fehler auftritt, dann ist dieser Schritt nicht notwendig.

Um diese Begrenzung zu entfernen:

1. Das Windows-Symbol drücken und regedit eingeben.

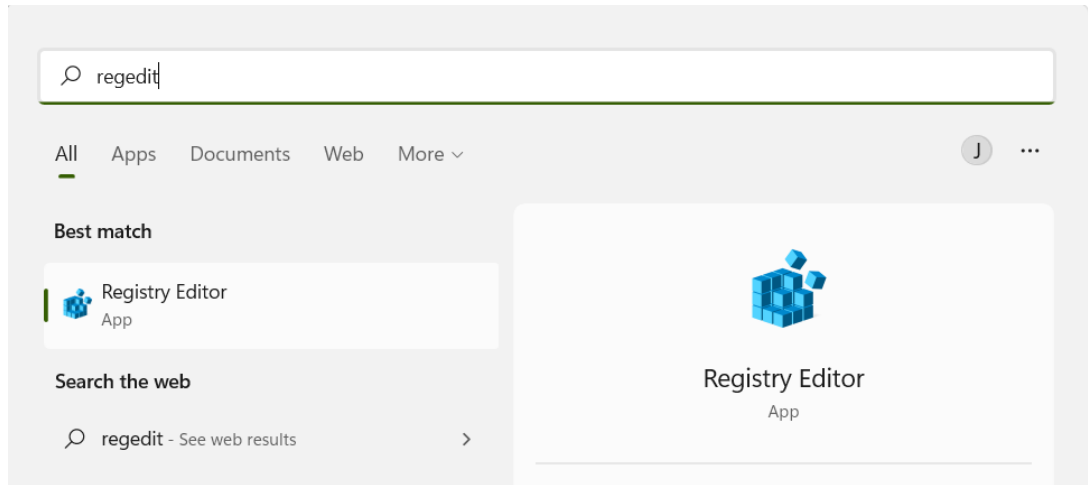


Abbildung 6: Registry Editor (1)

2. Dem folgenden Pfad folgen, um FileSystem zu erreichen.

Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\FileSystem

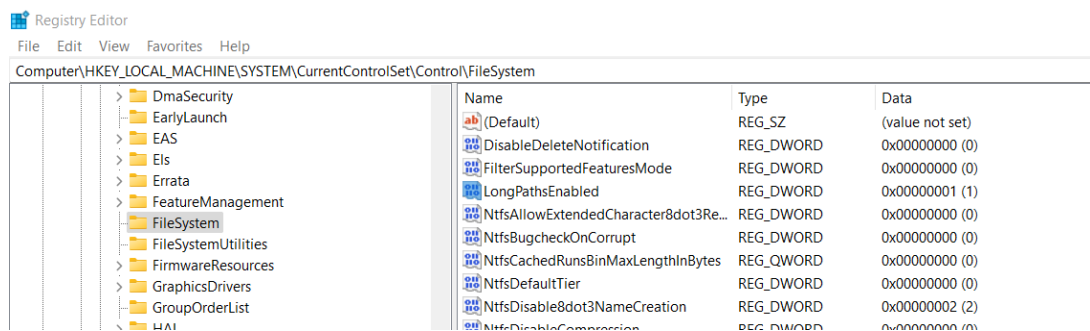


Abbildung 7: Registry Editor (2)

3. Doppelklick auf LongPathsEnabled.
4. Value data auf 1 setzen.

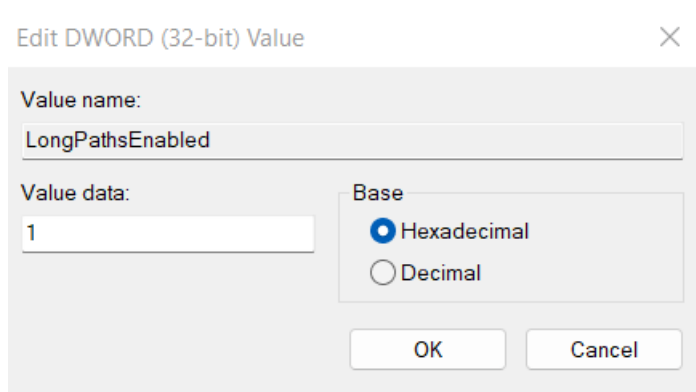


Abbildung 8: LongPathsEnabled Dialogfenster

5. OK drücken und Registry Editor schließen.

CUDA und cuDNN (Optional):

cuDNN ist eine GPU-beschleunigte Bibliothek mit Primitiven für DNNs. Tensorflow (sowie OpenCV) ist optimiert für die Arbeit mit GPUs. Um Tensorflow in die Lage zu versetzen, mit der GPU zu kommunizieren, ist es notwendig, CUDA und cuDNN zu installieren. Allerdings würde Tensorflow (sowie OpenCV) nach erfolgreicher Installation auch ohne CUDA oder cuDNN funktionieren.

CUDA ist mit bestimmten GPU-Treibern kompatibel²:

NVIDIA Driver Downloads

Select from the dropdown list below to identify the appropriate driver for your NVIDIA product. Help

Product Type:

Product Series:

Product:

Operating System:

Download Type:

Language:

?

Abbildung 9: NVIDIA-Treiber

Um die verfügbare GPU zu überprüfen, geht man zu Task-Manager > Leistung und klickt man auf GPU. In der oberen rechten Ecke steht der Name des verfügbaren GPU-Treibers.

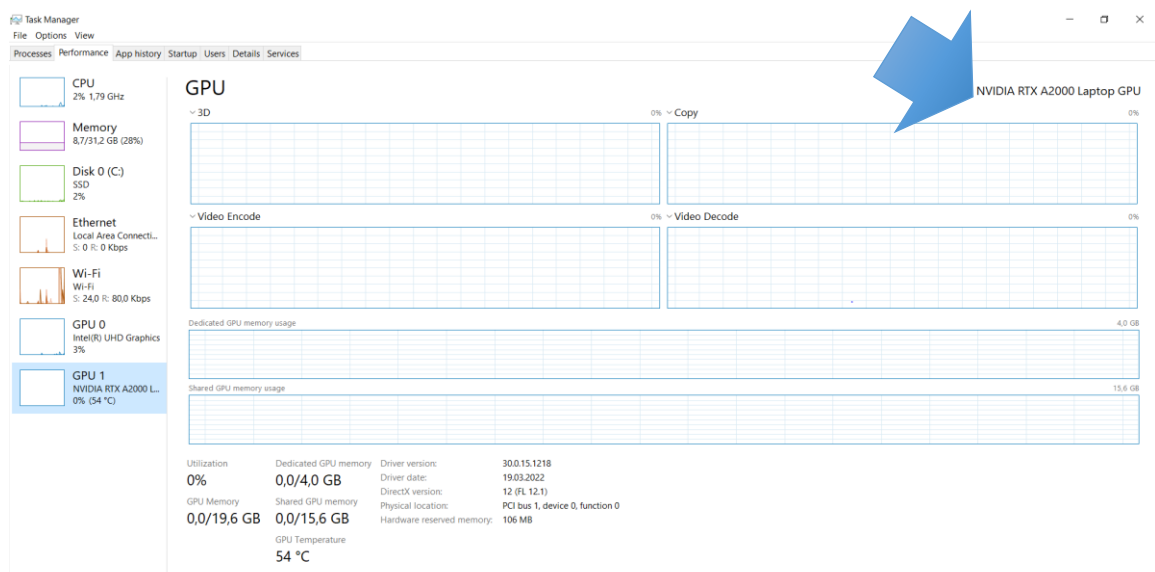


Abbildung 10: Task-Manager-Fenster

² Siehe Link: <https://www.nvidia.com/Download/index.aspx?lang=en-us>

Wenn der Grafikprozessor des Computers nicht auf der NVIDIA Webseite² aufgeführt ist (z.B. Intel® UHD Graphics), ist es nicht ratsam, mit diesem Schritt fortzufahren. Andere Grafikkarten (z.B. Intel-Grafikkarten) verwenden nicht das CUDA-Framework, sondern das OpenCL-Framework ("Open Computing Language", so dass sie weder mit CUDA noch cuDNN kompatibel sind.

Der 1. Schritt, der dafür notwendig ist, ist ein C-Compiler. CUDA ist für die Arbeit mit C, C+ und Fortran konzipiert, und im Gegensatz zu Linux, das gcc-Compiler („GNU Compiler Collection“) über das Terminalfenster ausführen kann, verfügt Windows standardmäßig nicht über einen geeigneten C-Compiler. Daher ist es notwendig, Visual Studio herunterzuladen³ und zu installieren. Die kostenlose Community-Edition ist ausreichend.

Starten Sie den Visual-Studio-Installer und gehen Sie zu den einzelnen Komponenten. Scrollen Sie nach unten zu den Compilern und wählen Sie die C++ Compiler.

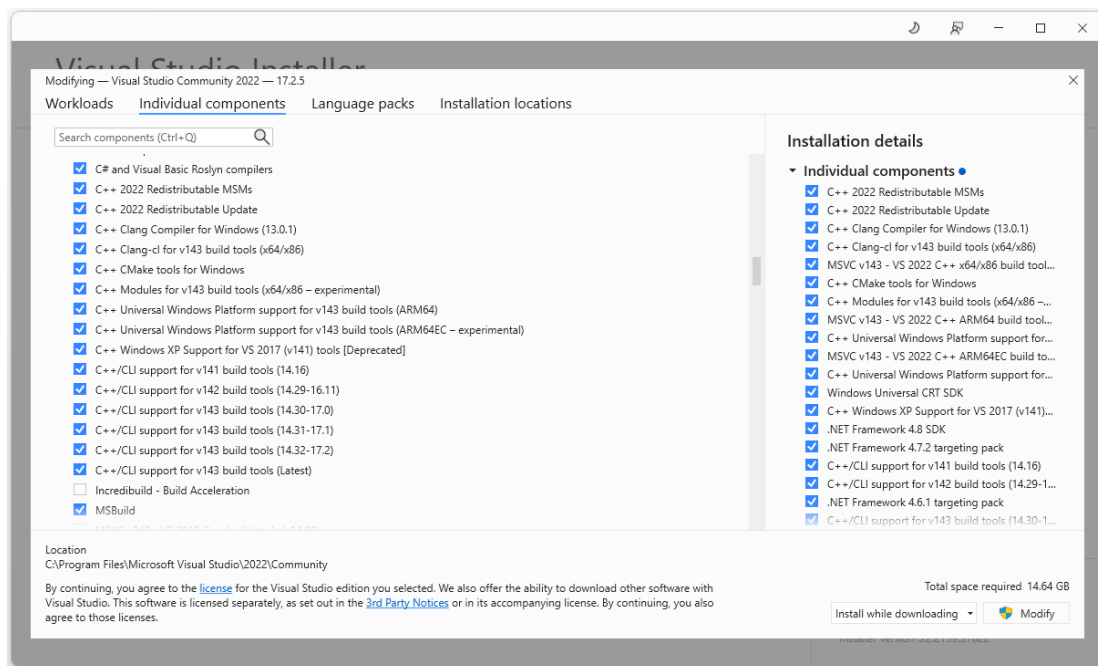


Abbildung 11: Visual-Studio-Installer

Auf der NVIDIA-CUDA-Toolkit-Website⁴ befindet sich den Button „Download-Now“, der zu einer Webseite führt, auf der der Benutzer nach dem Betriebssystem des Computers > der Architektur > der Version > dem Installer-Typ gefragt wird (siehe Abbildung 12). Das Installationsprogramm, das nach der Auswahl der korrekten Computerbeschreibung angezeigt wird, muss heruntergeladen werden.

³ Siehe Link: <https://visualstudio.microsoft.com/downloads/>

⁴ Siehe Link: <https://developer.nvidia.com/cuda-toolkit>

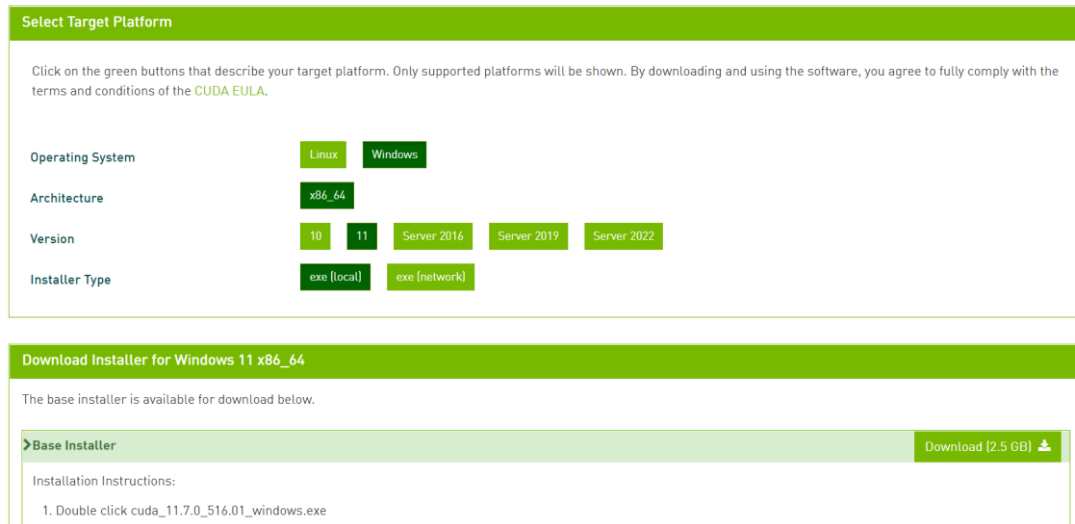


Abbildung 12: CUDA-Toolkit herunterladen

Klicken Sie auf die Standardeinstellungen und installieren Sie CUDA.

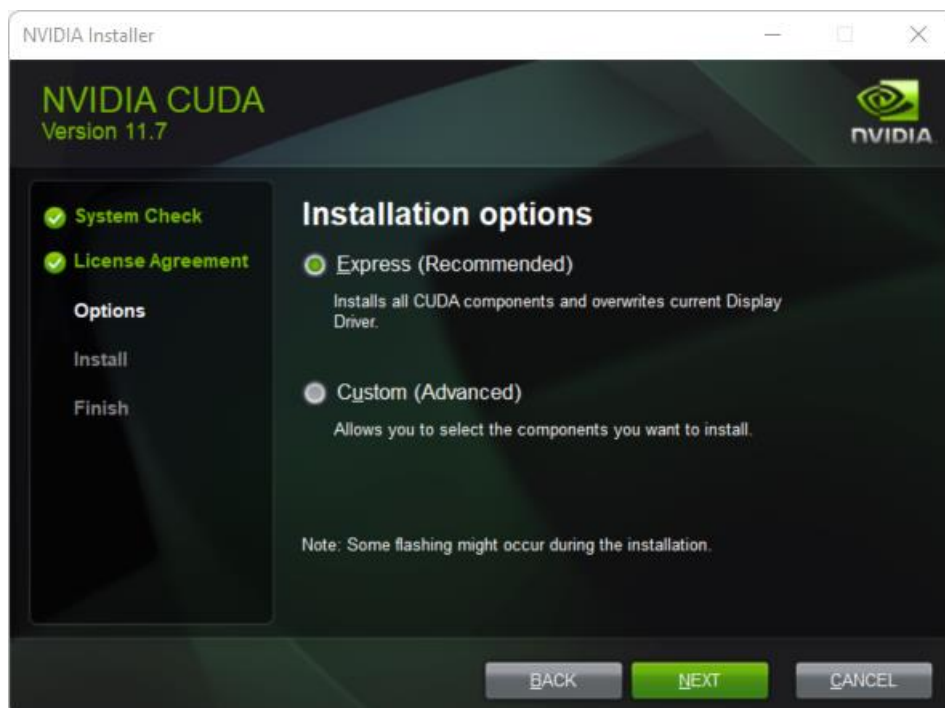


Abbildung 13: CUDA-Installation

Um die erfolgreiche Installation von CUDA zu bestätigen, gehen Sie zu Systemeigenschaften und klicken Sie auf Umgebungsvariablen. `CUDA_PATH` und `CUDA_PATH_<Versionsnummer>` sollten sichtbar sein.

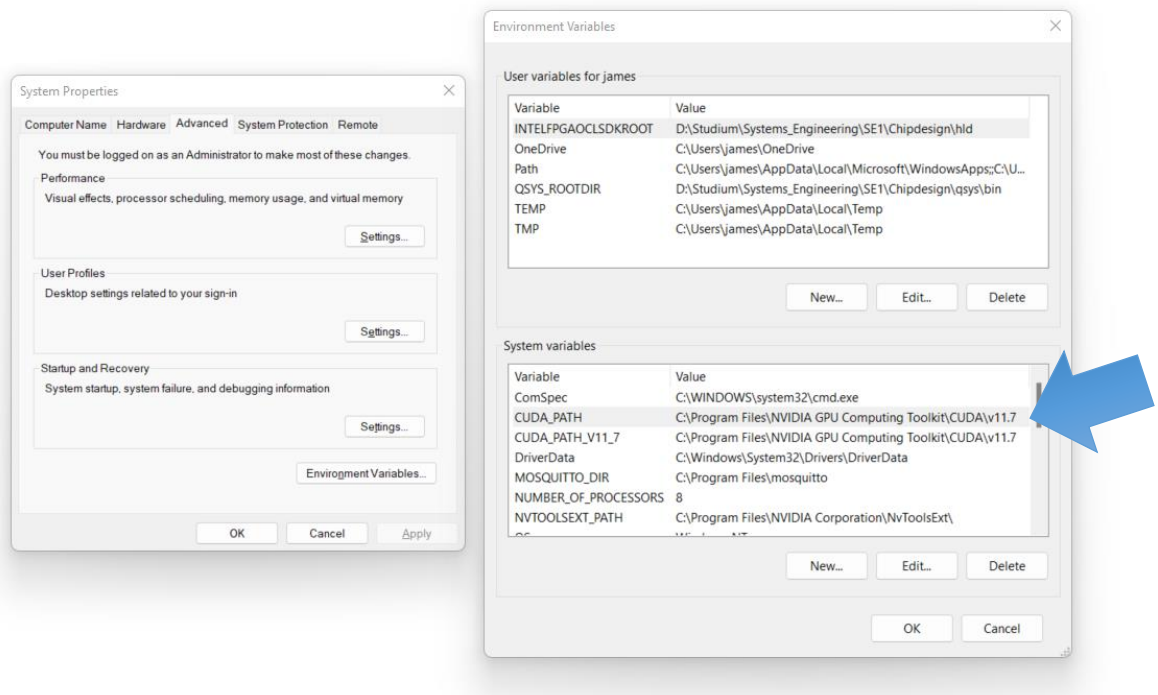


Abbildung 14: Umgebungsvariablen

Um cuDNN zu installieren, muss ein Konto registriert werden⁵. Laden Sie die cuDNN-Software herunter, die mit dem installierten CUDA kompatibel ist.

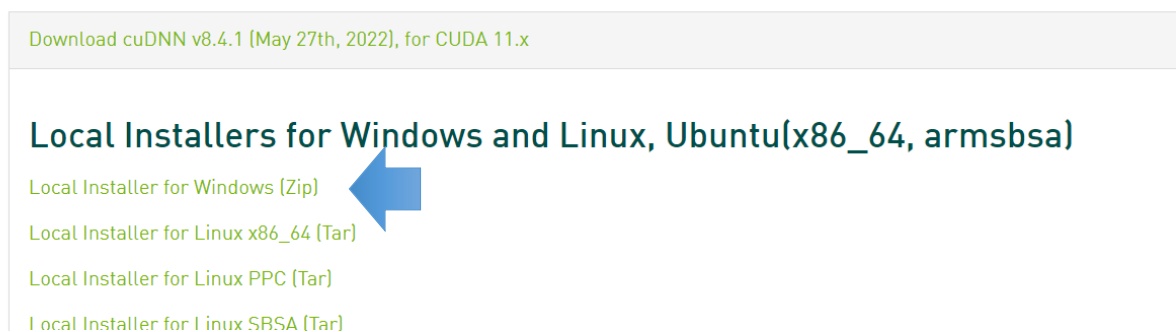


Abbildung 15: cuDNN-Download

Kopieren Sie die heruntergeladene cuDNN-Dateien in den Ordner unter folgendem Pfad:

C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\<Versionsnummer>

⁵ Siehe Link: <https://developer.nvidia.com/cudnn-download-survey>

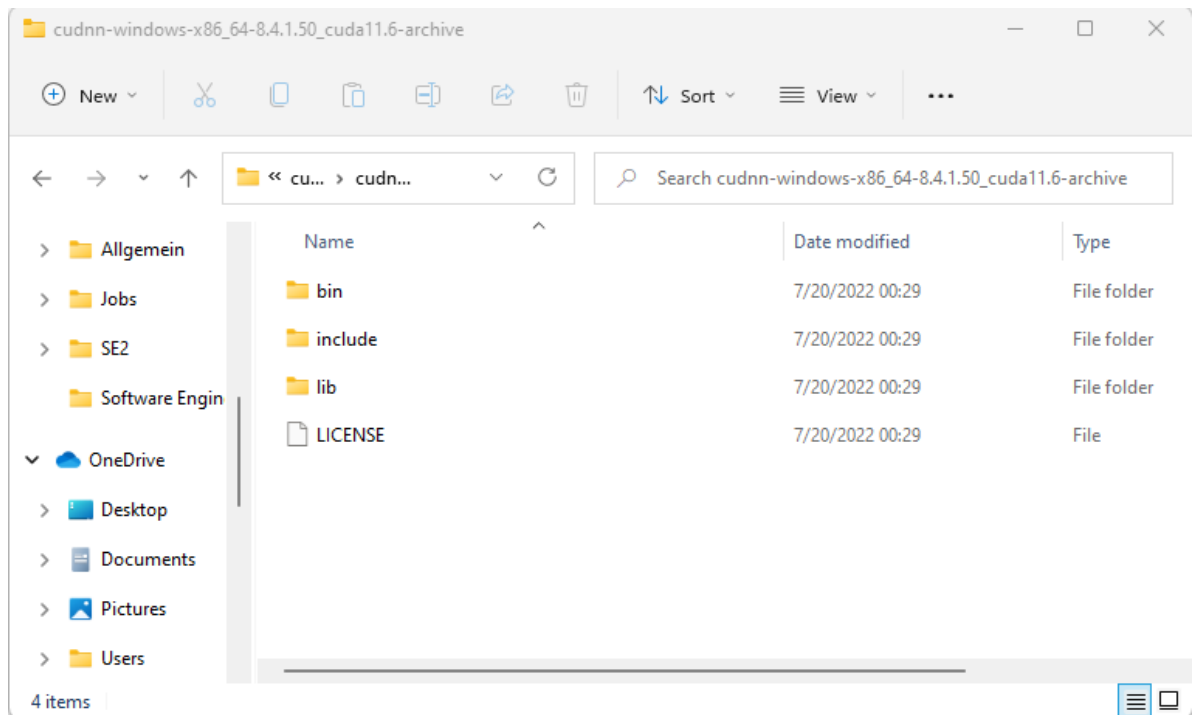


Abbildung 16: cuDNN-Ordner

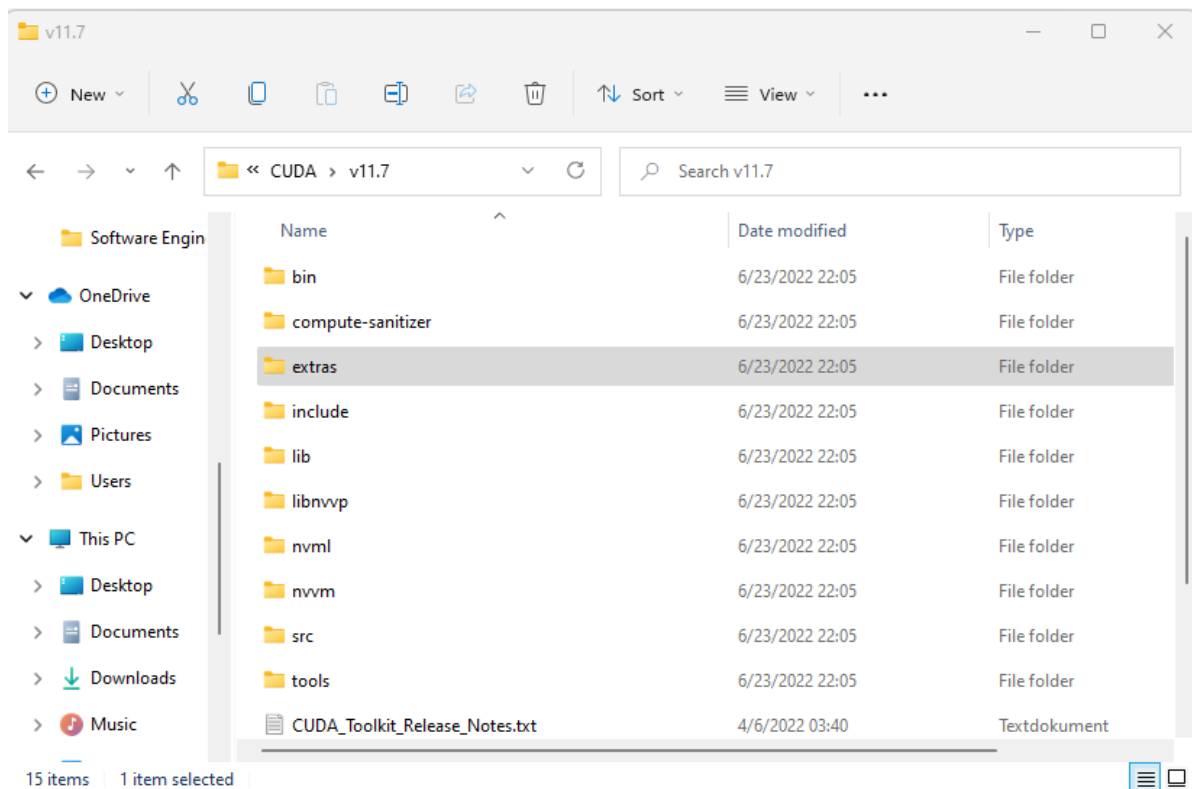


Abbildung 17: CUDA-Ordner (v11.7)

Setzen Sie sich schließlich die Systemvariable mit dem Namen CUDNN, um auf die Ordner bin, include und lib zu verweisen, die in das CUDA-Verzeichnis kopiert wurden.

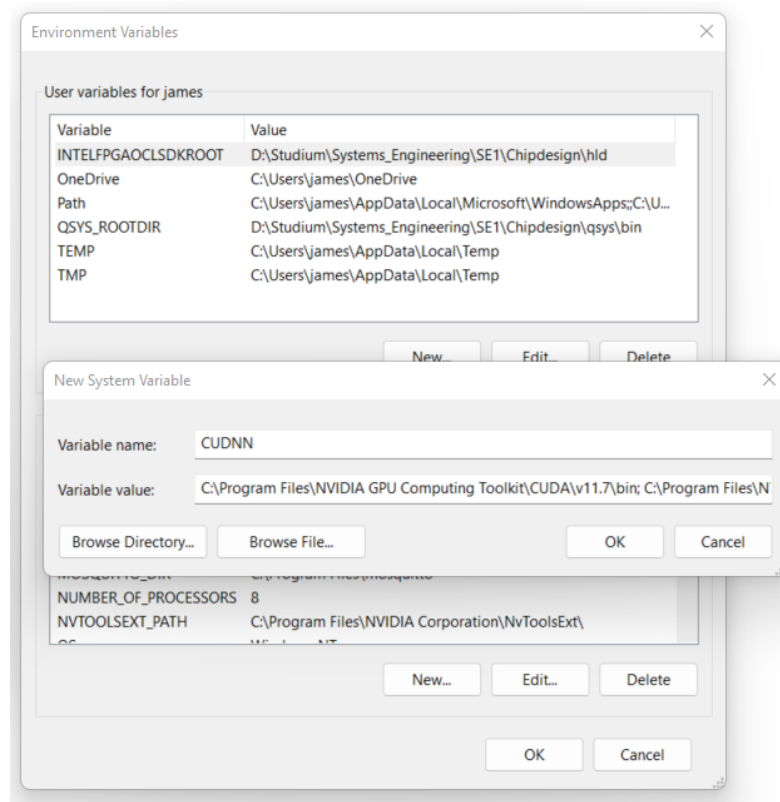


Abbildung 18: CUDNN-Pfad

Tensorflow sollte nun mit der GPU funktionieren. Um zu sehen, ob es funktioniert, kann man Tensorflow in Python importieren.

```
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow as tf
>>>
```

Abbildung 19: Tensorflow importieren (mit CUDA & cuDNN)

Tensorflow ohne CUDA funktioniert weiterhin, zeigt aber eine Warnung an, dass keine CUDA-Treiber gefunden werden.

```
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow as tf
2022-07-20 00:44:20.514167: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'cudart64_110.dll'; dlderror: cudart64_110.dll not found
2022-07-20 00:44:20.514459: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if you do not have a GPU set up on your machine.
>>>
```

Abbildung 20: Tensorflow importieren (ohne CUDA noch cuDNN)

Die Ausführungsgeschwindigkeit ist zwar etwas geringer, aber das hat keinen Einfluss auf die Gesamtleistung dieses Projekts.

2.2 Datenfluss

Anhand eines vereinfachten Flussdiagramms kann die Funktionsweise des Systems erklärt werden:

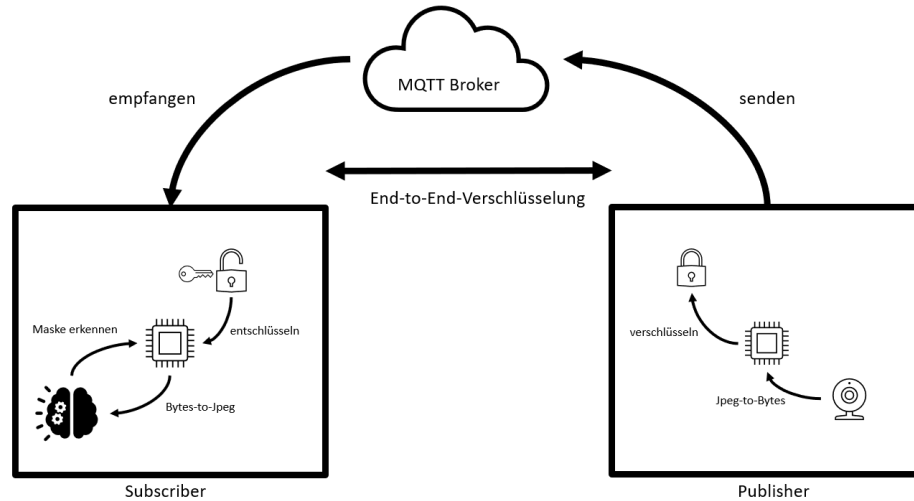


Abbildung 21: Flussdiagramm

Pseudocode:

Publisher

Video von der Webcam **empfangen**

Verbindung zum Broker

Videodaten in Bytes **umwandeln**

Bytes **verschlüsseln**

verschlüsselte Bytes **veröffentlichen („publish“)**

Subscriber

mit Broker **verbinden**

beim Publisher **abonnieren**

empfangene Bytes **entschlüsseln**

Bytes in Videodaten **umwandeln**

Gesichtsordnung und Maskenvorhersage **berechnen**

Video **anzeigen**

3 Systemarchitektur

3.1 MQTT

MQTT ist ein leichtgewichtiges Publish-Subscribe-Netzwerkprotokoll für die Kommunikation von Maschine zu Maschine. Es eignet sich ideal für die Verbindung entfernter Geräte mit geringem Code-Fußabdruck und minimaler Netzwerkbandbreite. [4]

Das MQTT-Protokoll definiert zwei Arten von Netzwerkeinheiten:

- einen Nachrichtenbroker (MQTT-Broker) und
- eine Reihe von Clients.

Um die Kommunikation zwischen einem MQTT-Broker und -Client herzustellen, muss zunächst eine MQTT-Bibliothek importiert werden. Für dieses Projekt wird die populäre paho-mqtt-Bibliothek importiert: `import paho.mqtt.client as mqtt`⁶ (siehe Abbildung 1 und 2).

MQTT-Broker:

Ein MQTT-Broker ist ein Server, der alle Nachrichten von den Clients empfängt und dann die Nachrichten an die entsprechenden Zielclients weiterleitet. Der Broker fungiert als Postamt, denn MQTT-Clients verwenden nicht die direkte Verbindungsadresse des Empfängers, sondern die Betreffzeile „Topic“.

Für dieses Projekt gibt es 2 Broker, die verwendet werden können, um Informationen von einem MQTT-Client zum anderen zu übertragen:

- HiveMQ und
- Mosquitto

Diese 2 Broker können austauschbar verwendet werden, da der Verbindungsprozess derselbe ist.

MQTT-Client:

Ein MQTT-Client ist ein beliebiges Gerät, auf dem eine MQTT-Bibliothek läuft und das sich über ein Netzwerk mit einem MQTT-Broker verbindet. Clients können Daten durch „*Publish*“ oder „*Subscribe*“ senden bzw. empfangen.

⁶ Die Bibliothek kann über die `requirements_sender.txt`, `requirements_receiver.txt` oder separat über `pip install paho-mqtt` installiert werden.

Für dieses Projekt wird eine One-To-One-Verbindung verwendet. Das bedeutet, dass nur ein Publisher und ein Subscriber verwendet werden.

Ablauf:

Die Kommunikation über MQTT funktioniert wie folgt:

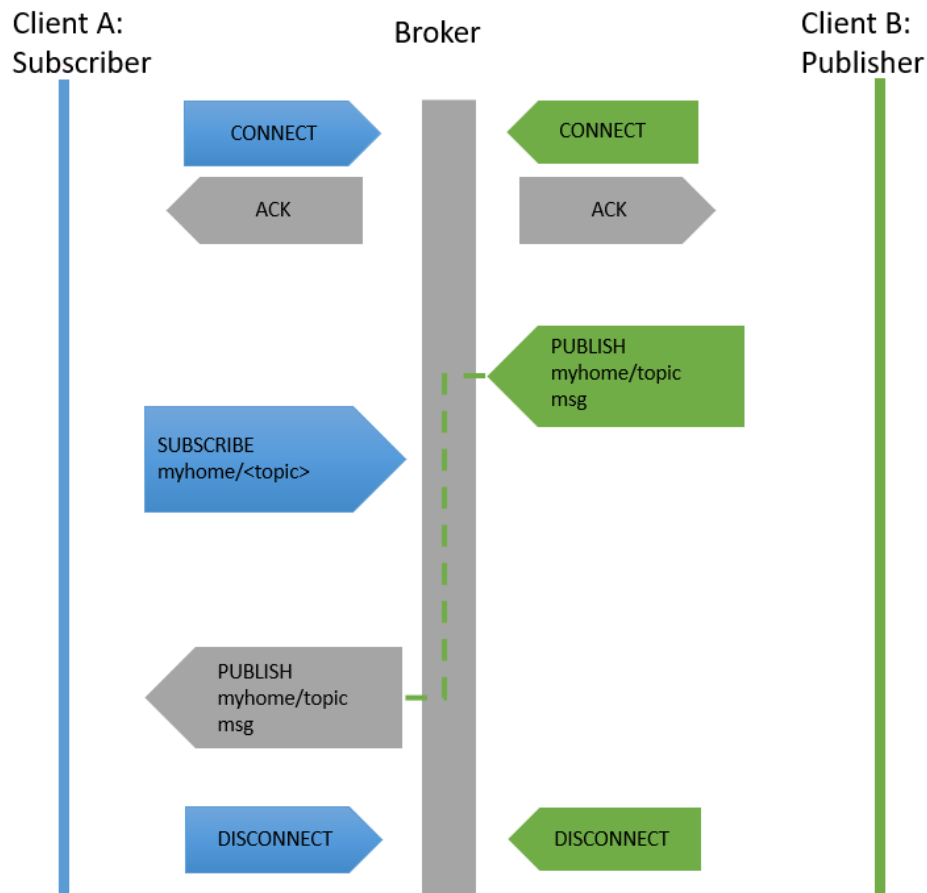


Abbildung 22: MQTT-Kommunikation

- **Publisher:** Eine VideoStream-Klasse wird mit den folgenden Konstruktorparametern erstellt:

```

25
26 class VideoStream(object):
27     def __init__(self):
28         self.__video = cv2.VideoCapture(0)
29         self.__client = None
30         # self.__MQTT_BROKER = "test.mosquitto.org"
31         self.__MQTT_BROKER = "broker.hivemq.com"
32         self.__PORT = 1883
33         # self.__PORT = 8883
34         self.__MQTT_SEND_VIDEO = subscription.topic["subscribe video"]
35         self.__cipher = self.__create_cipher()
36
  
```

Abbildung 23: VideoStream-Konstruktor (Publisher)

Dann wird die connect-Funktion aufgerufen, ein MQTT-Client-Objekt erstellt und die Funktion `self.__client.connect` mit den Eingabeparametern `self.__MQTT_Broker` und die Portnummer aufgerufen.

```

56
57
58 def connect(self):
59     if self.__client is None:
60         self.__client = mqtt.Client()
61         self.__client.connect(self.__MQTT_BROKER, self.__PORT)
62

```

Abbildung 24: connect-Funktion

Für MQTT stehen 2 Standardports zur Verfügung:

PORT	
1883:	bei der IANA für die Verwendung mit MQTT
8883:	registriert, um MQTT über SSL zu verwenden

Tabelle 1: Standard-MQTT-Ports

Die gewonnenen Videodaten werden dann über `self.__client.publish()`.

```

77 encrypted_video_data = self.__encrypt(video_data, self.__cipher)
78 # secure_video_data = base64.b64encode(encrypted_video_data)
79 self.__client.publish(self.__MQTT_SEND_VIDEO, encrypted_video_data)
80

```

Abbildung 25: publish-Funktion

- **Subscriber:** Eine VideoStream-Klasse wird ebenfalls innerhalb des Subscriber-Programms mit den folgenden Konstruktorparametern erstellt:

```

25
26 class VideoStream(object):
27     def __init__(self,):
28         self.__frame = np.zeros((240, 320, 3), np.uint8)
29         self.__MQTT_RECEIVE = subscription.topic["subscribe video"]
30         # self.__MQTT_BROKER = "test.mosquitto.org"
31         self.__MQTT_BROKER = "broker.hivemq.com"
32         self.__PORT = 1883
33         # self.__PORT = 8883
34         self.__args = None
35         self.__cipher = self.__create_cipher()
36

```

Abbildung 26: VideoStream-Konstruktor (Subscriber)

Die Funktion `connect` wird aufgerufen, wobei zunächst ein MQTT-Client-Objekt erstellt wird. Die Funktion `on_connect` wird aufgerufen, bei der der Client das MQTT-Topic abonniert und eine Verbindung zum Broker und Port herstellt.

```

51
52 def __connect(self) -> mqtt:
53     def on_connect(client, userdata, flags, rc):
54         print("Connected with result code " + str(rc))
55         client.subscribe(self.__MQTT_RECEIVE)
56
57     client = mqtt.Client()
58     client.on_connect = on_connect
59     client.connect(self.__MQTT_BROKER, self.__PORT)
60     return client
61

```

Abbildung 27: connect-Funktion

Anschließend wird die Funktion `read_message` aufgerufen, in der die empfangenen Message-Payload in nützliche Information umgewandelt werden, die der Client benötigt.

```

60
61
62
63
64
65
66
67
68
69
def __read_message(self, client: mqtt):
    def on_message(client, userdata, msg):
        token = self.__decrypt(msg.payload)
        img = base64.b64decode(token)
        npimg = np.frombuffer(img, dtype=np.uint8)
        self.__frame = cv2.imdecode(npimg, 1)

    client.on_message = on_message

```

Abbildung 28: read_message-Funktion

3.2 OpenCV

OpenCV ist eine Programmbibliothek mit Algorithmen für die Bildverarbeitung und Computer Vision. Damit lassen sich Bilder und Videos verarbeiten, um Objekte, Gesichter oder sogar die Handschrift eines Menschen zu erkennen.

Wenn OpenCV mit verschiedenen Bibliotheken wie NumPy integriert wird, kann Python die OpenCV-Array-Struktur für die Analyse verarbeiten. Zur Identifizierung von Bildmustern und ihren verschiedenen Merkmalen verwenden wir den Vektorraum und führen mathematische Operationen mit diesen Merkmalen durch.

OpenCV wird sowohl im Sender- als auch im Empfängerprogramm verwendet, um die folgenden Aufgaben zu erfüllen:

Sender

OpenCV wird verwendet, um einen Live-Video vom Sender zu erfassen, Bildformate in Streaming-Daten zu kodieren und sie im Cache zu speichern.

Nach dem Importieren der OpenCV-Bibliothek durch `import cv2` (siehe Abbildung 1) wird ein Video-Capture-Objekt mit dem Eingangsparameter als Geräteindex oder dem Namen einer Videodatei erstellt (siehe Abbildung 23). Ein Geräteindex wird verwendet, um die Kameraquelle zu spezifizieren. Im Normalfall ist nur 1 Kamera angeschlossen, daher übergibt man einfach 0 (oder -1) als Parameter. Wenn mehr als eine Kamera an das Gerät angeschlossen ist (z.B. eine externe Webcam), kann die Kamera durch Übergabe von 1 usw. ausgewählt werden, je nach der Anzahl der verfügbaren Kameras.

Die Funktion `read()` erfasst die Videobilder, die dann an die Funktion `imencode()` übergeben werden, wo die Bilder in Streaming-Daten umgewandelt und vorübergehend als Puffer gespeichert werden.

```

61
62 def send(self):
63     try:
64         print("Streaming Data")
65         while True:
66             start = time.time()
67             _, frame = self.__video.read()
68
69             _, buffer = cv2.imencode('.jpg', frame)
70             video_data = base64.b64encode(buffer)
71
72             encrypted_video_data = self.__encrypt(video_data, self.__cipher)
73             self.__client.publish(self.__MQTT_SEND_VIDEO, encrypted_video_data)
74
75             end = time.time()
76             t = end - start
77             fps = 1 / t
78             # print("FPS: " +str(fps))
79         except:
80             self.__video.release()
81             print("Stopping Video Transmission")
82             self.__client.disconnect()
83

```

Abbildung 29: send-Funktion

Receiver

Im Empfängerprogramm liest OpenCV die Bilddaten aus dem entschlüsselten Speichercache und konvertiert sie mit der Funktion `imdecode` in das Bildformat (siehe Abbildung 28).

Die erhaltenen Bilder werden mit einem ML-Modell analysiert, das auch einige OpenCV-Funktionen implementiert (siehe 3.3), bevor die Funktion `imshow` aufgerufen wird, die das fertige Video anzeigt.

3.3 Maschinelles Lernen

Nachdem die Bilddaten vom Sender zum Empfänger übertragen wurden, können sie nun in ein Bildformat decodiert und angezeigt werden. Die angezeigten Informationen können dann verwendet werden, um aus der Ferne Entscheidungen zu treffen, die auf den Beobachtungen des Videos basieren.

Dies ist zwar sehr hilfreich, erfordert aber einen Bediener, der das System während der gesamten Betriebszeit betreut. Wenn eine Überwachung rund um die Uhr erforderlich ist, kann dies kostspielig sein, was die menschliche Arbeitskraft angeht. Glücklicherweise ist es möglich, das System so zu trainieren, dass es Objekte von Interesse erkennt und intelligente und genaue Entscheidungen ohne menschliche Hilfe trifft.

In diesem Projekt wird ein Modell trainiert, um menschliche Gesichter mit und ohne Maske in dem vom Sender übertragenen Live-Video zu erkennen. Der Ansatz für das Training des ML-Modells wird durch „*supervised learning*“ erfolgen, da die zu suchenden Muster bereits bekannt sind.

3.3.1 Design des Trainingsprogramms

Dataset

Innerhalb des Ordners dataset gibt es 2 Ordner mit den Bezeichnungen with_mask beziehungsweise without_mask, die sowohl Bilder mit maskierten als auch mit nicht maskierten Personen enthalten (siehe Anhang C).

Die Bilder sind eine Kombination aus Bildern von Kaggle, Google-Bildern und persönlichen Bildern. Die Bilder werden zum Trainieren des ML-Modells verwendet.

In der Informatik gilt die Redewendung *"Garbage in, garbage out"* (GIGO), denn eine falsche Beschriftung des Dataset oder das Ablegen der Bilder im falschen Ordner (unsinnige Eingabe) führt zu fehlerhaften Ergebnissen (unsinnige Ausgabe). Daher ist beim Hinzufügen der Daten Vorsicht geboten, und Bilder, die einige Personen mit Masken und einige ohne Masken enthalten, sollten am besten vermieden werden.

Daten-Vorverarbeitung

JPEG- und PNG-Bilddaten können nicht direkt zum Trainieren des Modells verwendet werden und müssen daher in Datenarrays umgewandelt werden.

Um die Konvertierung durchführen zu können, müssen die notwendigen Bibliotheken installiert sein (siehe Abbildung 5).

Um das Auffinden der verschiedenen benötigten Pfade zu vereinfachen, werden Argument-Parser verwendet, um den Pfad zu liefern.

```

28
29     # Argument parser for the dataset (with or without mask),
30     # loss/accuracy plot & trained mask-detector model
31     ap = argparse.ArgumentParser()
32     ap.add_argument("-d", "--dataset", required=True,
33                     help="path to input dataset")
34     ap.add_argument("-p", "--plot", type=str, default="plot.png",
35                     help="path to output loss/accuracy plot")
36     ap.add_argument("-m", "--model", type=str,
37                     default="mask_detector.model",
38                     help="path to output face mask detector model")
39     args = vars(ap.parse_args())
40

```

Abbildung 30: Argument-Parser

Nach der Übergabe des Pfads zum Dataset-Ordner an den Argumentparser wird die Funktion `paths.list_images` aus `imutils` verwendet, um den Pfad aller Bilder im Dataset-Ordner als Liste zu erhalten. Wenn der angegebene Ordner keine Bilder enthält, gibt die Funktion eine leere Liste zurück. Es wird auch eine leere Liste für `data` und `labels` erstellt.


```

50
51 imagePaths = list(paths.list_images(args["dataset"]))
52 data = []
53 labels = []
54
55 # loop over the image paths
56 for imagePath in imagePaths:
57     # extract the class label from the filename
58     label = imagePath.split(os.path.sep)[-2]
59
60     # load the input image (224x224) and preprocess it
61     image = load_img(imagePath, target_size=(224, 224))
62     image = img_to_array(image)
63     image = preprocess_input(image)
64
65     data.append(image)
66     labels.append(label)
67
68 data = np.array(data, dtype="float32")
69 labels = np.array(labels)
70

```

Abbildung 31: Initialisierung der Bilddaten

Mithilfe einer for-Schleife werden die Labels extrahiert, um die Bilder zu kategorisieren. In diesem Fall ist dies der Name des Ordners, in dem die Bilder gespeichert sind (with_mask, without_mask).

Die Keras-Funktion `load_img` wird verwendet, um alle Bilder in der gleichen Höhe und Breite (224x224) zu laden, `img_to_array` konvertiert die Bilder in ein Array und da die Applikation `mobilnet_v2` verwendet wird (wird später erklärt), wird die Funktion `preprocess_input` aufgerufen. Die vorverarbeiteten Bilddaten werden an die `data`-Liste und ihre Kategorie an die `labels`-Liste angehängt.

Die Listen werden dann in Arrays umgewandelt, da dies der akzeptierte Datentyp für das Training der Daten ist.

One-Hot-Kodierung

Da die `labels` immer noch kategorisch sind, was für die meisten ML-Algorithmen nicht nützlich ist, müssen sie in numerische Daten umgewandelt werden. Hierfür wird die One-Hot-Kodierung verwendet.

One-Hot (auch 1-aus-n-Code) ist eine Gruppe von Bits, bei der nur die Kombinationen von Werten zulässig sind, bei denen ein einziges Bit „TRUE“ (1) ist und alle anderen „FALSE“ (0) sind.

1-aus-n-Code mit $n=5$ kann wie folgt dargestellt werden:

Dezimal	1-aus-5-codiert	Binär
0	00001	000
1	00010	001
2	00100	010
3	01000	011
4	10000	100

Tabelle 2: 1-aus-n mit n=5

Obwohl es möglich ist, die Datensätze als `with_mask=0` und `without_mask=1` zu kennzeichnen, interpretiert ML-Algorithmen oft fälschlicherweise, dass es eine Hierarchie in den Kennzeichnungen gibt. Durch diese Voreingenommenheit würde `without_mask` stärker bevorzugt werden, da $1 > 0$, obwohl idealerweise beide labels für den Dataset gleich wichtig sind. Daher werden die Kategorien durch One-Hot-Kodierung "binarisiert", um dieses mögliche Problem zu vermeiden.

```

71
72     # one-hot encoding for the labels
73     lb = LabelBinarizer()
74     labels = lb.fit_transform(labels)
75     labels = to_categorical(labels)
76

```

Abbildung 32: One-Hot-Kodierung

Nachfolgend werden die Schritte zur Durchführung der One-Hot-Codierung beschrieben:

1. Ein Objekt der Klasse `LabelBinarizer`, die aus `sklearn.preprocessing` importiert wurde (siehe Abbildung 5).
2. Die Funktion `fit_transform` aufrufen und das Array `labels` als Eingabeparameter angeben. Dadurch erhält `with_mask` den Wert 0 und `without_mask` den Wert 1.
3. Die Funktion `to_categorical` von Keras (siehe Abbildung 5) aufrufen, um eine One-Hot-Kodierung durchzuführen. Dies erzeugt ein Numpy-Array mit den Werten $[1 \ 0]^T$ für 0 und $[0 \ 1]^T$ für 1.

Train-Test-Split

```

77
78     # 80% -> Training
79     #20% -> Testing
80
81     (trainX, testX, trainY, testY) = train_test_split(data, labels,
82     test_size=0.20, stratify=labels, random_state=42)
83

```

Abbildung 33: Train-Test-Split-Funktion

⁷ [LSB, MSB]

Mit Hilfe des Dataset wird die Funktion `train_test_split` verwendet, um zu simulieren, wie ein Modell bei neuen/ungesehenen Daten abschneiden würde.

Die Daten werden in 2 aufgeteilt: Trainingsmenge und Testmenge. Die Daten werden nach dem Zufallsprinzip aufgeteilt und in 80 % der Daten für das Training und 20 % für die Tests aufgeteilt. Die Prozentsätze können nach Belieben geändert werden, indem der Eingabewert für `test_size` variiert wird.

Die Daten werden nach den labels geschichtet (`stratify=labels`), um Verzerrungen zu vermeiden, da es 2190 Bilder für `with_mask` und 1963 Bilder für `without_mask` gibt. Zusätzlich erhält `random_state` den Wert 42, um die Ergebnisse reproduzierbar zu machen. Dies ist jedoch optional und nur im akademischen Bereich für die Benotung oder in der Praxis für den Nachweis eines Konzepts wichtig.

Training

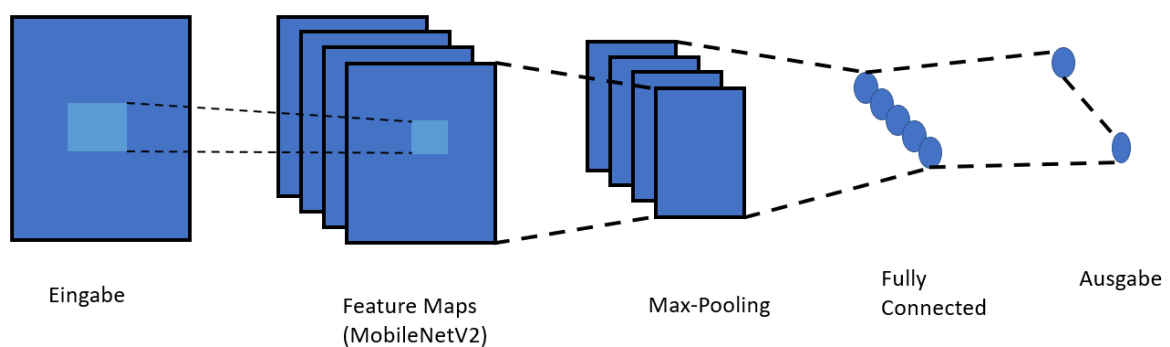


Abbildung 34: Convolutional Neural Networks

Der Trainingsprozess des ML-Modells ähnelt dem normaler Convolutional Neural Networks (CNNs), mit dem kleinen Unterschied, dass die Faltungen, die bei den Feature Maps durchgeführt werden, durch MobileNetV2 ersetzt werden.

MobileNetV2 ist eine neuronale Faltungsnetzwerk-Architektur, die sich für den Einsatz auf mobilen Geräten eignet. MobileNetV2 ist schneller als CNNs und benötigt weniger Parameter, wodurch es sich besser für den Einsatz in mobilen Geräten wie Raspberry Pi. Der Nachteil ist jedoch, dass MobileNetV2 im Vergleich zu herkömmlichen CNNs weniger genau ist. [2]

Um den Trainingsprozess zu beginnen, werden die folgenden Hyperparameter initialisiert.

```

41
42 # initialize the initial learning rate, number of epochs to train for,
43 # and batch size
44 INIT_LR = 1e-4
45 EPOCHS = 20
46 BS = 32
47

```

Abbildung 35: Initialisierung der Trainings-Parameter

Die Initial-Lernrate bestimmt die Schrittgröße bei jeder Iteration, während es sich zu einer Funktion mit minimalem Verlust bewegt. Daher ist die Genauigkeit um so besser, je kleiner die Lernrate ist. Epoche = 20 bedeutet, dass die Trainingsdaten den Algorithmus 20 Mal durchlaufen. Die internen Modellparameter des Datensatzes werden mit jeder Epoche aktualisiert. Die Batch-Größe bezieht sich auf die Anzahl der Trainingsdaten, die in einer Iteration verwendet werden. Je größer die Batch-Größe, desto höher ist der Trainingsverlust⁸. Die Werte können nach Belieben angepasst werden, um unterschiedliche Ergebnisse zu erzielen.

Der Image-Data-Generator von Keras wird für die Erzeugung von Batches verwendet, die die Daten von Tensor-Bildern enthalten, und wird im Bereich der Echtzeit-Datenerweiterung eingesetzt. Mit dem Image-Data-Generator in Keras kann eine Schleife über die Daten in Batches gezogen werden. Es gibt verschiedene Eingabeargumente für die Klasse des Bilddatengenerators, die helfen, das Verhalten der Datengenerierung zu definieren.

```

86
87 # construct the training image generator for data augmentation
88 aug = ImageDataGenerator(
89     rotation_range=20,
90     zoom_range=0.15,
91     width_shift_range=0.2,
92     height_shift_range=0.2,
93     shear_range=0.15,
94     horizontal_flip=True,
95     fill_mode="nearest")
96

```

Abbildung 36: Der Image-Data-Generator

Die Bilder können auf maximal 20 gedreht und auf 0,15 gezoomt werden, und die neuen Batches werden aus den geänderten Bilddaten erstellt, wodurch die Genauigkeit des trainierten Modells erhöht wird.

```

97
98 # load the MobileNetV2 network
99 # include pretrained imagenet weights
100 baseModel = MobileNetV2(weights="imagenet", include_top=False,
101     input_tensor=Input(shape=(224, 224, 3)))
102
103
104 # construct the head before including to the base
105 headModel = baseModel.output
106 headModel = AveragePooling2D(pool_size=(7, 7))(headModel)
107 headModel = Flatten(name="flatten")(headModel)
108 headModel = Dense(128, activation="relu")(headModel)
109 headModel = Dropout(0.5)(headModel)
110 headModel = Dense(2, activation="softmax")(headModel)
111
112 # create final model baseModel -> headModel
113 model = Model(inputs=baseModel.input, outputs=headModel)
114

```

Abbildung 37: Erstellung des Modells

Das Modell wird zunächst durch die Erstellung eines Objekts für das Base-Modell aus der Klasse MobileNetV2 erstellt. Bei den Eingabeparametern wird das trainierte

⁸ Nur wenn die Initial-Lernrate konstant bleibt

imagenet als Gewichte übergeben, `include_top` wird auf `False` gesetzt, da das Head-Modell später erstellt wird, bevor es in das Modell aufgenommen wird, und der Tensoreingabe werden die Eigenschaften der erzeugten Bilddaten gegeben: `224x224` (siehe Abbildung 31) und `3` ist gesetzt, da farbige Bilder verwendet werden (RGB-Farbcode).

Um die Fully-Connected-Layer zu erstellen, wird das Head-Modell zunächst durch Übergabe der Ausgabe des Base-Modells erstellt. `AveragePooling2D` verkleinert die Eingabe entlang ihrer räumlichen Dimensionen (Höhe und Breite), indem der Durchschnittswert über ein Eingabefenster (mit der durch `pool_size` definierten Größe) für jeden Kanal der Eingabe ermittelt wird.

Die Layers werden abgeflacht, bevor die Dense-Layer mit 128 Neuronen hinzugefügt und mit der ReLu-Funktion aktiviert wird, die für die nichtlineare Aktivierung von mehrschichtigen neuronalen Netzen nützlich ist. Die Dropout-Layer setzt die Eingabeeinheiten bei jedem Schritt (`rate=0,5`) während des Trainings zufällig auf 0, was eine Overfitting verhindert. Eine Dense-Layer wird hinzugefügt und durch die Softmax-Funktion aktiviert, die die Werte in Wahrscheinlichkeiten zwischen 0 und 1 umwandelt. Die Ausgabe wird auf 2 gesetzt, da es 2 mögliche Ergebnisse gibt: mit Maske und ohne Maske.

Das endgültige Modell, das verwendet wird, wird durch Aufruf von `Model`-Funktion erstellt, indem die Eingabe des Base-Modells als Eingabe des Modells und das Head-Modell als Ausgabe übergeben wird.

```
115
116 # Freeze all layers to prevent updating on the 1st training process
117 for layer in baseModel.layers:
118     layer.trainable = False
119
```

Abbildung 38: Alle Layers einfrieren, um Aktualisierungen zu verhindern

Der 1. Trainingsprozess sollte die Layers im Base-Modell nicht aktualisieren, daher setzt eine for-Schleife das boolesche Flag `trainable` für alle Schichten auf `False`.

```
120
121 # Compile the model and track accuracy
122 print("[INFO] compiling model...")
123 opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
124 model.compile(loss="binary_crossentropy", optimizer=opt,
125               metrics=["accuracy"])
126
```

Abbildung 39: Modell kompilieren

Das Modell wird mit dem Adam-Optimizer kompiliert, der bei großen Datenmengen nützlich ist, die Binary-Cross-Entropy für die Verlustfunktion ermittelt und die Accuracy des Modells verfolgt.

```

129
130
131 H = model.fit(
132     aug.flow(trainX, trainY, batch_size=BS),
133     steps_per_epoch=len(trainX) // BS,
134     validation_data=(testX, testY),
135     validation_steps=len(testX) // BS,
136     epochs=EPOCHS)
137

```

Abbildung 40: Model-Fitting

Das Modell-Training erfolgt anhand der erweiterten Daten aus dem Image-Data-Generator, und das Modell wird anhand der Testdaten (20% des Datasets) validiert.

```

136
137 # predictions on the testing dataset
138 print("[INFO] evaluating network...")
139 predIdxs = model.predict(testX, batch_size=BS)
140
141 # Find index of label with corresponding largest prediction
142 predIdxs = np.argmax(predIdxs, axis=1)
143
144 # classification report
145 print(classification_report(testY.argmax(axis=1), predIdxs,
146     target_names=lb.classes_))
147
148

```

Abbildung 41: Netzwerk-Evaluierung

Das Netzwerk wird mit der Funktion `predict` ausgewertet und mit der Funktion `argmax` von `numpy` wird die Kategorie mit der höchsten Probabilität gefunden, bevor ein Klassifizierungsbericht ausgedruckt wird.

```

148
149 # serialize the model
150 print("[INFO] saving mask detector model...")
151 model.save(args["model"], save_format="h5")
152
153 # training loss and accuracy matplotlib-plot
154 N = EPOCHS
155 plt.style.use("ggplot")
156 plt.figure()
157 plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
158 plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
159 plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")
160 plt.plot(np.arange(0, N), H.history["val_accuracy"], label="val_acc")
161 plt.title("Training Loss and Accuracy")
162 plt.xlabel("Epoch #")
163 plt.ylabel("Loss/Accuracy")
164 plt.legend(loc="lower left")
165 plt.savefig(args["plot"])
166

```

Abbildung 42: Modell speichern und Plot erstellen

Das Modell wird im h5-Format gespeichert, und es wird ein Matplotlib-Plot erstellt, das den Trainingsverlust und die Genauigkeit anzeigt.

3.3.2 Durchführung des Modell-Trainingsprogramm

Die Ausführung des Trainingsprogramms dauert etwa 20 bis 30 Minuten. Das Training muss jedoch nur einmal auf dem Empfängergerät durchgeführt werden.

Um das Trainingsprogramm zu starten, wird das Kommando-Fenster geöffnet und in das Verzeichnis gewechselt, das das Trainingsprogramm enthält. Der Befehl `python model_training.py --dataset dataset` wird getippt und das Programm beginnt.

```

[INFO] compiling model...
[INFO] training head...
2022-07-22 18:53:28.544790: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:143] Filling up shuffle buffer (this may take a while): 58 of 104
2022-07-22 18:53:36.421603: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:193] Shuffle buffer filled.
Train for 103 steps, validate on 828 samples
Epoch 1/20
2022-07-22 18:53:47.889691: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:143] Filling up shuffle buffer (this may take a while): 57 of 104
2022-07-22 18:53:55.944450: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:193] Shuffle buffer filled.
103/103 [=====] - 112s 1s/step - loss: 0.5632 - accuracy: 0.7156 - val_loss: 0.3780 - val_accuracy: 0.8500
Epoch 2/20
2022-07-22 18:55:38.538302: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:143] Filling up shuffle buffer (this may take a while): 57 of 104
2022-07-22 18:55:46.602572: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:193] Shuffle buffer filled.
103/103 [=====] - 107s 1s/step - loss: 0.3074 - accuracy: 0.8776 - val_loss: 0.2747 - val_accuracy: 0.8988
Epoch 3/20
2022-07-22 18:57:25.071472: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:143] Filling up shuffle buffer (this may take a while): 57 of 104
2022-07-22 18:57:33.112675: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:193] Shuffle buffer filled.
103/103 [=====] - 106s 1s/step - loss: 0.2270 - accuracy: 0.9161 - val_loss: 0.3458 - val_accuracy: 0.8487
Epoch 4/20

```

Abbildung 43: Ausführung des Trainingsprogramms

Es wird ein Klassifizierungsbericht mit Details wie precision, recall, f1-score und accuracy des ML-Modells angezeigt.

```

[INFO] evaluating network...
              precision    recall  f1-score   support

   with_mask           0.99      0.85      0.92       436
 without_mask          0.86      0.99      0.92       392

   accuracy                   0.92       828
  macro avg           0.93      0.92      0.92       828
 weighted avg          0.93      0.92      0.92       828

[INFO] saving mask detector model...

```

Abbildung 44: Klassifizierungsbericht

Ein Diagramm, das den Trainings- und Validierungsverlust und die Genauigkeit gegen die Epochen zeigt, ist zu sehen. Je höher die Genauigkeit und je niedriger der Verlust, desto besser ist die Vorhersagefähigkeit des Modells.

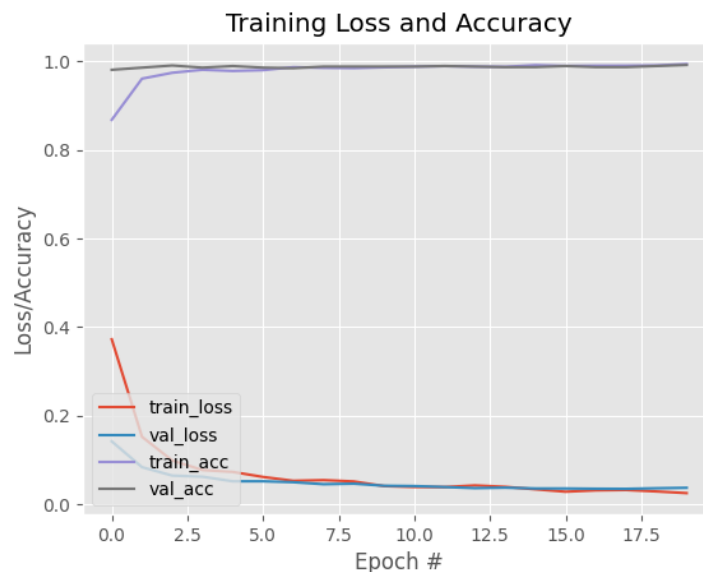


Abbildung 45: Trainings-Loss und -Accuracy

3.3.3 Implementierung des ML-Modells in das System

Sobald das Modell erfolgreich trainiert wurde, kann es in das Smart-Kamerasystem integriert werden, um Gesichter mit Masken zu erkennen. OpenCV bietet bereits Tools zur Gesichtserkennung. Obwohl Haarcascades beliebter sind, bietet OpenCV eine schnellere und genauere Erkennung mit seinem Deep-Learning-Gesichtserkennungsmodell⁹ [8]. Die folgenden Dateien werden daher in den Ordner `face_detector` eingefügt (siehe Abbildung 46).

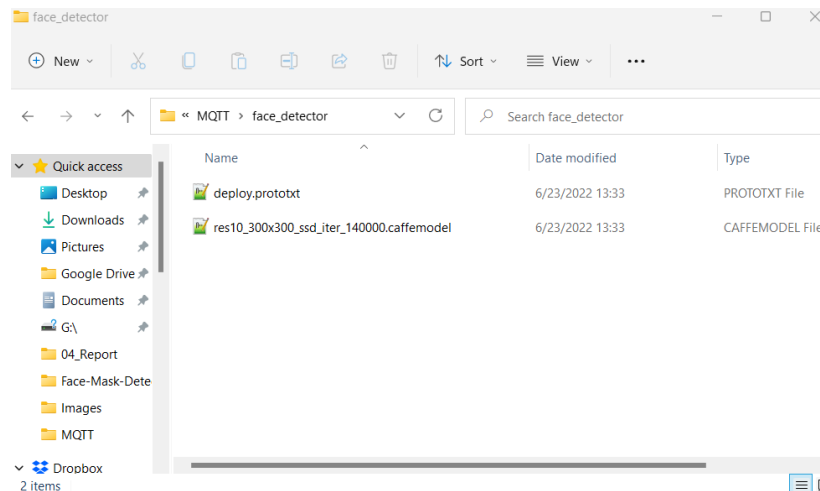


Abbildung 46: face_detector-Ordner

Nach dem Importieren der erforderlichen Bibliotheken (siehe Abbildung 2) werden 3 Argumente mit `argparse` definiert:

- `--face detector config`: Pfad zum `face_detector`-Verzeichnis
- `--model`: Pfad zum trainierten ML-Modell
- `--confidence`: Mindestwahrscheinlichkeit (50%), um schwache Erkennungen herauszufiltern

Die beiden Modelle, das Face-Detector-Modell und das Mask-Detektor-Modell, werden mit der Funktion `load_model` geladen und als `faceNet` bzw. `maskNet` zurückgegeben.

```

141 def __load_model(self):
142     # load the serialized face detector model
143     print("Loading face detector model...")
144     prototxtPath = os.path.sep.join([self.__args["face detector config"], "deploy.prototxt"])
145     weightsPath = os.path.sep.join([self.__args["face detector config"],
146                                     "res10_300x300_ssd_iter_140000.caffemodel"])
147     faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)
148
149     # load the face mask detector model from disk
150     print("Loading face mask detector model...")
151     maskNet = load_model(self.__args["model"])
152
153     return faceNet, maskNet


```

Abbildung 47: Modelle laden

⁹ Siehe Link: https://github.com/opencv/opencv/tree/4.x/samples/dnn/face_detector

Die über MQTT empfangenen dekodierten Frames werden verkleinert und zusammen mit dem faceNet und maskNet als Eingabeparameter an die Funktion detect_mask übergeben.

```

70
71  def __detect_mask(self, frame, faceNet, maskNet):
72     # grab the dimensions of the frame and then construct a blob
73     # from it
74     (h, w) = frame.shape[:2]
75     blob = cv2.dnn.blobFromImage(frame, 1.0, (300, 300), (104.0, 177.0, 123.0))
76
77     # pass the blob through the network and obtain the face detections
78     faceNet.setInput(blob)
79     detections = faceNet.forward()
80

```

Abbildung 48: detect_mask-Funktion

Aus den Frames werden BLOB („*Binary Large Object*“) [1] erstellt, das durch das faceNet-Netzwerk geleitet, um Gesichterprädiktionen zu erhalten. [7]

Die Konfidenz in die Erkennungen wird extrahiert, schwache Vorhersagen werden herausgefiltert und die Dimensionen eines Rechtecks um das Gesicht wird berechnet. Die Gesichts-ROI („*Region of Interest*“) wird extrahiert und vom BGR- in den RGB-Kanal konvertiert, der dann in der Größe angepasst wird. Jedes erkannte Gesicht wird mit der Keras-Funktion preprocess_input vorverarbeitet, und das Gesicht und die Position im Bild werden an eine Liste angehängt. Wenn ein Gesicht gefunden wird, wird maskNet aufgerufen und die Maskenprädiktionen berechnet (siehe Anhang A).

Eine Schleife durchläuft die von der Funktion detect_mask zurückgegebenen Orte und Prädiktionen und packt das Begrenzungsrechteck und die Vorhersagen aus. Das Label hängt davon ab, welche Kategorie eine höhere Prädiktion hat, was auch die Farbe des zu zeichnenden Rechtecks bestimmt (grün für Maske, rot für ohne Maske). Die Wahrscheinlichkeit, die sich aus der Prädiktion ergibt, wird ebenfalls an das Label angehängt und als Text im Video angezeigt (siehe Anhang B).

3.4 Verschlüsselung

Die Übertragung des Videos vom Sender zum Empfänger ist nun erfolgreich. Die Analyse und Maskenprädiktion ist ebenfalls möglich. Es ergibt sich jedoch ein Sicherheitsproblem - jeder, der irgendwo auf der Welt Zugriff auf das MQTT-Topic hat, kann die base64-Verschlüsselung umkehren und so ebenfalls Zugriff auf das Videofeed erhalten.

Laut Prof. Dr. Martin Rieger geht es beim Datenschutz um den Schutz der persönlichen Daten vor Missbrauch durch Dritte. Das deutsche Bundesdatenschutzgesetz (BDSG) definiert die Anforderungen zum Umgang mit persönlichen Daten. [6]

Eines der IT-Schutzziele ist die Vertraulichkeit, was bedeutet, dass nur befugte Personen Zugang zu den Informationen erhalten sollten. Obwohl die Geheimhaltung des MQTT-Topic ein Schritt in Richtung Vertraulichkeit sein könnte, widerspricht

dies mit dem Kerckhoffs' Prinzip. Das Kerckhoffs'sche Prinzip besagt, dass die Sicherheit eines Verschlüsselungsverfahrens auf der Geheimhaltung des Schlüssels beruht anstatt auf der Geheimhaltung des Verschlüsselungsalgorithmus.

« Il faut qu'il n'exige pas le secret, et qu'il puisse sans inconvénient tomber entre les mains de l'ennemi. »

„Es darf nicht der Geheimhaltung bedürfen und soll ohne Schaden in Feindeshand fallen können.“

– AUGUSTE KERCKHOFFS: La cryptographie militaire 1883

Ein unsicheres Produkt mindert die Produktqualität, unabhängig davon, wie ausgefeilt die Software ist. Daher ist es notwendig, alle möglichen Bedrohungen für das System zu identifizieren.

Die größte Gefahr für die Datensicherheit besteht bei der Übertragung vom Sender zum Empfänger. Dies liegt daran, dass Tapping leicht aus der Ferne durchgeführt werden kann, ohne dass das befugte Personal darüber informiert wird, dass es ein Informationsleck gibt.

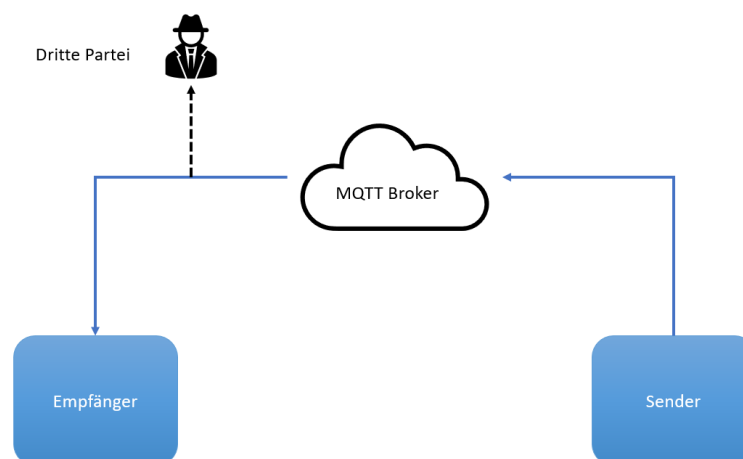


Abbildung 49: Informationsleck

Es gibt viele Möglichkeiten, die MQTT-Verbindung zu sichern. Die gebräuchlichsten Möglichkeiten sind die Angabe eines Benutzernamens und eines Passworts oder die Verwendung eines TLS-Zertifikats („*Trusted Layer Security*“). Anhand eines ISO-OSI-Modells¹⁰ kann man sehen, wie die Daten gesichert sind. Die oben erwähnten Sicherheitsmethoden sichern Daten auf der Vermittlungsschicht. Dies verhindert zwar, dass eine Dritte-Partei auf die Daten zugreift, nicht aber, dass der MQTT-Broker oder jemand, der Zugriff auf den MQTT-Server hat, auf die Daten zugreifen kann. [3]

Das bedeutet, dass man darauf vertrauen muss, dass der MQTT-Server sehr sicher ist, was vor allem bei sensiblen Informationen wie Videoaufnahmen von Personen sehr riskant sein kann. Daher werden die Daten in diesem Projekt auf der

¹⁰ Siehe Link: <https://de.wikipedia.org/wiki/OSI-Modell>

Anwendungsschicht mit Ende-zu-Ende-Verschlüsselung (E2EE) verschlüsselt. Bei E2EE werden die Daten auf dem System oder Gerät des Absenders verschlüsselt, und nur der vorgesehene Empfänger kann die Daten entschlüsseln. Auf dem Weg zum Ziel kann die Nachricht nicht von einem Internetdienstanbieter (ISP), einem Anwendungsdienstanbieter (in dem Fall MQTT), einem Hacker oder einer anderen Einrichtung oder einem Dienst gelesen oder manipuliert werden. Dies geschieht durch symmetrische Verschlüsselung. [3]

Symmetrische-Verschlüsselung

In einem symmetrischen System müssen zwei Kommunikationspartner einen gemeinsamen, geheimen Schlüssel verwenden, wenn sie miteinander vertraulich kommunizieren möchten. Das bedeutet, dass sie sich vorab über diesen gemeinsamen Schlüssel verständigen müssen. Nach dem Importieren der notwendigen Bibliotheken (siehe Abbildung 4) wird zunächst ein geheimer Schlüssel erzeugt und im Sender- und Empfängergerät gespeichert. Dieser wird nicht über MQTT ausgetauscht.

```

13
14     secretKey = Fernet.generate_key()
15     current_time = time.strftime("%H:%M:%S", time.localtime())
16
17     config = ConfigParser()
18     config.read('secret_key.ini')
19     config.add_section('Symmetric Encryption')
20     config.set('Symmetric Encryption', 'Secret Key', base64.b64encode(secretKey).decode('utf-8'))
21
22     with open('secret_key.ini', 'w') as configfile:
23         config.write(configfile)
24
25     print("Key was generated successfully at " +str(current_time))
26     print("Key Length: " +str(len(secretKey)))
27     print("To view how long it would take to crack your password visit")
28     print("https://www.security.org/how-secure-is-my-password/")
29     print("& enter your password")

```

Abbildung 50: Generierung eines geheimen Schlüssels

Eine Config-Datei wird erstellt und der geheime Schlüssel wird in einer base64-Kodierung gespeichert.

```

[Symmetric Encrvption]
secret key = 

```

Abbildung 51: Geheimer Schlüssel

Implementierung des geheimen Schlüssels in das System

Der geheime Schlüssel wird sowohl im Sender- als auch im Empfängerprogramm durch eine statische Methode `create_cipher` aufgerufen, wenn der `VideoStream`-Konstruktor aufgerufen wird.

```

39     @staticmethod
40     def __create_cipher():
41         # Read config file
42         config = configparser.ConfigParser()
43         config.read('secret_key.ini')
44         secretKey_str = config['Symmetric Encryption']['Secret Key']
45         secretKey = base64.b64decode(secretKey_str.encode('utf-8'))
46         cipher = Fernet(secretKey)
47         return cipher
48

```

Abbildung 52:create_cipher-Funktion

In der Funktion `encrypt` wird die Nachricht mit dem geheimen Schlüssel verschlüsselt und die verschlüsselten Daten werden veröffentlicht.

```

48
49     def __encrypt(self, msg, cipher):
50         cipher_msg = cipher.encrypt(msg)
51         return cipher_msg
52

```

Abbildung 53: encrypt-Funktion

Im Empfängerprogramm wird die Nachricht zunächst mit Hilfe des geheimen Schlüssels entschlüsselt, bevor sie in ein Bildformat umgewandelt werden kann (siehe Abbildung 28).

```

48
49     def __decrypt(self, encrypted_msg):
50         decrypted_msg = self.__cipher.decrypt(encrypted_msg)
51         return decrypted_msg
52

```

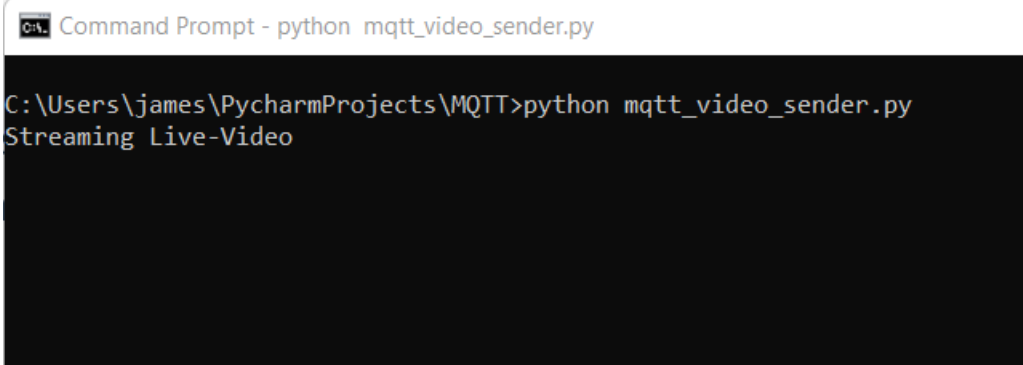
Abbildung 54: decrypt-Funktion

4 Diskussion

4.1 Ergebnis

Sender-Programm

Wenn das Sendeprogramm startet, wird im Befehlsfenster eine Meldung angezeigt, dass ein Live-Video gestreamt wird.



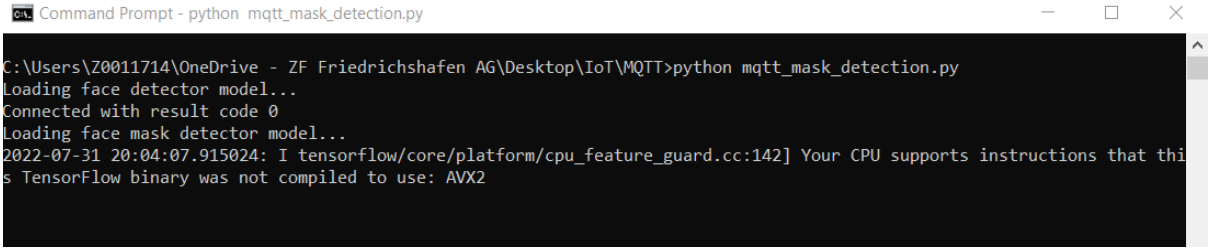
```

Command Prompt - python mqtt_video_sender.py

C:\Users\james\PycharmProjects\MQTT>python mqtt_video_sender.py
Streaming Live-Video
  
```

Abbildung 55: Live-Stream-Benachrichtigung

Empfänger-Programm



```

Command Prompt - python mqtt_mask_detection.py

C:\Users\Z0011714\OneDrive - ZF Friedrichshafen AG\Desktop\IoT\MQTT>python mqtt_mask_detection.py
Loading face detector model...
Connected with result code 0
Loading face mask detector model...
2022-07-31 20:04:07.915024: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2
  
```

Abbildung 56: Ausführen des Empfängerprogramms

Um sicher zu sein, dass der Empfänger das MQTT-Topic erfolgreich abonniert hat, muss die Meldung "Connected with result code 0" erscheinen, sonst ist die Verbindung nicht erfolgreich. Es erscheint eine Benachrichtigung, dass sowohl das Gesichts- als auch das Maskendetektormodell angezeigt werden, und ein Bildschirm mit dem Video wird eingeblendet.

Maskenerkennung

Auf dem Bildschirm werden die erkannten Gesichter sowie die Wahrscheinlichkeit, ob die Person eine Maske trägt oder nicht, angezeigt.

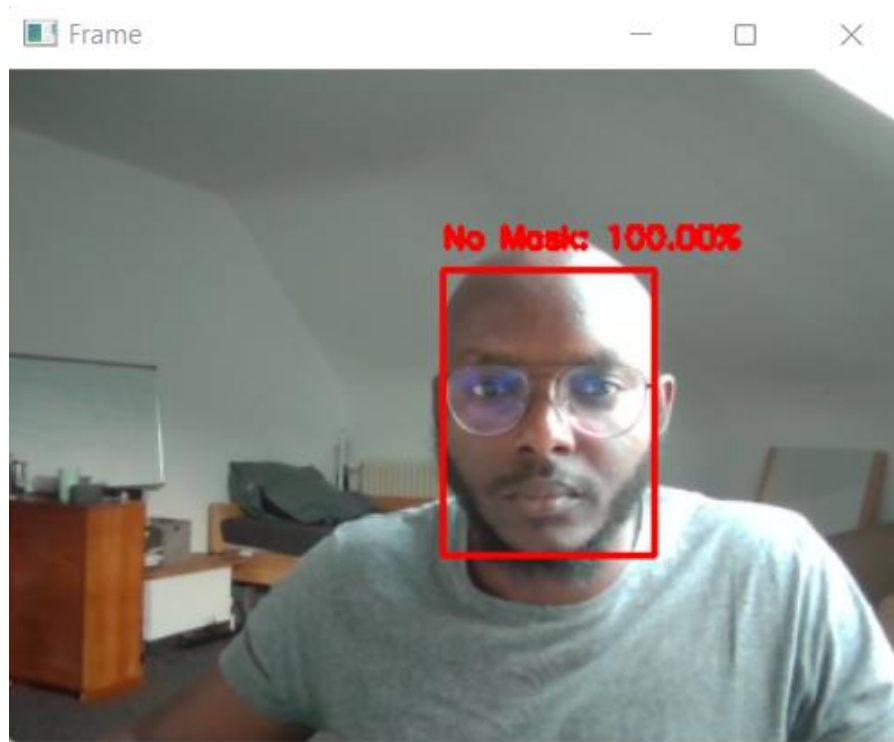


Abbildung 57: Ohne Maske

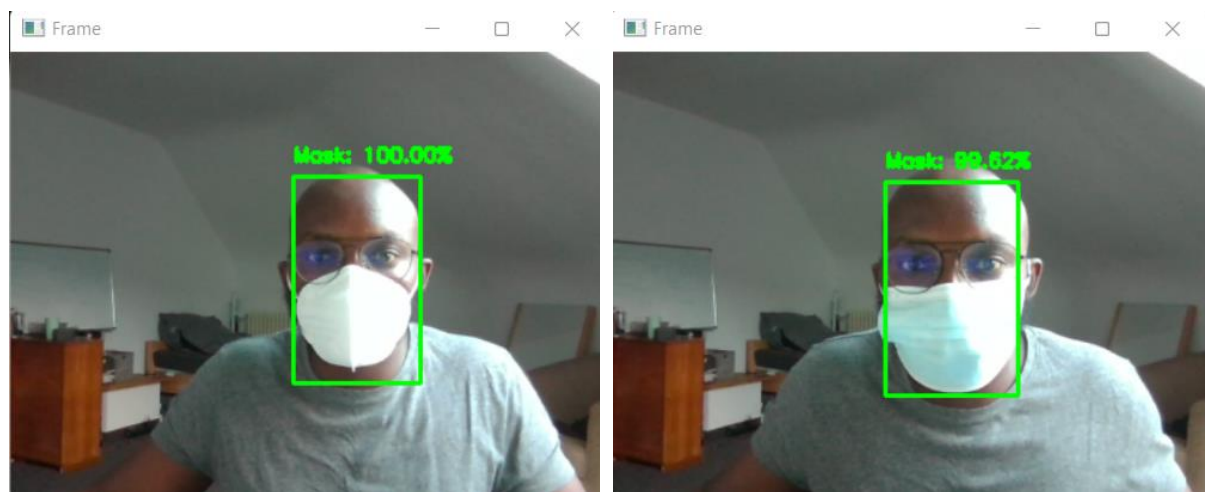


Abbildung 58: Mit einer Maske (rechts: FFP2-Maske, links: OP-Maske)

Es können auch mehrere Personen erkannt werden und die Prädiktion der Maske ist unabhängig voneinander.

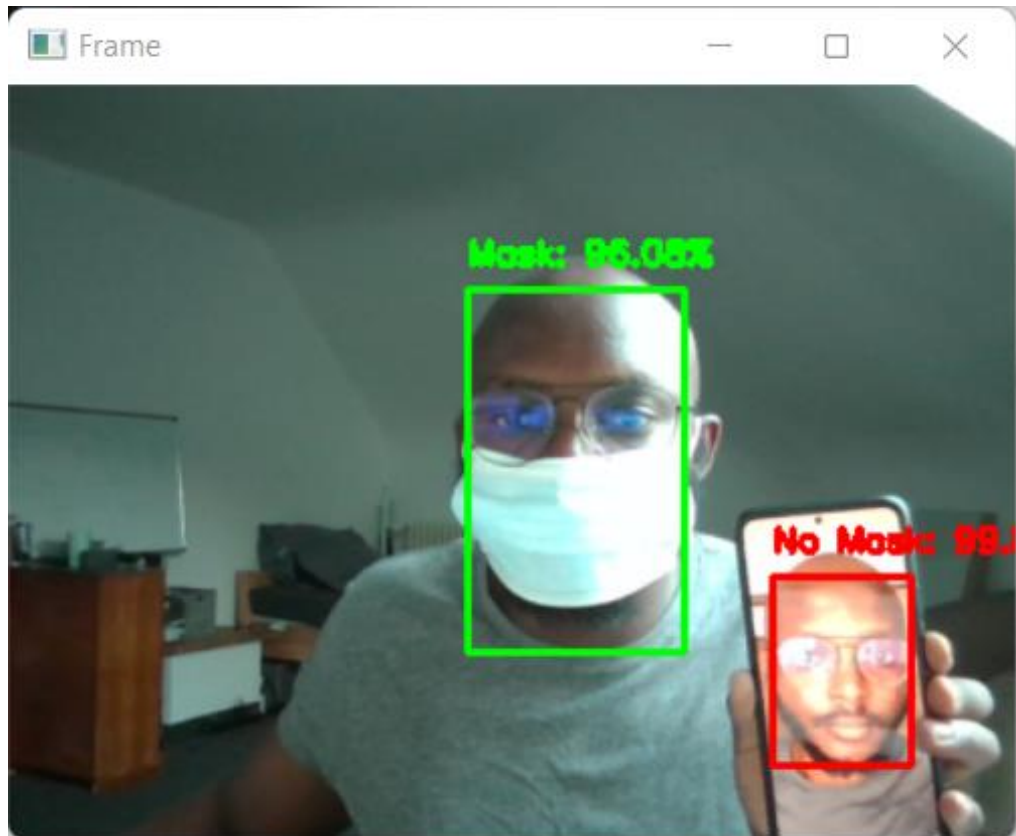


Abbildung 59: Maskenerkennung für mehrere Personen

Was würde passieren, wenn man versucht, die Software zu betrügen?

Der Versuch, Mund und Nase zu bedecken, ist nicht sinnvoll, da die Software dies erkennen und bescheinigen kann, dass die Person keine Maske trägt.

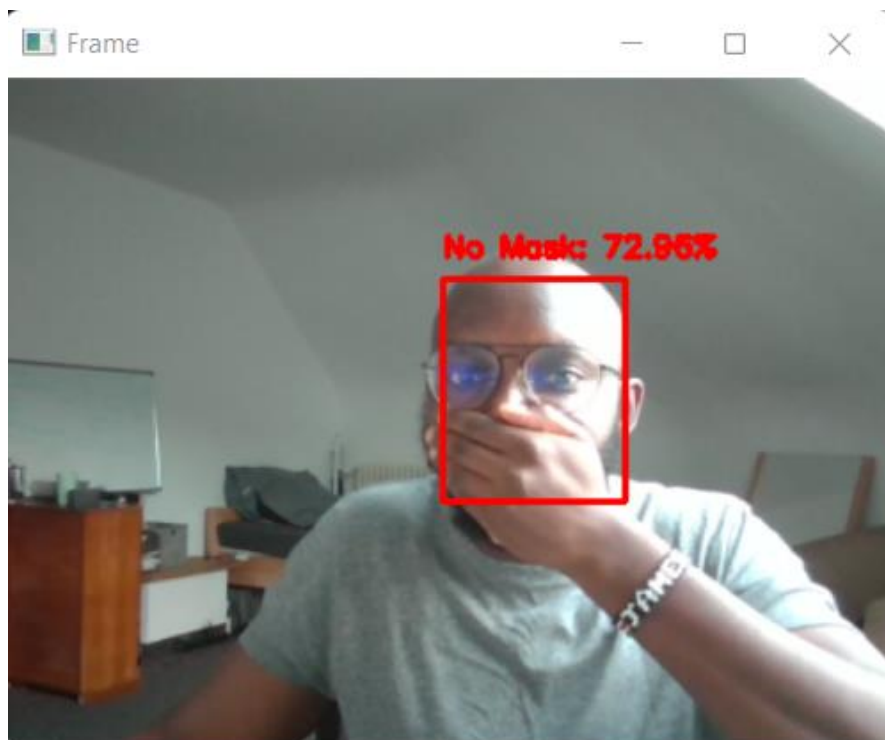


Abbildung 60: Bedecken von Mund und Nase

Wenn die Maske zwar vorhanden ist, aber in einer falschen Weise getragen wird, kann die Software nicht so gut getäuscht werden.

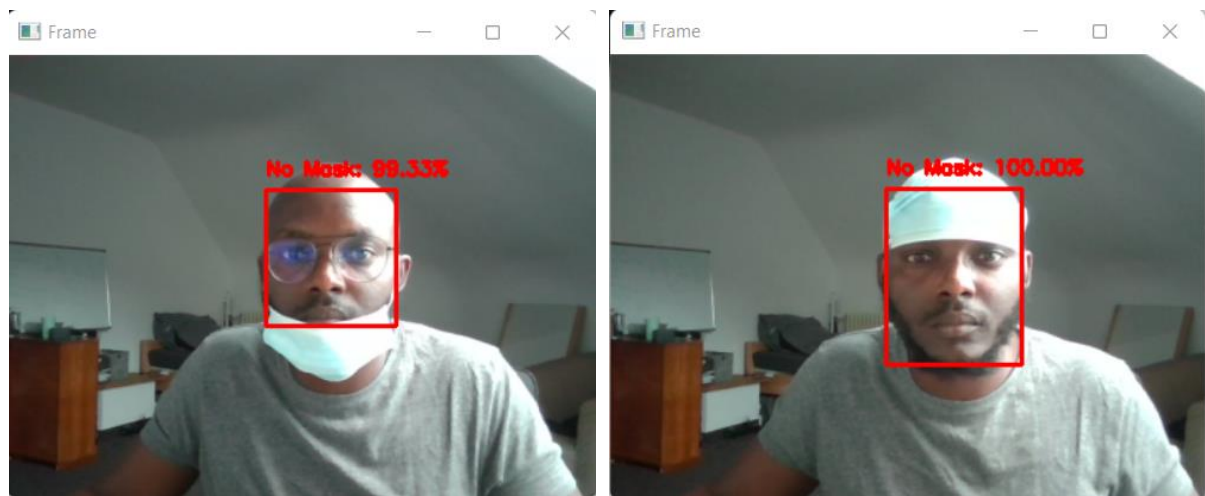


Abbildung 61: Unsachgemäßes Tragen der Maske

Sicherheit

Die Sicherung der Daten auf der Anwendungsschicht bedeutet, dass die Daten auch über einen unsicheren Kanal übertragen werden können. Dies bedeutet, dass die Daten immer noch von einer dritten Partei abgehört werden können, aber ohne den geheimen Schlüssel ist es sehr schwierig, die Nachricht zu entschlüsseln.

Die Website [security.org](https://www.security.org) gibt eine Vorstellung davon, wie lange es dauern würde, bis ein Computer den Schlüssel knacken könnte.

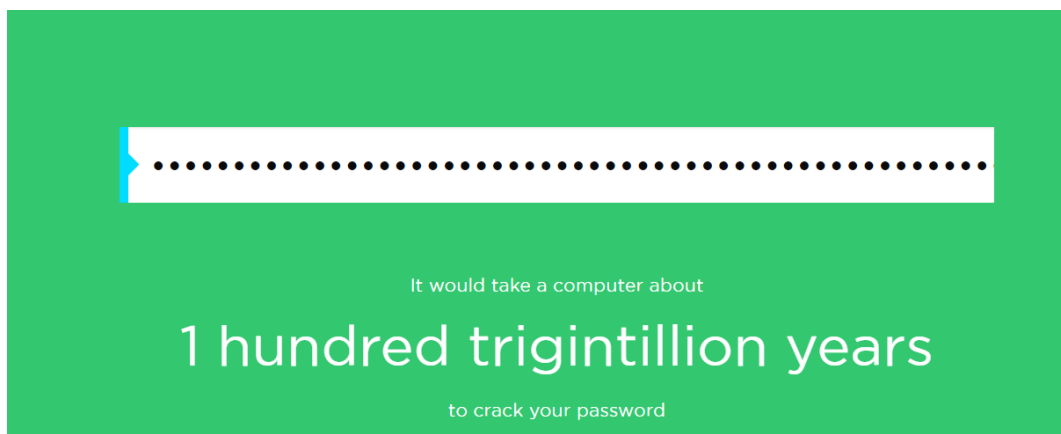


Abbildung 62: security.org-Webseite¹¹

Dadurch wird sichergestellt, dass nur der Sender und der Empfänger Zugriff auf die Videodaten haben.

¹¹ Siehe Link: <https://www.security.org/how-secure-is-my-password/>

4.2 Anwendung in der Praxis

Die Idee für dieses Projekt wurde von den deutschen öffentlichen Verkehrsmitteln während der Covid-Pandemie inspiriert. Obwohl das Tragen einer Maske für Passagiere obligatorisch ist, gibt es Passagiere, die die Maske abnehmen und sie nur tragen, wenn Zug- und Busbegleiter ihre Runden drehen. Die erforderliche Arbeitskraft, um sicherzustellen, dass jeder Passagier jederzeit eine Maske trägt, würde die Betriebskosten erhöhen und ist auch unrealistisch zu erreichen.

Die Implementierung dieser Software in bereits vorhandene Überwachungskameras kann jedoch als Lösung angeboten werden, ohne dass die Hardware der Kamera geändert werden muss. Dies würde auch dazu beitragen, die zusätzliche Belastung zu verringern, die Zug- und Busbegleiter seit dem Ausbruch der Covid-Pandemie tragen müssen, da die Überwachungskameras in der Lage wären, festzustellen, ob Fahrgäste eine Maske tragen oder nicht.

Die Software ist nicht nur auf öffentliche Verkehrsmittel beschränkt, sondern auch auf Flughäfen und Bereiche, in denen vor dem Betreten Masken erforderlich sind. Ein Raum auf der Intensivstation in einem Krankenhaus ist ein solcher Bereich, der den Patienten vor Kontamination schützt.

5 Zusammenfassung

Das Projekt demonstrierte die Möglichkeit, eine intelligente Kamera zu entwickeln. Die Daten wurden über einen ungesicherten Kanal vom Sender zum Empfänger gesendet und die Maskenprädiktionen sind erfolgreich.

MQTT

Das Projekt war erfolgreich beim Senden von Videodaten über MQTT. Die Nachricht wurde vom Empfängerprogramm empfangen und ohne Datenverlust dekodiert. Durch MQTT ist es möglich, überall auf der Welt Daten zu senden und zu empfangen. Dadurch entfällt die Begrenzung durch den Standort, da der Empfänger überall platziert werden kann und die Daten richtig empfangen und analysieren kann. Dieser Mobilitätsvorteil kommt besonders bei Zügen und Bussen auf langen Strecken zum Tragen.

Der verwendete MQTT-Server ist jedoch ein kostenloser Testserver, der sowohl von Mosquitto als auch von HiveMQ angeboten wird und nicht für irgendetwas in der Produktion verwendet werden sollte. Dennoch wurde für einen Prototypen bewiesen, dass es möglich ist, Video per MQTT zu übertragen. Mittels PyAudio wurden auch Audiosignale verlustfrei vom Sender zum Empfänger gesendet, was aber aufgrund der Komplexität in der Entwicklung des Projekts später im Rahmen des Projekts entfallen ist.

OpenCV

OpenCV ist ein sehr mächtiges Werkzeug, wie in diesem Projekt zu sehen ist. Es war nicht nur möglich, Live-Videodaten über OpenCV zu erfassen, sondern es war auch möglich, dass das Empfangsprogramm das Video anzeigt.

OpenCV bietet ein DNNs-Tool, das sehr nützlich ist, um Gesichter genau vorherzusagen.

Maschinelles Lernen

Das trainierte Modell erwies sich als in der Lage, genaue Vorhersagen zu treffen, und in Fällen, in denen ein Mensch absichtlich versucht, das Modell auszutricksen, erwies sich das Modell als in der Lage, den Trick zu bemerken.

Wie in Abbildung 59 zu sehen ist, ist ein Problem, das die Software nicht unterscheiden kann, echte Menschen von Bildern (z. B. von einem Bild oder Video). Dies ist ohne den Einsatz zusätzlicher Hilfsmittel wie Näherungssensoren nicht möglich zu unterscheiden. Dies wird von der Firma Apple mit seinem Face ID-Tool gut demonstriert, da die ML-Software auch auf IR-Sensoren und einen Punktprojektor setzt [\[11\]](#). Daher muss noch geforscht werden, um die Möglichkeit von Sensoren mit großer Reichweite zu ermitteln, um zwischen echten Menschen und Abbildungen von Menschen unterscheiden zu können.

Der Vorteil des implementierten Trainingsverfahrens besteht darin, dass das Modell nicht nur zum Erkennen von Masken trainiert werden muss. Mit ein paar Anpassungen kann das ML-Modell so trainiert werden, dass es je nach Wunsch des Benutzers andere Elemente erkennt.

Sicherheit

Auch die Sicherheit des Systems hat sich bewährt. Die symmetrische Verschlüsselung ermöglicht die Übertragung der Videodaten auf beliebigen Kanälen, auch auf unsicheren.

Da es nur einen Sender und einen Empfänger gibt, ist die Verschlüsselung auch ausreichend, um die Sicherheit zu gewährleisten. Bei vielen Absendern sollte jedoch eine robustere Methode der Schlüsselverwaltung verwendet werden. In diesem Fall sollte stattdessen eine Public-Key-Infrastruktur verwendet werden. Verfahren wie das Diffie-Hellman-Verfahren und das RSA-Verfahren sind solche Verfahren. [5]

Denn es gibt nur einen Absender und einen Empfänger, reicht auch die Verschlüsselung aus, um die Sicherheit zu gewährleisten. Für viele Absender sollte jedoch eine robustere Methode der Schlüsselverwaltung verwendet werden. In diesem Fall bietet ein Verfahren wie das Diffie-Hellman-Verfahren und das RSA-Verfahren eine Lösung. Dies erfordert jedoch mehr Rechenleistung, daher muss ein Gleichgewicht zwischen Sicherheit und Leistung hergestellt werden.

Zusammenfassend lässt sich sagen, dass das Projekt seine Hauptziele erreicht hat. Es bedarf noch weiterer Forschung, um die Qualität des Prototyps zu verbessern. Dennoch kann das Projekt als erfolgreich bezeichnet werden.

6 Literaturquellen

- [1] Geeks, G. f. (10. 06 2022). *BLOB Full Form*. Von GeeksforGeeks: <https://www.geeksforgeeks.org/blob-full-form/> abgerufen
- [2] Keras. (05. 06 2022). *MobileNet, MobileNetV2, and MobileNetV3*. Von Keras: <https://keras.io/api/applications/mobilenet/> abgerufen
- [3] Lutkevich, B. (29. 06 2022). *end-to-end encryption (E2EE)*. Von TechTarget: <https://www.techtarget.com/searchsecurity/definition/end-to-end-encryption-E2EE> abgerufen
- [4] MQTT.org. (22. 06 2022). *MQTT*. Von MQTT: The Standard for IoT Messaging: <https://mqtt.org/> abgerufen
- [5] Rieger, M. (2022). Grundlagen kryptografischer Verfahren. In P. D. Rieger, *Vorlesung IT-Sicherheit* (S. 104). Albstadt: Hochschule Albstadt-Sigmaringen.
- [6] Rieger, M. (2022). Grundlegende Begriffe zur IT-Sicherheit. In P. D. Rieger, *Vorlesung IT-Sicherheit* (S. 1-6). Albstadt: Hochschule Albstadt-Sigmaringen.
- [7] Rosebrock, A. (10. 06 2022). *Deep learning: How OpenCV's blobFromImage works*. Von PyImageSearch: <https://pyimagesearch.com/2017/11/06/deep-learning-opencvs-blobfromimage-works/> abgerufen
- [8] Rosebrock, A. (10. 06 2022). *Face detection with OpenCV and deep learning*. Von PyImageSearch: <https://pyimagesearch.com/2018/02/26/face-detection-with-opencv-and-deep-learning/> abgerufen
- [9] Shaligram, S. S. (28. 06 2022). *Face Mask Detection: Simple OpenCV Based Program*. Von Medium: <https://medium.com/@saurabh.shaligram/face-mask-detection-simple-opencv-based-program-417bbcf0abd8> abgerufen
- [10] Team, T. H. (22. 06 2022). *MQTT Security Fundamentals*. Von Payload Encryption - MQTT Security Fundamentals: <https://www.hivemq.com/blog/mqtt-security-fundamentals-payload-encryption/> abgerufen
- [11] Tillman, M. (7. 31 2022). *What is Apple Face ID and how does it work?* Von Pocket-lint: <https://www.pocket-lint.com/phones/news/apple/142207-what-is-apple-face-id-and-how-does-it-work> abgerufen

Anhang

Anhang A: Verfahren zur Prädiktion von Gesicht und Maske

```
# loop over the detections
for i in range(0, detections.shape[2]):
    # extract the confidence (i.e., probability) associated with the detection
    confidence = detections[0, 0, i, 2]

    # filter out weak detections by ensuring the confidence is
    # greater than the minimum confidence
    if confidence > self.__args["confidence"]:
        # compute the (x, y)-coordinates of the bounding box for the object
        box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
        (startX, startY, endX, endY) = box.astype("int")

        # ensure the bounding boxes fall within the dimensions of
        # the frame
        (startX, startY) = (max(0, startX), max(0, startY))
        (endX, endY) = (min(w - 1, endX), min(h - 1, endY))

        # extract the face ROI, convert it from BGR to RGB channel
        # ordering, resize it to 224x224, and preprocess it
        face = frame[startY:endY, startX:endX]
        if face.any():
            face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
            face = cv2.resize(face, (224, 224))
            face = img_to_array(face)
            face = preprocess_input(face)

            # add the face and bounding boxes to their respective
            # lists
            faces.append(face)
            locs.append((startX, startY, endX, endY))

# only make a predictions if at least one face was detected
if len(faces) > 0:
    # for faster inference we'll make batch predictions on *all*
    # faces at the same time rather than one-by-one predictions
    # in the above `for` loop
    faces = np.array(faces, dtype="float32")
    preds = maskNet.predict(faces, batch_size=32)

# return a 2-tuple of the face locations and their corresponding
# locations
return (locs, preds)
```

Anhang B: Stream-Funktion

```

171 def __stream(self):
172     self.__define_args()
173     faceNet, maskNet = self.__load_model()
174     while True:
175         frame = imutils.resize(self.__frame, width=400)
176
177         # detect faces in the frame and determine if they are wearing a
178         # face mask or not
179         (locs, preds) = self.__detect_mask(frame, faceNet, maskNet)
180
181         # loop over the detected face locations and their corresponding
182         # locations
183         for (box, pred) in zip(locs, preds):
184             # unpack the bounding box and predictions
185             (startX, startY, endX, endY) = box
186             (mask, withoutMask) = pred
187
188             # determine the class label and color we'll use to draw
189             # the bounding box and text
190             label = "Mask" if mask > withoutMask else "No Mask"
191             color = (0, 255, 0) if label == "Mask" else (0, 0, 255)
192
193             # include the probability in the label
194             label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)
195
196             # display the label and bounding box rectangle on the output
197             # frame
198             cv2.putText(frame, label, (startX, startY - 10),
199                         cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
200             cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)
201
202             # self.__detect_face()
203             cv2.imshow("Stream", frame)
204             if cv2.waitKey(1) & 0xFF == ord('q'):
205                 break
206
207     # client.loop_stop()

```

Anhang C: Dataset mit Bildern mit und ohne Maske

