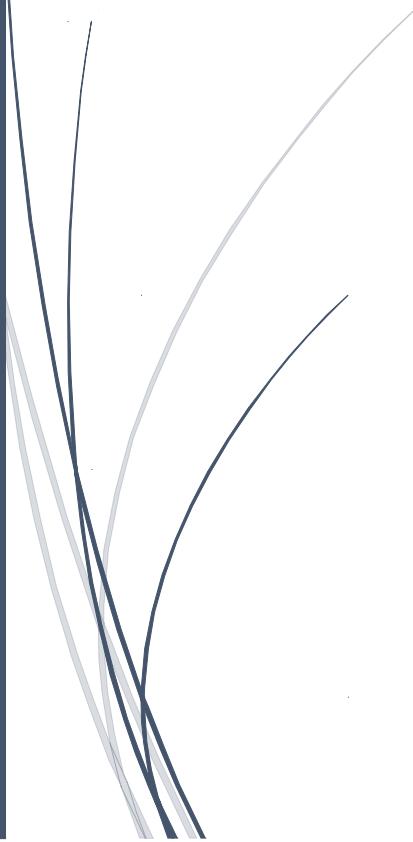


# Deep Learning in der Bilderkennung

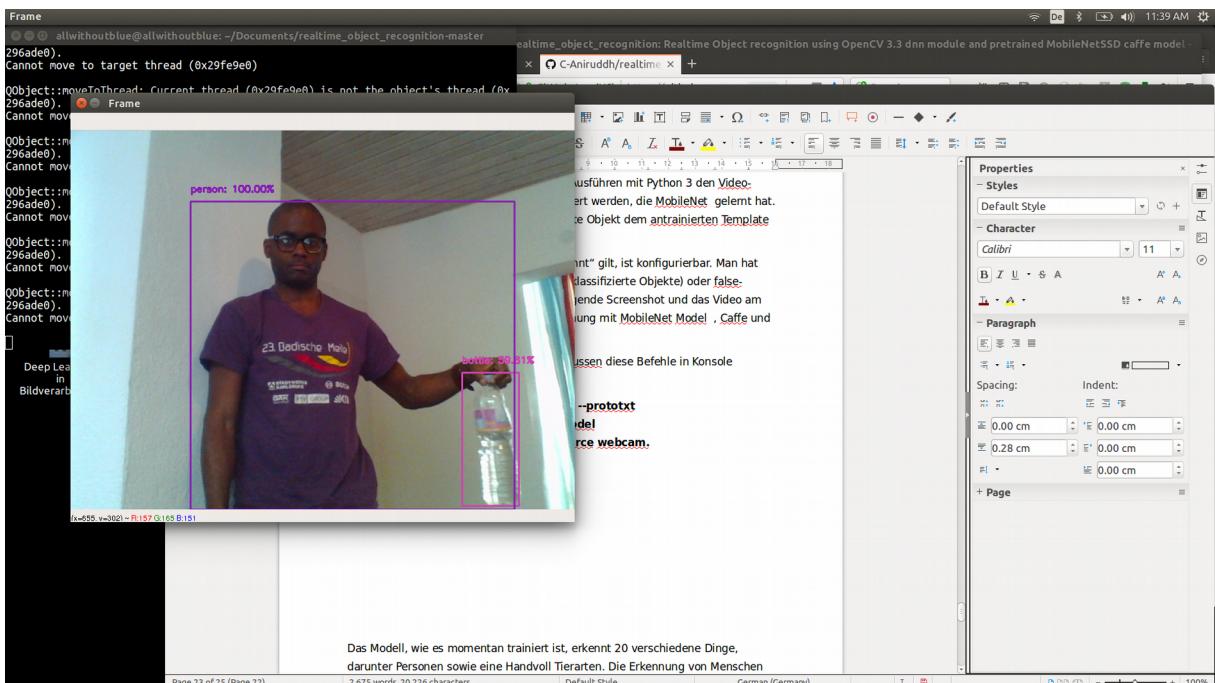
10.1.2018

Experiment: Echtzeit Objekterkennung  
unter Einsatz von OpenCV 3 and Deep  
Learning Framework Caffe

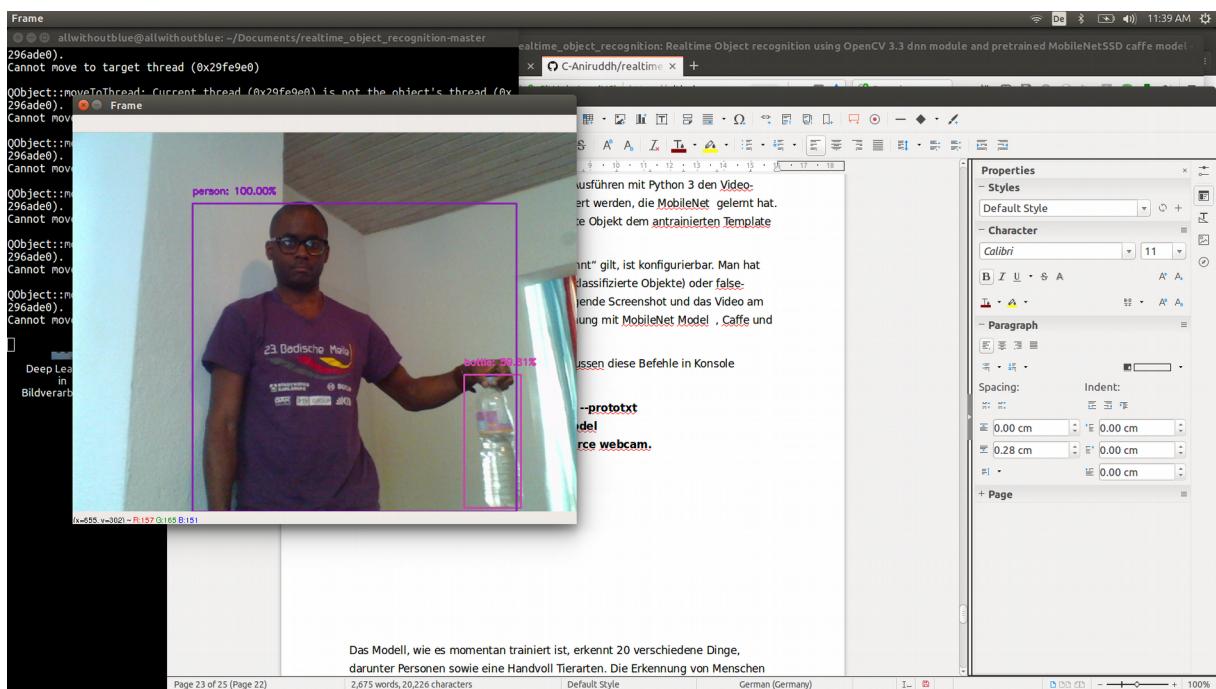
Projekt Bericht



Patrick Virgile Djimgou  
MASTER INFORMATIK - HOCHSCHULE AALEN

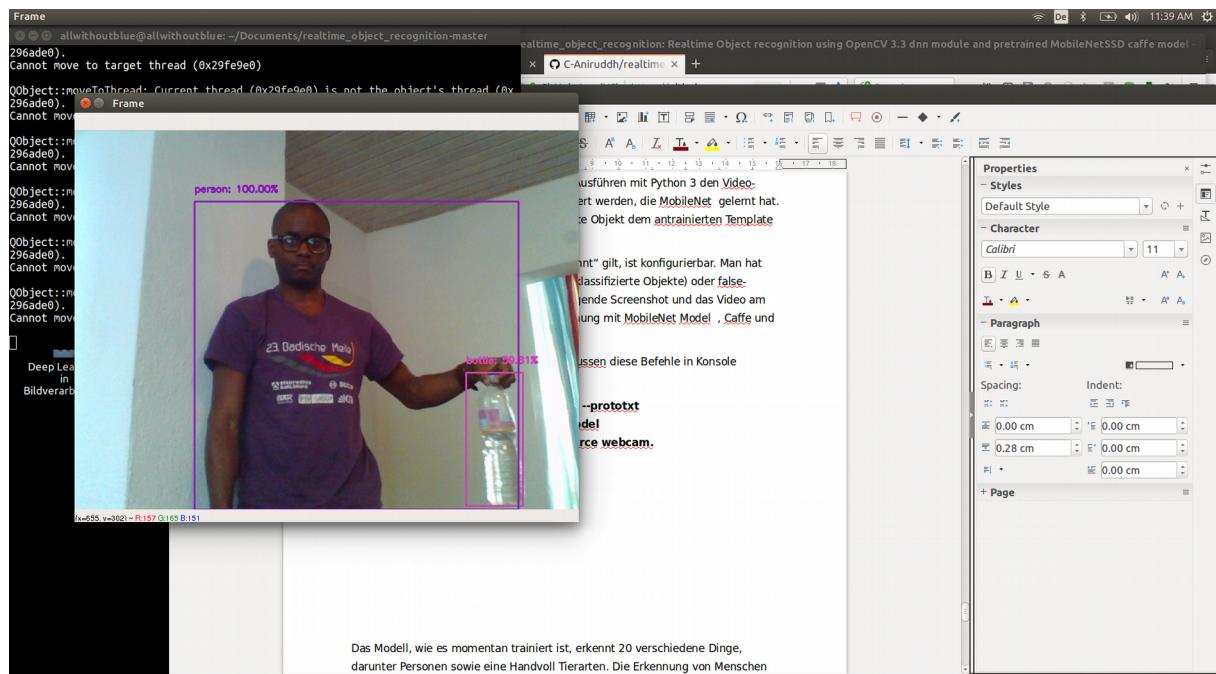


# I. Inhaltsverzeichnis

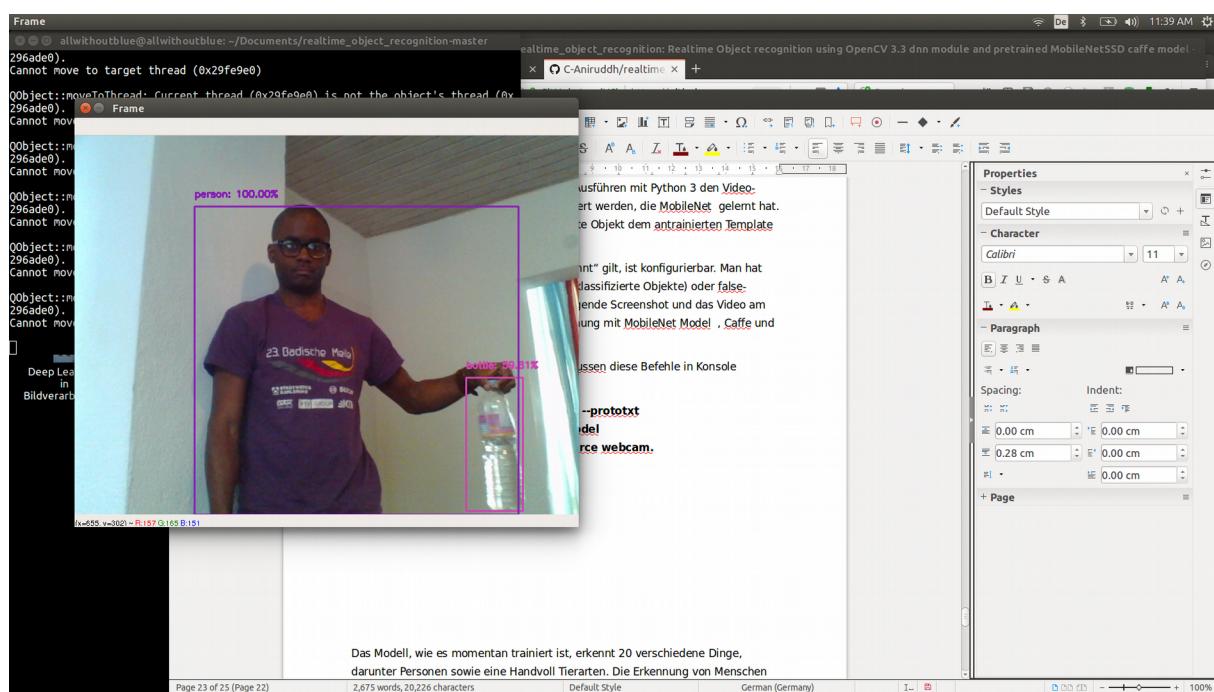


# Inhalt

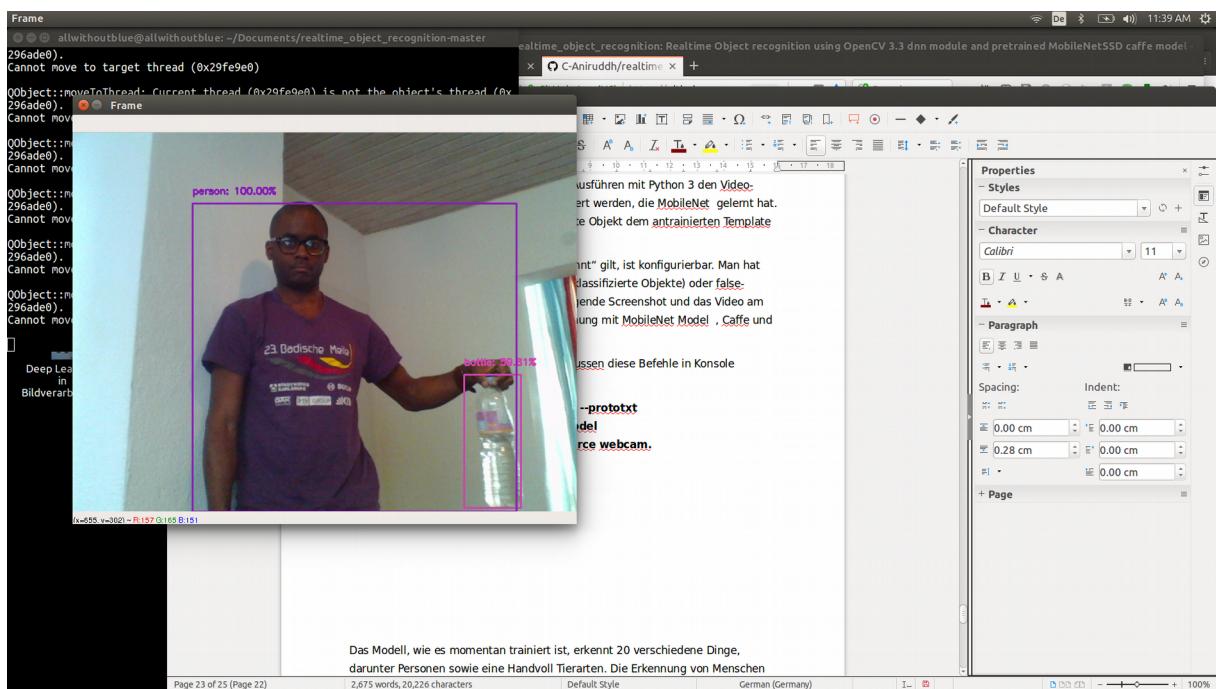
I.	Inhaltsverzeichnis.....	2
II.	Abbildungsverzeichnis.....	5
III.	Abkürzungsverzeichnis.....	6
IV.	Quellcodeverzeichnis.....	7
V.	Tabellenverzeichnis.....	8
1	Einführung.....	9
1.1	Problemanalyse.....	9
1.2	Ziel des Projekts.....	9
1.3	Projektumfeld.....	9
1.3.1	Hardwareumfeld.....	9
1.3.2	Softwareumfeld.....	9
2	Stand der Technik und Grundlagen.....	10
2.1	Open-Source vergleichbare Projekte.....	10
2.1.1	Projekt 1.....	10
2.1.2	Projekt 2.....	10
2.2	Grundlagen.....	10
2.2.1	Sensorik.....	10
2.2.2	Datenvizualisierung.....	10
3	Konzept.....	11
3.1	Systemarchitektur.....	11



4.2.2 Installation und Konfiguration der MySQL-Datenbank.....	11
5 Fazit.....	11
5.1 Zusammenfassung.....	11
5.2 Aktueller Projektstand.....	11
5.3 Ausblick.....	11
VI. Literaturverzeichnis.....	12



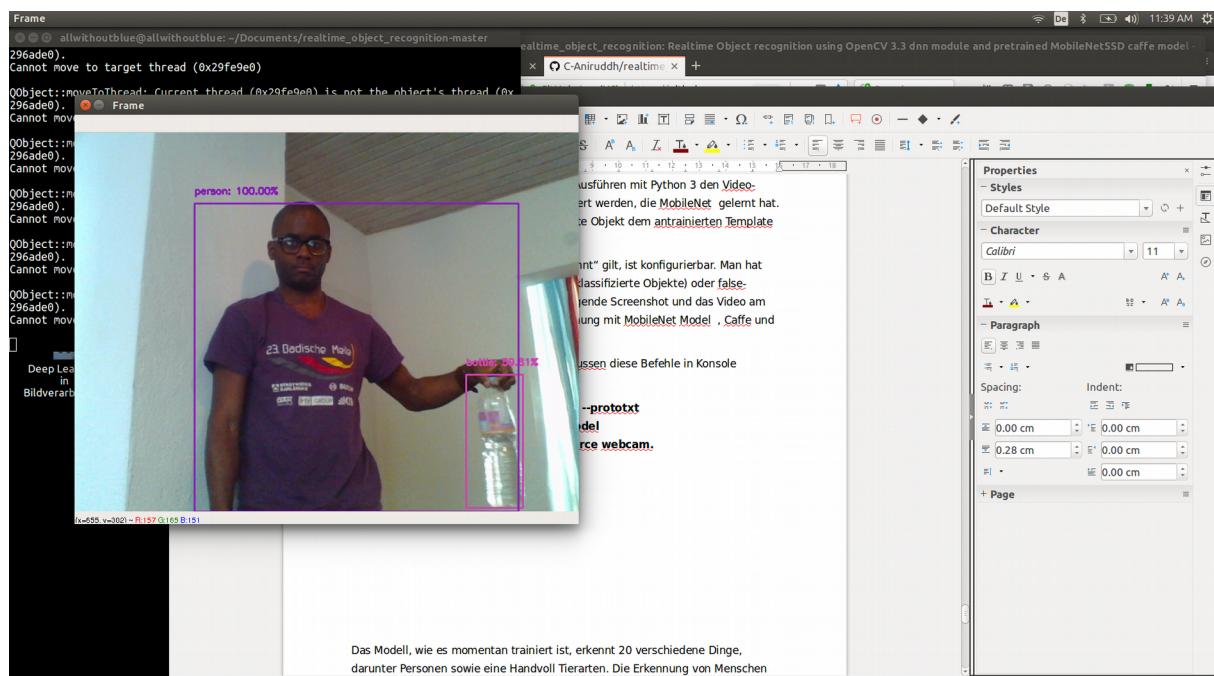
## II. Tabellenverzeichnis



# 1 Einführung.

Das Sehen ist einer der wichtigsten, wenn nicht der wichtigste Sinn des Menschen. Mit dem derzeitigen schnellen Fortschritt in der Entwicklung von autonomen Systemen wie Autos, Drohnen und Robotern für viele mögliche Anwendungen in allen Tätigkeitsbereichen ist es extrem wichtig, dass intelligente Systeme nicht nur sehen können, sondern auch verstehen was Sie sehen um die Komplexität ihrer Umgebung wahrzunehmen.

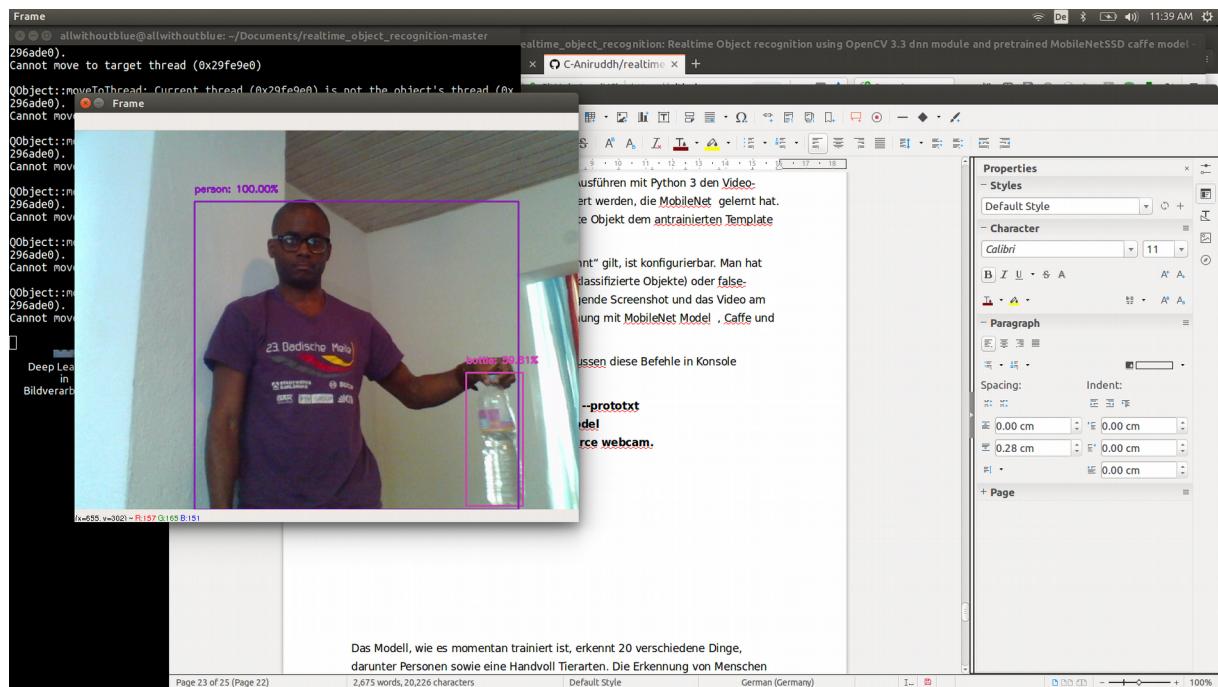
Während das Feld der Computer Vision schon seit langer Zeit existiert, haben wir in die letzten Jahre gesehen große Fortschritte und eine erneute Fokussierung auf die Anwendung von Deep Learning-Techniken in Form von neuronalen Netzwerken für Objekterkennungsaufgaben.



## 1.1 Ziel der Arbeit

Insbesonders Convolutional Neural Networks (CNNs) haben vielversprechende Ergebnisse bei der Erkennung von Objekten gezeigt. Im Hinblick Vor diesem Hintergrund war es interessant , zu recherchieren wie eine mächtige Bildverarbeitung Frameworks wie „Opencv“ kann in Zusammenhang mit extrem flexible Deep Learning Framework „Caffe“ unterschiedliche Objekte erkennen .

Im Rahmen der vorliegenden Projekt wird aufgezeigt, wie eine einsatzbereite Echzeit Objekterkennung mit Convolutional Neural Networks implementiert werden kann, basierend auf vortrainierten Modellen von Caffe Framework. Die gleichzeitige Erkennung mehrerer Objekte wird nicht nur mit Opencv and Caffe Model sondern auch von unterschiedliche Python bibliothek mit realisiert. Die Erkennungsraten der Implementierung werden in Tests evaluiert und analysiert.



## 1.2 Methodologie.

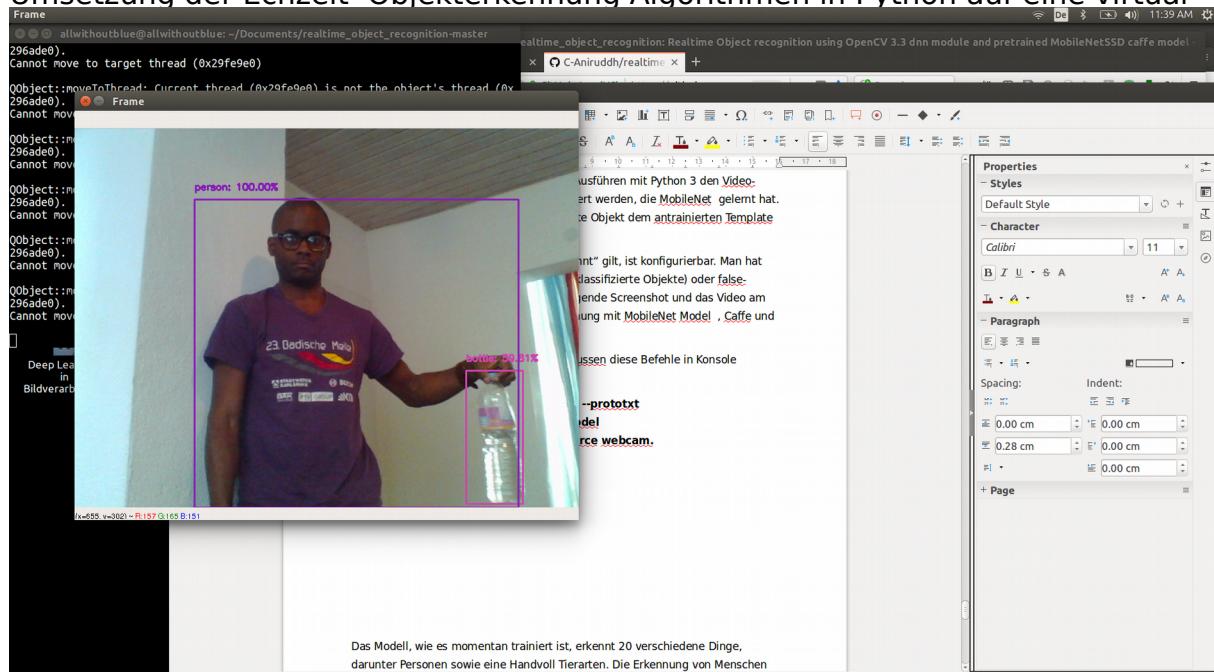
Es existieren verschiedene Frameworks für neuronales Lernen. Angefangen mit frühen Programmbibliotheken wie OpenCV aus dem Jahre 2000 existieren heute einige weitere Frameworks wie Torch, Theano, Caffe, Neon und TensorFlow.

Während Theano, Neon und TensorFlow auf Python basieren, wird für Caffe C++ und für Torch die weniger verbreitete Scriptssprache Lua verwendet. Bis auf Theano sind alle Frameworks dafür vorbereitet mehrere Grafikprozessoren zur Berechnung nutzen zu können.

Der thematische Einstieg erfolgte über einen Udacity-Kurs zu Deep Learning mit Caffe. In diesem Kurs erfolgte der Einstieg in die Thematik mit der Programmiersprache Python, mit der die Funktionsweise von Deep Learning einprägsam dargestellt werden konnte. Aufgrund der guten Dokumentation und des zur Verfügung gestelltem Sourcecodes, auch zu den behandelten Modellen, wurden Caffe , Opencv 3 und Python 3 für die praktische Umsetzung der Konzepte des Deep Learning in Bildverarbeitung in dieser Projekt ausgewählt.

## 1.3 Aufbau der Projekt

Die vorliegende Arbeit ist in die folgenden Teile gegliedert, wobei die ersten Teilen einen Überblick über die verwendeten primären Technologien und Frameworks geben , befassen sich die letzten Block mit der tatsächlichen Umsetzung der Echzeit Objekterkennung Algorithmen in Python auf eine virtual



- **Die Implementierung und Evaluierung von der programme .**
- **Fazit:** Fasst die aktuelle Arbeit zusammen und diskution über zukünftige Nutzung und mögliche Verbesserungen

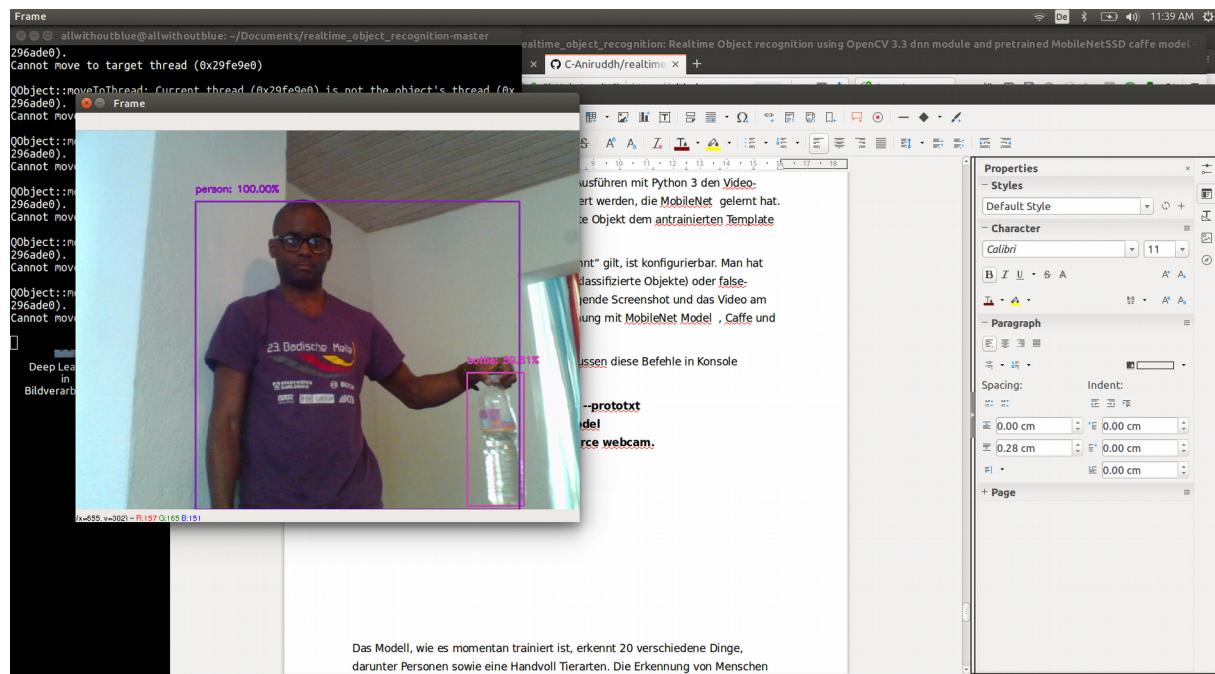
## 2 Grundlagen und Definition

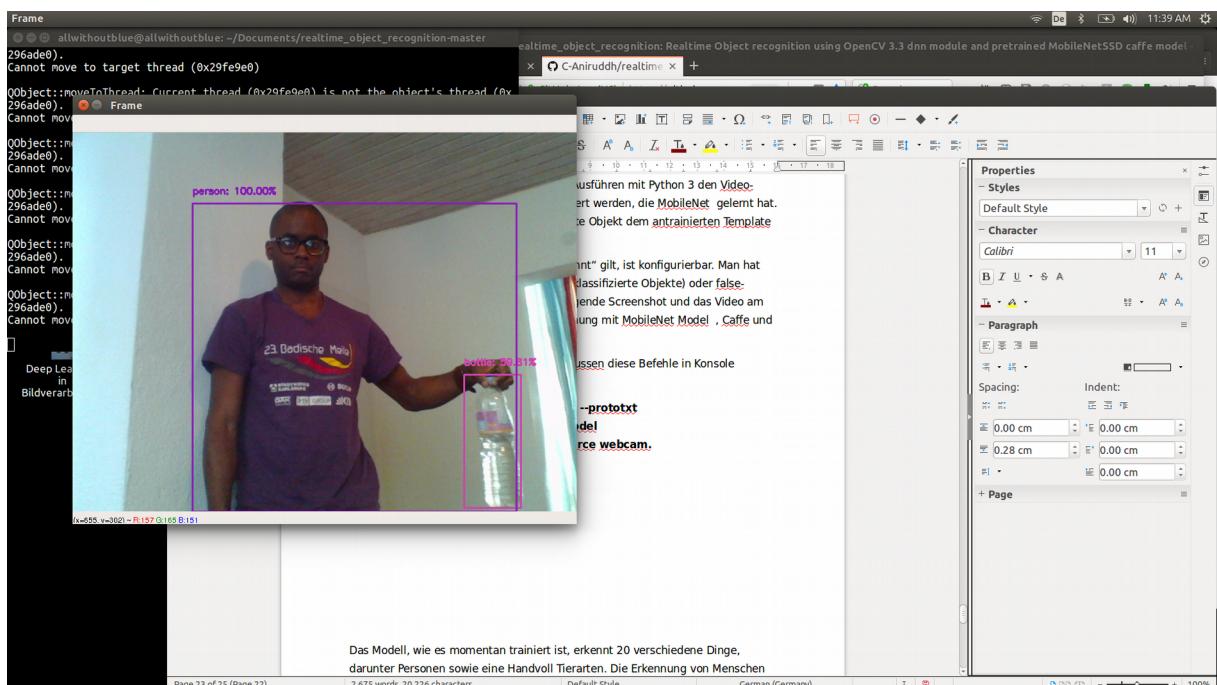
### 2.1.1 Bildverarbeitung

#### 2.1.1.1 Definition der Bildverarbeitung / Bilderkennung

Der Begriff Bilderkennung beschreibt den Prozess der Auswertung eines Einzelbildes in Hinblick auf definierte Bildinformationen. Diese Bildinformationen können dabei Farben, Formen, Muster oder auch Zusammensetzungen von Farben, Formen oder Mustern sein, also komplexe Muster wie beispielsweise menschliche Gesichter oder Alltagsgegenstände. Je nach gewünschter Anwendung kann nun die durch die Algorithmen gewonnene Bildinformation aus dem Bild extrahiert und maschinenlesbar kategorisiert, klassifiziert oder ausgegeben werden um daraufhin weiter verarbeitet zu werden.

Die Komplexität der Algorithmen steigt dabei mit der Komplexität der gesuchten Bildinformation. Sollen beispielsweise Farbinformationen in einem Bild analysiert werden, so genügt die Filterung von Farbwerten mit jeweils festgelegten Abweichungstoleranzen.

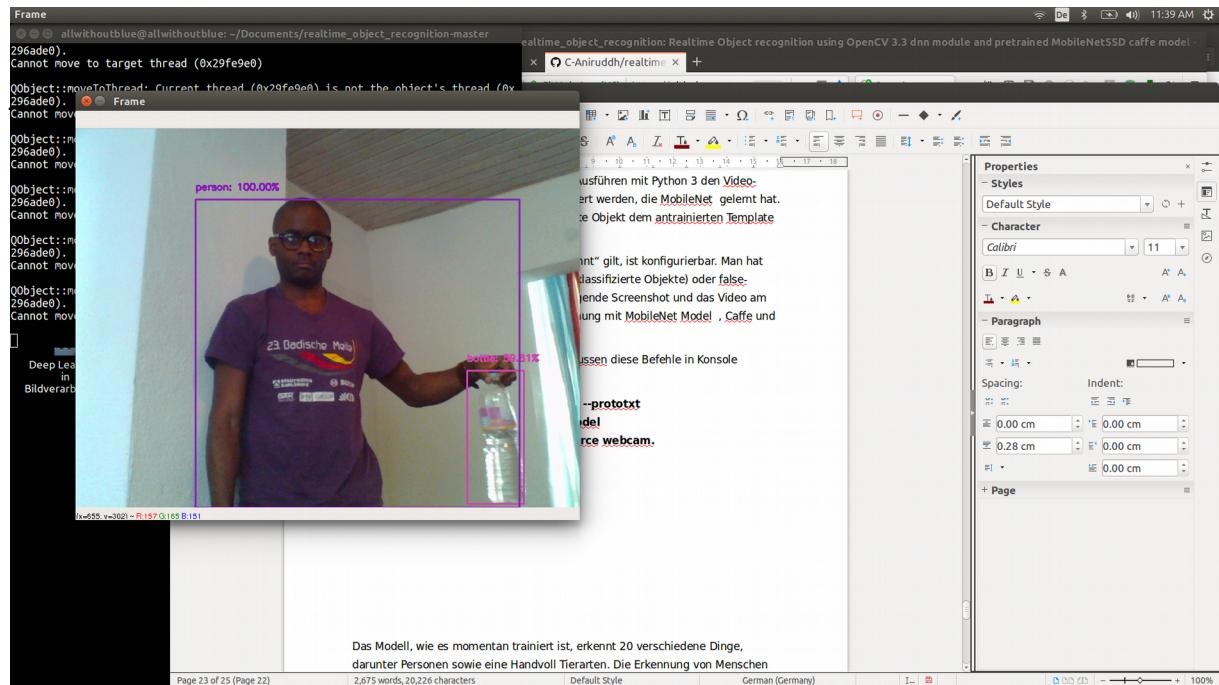




## 2.1.1.2 Opencv Framework .

**OpenCV** ist eine Open-Source Programmier-Bibliothek, die über 500 optimierte Algorithmen und Funktionen zur Video und Bildanalyse beinhaltet. Darunter auch die Filter, die für das „Experiment: Echtzeit Objekterkennung unter Einsatz von Opencv 3 and Deep Learning Framework Caffe“ genutzt wurden. OpenCV wird seit ihrer Einführung im Jahr 1999 von einer breiten Community im Feld der Bilderkennung genutzt. Ursprünglich wurde OpenCV von einem Team um Gary Bradski bei der Firma Intel entwickelt. 2006 wurde die Version 1.0 veröffentlicht.

OpenCV ist optimiert für Effizienz und Echtzeitanwendungen, es ist in der Programmiersprache C geschrieben und kann mehrere Prozessoren gleichzeitig nutzen. Das Ziel von OpenCV ist es, eine einfache Infrastruktur zu bieten, um kurzfristig Bilderkennungsanwendungen zu entwickeln.<sup>22</sup> Auch wenn OpenCV in der Programmiersprache C geschrieben wird, kann die Bibliothek unter anderem auch über Programmiersprachen wie Python und Java eingebunden werden. Genutzt wird OpenCV beispielsweise für die Analyse von Überwachungskameraaufnahmen und für die Identifizierung und Markierung von mangelhaften Produkten auf einer Produktionsstraße. Nur wenige Menschen wissen, dass Bilderkennung beispielsweise für Google Street View genutzt wird, aber gerade bei solchen Anwendungen werden die Techniken aus OpenCV, wie automatische Kamera-Kalibrierung und Bild-Zusammenfügung, in großem



## 2.1.2 Machine Learning, Deep Learning und “Caffe” Framework.

Schon früher haben Experten versucht, die Merkmale eines Bilds von Hand vorzugeben und manuell zu definieren. Doch das funktionierte nur für rudimentäre Aufgaben, wie das Finden eines runden Objekts auf einem eintönigen Hintergrund. Während ein Mensch einem anderen Menschen problemlos in wenigen Sätzen ein neues Objekt verständlich beschreiben kann, war eine Maschine bislang nicht in der Lage, ein solches Objekt eigenständig zu identifizieren.

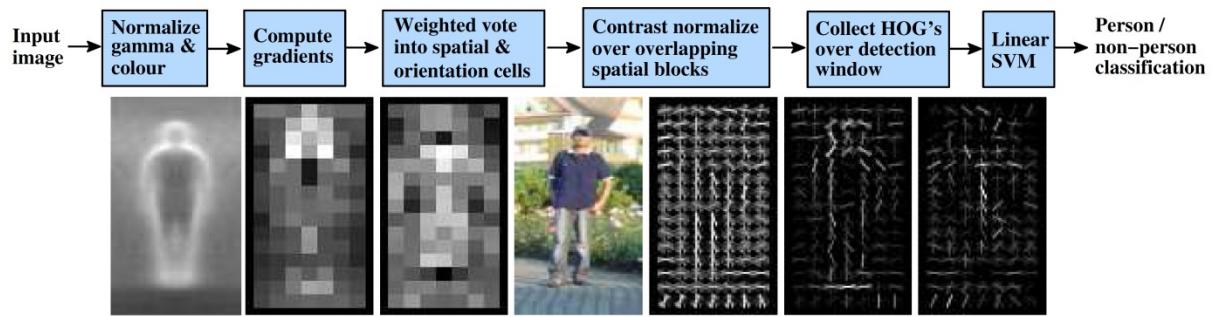
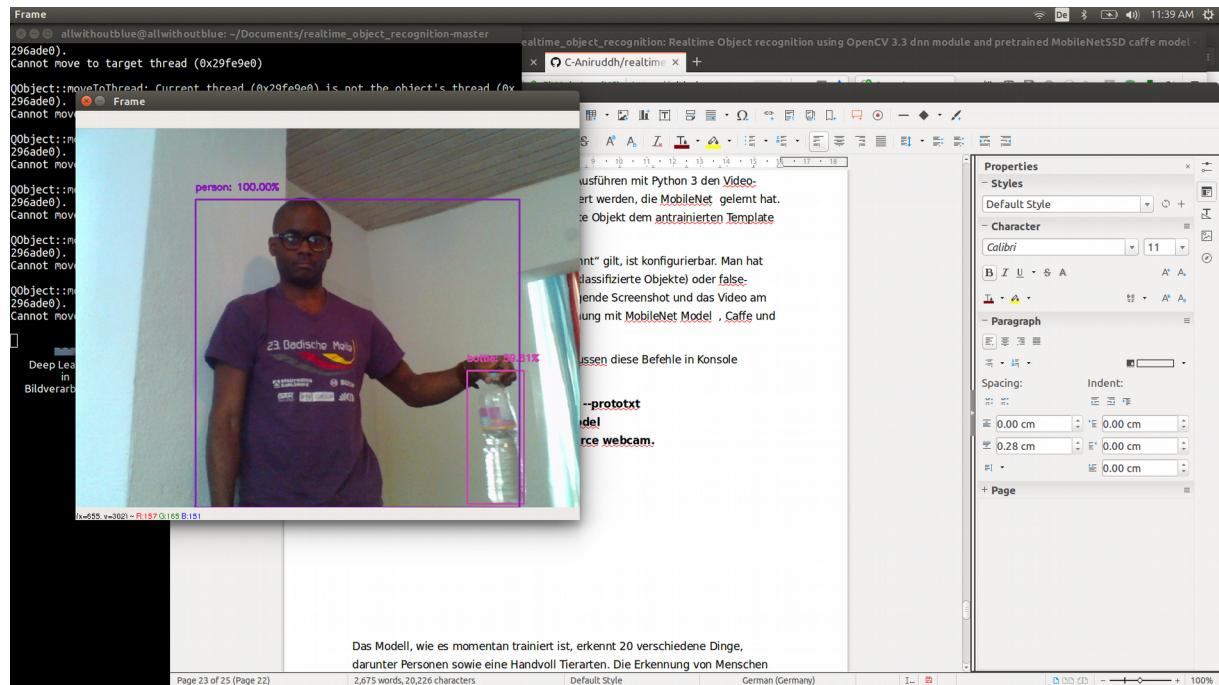


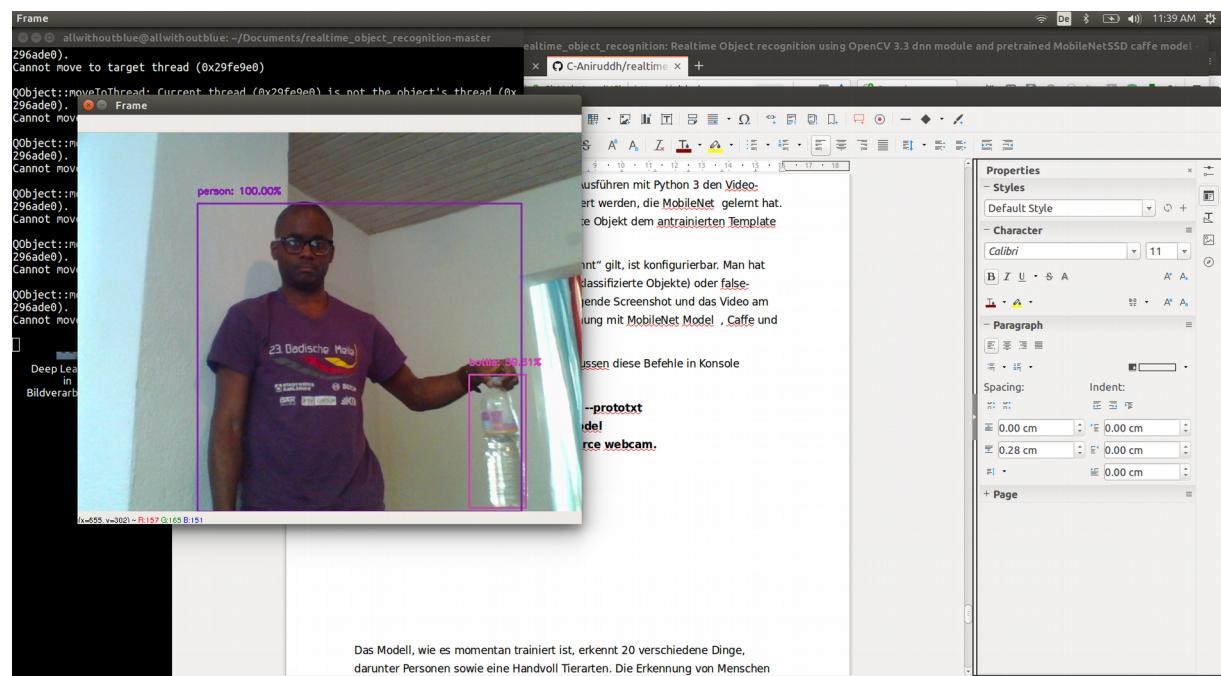
Abb.1



### 2.1.2.1 Deep Learning Definition

**Deep Learning** ist ein Ansatz im maschinellen Lernen, um Wissen aus Erfahrung zu gewinnen und das zu lösende Problem mit Hilfe einer Hierarchie von Lösungskonzepten zu verstehen, die jeweils durch einfachere Teillösungen definiert sind [10]. Durch diesen Ansatz kann die Notwendigkeit der Definition jeglichen für den Computer zur Problemlösung notwendigen Wissens durch den Programmierer vermieden werden. Die Hierarchie von Lösungskonzepten macht es dem Computer möglich, komplexe Lösungsansätze aus einfacheren Ansätzen zu bilden. Ein Graph, der diese Konzepte aufeinander abbildet, besteht aus vielen Schichten und ist somit „tief“. Deshalb spricht man bei diesem Ansatz der künstlichen Intelligenz von „Deep Learning“. Im Folgenden werden Techniken des Deep Learning und die Vorgehensweise der Bilderkennung in tiefen Netzen erläutert. Es werden mögliche Schwachstellen sowie deren Lösungsmöglichkeiten aufgezeigt und ein Machbarkeitsnachweis für die Bilderkennung durchgeführt.

### 2.1.2.2 Machinelle Learning and Deep Learning Merkmale



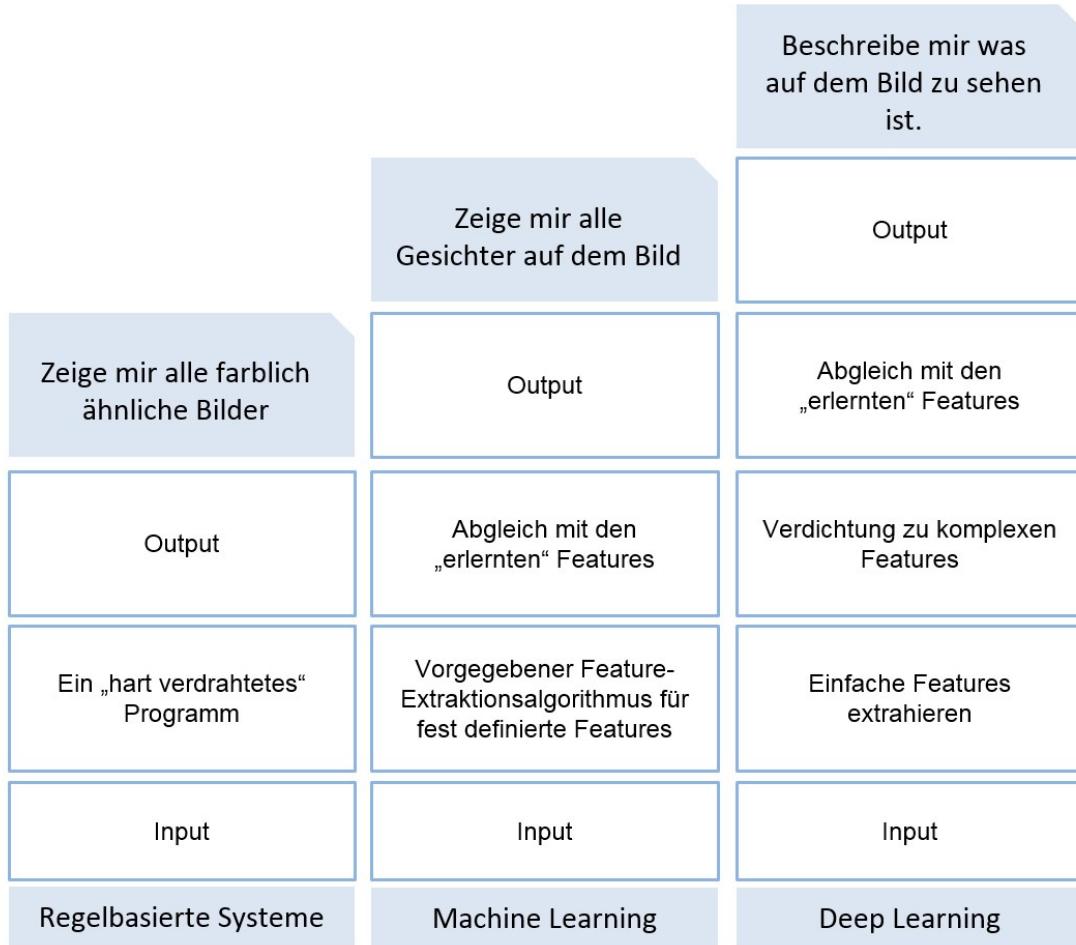
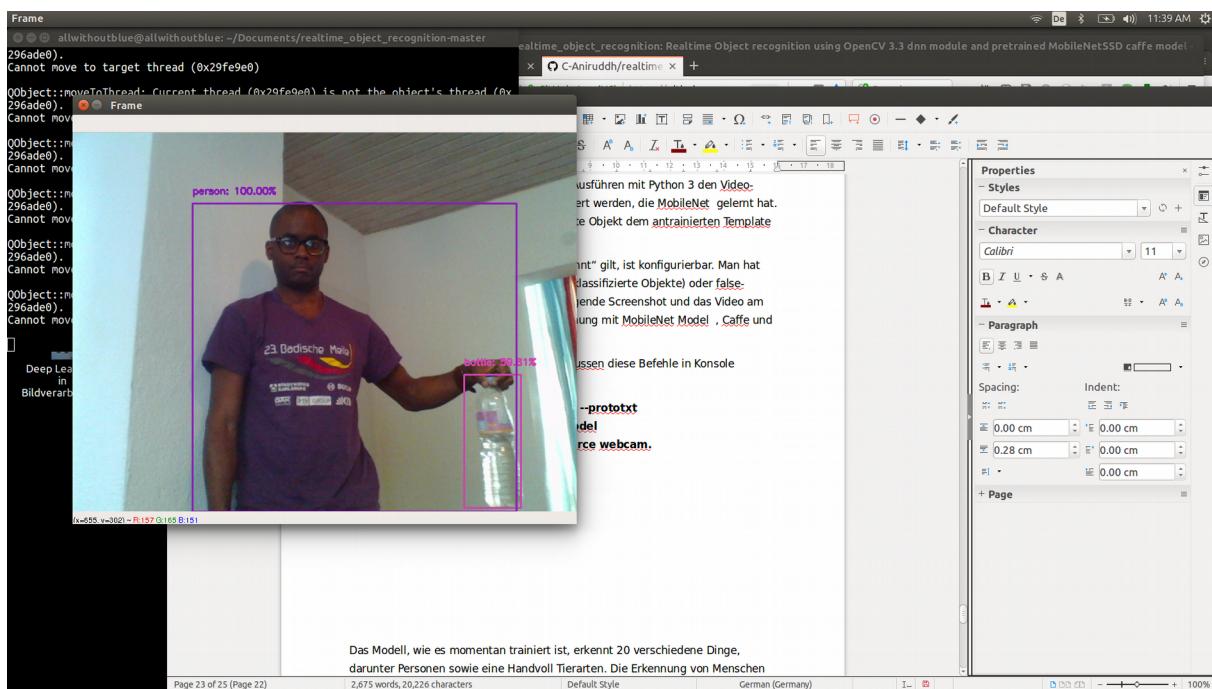


Abb 2.

Abbildung 2 zeigt Unterschiede zwischen regelbasierten Systemen, Machine Learning und Deep Learning. So kann jedes Verfahren, das einen dedizierten Algorithmus zur Featureextraktion benutzt, um eine bestimmte Aufgabenstellung zu lösen, dem Bereich Machine Learning zugeordnet werden. Dazu gehört z. B. das Erkennen von bestimmten Objekten, wie Gesicht, Menschen, Autos oder Fußgänger. Beim Deep Learning kennt das System nicht die repräsentativen



Experimente und den industriellen Einsatz bereitgestellt werden.

```
1 layers {
2   name: "conv1"
3   type: CONVOLUTION
4   bottom: "data"
5   top: "conv1"
6   convolution_param {
7     num_output: 20
8     kernel_size: 5
9     stride: 1
10  }
11 }
```

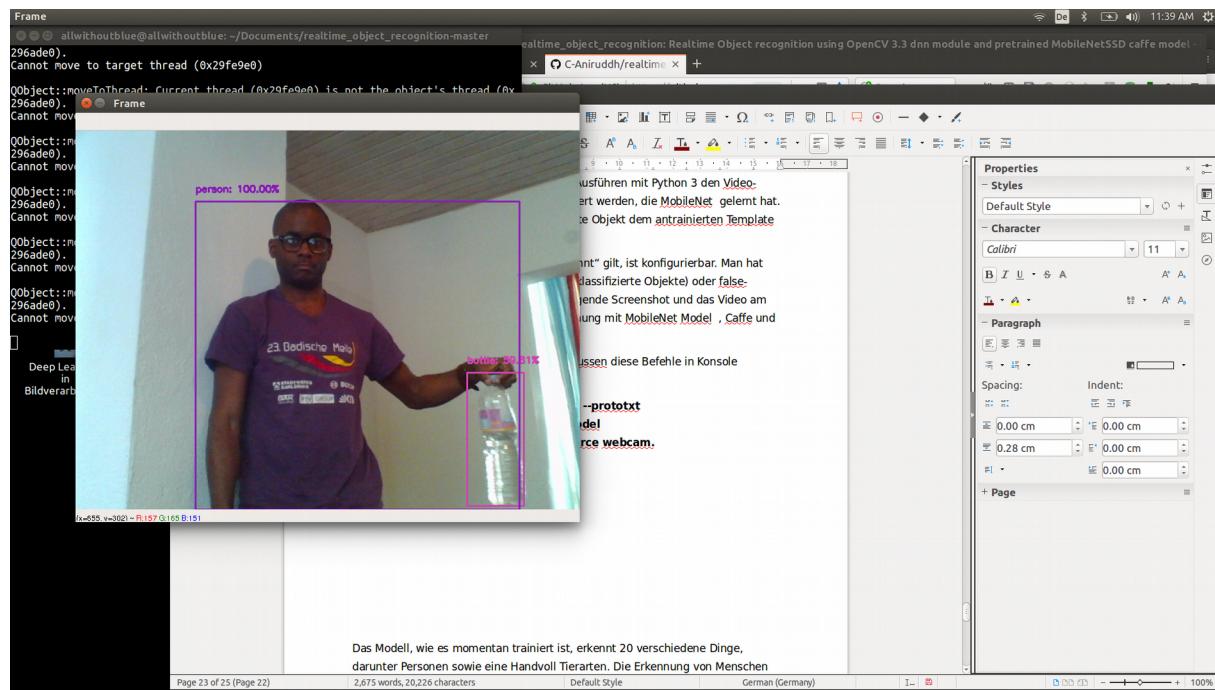
### [Abb Caffe Model .](#)

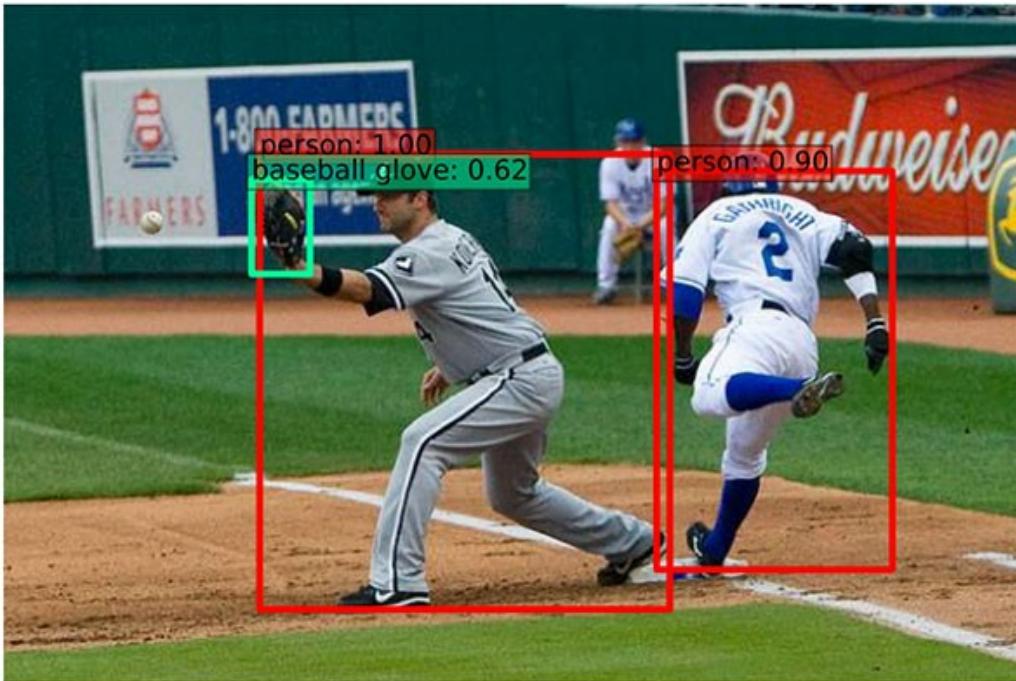
Jede Modellschicht ist auf diese Weise [definiert](#).. Es kann 99% Genauigkeit in weniger als einer Minute mit GPU-Training erreichen.

Kurz zusammengefasst kann sagen dass das Framework Caffe ist ein Deep Learning Framework , das uns vor allem Schnelligkeit und Modularität bietet. Es wird von Berkeley AI Research (BAIR) und von Community Contributors entwickelt. **Yangqing Jia** hat das Projekt während seiner Promotion an der UC Berkeley erstellt. Caffe wird unter der BSD 2-Clause Lizenz veröffentlicht.

Eine Offiziale definition der Abkürzung Caffe ist: **Convolution Architecture For Feature Extraction (CAFFE)** und seine wichtige Merkmale sind unter anderen :

- **Command Line , Python , Matlab Interfaces**
- **Referenz modelle , viele Demos.**
- **Funktioniert einwandfrei mit dem CPU und GPU**
- **Sehr häufig benutzt in Bereich Computer Vision.**
- **Pure C++/Cuda Architektur für Deep Learning.**

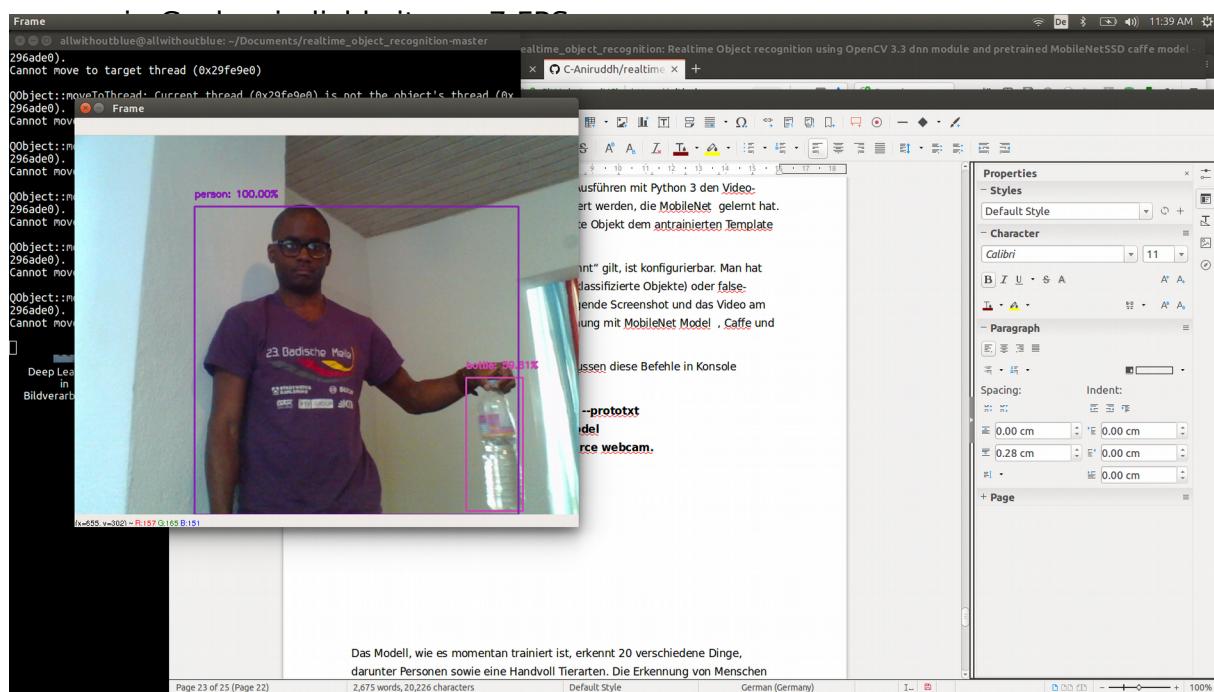


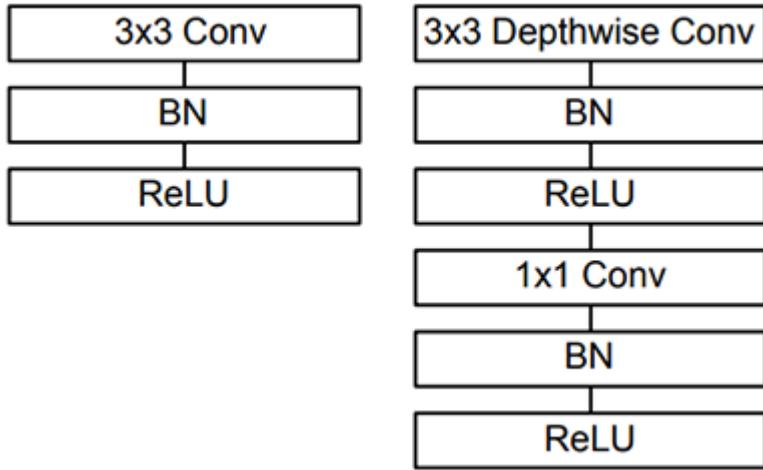


Biespiel von Objekterkennung mit dem Hilfe von Single Shot Detectors (SSD) von Liu and ai

Wenn es um Deep-Learning-basierte Objekterkennung geht, gibt es drei primäre Objekterkennungsmethoden, denen Sie wahrscheinlich begegnen werden:

- **Faster R-CNNs :** R-CNNs sind wahrscheinlich die am häufigsten gehörte Methode zur Objekterkennung mit Deep Learning. Die Technik kann jedoch schwierig zu verstehen sein (vor allem für Anfänger im Deep Learning), schwer zu implementieren und schwierig zu trainieren. Darüber hinaus kann der Algorithmus sogar mit den "schnelleren" Implementierungs-R-CNNs (wobei "R" für "Region Proposal" steht) ziemlich langsam sein, mit



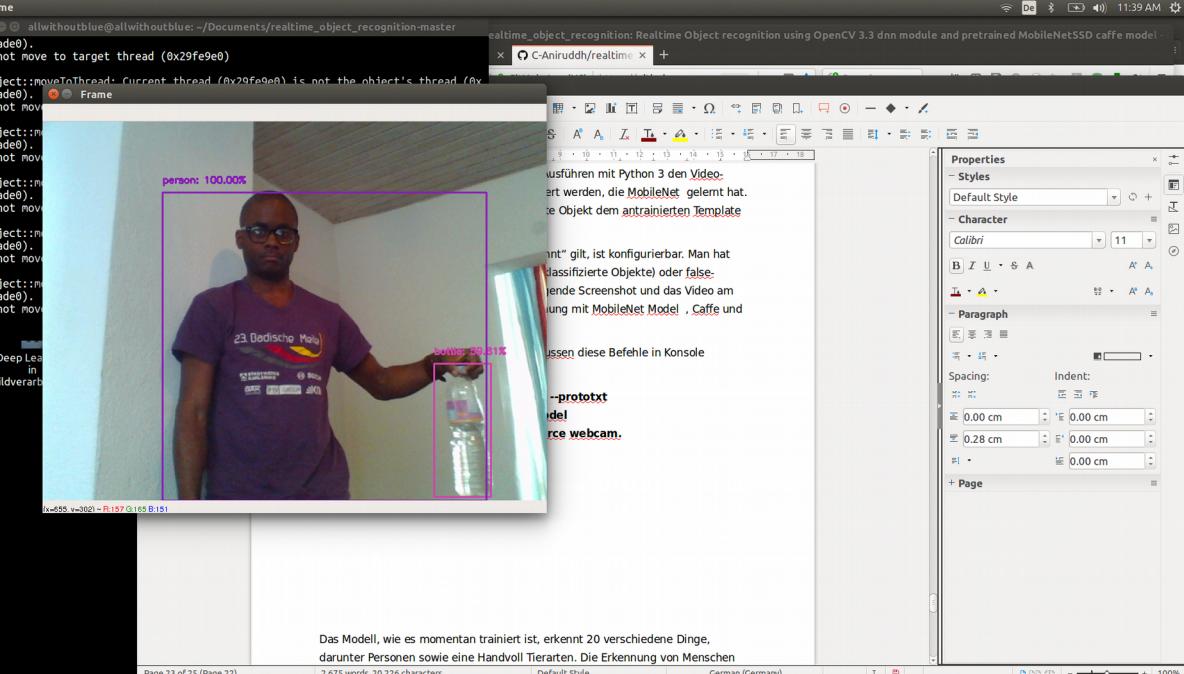


(Left) Standard convolutional layer with batch normalization and ReLU. (Right) Depthwise separable convolution with depthwise and pointwise layers followed by batch normalization and ReLU (figure and caption from Liu et al.).

Beim Erstellen von Objekterkennungsnetzwerken verwenden wir normalerweise eine vorhandene Netzwerkarchitektur wie VGG oder ResNet . Das Problem ist, dass diese Netzwerkarchitekturen in der Größenordnung von 200-500 MB sehr groß sein können. Netzwerkarchitekturen wie diese sind aufgrund ihrer Größe und der resultierenden Anzahl von Berechnungen für ressourcenbeschränkte Geräte ungeeignet.

Stattdessen können wir MobileNets , weil sie für ressourcenbeschränkte Geräte wie Ihr Smartphone ausgelegt sind. MobileNets unterscheiden sich von herkömmlichen CNNs durch die Verwendung von in der Tiefe trennbarer Faltung

Abb. 6



sollen wir eine schnelle, effiziente Deep-Learning-basierte Methode zur Objekterkennung erhalten .

Bereits im August 2017 wurde OpenCV 3.3 offiziell freigegeben und bringt ein stark verbessertes Modul "Deep Neural Networks" (dnn) mit.

Dieses Modul unterstützt eine Reihe von Deep-Learning-Frameworks, einschließlich Caffe, TensorFlow und Torch / PyTorch.

Wie bereits erwähnt werde ich im Rahmen dieser Projekt Caffe , MobileNet Model mit Opencv3 benutzen . Wenn man OpenCVs Deep neuronales Netzwerkmodul mit Caffe-Modellen verwenden möchte , benötigen wir zwei Sätze von Dateien: :

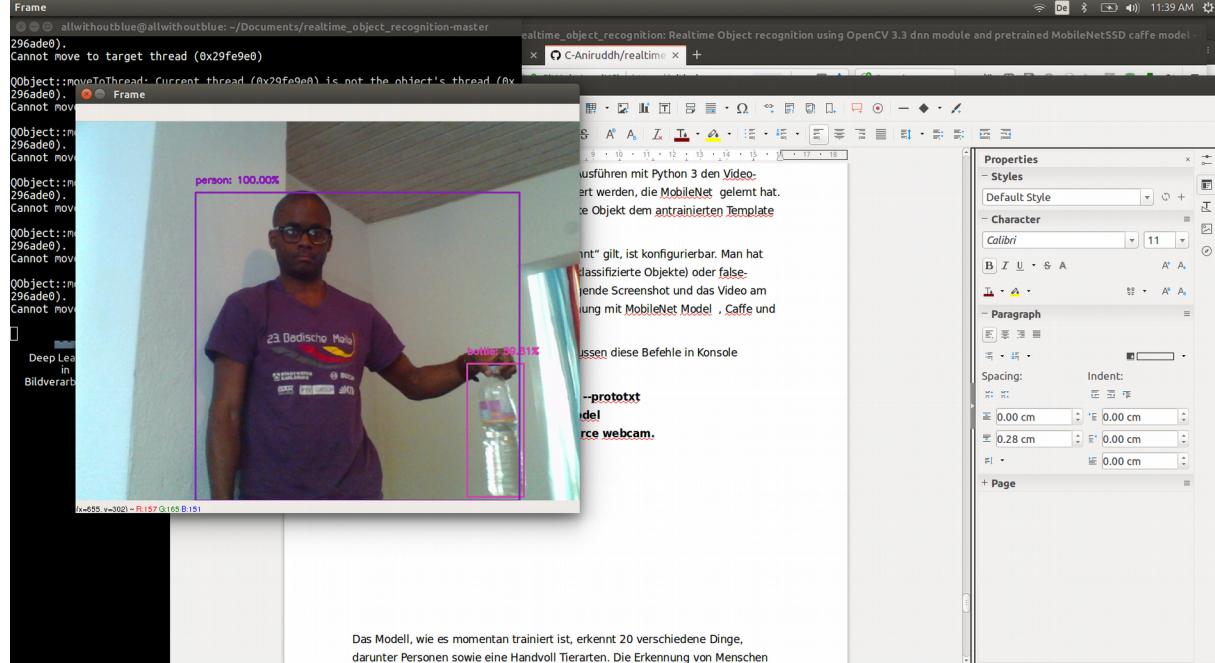
- Die **.prototxt-Datei** (en), die die Modellarchitektur definieren.
- Die **.caffemodel-Datei**, die die Gewichtungen für die tatsächlichen Ebenen enthält

Um unseren Deep Learning-basierten Echtzeit-Objektdetektor mit OpenCV zu erstellen, müssen wir (1) effizient auf unseren Webcam / Video-Stream zugreifen und (2) die Objekterkennung Modelle auf jeden Frame anwenden.

- Erste Etape Wir beginnen damit, Pakete auf den Zeilen 4-12 zu importieren.

```
# USAGE
# python real_time_object_detection.py --prototxt MobileNetSSD_deploy.prototxt.txt --model MobileNetSSD_depl
```

```
# import the necessary packages
from imutils.video import VideoStream
import numpy as np
import sys
import argparse
import imutils
import time
import cv2
```



Das Modell, wie es momentan trainiert ist, erkennt 20 verschiedene Dinge, darunter Personen sowie eine Handvoll Tierarten. Die Erkennung von Menschen

```

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("--prototxt", required=True,
    help="path to Caffe 'deploy' prototxt file")
ap.add_argument("--model", required=True,
    help="path to Caffe pre-trained model")
ap.add_argument("--source", required=True,
    help="Source of video stream (webcam/host)")
ap.add_argument("-c", "--confidence", type=float, default=0.2,
    help="minimum probability to filter weak detections")
args = vars(ap.parse_args())

```

Als nächste werden die Liste der Klassenbezeichnungen, auf denen MobileNet SSD trainiert wurde , initialisiert und anschliessend wir hier generiert eine Reihe von Bounding Box-Farben für jede Klasse

```

# Initialisiere die Liste der Klassenbezeichnungen, auf denen MobileNet SSD trainiert wurde
# detect und generiert dann eine Reihe von Bounding Box-Farben für jede Klasse
CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat",
    "bottle", "bus", "car", "cat", "chair", "cow", "diningtable",
    "dog", "horse", "motorbike", "person", "pottedplant", "sheep",
    "sofa", "train", "tvmonitor"]
COLORS = np.random.uniform(0, 255, size=(len(CLASSES), 3))

```

- Laden von unseren Model auf dem Fastplatte , erfolg mit diese Befehle :

```

# laden von unseren Model auf dem Fastplatte
print("[INFO] loading model...")
net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])

```

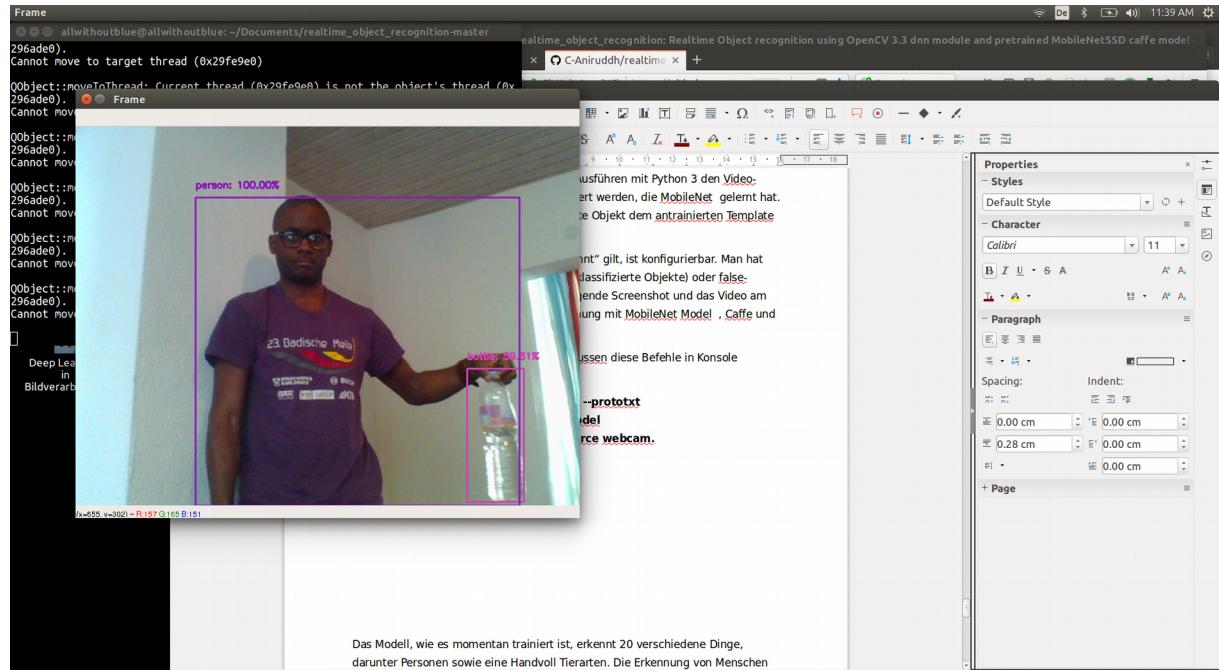
- Danach werden das Kamera initialisiert mit dem Befehle :

```

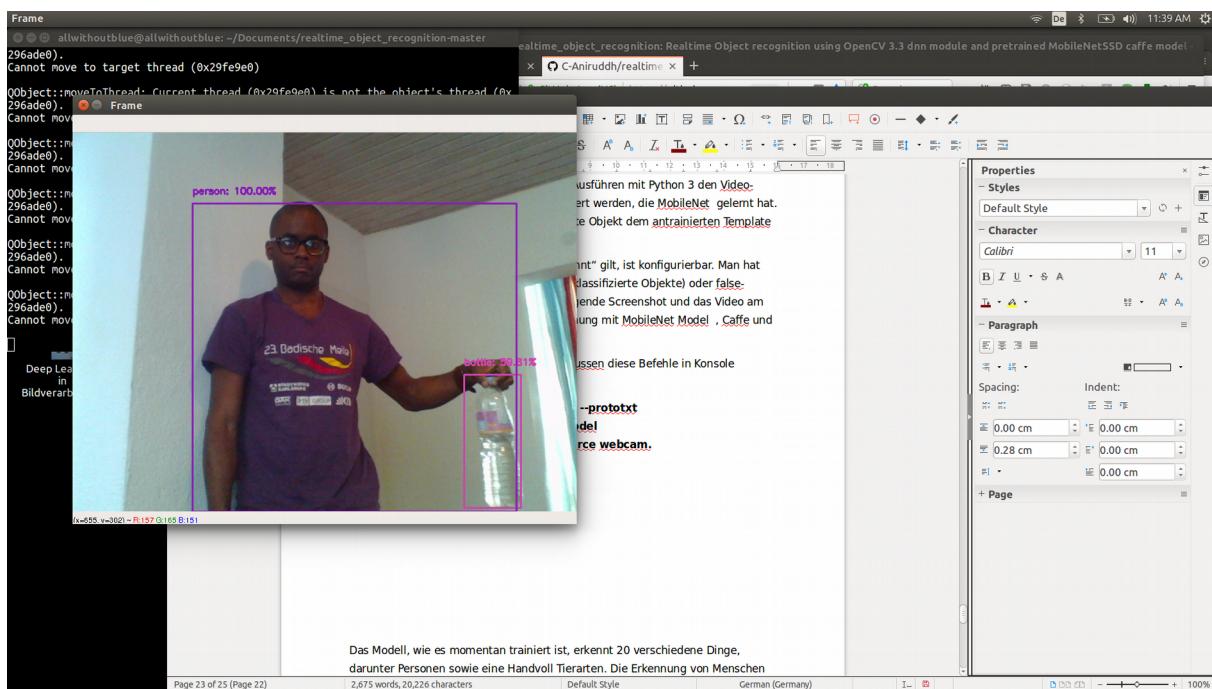
#Initialisierung von VideoStream und kurz Tempo
print("[INFO] starting video stream...")

if args["source"] == "webcam":
    vs = cv2.VideoCapture(0)

```



- Dann in der **For Schleife** müssen wir Extrahiere die , confidence ' (oder . Wahrscheinlichkeit), die mit der Vorhersage assoziiert ist , sowie Herausfiltern von schwachen Erkennungen, indem sichergestellt wird, dass die "confidence" größer ist als die minimale Confidence
- Anschliessend nach diese Reihe von berechnungen werden die verabreiten Frames in die Console dargestellet.
- Das Programm soll sich anhalten , wenn das Key „q“ gedrukt wird.



```
File Edit Format Run Options Window Help
while True:
    # grab the frame from the threaded video stream and resize it
    # to have a maximum width of 400 pixels
    if args["source"] == "webcam":
        ret, frame = vs.read()
    else:
        imgResp=urlopen(url)
        imgNp=np.array(bytearray(imgResp.read()),dtype=np.uint8)
        frame=cv2.imdecode(imgNp,-1)

    frame = imutils.resize(frame, width=800)

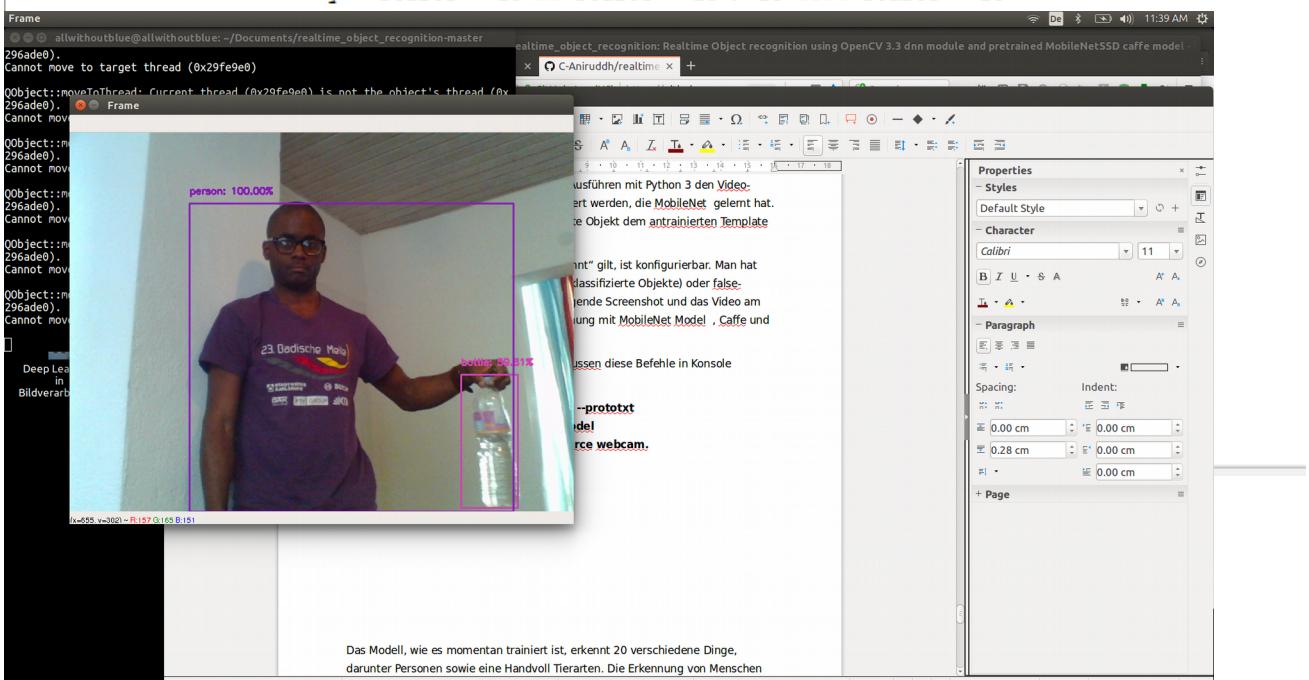
    # grab the frame dimensions and convert it to a blob
    (h, w) = frame.shape[:2]
    blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)),
        0.007843, (300, 300), 127.5)

    # pass the blob through the network and obtain the detections and
    # predictions
    net.setInput(blob)
    detections = net.forward()

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

        # filter out weak detections by ensuring the `confidence` is
        # greater than the minimum confidence
        if confidence > args["confidence"]:
            # extract the index of the class label from the
            # `detections`, then compute the (x, y)-coordinates of
            # the bounding box for the object
            idx = int(detections[0, 0, i, 1])
            box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
            (startX, startY, endX, endY) = box.astype("int")

            # draw the prediction on the frame
            label = "{}: {:.2f}%".format(CLASSES[idx],
                confidence * 100)
            detected_objects.append(label)
            cv2.rectangle(frame, (startX, startY), (endX, endY),
                COLORS[idx], 2)
            y = startY - 15 if startY - 15 > 15 else startY + 15
```



## 4 Evaluation von Experiment

Hat soweit alles geklappt, sieht man beim Ausführen mit Python 3 den Video-Stream der Webcam, in dem Objekte markiert werden, die MobileNet gelernt hat. Die Zahlen geben an, inwieweit das erkannte Objekt dem antrainierten Template ähnelt.

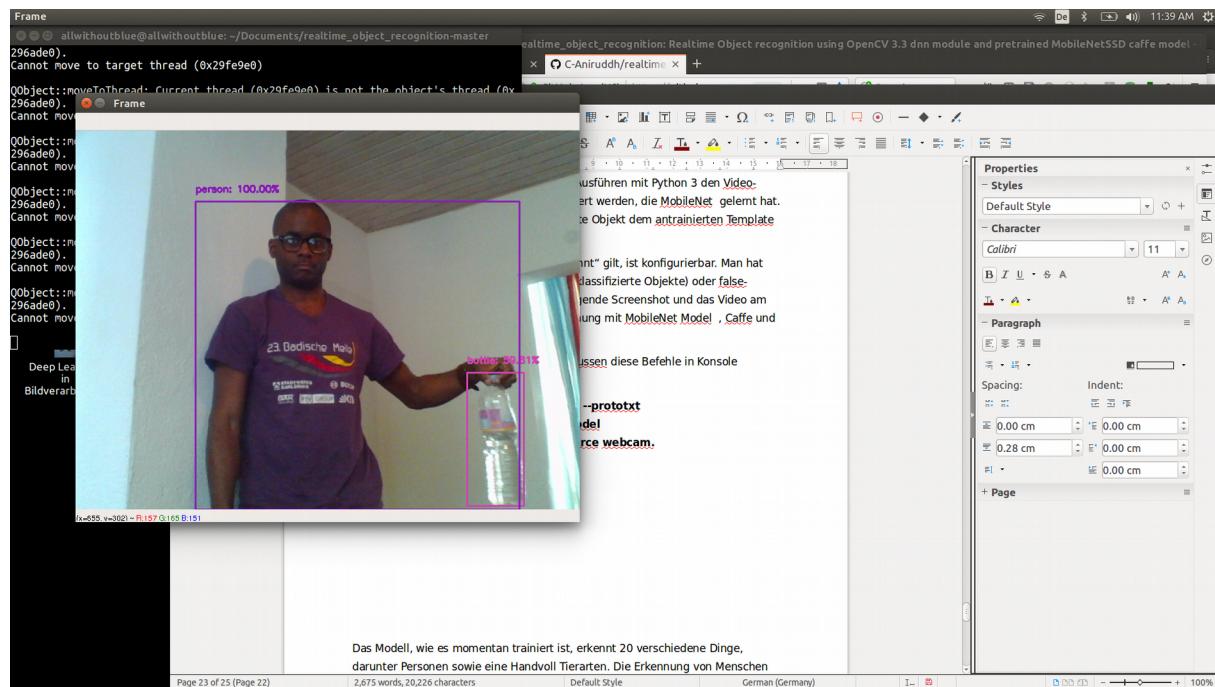
Ab welcher Ähnlichkeit ein Objekt als „erkannt“ gilt, ist konfigurierbar. Man hat also entweder mehr false-positives (falsch klassifizierte Objekte) oder false-negatives (nicht erkannte Objekte). Der folgende Screenshot und das Video am Textanfang demonstrieren die Objekterkennung mit MobileNet Model , Caffe und OpenCV in einem Webcam-Stream.

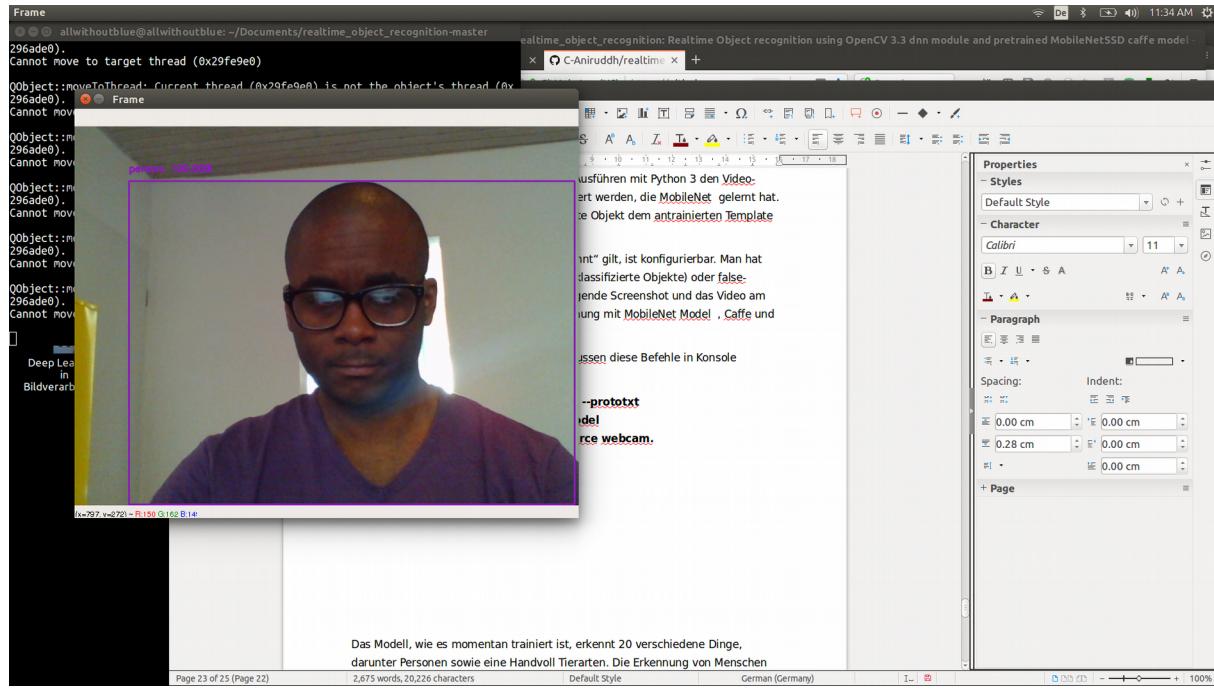
Wichtig : zum Starten das python Skript müssen diese Befehle in Konsole angeben :

```
python3 real_time_object_detection.py --prototxt  
MobileNetSSD_deploy.prototxt.txt --model  
MobileNetSSD_deploy.caffemodel --source webcam.
```

Das Modell, wie es momentan trainiert ist, erkennt 20 verschiedene Dinge, darunter Personen sowie eine Handvoll Tierarten. Die Erkennung von Menschen funktioniert am besten, bei „leblosen“ Gegenständen wie Flaschen arbeitet sie aber eher unzuverlässig.

Ergebnisse :



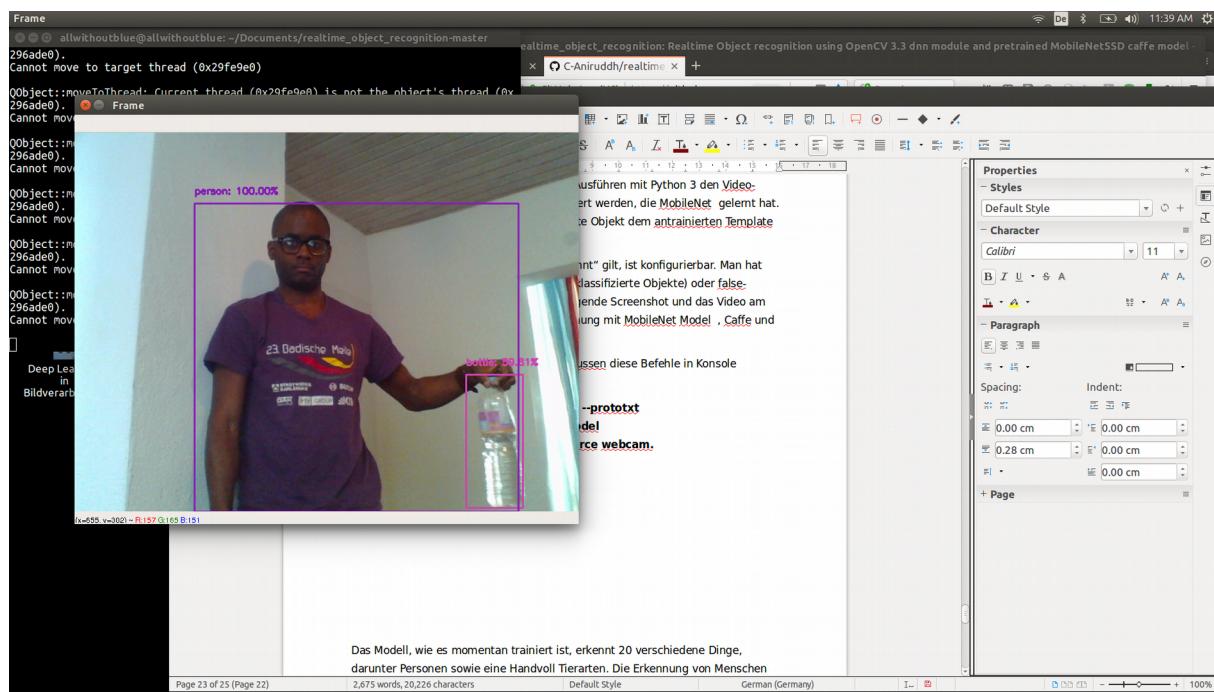


## 5 Fazit

### 5.1 Zusammenfassung

Die Bilderkennung und auch die komplexe Bilderkennung ist mit OpenCV3 (.dnn Module) und Caffe Model möglich. Eine Einschränkung bilden zeitkritische und Echtzeitanwendungen wie beispielsweise das autonome Fahren. Ist die Zeitspanne zwischen Ereignis und Ergebnis der Auswertung zu vernachlässigen, so lassen sich viele möglichen Szenarien der Bilderkennung mit dem Aktoren OpenCV3 MobileNet Model und Caffe Framework umsetzen.

Sollen komplexere Muster erkannt werden, so kann in vielen Fällen ebenfalls auf die Funktionen von OpenCV3 und Caffe Framework zurückgegriffen werden und muss seine eigene Model oder neuronale Netze trainieren. Jedoch ist hier ein



- 3- <https://prateekvjoshi.com/2016/02/23/deep-learning-with-caffe-in-python-part-iv-classifying-an-image/>
- 4- [http://caffe.berkeleyvision.org/installation.html?utm\\_source=jiji.io](http://caffe.berkeleyvision.org/installation.html?utm_source=jiji.io)
- 5- <https://github.com/opencv/opencv/wiki/Deep-Learning-in-OpenCV>
- 6- <https://jaxenter.de/big-data-bildanalyse-50313>
- 7- <http://www.kreissl.info/bilderkennung.php>
- 8- <http://unsicolta.tk/Aufsatzaufgabe-objekterkennung-open-cv-tutorial-c--1103.html>
- 9- <https://www.pyimagesearch.com/2018/02/26/face-detection-with-opencv-and-deep-learning/>

## 10-A Practical Introduction to Deep Learning with Caffe :

<http://www.panderson.me/images/Caffe.pdf>

11-<http://caffe.berkeleyvision.org/>

12-<http://caffe.berkeleyvision.org/tutorial/>

13-<https://de.wikipedia.org/wiki/Caffe>

14-[http://caffe.berkeleyvision.org/model\\_zoo.html](http://caffe.berkeleyvision.org/model_zoo.html)

15-<https://arxiv.org/abs/1512.02325>

16-<https://github.com/chuanqi305/MobileNet-SSD>

17-

