Hochschule Darmstadt
M. Sc. Data Science

SS 2022

Computer Vision

DOKUMENTATION ABSCHLUSSPROJEKT

LORENC ZHUKA & PATRICK EIDEMÜLLER

Abstract

Die Erkennung von Gesichtern ist ein gefragtes Feld in der Computer Vision. Hierbei sorgt besonders die Varianz der Daten in Bezug zu den verschiedenen Blickwinkeln und Beleuchtungen für Probleme. Die vorliegende Arbeit gewährleistet einen Einblick in das Erstellen und Anwenden eines Gesichtserkennungssystem auf Basis eines gefalteten neuronalen Netzwerks (CNN) umgesetzt auf einen Raspberry Pi zum privaten Gebrauch, speziell dem elektrischen Öffnen von Türen. Im Zuge dessen werden von der Erstellung des Bilddatensatzes der zu identifizierenden Personen, über den Vergleich eines selbst-trainierten Netzwerkes nach der LeNet Architektur zu dem vor-trainierten VGG-16 Netzwerks und schließlich den Übergang der Anwendung auf einem Raspberry Pi mit Kamera, pragmatisch dargestellt. Auf dieser Grundlage werden die Ergebnisse komplementär zu den angeführten Kapiteln im Anschluss zusammengefasst und in Hinblick auf eine realitätsnahe Anwendung reflektiert.

Inhaltsverzeichnis

Abstract **1**

Abbildungsverzeichnis **3**

Abkürzungsverzeichnis **3**

1. Konzeption **4**

1.1 Problemstellung *4*

1.2 Zielsetzung, Forschungsfrage und Hypothesen *4*

1.3 Methodik *4*

2. Convolutional Neural Network **5**

2.1 Überblick *5*

2.2 Vorteile *5*

2.3 Arbeitsweise *5*

2.4 LeNet Architektur *6*

2.5 VGG-16 ..... *7*

3. Modell nach LeNet Architektur **8**

3.1 Erstellung Bilddatensatz der zu identifizierenden Personen *8*

3.2 Datenvorbereitung *9*

3.3 Model Erstellung *10*

3.4 Perfomance *10*

4. Implementierung des VGG-16 Modells **11**

4.1 Datenvorbereitung *11*

4.2 Model Erstellung *11*

4.3 Performance *12*

5. Anwendung auf Raspberry Pi **13**

5.1 Hardware & Software *13*

5.2 Vorbereitung *13*

5.3 Test *14*

6. Zusammenfassung und Reflektion **15**

6.1 Zusammenfassung *15*

6.2 Reflektion *15*

6.3 Ausblick *16*

7. Anhang **16**

8. Literaturverzeichnis **16**

Abbildungsverzeichnis

Abbildung 1 Aufbau CNN 6
Abbildung 2 Die LeNet Architektur 7
Abbildung 3 Die LeNet Architektur im Detail 7
Abbildung 4 Die VGG-16 Architektur. Quelle: Researchgate.net 8
Abbildung 5 Kaggle Datensatz Beispielbilder 9
Abbildung 6 Code zur Modellerstellung nach LeNet 10
Abbildung 7 Performance LeNet Architektur 10
Abbildung 8 Schichten der angepassten VGG16 Architektur 12
Abbildung 9 Performance VGG-16 12
Abbildung 10 Raspberry Pi System 14
Abbildung 11 Funktionierendes System 14

Abkürzungsverzeichnis

CNN ..... *Convolutional Neural Network, Convolutional Neural Network*
Kaggle ..... *Online Plattform zum Wissensaustausch rund um Daten*
mtcnn ..... *Multi-Task Cascaded Convolutional Neural Network*
Pi ..... *Minicomputer*
ssh ..... *Secure Shell Protokoll*

1. Konzeption

1.1 Problemstellung

Die weiteste Verbreitung fand die Gesichtserkennung bisher vor allem in der Unterhaltungsbranche. Apps, welche die Gesichter spielerisch verändern oder Kameras, die erst auslösen, wenn man lächelt. Heutzutage wird von der Website bis zum Türschloss bei immer mehr geschützten Bereichen die Passworteingabe durch eine Gesichtskontrolle ersetzt. Seit 2017 gehört die Face ID, eine Technik zur Biometrischen Gesichtserkennung des US-amerikanischen Unternehmens Apple, zu dem Standard zur Entsperrung jedes iPhones sowie dem Schlüsselbund und der Passwort Eingabe im Internet.

In unserem alltäglichen Leben ist das Smartphone absolut integriert, wodurch die Technik der Gesichtserkennung zur Bildschirmentsperrung von dem Menschen schnell adaptiert wurde. Laut Apple liegt die Wahrscheinlichkeit bei 1 zu 1.000.000, dass von Fremden ein Gerät mit Face ID entsperrt werden könnte. Im Gegensatz dazu ist die Gesichtserkennung zur Haustüröffnung nicht wirklich weit verbreitet, obwohl diese großen Nutzen stiften könnte.

1.2 Zielsetzung, Forschungsfrage und Hypothesen

Vor dem Hintergrund der Problemstellung definiert sich das Ziel, ein System zu entwickeln, um die Haustür mit Gesichtserkennung öffnen zu können. Das System soll mit Hilfe eines CNN, welches auf die zu erkennende Person trainiert wird, wiedererkennen und ein Signal geben. Anschließend wird das Modell auf einem Raspberry Pi (Pi) eingesetzt, um Bilder der damit verbundenen Kamera auszuwerten. Bei den richtigen Personen soll über die GPIU Pins ein Impuls gegeben werden, der eine LED-Lampe zum Leuchten bringt. Damit wird die elektrische Tür Öffnung simuliert. Man könnte somit das Gerät am Verteilerkasten mit dem Kabel der elektrische Tür Öffnung verbinden und es nutzen.

Dabei stellt sich folgende Forschungsfrage:

Inwiefern ist es möglich ein sicheres System zu konstruieren, durch welches sich die Haustür mit Gesichtserkennung über ein CNN öffnen lässt? Wie hoch ist die falsch-positiv Rate?

Daraus ergibt sich folgende Hypothese:

Es ist möglich mit einem CNN ein sicheres System zu konstruieren mit dem sich Gesichter verlässlich erkennen lassen und somit die Haustür sicher geöffnet werden kann.

1.3 Methodik

Der erste Teil der Arbeit führt den Leser in das Themengebiet der gefalteten neuronalen Netze und deren Arbeitsweise ein. Dabei wird die LeNet Architektur sowie das VGG-16 Modell kurz erklärt.

Im zweiten Teil geht es um die Erstellung und das Trainieren eines Modells nach der LeNet Architektur. Von der Erstellung des Bilddatensatzes über die Datenvorbereitung zur Modellerstellung und schließlich der Performance dieses selbst trainierten Modells.

Der dritte Teil beschäftigt sich mit der Implementierung des vor-trainierten VGG-16 Modells. Und behandelt die Vorbereitung und dessen Implementierung.

Anschließend wird im vierten Teil die Anwendung auf dem Raspberry Pi erläutert. Von der Hardware, über die Einrichtung und dem Anwenden des Modells auf dem Mini Computer.

Schließlich folgen eine Zusammenfassung und Reflektion.

2. Convolutional Neural Network

Dieses Kapitel bietet einen kurzen Überblick über die Grundlagen. Es behandelt die Convolutional Neural Networks, sowie die im anschließenden Kapitel verwendete LeNet Architektur und einen Einblick in das vortrainierte VGG-16 Modell.

2.1 Überblick

Ein Convolutional Neural Network (CNN) ist eine besondere Form eines künstlichen neuronalen Netzwerkes. Der Unterschied zeigt sich dadurch, dass ein CNN, wie sich aus der deutschen Übersetzung „Gefaltetes Neuronales Netzwerk“ ableiten lässt, aus mehreren Faltungsschichten aufgebaut ist, wodurch es besonders gut für Bild- und Spracherkennung geeignet ist. Es lässt sich behaupten, dass die Funktionsweise zu einem gewissen Teil biologischen Vorgängen, wesentlich der Sehrinde des Gehirnes, nachempfunden ist.

2.2 Vorteile

Herkömmliche nicht gefaltete Neuronale Netze kommen bei der Erkennung von Bildern schnell an ihre Grenzen, da die Anzahl der Knoten der Eingabeschicht gleich der Pixelanzahl des Bildes sein muss. Dadurch, dass Bilder eine hohe Anzahl an Pixel haben, wird das sehr rechenintensiv. CNN Netzwerke sind sehr robust und gegenüber Verzerrungen oder anderen optischen Veränderungen unempfindlich. Ein weiterer Vorteil ist, dass bei Eingabebildern unter verschiedenen Lichtverhältnissen und Perspektiven dennoch die typischen Merkmale eines Bildes erkannt werden.

2.3 Arbeitsweise

Wie bereits erwähnt besteht ein CNN aus verschiedenen Schichten. Grundsätzlich kann man sagen, dass die ersten Schichten Details aus dem Bild filtern, danach werden die Bildstrukturen herausgefiltert und dann zunehmend die Bildinformationen abstrahiert bis schließlich von einer vollständig verbundenen Schicht die Klassifizierung erfolgt.

Im Folgenden betrachten wir die Arbeitsweise genauer. Die nachfolgende Abbildung zeigt wie ein CNN ein Bild einer handschriftlich geschrieben Zahl erkennt.

In dem Convolutional Layer tasten zunächst 32 mit zufälligen Gewichten belegte Faltungskerne der Größe 3x3 pixelweise mit der Schrittweite 1 das Eingabebild ab. Zusätzlich bewirkt die Aktivierungsfunktion RelU das negative Werte auf null abgebildet werden und positive Werte ihren Wert behalten. Dadurch erhalten wir 32 unterschiedlich gefaltete Bilder mit der Größe 28x28 Pixel.

Anschließend kommt die Pooling Schicht zum Einsatz. In diesem Fall das Max-pooling mit einem Kern der Größe 2x2 Pixel, welches dafür sorgt das die 28x28 Pixel in ihrer Auflösung auf 14 x14 Pixel reduziert werden, indem nur der Pixel mit dem maximalen Wert in dem 2x2 Kern behalten wird, welcher mit einer Schrittweite von 2 über die Eingabebilder läuft. Die Idee dahinter ist, dass wir den Pixel mit der größten Information behalten möchten.

Beide Schichten wiederholen sich abwechselnd und nach der letzten Max-pooling Schicht erhalten wir 64 Bilder mit der Größe von 7x7 Pixel. Das bedeutet wir haben 3136 Eingabe Pixel für das neuronale Netz, welches in diesem Fall mit einem „hidden layer“ trainiert. Schließlich erhalten wir eine Klassifikation. Bevor das Modell bereit ist zum Einsatz muss es trainiert werden.

In der Trainingsphase werden die Fehler der Klassifikationen der Trainingsdaten durch „back propagation“ zurück in das Netz verteilt und mit dem Gradienten Abstiegsverfahren minimiert. Dadurch werden die in den Convolutional Layer zufällig gesetzten Gewichte der Faltungskerne optimiert.

Wenn das Modell vollständig trainiert ist, kann ein neues fremdes Bild als Input genommen werden und das Netz klassifiziert dieses Bild.

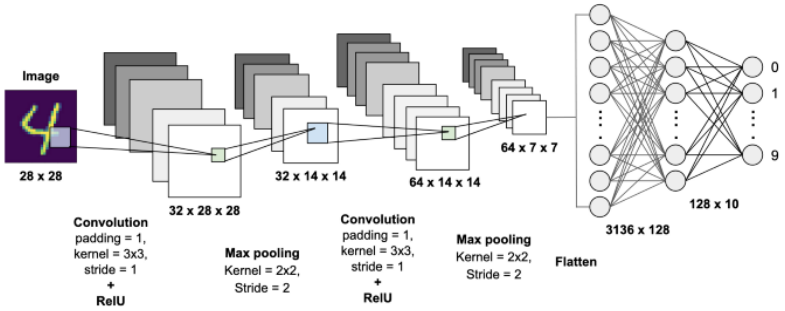


Abbildung 1 Aufbau CNN

2.4 LeNet Architektur

Die LeNet Architektur wurde erstmal in dem Forschungsbericht „Gradient-Based Learning Applied To Document Recognition“ im Jahr 1998 von Yann LeCun, Leon Bottou, Yoshua Bengio und Patrick Haffner, erwähnt. Besonders durch die einfache und unkomplizierte Art der Architektur wurde diese sehr bekannt.

Die LeNet Architektur besteht aus fünf Schichten. Drei dieser Schichten sind Faltungsschichten mit der Aktivierungsfunktion tanh jeweils kombiniert mit dem durchschnittlichen Pooling und schließlich gibt es zwei vollständig verbundene Schichten, wobei letztere die Aktivierungsfunktion Softmax benutzt, um die Bilder zu klassifizieren.

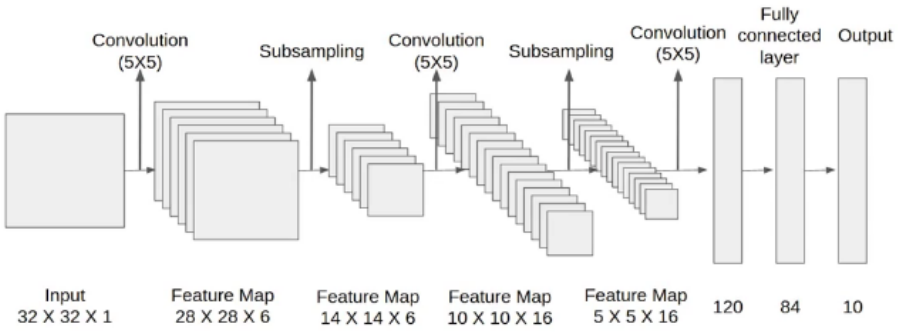


Abbildung 2 Die LeNet Architektur

Das Eingabebild ist ein 32x32 Pixel Graubild mit einem Kanal, auf welches sechs Faltungskerne der Größe 5x5 Pixel pixelweise angewendet wird. Dadurch erhalten wir sechs Bilder der Größe 28x28.

Anschließend folgt die Poolingschicht welche nach dem average-pooling Prinzip funktioniert. Das bedeutet, dass im Gegensatz zu dem max-pooling Verfahren, bei welchem wir das Pixel mit dem größten Wert weitergeben, den Durchschnitt der Pixel in dem Kern berechnen. Somit wird die Größe der sechs Bilder, durch den 2x2 Pixel Kern mit der Schrittweite eins, um die Hälfte auf 14x14 Pixel reduziert.

Anschließend folgt eine Faltungsschicht mit 16 Kernen der Größe 5x5 Pixel mit Schrittweite eins, womit die Größe der Bilder auf 10x10 Pixel reduziert wird. Nun wird wieder eine Poolingschicht angewendet und die Größe der Bilder erneut reduziert.

Zuletzt folgt eine Faltungsschicht mit 120 Kernen der Größe 5x5 Pixel, wodurch 120 Werte entstehen, welche in das Neuronale Netz mit 84 Neuronen übergeben werden. Die Größe der Ergebnisschicht hängt von der Anzahl der zu klassifizierenden Klassen ab.

Die Anzahl der trainierbaren Parameter dieser Architektur sind ungefähr 60.000.

Layer	# filters / neurons	Filter size	Stride	Size of feature map	Activation function
Input	-	-	-	32 X 32 X 1	
Conv 1	6	5 * 5	1	28 X 28 X 6	tanh
Avg. pooling 1		2 * 2	2	14 X 14 X 6	
Conv 2	16	5 * 5	1	10 X 10 X 16	tanh
Avg. pooling 2		2 * 2	2	5 X 5 X 16	
Conv 3	120	5 * 5	1	120	tanh
Fully Connected 1	-	-	-	84	tanh
Fully Connected 2	-	-	-	10	Softmax

Abbildung 3 Die LeNet Architektur im Detail

2.5 VGG-16

VGG-16 ist eine Faltungsarchitektur von Karen Simonyan und Andrew Zisserman von der Visual Geometry Group Lab der Universität Oxford und wurde 2014 durch eine Top Platzierung in der ImageNet Large Scale Visual Recognition Challange (ILSVRC) bekannt.

Als Eingabe werden RGB Bilder der Größe 224x224 Pixel benötigt. Es gibt insgesamt 13 Faltungsschichten und drei vollständig verbundene Schichten. Die Filterkerne in VGG16 sind mit 3x3 Pixel kleiner als in der LeNet Architektur.

Die ersten beiden sind Faltungsschichten mit 64 3x3 Pixel Kernen und der Schrittweite eins, wodurch 64 Bilder mit der Größe von 224x224 Pixel entstehen.

Die erste Pooling Schicht, welche das max-pooling Prinzip mit einem Kern der Größe 2x2 und Schrittweite zwei nutzt, reduziert die Größe der Bilder auf 112x112 Pixel.

Nun folgen wieder zwei Faltungsschichten mit 128 Kernen, wodurch sich die Anzahl der Bilder auf 128 erhöht. Die Nachfolgende max-Poolingschicht reduziert die Größe wiederum auf 56x56 Pixel.

Es folgen zwei weitere Faltungsschichten mit 256 Kernen. Gefolgt von einer Pooling Schicht, welche die Größe auf 28x28 Pixel reduziert.

Nun folgen zwei mal drei Faltungsschichten jeweils getrennt von einer max-Poolingschicht. Nach der letzten Poolingschicht wird eine Größe von 7x7 Pixel mit der Anzahl von 512 Bildern erreicht. Das bedeutet wir haben 4096 Eingabepixel für das Netz.

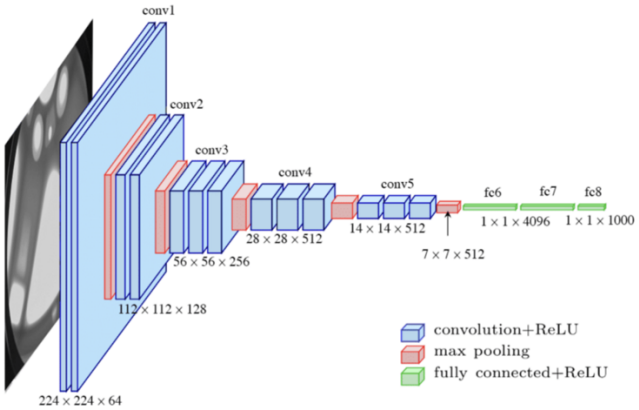


Abbildung 4 Die VGG-16 Architektur. Quelle: Researchgate.net

3. Modell nach LeNet Architektur

Dieses Kapitel erläutert die Erstellung und das Trainieren eines Modells nach der LeNet Architektur.

3.1 Erstellung Bilddatensatz der zu identifizierenden Personen

Um das CNN trainieren zu können benötigen wir einen großen Datensatz mit möglichst vielen verschiedenen Aufnahmen von Gesichtern. Dazu haben wir einen Datensatz von Kaggle benutzt, welcher mit über 7000 verschiedensten Aufnahmen von Gesichtern von jung bis alt in zeitgemäßer Qualität unsere Ansprüche erfüllt. Diese Bilder wurden mit dem Label „andere“ markiert.

Abbildung 5 Kaggle Datensatz Beispielbilder

Zusätzlich benötigen wir noch möglichst verschiedene Bilder von den zu identifizierenden Personen. In unserem Fall gibt es zwei zu identifizierende Personen: Lorenc und Patrick. Es ist jedoch möglich die Anzahl der zu identifizierenden Personen zu erhöhen. Die Bilder sollten möglichst divers in Hinblick auf die Beleuchtungen, Neigungen und Hintergründe sein. Um die Bilder aufzunehmen haben wir ein kurzes Skript geschrieben, welches über die Webcam des Computers Aufnahmen macht. Damit kann man zu unterschiedlichen Belichtungen, Abständen und Neigungen an unterschiedlichen Orten mit verschiedenen Hintergründen Bilder aufnehmen. Die Bilder werden jeweils in einem Ordner mit dem entsprechenden Label gespeichert.

3.2 Datenvorbereitung

Nun erstellen wir ein Python Dictionary aus den Bildpfaden mit den zugehörigen Labels. Mit dem Wählen einer zufälligen Stichprobe aus den Bildern können wir auch das Verhältnis der Labels untereinander bestimmen.

Anschließend haben wir mit dem mtcnn die Gesichter aus den Bildern geschnitten. Dadurch erhöhen wir die Stabilität des Modells. Die Selbsterstellten Bilder hatten viel Hintergrund im Gegensatz zu den Bildern aus dem Kaggle Datensatz, welche sehr wenig Hintergrund im Bild haben. In der Retrospektive ist dies ein bedeutender Schritt für die Güte des Modells.

Bevor die Bilder weiterverarbeitet werden, muss noch deren Größe auf 400 x 400 Pixel angepasst werden. Damit wir ein Neurales Netz trainieren können, müssen alle Bilder die gleiche Größe haben.

Anschließend wenden wir einen Graufilter an, der das Farbschema der Bilder in Graustufen transformiert und die Pixel der Bilder werden zwischen 0-1 skaliert. Mit drei Kanälen würde unser Datensatz viel grösser sein und daher müssten mehr Gewichte trainiert werden. Damit das Gradienten Abstiegsverfahren schneller konvergiert skalieren wir die Daten.

Schließlich werden die Bilder noch gemischt und in ein Training und Testdatensatz im Verhältnis von 80 zu 20 aufgeteilt.

Die Bilder durch den Computer zu drehen, neigen oder zu filtern hat keine bedeutende Verbesserung der Güte ergeben. Daher haben wir uns entschieden diesen Schritt zur Veränderung der selbst aufgenommenen Bilder wegzulassen.

3.3 Model Erstellung

CNN (LeNet Architektur) mit:

- Conv2D mit 32 Kernen der Größe 3x3 Pixel und RelU Aktivierungsfunktion
- Maxpooling2D mit Kern der Größe 2x2 Pixel
- Conv2D mit 64 Kernen der Größe 3x3 Pixel und RelU Aktivierungsfunktion
- Maxpooling2D mit Kern der Größe 2x2 Pixel
- Conv2D mit 64 Kernen der Größe 3x3 Pixel und RelU Aktivierungsfunktion
- Dropout mit Wahrscheinlichkeit 0.5
- Dense
- Dropout mit Wahrscheinlichkeit 0.5

```python
import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, AveragePooling2D, Dense, Flatten, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import TensorBoard
from sklearn.model_selection import train_test_split

def build_model():
    model = Sequential()

    model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=X_train_full_one_channel.shape[1:]))
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))

    model.add(Flatten())
    model.add(Dropout(0.5))
    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.5))


    model.add(Dense(units=3, activation = 'softmax'))

    # we must use sparse_categorical_crossentropy since we have not encoded the target feature into one hot features
    model.compile(loss ='sparse_categorical_crossentropy', optimizer=Adam(lr=0.001),metrics='accuracy')

    return model
```

Abbildung 6 Code zur Modellerstellung nach LeNet

3.4 Perfomance

Die Accuracy ist 0.96. Das heißt 96% der Bilder wurden von unserem Modell richtig klassifiziert. Am meisten wurden die Bilder von dem Label andere richtig klassifiziert, da dieser f1-score am größten ist. 10 Bilder der Klasse „patrick“ wurden fälschlicherweise in die Klasse andere klassifiziert. Bei der Klasse „lorenc“ wurden 7 Bilder zu andere und 2 zu Patrick falsch klassifiziert. Die Performance dieses Modells ist trotzdem sehr gut. Die falsch-positiv Rate ist hervorragend kein Bild der Klasse andere wurde als „lorenc“ oder „patrick“ klassifiziert.

	precision	recall	f1-score	support
0	1.00	0.95	0.97	317
1	0.90	0.98	0.94	92
2	0.91	1.00	0.95	91
accuracy			0.96	500

	andere	patrick	lorenc
andere	300	0	0
patrick	10	90	0
lorenc	7	2	91

Abbildung 7 Performance LeNet Architektur

4. Implementierung des VGG-16 Modells

Dieses Kapitel handelt über die Implementierung des vortrainierten VGG-16 Modells.

4.1 Datenvorbereitung

Die Vorbereitung der Bilder für das Training des vortrainierten VGG-16 Modells erfolgt fast genauso wie bei dem anderen Netz, also das Ausschneiden der Gesichter, die Anpassung der Größe, die Transformation in Graustufen und die Skalierung der Bilder in den Bereich von null bis 1. Hier müssen wir nur aufpassen, dass die Bilder 3 Kanäle (RGB) haben statt nur einen, denn das vortrainierte Netz erwartet 3 Kanäle als Eingabe.

4.2 Model Erstellung

Wir haben die Architektur von dem vortrainierten VGG-16 genommen und die Gewichte eingefroren. Daher werde die Gewichte des VGG-16 nicht weitertrainiert. Dazu fügen wir noch zwei Denseschichten und eine Dropoutschicht ein. Dann werden nur die Gewichte von diesen neuen Schichten trainiert, um sehr lange Trainingszeiten zu vermeiden.

Die Architektur von unserem aufgebauten vortrainierten Modell (VGG-16 + neue Schichten) sieht so aus:

- VGG-16 Architektur
- Flatten
- Dense
- Droput
- Dense

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 400, 400, 3)]	0
block1_conv1 (Conv2D)	(None, 400, 400, 64)	1792
block1_conv2 (Conv2D)	(None, 400, 400, 64)	36928
block1_pool (MaxPooling2D)	(None, 200, 200, 64)	0
block2_conv1 (Conv2D)	(None, 200, 200, 128)	73856
block2_conv2 (Conv2D)	(None, 200, 200, 128)	147584
block2_pool (MaxPooling2D)	(None, 100, 100, 128)	0
block3_conv1 (Conv2D)	(None, 100, 100, 256)	295168
block3_conv2 (Conv2D)	(None, 100, 100, 256)	590080
block3_conv3 (Conv2D)	(None, 100, 100, 256)	590080
block3_pool (MaxPooling2D)	(None, 50, 50, 256)	0
block4_conv1 (Conv2D)	(None, 50, 50, 512)	1180160
block4_conv2 (Conv2D)	(None, 50, 50, 512)	2359808
block4_conv3 (Conv2D)	(None, 50, 50, 512)	2359808
block4_pool (MaxPooling2D)	(None, 25, 25, 512)	0
block5_conv1 (Conv2D)	(None, 25, 25, 512)	2359808
block5_conv2 (Conv2D)	(None, 25, 25, 512)	2359808
block5_conv3 (Conv2D)	(None, 25, 25, 512)	2359808
block5_pool (MaxPooling2D)	(None, 12, 12, 512)	0
flatten_4 (Flatten)	(None, 73728)	0
dense_6 (Dense)	(None, 128)	9437312
dropout_2 (Dropout)	(None, 128)	0
dense_7 (Dense)	(None, 3)	387

Total params: 24,152,387
Trainable params: 9,437,699
Non-trainable params: 14,714,688

Abbildung 8 Schichten der angepassten VGG16 Architektur

4.3 Performance

Das vortrainierte Netz hat eine bessere Performance. Die Bilder von Patrick und Lorenc wurde alle richtig klassifiziert. Deswegen ist f1-score für diese Klassen 100%. Nur ein Bild ist falsch vom Netz klassifiziert worden. Die Accuracy ist fast 100% und es besteht daher das Risiko von „Overfitting“. Deswegen wählen wir das andere Netz (LeNet), da die Wahrscheinlichkeit bei diesem Netz höher ist, bei neuen Daten besser zu generalisieren.

	precision	recall	f1-score	support
0	1.00	1.00	1.00	300
1	1.00	1.00	1.00	100
2	0.99	1.00	1.00	100

	andere	patrick	lorenc
andere	399	0	1
patrick	0	100	0
lorenc	0	0	100

Abbildung 9 Performance VGG-16

5. Anwendung auf Raspberry Pi

In diesem Kapitel widmen wir uns der Nutzung des Modells auf dem Raspberry Pi. Es wird die benötigte Hardware und Software dargelegt und die Einrichtung sowie die Nutzung des Systems erklärt.

5.1 Hardware & Software

Im Folgenden findet sich eine Auflistung der benötigten Hardware und Software um das System auf dem Raspberry einrichten und nutzen zu können.

Hardware	**Software**
Raspberry Pi	Raspbian auf SD-Karte schreiben
SD-Karte	Wlan & ssh einrichten
Draht	Environment und packages installieren
LED´s	Modell auf SD-Karte laden
Widerstände	Skript zum Modell ausführen erstellen
Steckbrett	Kamera aktivieren
Kamera	Tensorflow installieren

5.2 Vorbereitung

Nachfolgend sind die essenziellen Schritte aufgelistet, um das Gesichtserkennungssystem einzurichten. Eine ausführliche Beschreibung und der benötigte Code findet sich im Jupyter Notebook unter 5.2 Set up Raspberry Pi wieder.

1. Raspbian runterladen und auf SD-Karte schreiben. In den Einstellungen die Zugangsdaten für das Heimnetzwerk eingeben. Die SD-Karte in das Pi stecken und Installation abwarten.
2. IP-Adresse des Pi in den Netzwerkeinstellungen des Routers herausfinden. Mit dem Pi über ssh verbinden. Anschließend Benutzer Name und Passwort festlegen.
3. Ein weiteres Netzwerk (z.B. das Hochschulnetzwerk eduram) einrichten.
4. Die Kamera verbinden und einrichten.
5. Ein Projekt Ordner (z.B. mit dem Namen CV) und das virtuelle Environment erstellen. Danach die benötigte Software in diesem Environment installieren.
6. Tensorflow über die wheel version installieren.
7. Skript zum Aktivieren der Kamera, ausführen des Modells und ansteuern der LED´s erstellen.
8. Das erstellte Modell in den Projekt Ordner laden.
9. Erstelle den Schaltkreis.
10. Führe das Skript aus.

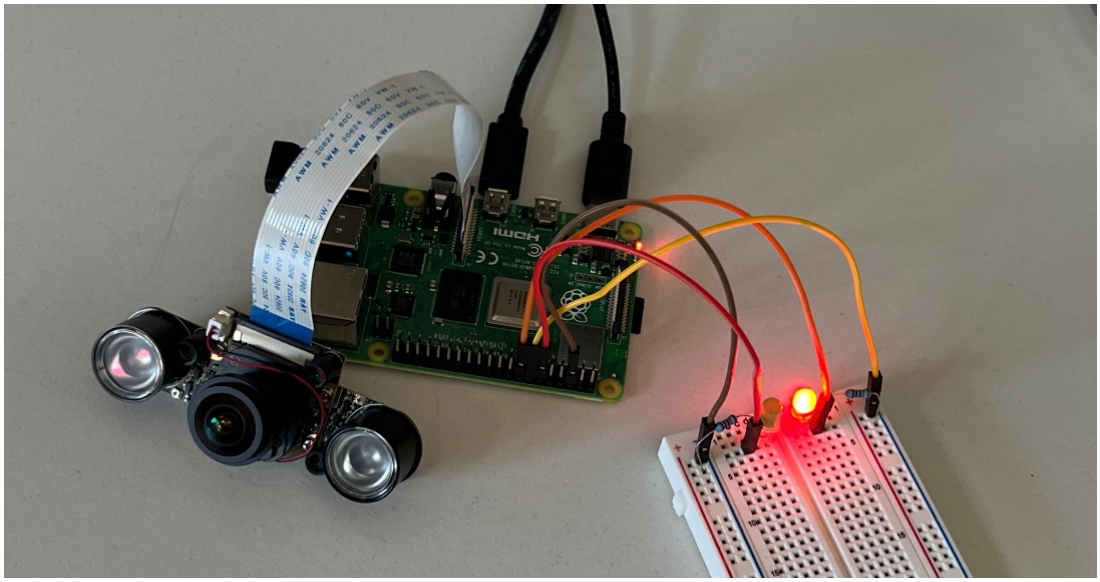


Abbildung 10 Raspberry Pi System

5.3 Test

Mehrere Versuche mit der Testperson Patrick haben durch das Leuchten der grünen LED ergeben, dass diese zuverlässig erkannt wird. Weitere Versuche mit 2 fremden Personen und lediglich dem Hintergrund ohne Personen zeigten durch das Leuchten der roten LED, dass diese von dem System korrekterweise nicht erkannt wurden. Es gilt weiterhin zu prüfen wie das System auf ähnlich aussehende Menschen sowie auf Bilder der zu identifizierenden Personen Patrick und Lorenc reagiert.

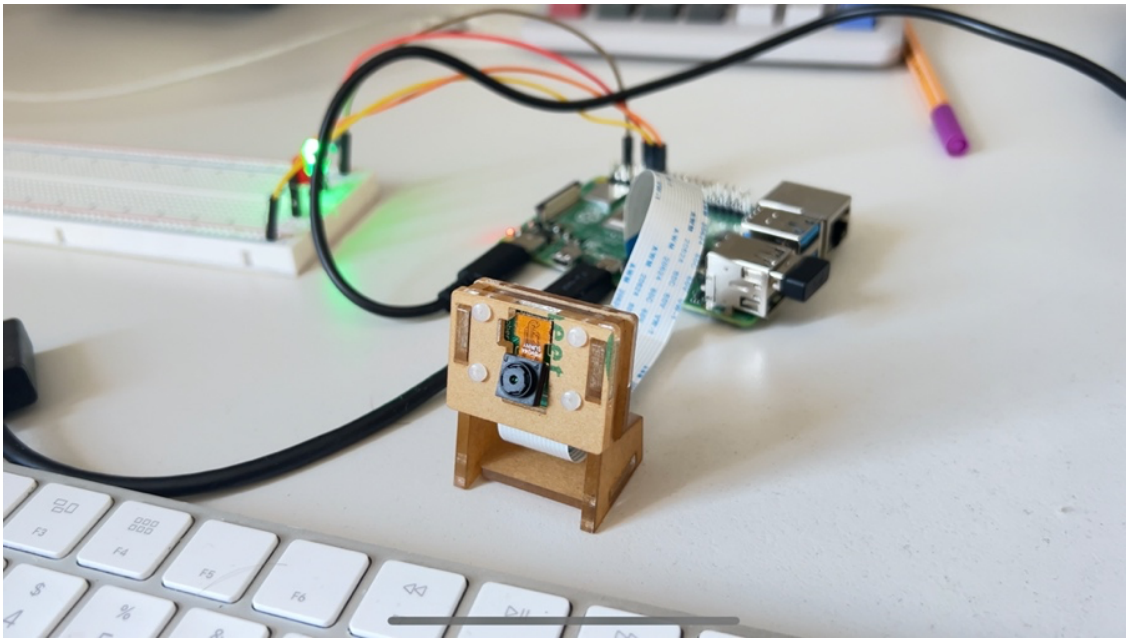


Abbildung 11 Funktionierendes System

6. Zusammenfassung und Reflektion

6.1 Zusammenfassung

Zusammenfassend lässt sich sagen, dass sowohl das selbst trainierte Modell nach der LeNet Architektur gleichermaßen wie das vortrainierte Modell nach der VGG16 Architektur funktionieren. Dabei hat sich deutlich gezeigt, dass das VG16 Modell robuster und genauer ist, jedoch mit großer Wahrscheinlichkeit „Overfitted“.

Die Nutzung des Modells der LeNet Architektur auf dem Pi war erfolgreich. Die zu identifizierenden Personen wurden erfolgreich erkannt und die Haustür würde sich öffnen.

Die Sicherheit des Systems ist nach dem aktuellen Stand schwer zu beurteilen und es müssten weitere Test folgen. Theoretisch sind die Ergebnisse des Modells der LeNet Architektur hervorragend und kein Bild der Klasse andere wurde als „lorenc“ oder „patrick“ klassifiziert, somit wäre die Sicherheit gegeben. Aktuell lässt sich sagen, dass die grüne LED-Lampe nur bei Patrick und Lorenc grün leuchtet. Hinzufügend muss man dazu sagen, dass keine ausgiebigen Tests mit realen Personen stattgefunden haben. Des weiteren haben erste, nicht umfangreiche, Tests gezeigt, dass das System fremde Personen als „andere“ klassifiziert und diese mit dem roten leuchten der LED-Lampe nicht in das Haus lassen würde.

6.2 Reflektion

Die Kombination von Hardware und Software war erfolgreich. Es ist möglich ein Gesichtserkennungssystem mit einem Pi zu konstruieren und zu benutzen. Ausschlaggebend für die Nutzung ist eine geeignete Kamera. Es hat sich gezeigt, dass die Gesichts Erkennung mit einer Weitwinkel Kamera nicht robust funktioniert. Seitdem benutzen der „normalen“ Kamera ist das System voll funktionsfähig einsetzbar.

Besonders wichtig ist mit besonderer Sorgfalt einen sehr guten Trainingsdatensatz zu generieren. Vor allem in Hinblick auf die Varietät der Bilder der zu identifizierenden Personen. Ebenso sollten die Varietät der zu identifizierenden Personen nicht zu extrem zu den anderen Trainingsbildern abweichen, damit das Netz nicht nur die Unterschiede in den Daten lernt, sondern wirklich die Gesichter unterscheidet.

Des weiteren ist es Ratsam zum Trainieren der Modelle auf externe Rechenressource zu nutzen.

Die Forschungsfrage *„Inwiefern ist es möglich ein sicheres System zu konstruieren, mit welchem sich die Haustür mit Gesichtserkennung über ein CNN öffnen lässt? Wie hoch ist die falsch-positiv Rate?“* lässt sich schließlich nicht vollständig beantworten.

Es ist möglich die Haustür mit dem System unter der Benutzung eines CNN zu öffnen. Um in Bezug zu der Sicherheit eine stichhaltige Aussage treffen zu können müsste man tiefgründigere Tests mit realen Personen machen. Aktuell macht es den Anschein, dass die grüne LED, also die Türöffnung bei fremden Personen nicht funktioniert. Auch die Auswertung auf dem Testdaten lässt das System sicher erscheinen. Die falsch-positiv Rate ist auf den Testdaten null.

6.3 Ausblick

Um das System in der Haustür zu einzubauen, werden als nächstes weiter Tests mit realen Personen folgen. Es ist ebenso nötig zu überprüfen, wie das System auf ausgedruckte Bilder der zu identifizierenden Personen reagiert.

Anschließend kann das System am Verteilerkasten an die elektrische Türöffnung angeschlossen werden. Hierzu wird eine optisch angemessene Kamera mit Gehäuse benötigt, welche am Eingang eingebaut wird.

7. Anhang

- Juypter Notebook als Datei
- Jupyter Notebook als PDF
- Power Point Präsentation mit Video des funktionierenden Systems
- Video funktionierendes System

8. Literaturverzeichnis

https://www.analyticsvidhya.com/blog/2021/03/the-architecture-of-lenet-5/

https://towardsdatascience.com/mnist-handwritten-digits-classification-using-a-convolutional-neural-network-cnn-af5fafbc35e9

https://medium.com/mlearning-ai/an-overview-of-vgg16-and-nin-models-96e4bf398484

Die Videos und Skripte der Vorlesung Computer Vision SS 22 von Prof. Dr. Elke Hergenröther

Datensatz: https://www.kaggle.com/datasets/ashwingupta3012/human-faces

Computer Vision Praktikum 3

Building a CNN Face Recognition System

Lorenz Chase & Patrick Gabriel

```
In [1]: from IPython.display import Image
```

```
In [2]: Image(filename='/home/patrick/videos/Screenshots/Screenshot from 2025-05-23 15-17-11.jpg', height=500, width=700)
```

Table of Contents

- Introduction and Dataset Generation
  - 1.1 Load Packages
  - 1.2 Generate Pictures
- Dataset Preprocessing
  - 2.1 Random Seedsplitting for each label
  - 2.2 Build the datasets with cropped images
- CNN Model Definition
  - 3.1 Model Architecture
  - 3.2 Loss and Optimizer
  - 3.3 Learning Rate Scheduler
  - 3.4 Early Stopping
  - 3.5 Model Summary
- Training
  - 4.1 Hyperparameters
  - 4.2 Run Training
  - 4.3 Training Curves / Diagnostics
  - 4.4 Results
  - 4.5 Save Model
- Test Set

1 Introduction

1.1 Load Packages

```
In [3]: !pip install numpy==1.26.4
        !pip install tqdm
        !pip install cv2
        !pip install matplotlib
        !pip install tensorflow
        !pip install keras
        !pip install sklearn
        !pip install seaborn
        !pip install pandas
        !pip install pillow
```

```
In [4]: import os
```

1.2 Generate Pictures

```
In [5]: !mkdir -p ~/videos/Screenshots and cd ~/videos/Screenshots
        !ffmpeg -i ~/videos/Screenshots/Screenshot\ from\ 2025-05-23\ 15-17-11.mp4 -vf "crop=1280:720:0:0,fps=1" -qscale:v 2 -start_number 1 frame_%04d.jpg
```

```
In [6]: !ls ~/videos/Screenshots | wc -l
```

```
In [7]: !ls ~/videos/Screenshots | head -5
```

2 Dataprepage Processing

```
In [8]: !pip install -U "numpy<2" --force-reinstall
        !pip install -U "tensorflow==2.15.0" "keras==2.15.0" --force-reinstall
        !pip install -U "tensorflow-datasets==4.9.4" "tensorflow-metadata==1.14.0" --force-reinstall
        !pip install -U "scikit-learn==1.4.2" "scipy==1.11.4" --force-reinstall
        !pip install -U "matplotlib==3.8.4" --force-reinstall
```

```
In [9]: !pip install -U "numpy<2" --force-reinstall
```

2.1 Random Seedsplitting for each label

```
In [10]: import random
         import os

         for dir in os.listdir():
             if os.path.isdir(dir):
                 directory = os.listdir(dir)
                 print("Dir: ", dir, "count: ", len(directory))
                 for file in directory:
                     if file.endswith(".jpg"):
                         print("  File: ", file)
```

```
In [11]: !ls ~/videos/Screenshots | head -5
         !ls ~/videos/Screenshots | wc -l
         !ls ~/videos/Screenshots | head -5 | xargs -I{} file ~/videos/Screenshots/{}
         !ls ~/videos/Screenshots | head -5 | xargs -I{} ffmpeg -i ~/videos/Screenshots/{} -vf "scale=320:180" -f null -
```

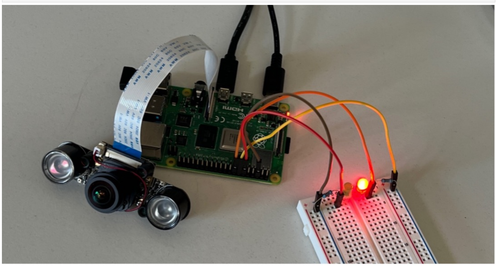
In [3]:
```
## TEST MODEL USING LIVE CAMERA

import cv2
import numpy as np
from tensorflow import keras
import tensorflow as tf

model = keras.models.load_model('new_pretrained.h5')

full_width = 400
full_height = 400

# Connects to your computer's default camera
cap = cv2.VideoCapture(0)

label_dic = {0 : 'andere' , 1 : 'patrick', 2 : 'lorenc'}

while True:

    # Capture frame-by-frame
    ret, frame = cap.read()
    resized_img = cv2.resize(frame, (full_width, full_height))
    resized_img = np.expand_dims(resized_img, axis=-1)
    resized_img = np.expand_dims(resized_img, axis=0)
    resized_img = resized_img / 255
    predictions = model(resized_img).numpy()
    probs = tf.nn.softmax(predictions).numpy()
    cv2.putText(frame, f"Prediction : {label_dic[np.argmax(probs[0])]}", (50,50), cv2.FONT_HERSHEY_SIMPLEX, 1, (255,0,0), 2,

    # Display the resulting frame
    cv2.imshow('frame',frame)#live cam oder frame

    # This command let's us quit with the "q" button on a keyboard.
    # Simply pressing X on the window won't workq!
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# When everything done, release the capture and destroy the windows
cap.release()
cv2.destroyAllWindows()
```

5 Transfer to Raspberry Pi

5.1 Requirements

In [4]:
```
Image(filename='/Users/patrickeidemuller/Documents/Uni/MasterDataScience/2_CV/projekt/bilder präsi/facerecsystem.jpg')
```

Out[4]:



Hardware

- Raspberry Pi
- SD-Card
- Jumpwire
- LED´s
- Electrical resistance
- breadboard
- Camera

Software

- flash Raspbian to SD-Card
- set up wlan & ssh
- set up environment and install packages
- load model via ssh to Pi
- create skript to run model and light the LED´s
- activate Camera

5.2 Set-up the Raspberry Pi

1. Download Raspbian and flash on SD-Card (activate youur local network before you start the flash process). Insert SD-Card in Pi and wait.
2. Find out IP-Adress of Pi in your Networksettings to connect via ssh (ssh pi@ip-adress)to your pi. Select username and password for futher log in.
3. Activate Camera type in the command line: sudo raspi-config and a settings window will open. Follow the following path to activate the camera: > interfacing options > enable camera > reboot. Test with vcgencmd get_camera. If supported & detected = 1 everything is okay.
4. Connect more than one network in our case eduram. Open interfaces with typing the following path into the terminal: sudo nano /etc/network/interfaces. Make sure that the following 2 lines are in the file:

- iface wlan0 inet manual
- wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf Complement the following 2 lines:
- iface netzwerk_a inet dhcp
- iface netzwerk_b inet dhcp

Now open the supplicant.conf file to add the network. sudo nano /etc/wpa_supplicant/wpa_supplicant.conf and add the following information for the network. Add the id to the existing network as well

network={ ssid="eduroam" proto=RSN key_mgmt=WPA-EAP eap=PEAP identity=„stpaeide" password=„xxx" phase1="peaplabel=0" phase2="auth=MSCHAPV2" id_str="netzwerk_b" }

1. Set up Virtual Environment and install packages. Create project folder CV with mkdir CV. For setting up the virtual environment install the packages virtualenv with sudo pip3 install virtualenv. Then create env with virtualenv -p python3 .venv. Now activate the env with source env/bin/activate and install the following packages:

- pip3 install jupyter
- Pip3 install matplotlib
- pip3 install pandas
- pip3 install numpy
- pip install opencv-python
- Pip install pathlib
- Pip install mtcnn
- pip install scikit-learn
- Pip install imagesize
- pip install RPi.GPIO :ssh GPIO controlling

1. installing tensorflow on raspberry is kind of difficult. Look up the right version with the following link : https://github.com/PINTO0309/Tensorflow-bin/tree/main/previous_versions. With wget and the raw link you can download the tensorflow wheel: wget https://raw.githubusercontent.com/PINTO0309/Tensorflow-bin/main/previous_versions/download_tensorflow-2.8.0-cp39-none-linux_aarch64_numpy1221.sh. With sudo chmod +x download_tensorflow-2.8.0-cp39-none-linux_aarch64_numpy1221.sh you can install tensorflow on the raspberry.
2. Create a script with the code from 5.3 to activate the camera, run the model and light the led´s
3. Transfer the model to the project folder with:

- scp path/on/computer/model.h5 pi@ip:/home/pi/projectfolder

1. Build the circuit. Connect the jumpwire to a pin with a number for example number 18, plug in the led in a column with a letter but one end in the same row as the jumpwire. Use a restiance to connect the green led on the other side with the next jumpwire wich is connected to the raspberry with a pin with the label gnd and plugged in in the breadboard on the minus column. To the same for the number 23 with a red led.
2. run the script

5.3 Run Model on Raspberry

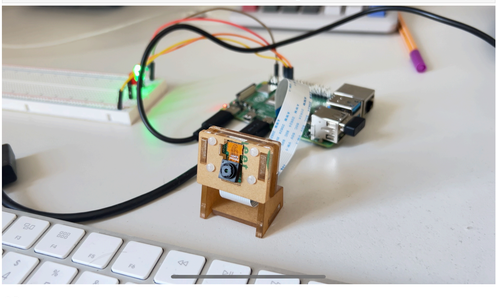
In [ ]:
```
## TEST MODEL USING LIVE CAMERA on RASPBERRY PI

import cv2
import numpy as np
from tensorflow import keras
import tensorflow as tf
import RPi.GPIO as GPIO
import time

model = keras.models.load_model('model_selfmade.h5')

full_width = 400
full_height = 400

# Connects to your computer's default camera
cap = cv2.VideoCapture(0)

label_dic = {0 : 'andere' , 1 : 'patrick', 2 : 'lorenc'}

while True:

    # Capture frame-by-frame
    ret, frame = cap.read()
    new_img = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    resized_img = cv2.resize(new_img, (full_width, full_height))
    resized_img = np.expand_dims(resized_img, axis=-1)
    resized_img = np.expand_dims(resized_img, axis=0)
    print(resized_img.shape)
    resized_img = resized_img / 255
    predictions = model(resized_img).numpy()
    probs = tf.nn.softmax(predictions).numpy()
    cv2.putText(frame, f"Prediction : {label_dic[np.argmax(probs[0])]}", (50,50), cv2.FONT_HERSHEY_SIMPLEX, 1, (255,0,0), 2,

    if label_dic[np.argmax(probs[0])] == "andere":
        GPIO.setmode(GPIO.BCM)
        GPIO.setwarnings(False)
        GPIO.setup(23,GPIO.OUT)
        print "LED on"
        GPIO.output(23,GPIO.HIGH)
        time.sleep(3)
        print "LED off"
        GPIO.output(23,GPIO.LOW)

    if label_dic[np.argmax(probs[0])] == "patrick":
            GPIO.setmode(GPIO.BCM)
            GPIO.setwarnings(False)
            GPIO.setup(18,GPIO.OUT)
            print "LED on"
            GPIO.output(18,GPIO.HIGH)
            time.sleep(3)
            print "LED off"
            GPIO.output(18,GPIO.LOW)

    if label_dic[np.argmax(probs[0])] == "lorenc":
            GPIO.setmode(GPIO.BCM)
            GPIO.setwarnings(False)
            GPIO.setup(18,GPIO.OUT)
            print "LED on"
            GPIO.output(18,GPIO.HIGH)
            time.sleep(3)
            print "LED off"
            GPIO.output(18,GPIO.LOW)

    else:
        continue

    # Display the resulting frame
    cv2.imshow('frame',frame)#live cam oder frame

    # This command let's us quit with the "q" button on a keyboard.
    # Simply pressing X on the window won't workq!
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# When everything done, release the capture and destroy the windows
cap.release()
cv2.destroyAllWindows()
```

5.4 Result

As you can see in the picture below is that the system works quiete well with the new camera without wide angel.

In [3]:
```
Image(filename='/Users/patrickeidemuller/Documents/Uni/MasterDataScience/2_CV/projekt/bilder präsi/funktionierendes system.jp
```

Out[3]:



6 Resume

6.1 Fazit

- most impact have the quality of the pictures for training the model in terms of lighting, angel and background variety
- pre-trained models are more efficient and easier to use
- the score and false-positiv rate is surprisingly good
- there need to be more specific test with faces similar to the person who should be identified to check the security of the system. Also with pictures of them to verify if the system can be tricked out with pictures.

In [ ]: