DEPARTMENT OF INFORMATICS

CHAIR OF ROBOTICS, ARTIFICIAL INTELLIGENCE AND REAL-TIME SYSTEMS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Information Systems

Enabling Real Time Multi Face Emotion Recognition for Embedded Devices

Nikan Moghadam

DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Information Systems

Enabling Real Time Multi Face Emotion Recognition for Embedded Devices

Effiziente Implementierung eines Echtzeit-Emotionserkennungssystems für Fahrzeuge

Author:	Nikan Moghadam
Supervisor:	Prof. Dr.-Ing. Alois Knoll
Advisor:	M.Sc. Sina Shafaei
Submission Date:	15th August 2019

I confirm that this bachelor's thesis in information systems is my own work and I have documented all sources and material used.

Munich, 15th August 2019 Nikan Moghadam

Acknowledgments

First of all, I would like to thank my parents for their continuous support that enabled me to pursue this degree. Without their help I would not have had many of the opportunities that got me where I am today.
In addition to that I would like to express my deepest sense of gratitude to my advisor Sina Shafaei. His continuous advice and consultations allowed me to avoid many common mistakes and significantly improve the quality of my work.
I would also like to thank the Microsoft Corporation for providing me with valuable cloud resources to train my models and the Chair of Robotics, Artificial Intelligence and Real-time Systems for making my research possible.

Abstract

The rise of autonomous vehicles in the transportation industry creates new challenges and opportunities for driver-vehicle interaction. As cars gain more and more autonomy new sensors and methods are needed to ensure the driver's wellbeing and a clear communication between the driver and the digital systems in the car. One important aspect of communication is emotion and more specifically emotion recognition.
As part of this thesis we implement a system that can detect the faces of multiple passengers on an embedded device with very high accuracy. To achieve this, we combine a mobile convolutional neural network that was trained on a large amount of faces (Wider Face data set) with a motion tracker (KCF).
To choose the best approach extensive testing and benchmarking was performed on a Raspberry Pi 3 B+, assessing accuracy and performance of various approaches. As for the face detection multiple neural networks as well as the dlib HoG approach and the OpenCV implementation of the Viola Jones Algorithm were benchmarked. The quality of the face detector as well as the object tracker were then evaluated using a benchmark video which most accurately represents our use case (camera mounted inside a vehicle). The benchmark results were then used to finetune the parameters of our final implementation.
The tracking system achieves an average frame rate of 10 FPS with the KCF algorithm and 25 FPS with the MOSSE algorithm on the Raspberry Pi 3B+.

Contents

1. Introduction

As the speed and - more importantly - the efficiency of CPUs is rapidly increasing, new opportunities for research in embedded devices are constantly arising. One recent megatrend that is enabled by cheap, small and energy efficient computing devices is edge computing. Embedded devices have the potential to bring the power of digitalization to many places that were previously too expensive or inaccessible to digitalize. This is especially significant for the area of computer vision where a combination of powerful embedded devices and new algorithms and techniques in the field of A.I. enable a variety of new use cases.
Traditional industries like manufacturing and automotive are the biggest beneficiaries of this trend. Today's cars are packed with sensors and assistants that increase passenger safety and comfort. Vehicles are expected to become increasingly autonomous which will further increase the significance of embedded devices in cars.
One use case for embedded devices in vehicles is to improve driver-vehicle interaction and communication. This reaches from better infotainment and entertainment features to safety features like drowsiness detection. The scope of this thesis will be to develop a system that enables real-time emotion recognition for up to 4 passengers inside a vehicle. There are multiple ways to achieve this goal and some can even be combined to form multimodal emotion recognition systems. We will focus on emotion recognition using a camera mounted inside a car.
The field of emotion recognition using computer vision provides many algorithms with varying accuracy and performance for this problem. Most of the research however is focused on subcomponents of the pipeline like face detection or emotion classification. The existing research is also mostly focused on accuracy and not optimized to run in real-time on an embedded device. So far only one paper on emotion recognition for an older Raspberry PI 2 was published.[1]
The primary challenge of our work is to scale algorithms with a high accuracy to the Raspberry PI. A high detection quality is necessary as difficult lighting conditions and face occlusion are common in cars. Additionally, our system needs to run in real time whilst meeting the resource constraints of an embedded device.
Our goal is to combine the existing research and optimize it for our use case to achieve a better detection quality and performance (frames per second).
We solved the problem by first benchmarking three major techniques that enable face detection to determine whether they are fast enough to run on an embedded device. As part of the benchmarking all parameters were fine-tuned so that the algorithms deliver comparable results. For instance, one requirement was to enable face detection up to 1.8m and to run on

[1] More information in the Related Work section

the Raspberry PI with more than one frame per second.
After a first qualitative comparison we investigated the specifics of our use case and whether there was any way to improve the system. As we expect the number of passengers in a car and the position of their faces to remain relatively constant over time, we decided to combine our face detector with an object tracker.
After studying the existing literature two object trackers were combined with our face detector and then additional benchmarking was performed. The factors assessed as part of the benchmark were the overall speed and performance and the fit for our use case.
Finally, some smaller improvements were implemented in the final system to further improve its capabilities like face loss detection and regular validity scans.
The final proposed system detects faces with a small and efficient convolutional neural network optimized for embedded devices based on the SSDLite MobileNetV2 [1]. The network was trained for our use case with the Wider Face [2] data set which contains more than 393.000 faces. As part of the data transformation pipeline small faces were removed from the data set so that the final training set contains about 30.000 faces. Transferred learning was then used to train the existing object detector on faces.
The convolutional neural network was then combined with the KCF object tracker to significantly increase the performance of the system (>10x speedup) whilst maintaining almost the same level of accuracy.
Our proposed system runs at more than 10 FPS on a Raspberry PI and can detect 91% of the faces compared to the OpenCV CNN implementation in our benchmarking video.
The structure of the thesis is as follows:
First an overview of the different algorithms enabling face detection and object tracking is provided in the related work section.
Then in the comparison section the different face detection approaches and the different object trackers are compared. This section contains both qualitative and quantitative assessments.
Afterwards our proposed emotion recognition system is explained in detail, including the training process for our model.
The evaluation section describes in detail the methods we used for the quantitative benchmarks and the qualitative assessments. Additionally, the final proposed system is evaluated and the characteristics as well as advantages and disadvantages are discussed.
Lastly future research areas are highlighted and the content of this thesis is concluded.

2. Related Work

Most of the research in the area of emotion recognition on embedded devices is focused on the subsystems face detection, emotion classification and object tracking. Only in one paper a combination of face detection and emotion classification was implemented in an optimized way for embedded devices.
In the paper "Real-Time Emotion Recognition from Facial Images using Raspberry Pi II" [3] a real-time emotion detector was implemented on the Raspberry Pi II using the Viola Jones Algorithm and an Adaboost classifier. The proposed solution can recognize up to 5 different emotions in real-time (120 ms delay) for one person. The solution proposed in this paper extends the functionality of that system by implementing a convolutional neural network and classifying up to 7 emotions. Additionally, the approach is extended so that multiple faces can be detected. At the same time the framerate is as high or even higher (depending on external parameters).
In the following section the face detection algorithms and techniques used in our benchmarks and in the final proposed system are summarized.
The face detector used in the proposed system is based on the MobileNetV2 SSDLite object detector proposed in "MobileNetV2: Inverted Residuals and Linear Bottlenecks "[1]. The network can detect a variety of objects in different environments, lighting conditions and under occlusion. It was developed by a group of researchers at Google Inc. in 2018. The MobileNetV2 convolutional neural network is optimized for mobile devices and provides a very high accuracy with limited resources. This neural network architecture provides the foundation for the model we trained for our system.
We trained the MobileNetV2 model on the Wider Face data set introduced in "WIDER FACE : A Face Detection Benchmark"[2] by a group of researcher at the university of Hong Kong in 2015. The data set contains 393.703 faces in different shapes and environments and is one of the largest face data sets publicly available.
For benchmarking the OpenCV implementation of the Viola Jones Algorithm proposed in "Rapid Object Detection Using a Boosted Cascade of Simple Features"[4] by Paul Viola and Michael Jones in 2004 was also included. They introduce three new techniques for their detection algorithm which provide significant improvements in speed and detection quality compared to traditional approaches. First integral images are used to speed up the feature extraction. A technique called AdaBoost is then used to focus only on the relevant features of an object for model training. Finally, cascading classifiers are used to eliminate many unnecessary checks by grouping them and introducing detection stages. If a feature fails at the first stage all subsequent stages are no longer evaluated.
The dlib face detection implementation used in our benchmark is enabled by an approach introduced in "Histograms of Oriented Gradients for Human Detection"[5] by Navneet Dalal

and Bill Triggs published in 2005. The algorithm starts by converting the image into a Histograms of Oriented Gradients that highlights differences and directions of contrast. Afterwards a detection window with a linear SVM slides over the image to detect faces.
In the following section the object tracking algorithms used in our benchmarks and in the final proposed system are summarized.
To increase the overall performance of our implementation the detected face bounding boxes are passed to an object tracker which is based on kernelized correlation filters introduced in "High-Speed Tracking with Kernelized Correlation Filters"[6]. The KCF object tracker eliminates the visual lag imposed by the neural network and it provides the best balance between accuracy and performance for our use case (the object tracker runs in n*log(n)).
The MOSSE object tracker based on the paper "Visual Object Tracking using Adaptive Correlation Filters"[7] published by a group of researchers at the Colorado State University was also included in our benchmark. It tracks objects with correlation filters and is currently the fastest object tracker within the OpenCV library.
Lastly the detected face is processed by a facial landmark detector and an emotion classifier. The emotion classifier in our system was trained on "The Extended Cohn-Kanade Dataset (CK+)"[8] published by Cohn Kanade in 2010 which consists of the faces of 123 subjects and is labeled with their emotions.

3. Comparison of Different Approaches

3.1. Comparison of Face Detectors

The most resource intensive component of the proposed emotion recognition system is the face detection algorithm that constantly scans the input video stream for faces. Multiple approaches were implemented and tested for our use case to determine the optimal algorithm for our system. A qualitative assessment of each algorithm is given in the following section.

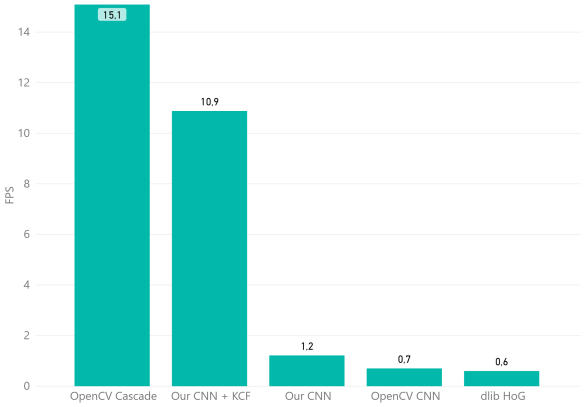

Figure 3.1.: Frames per second by approach

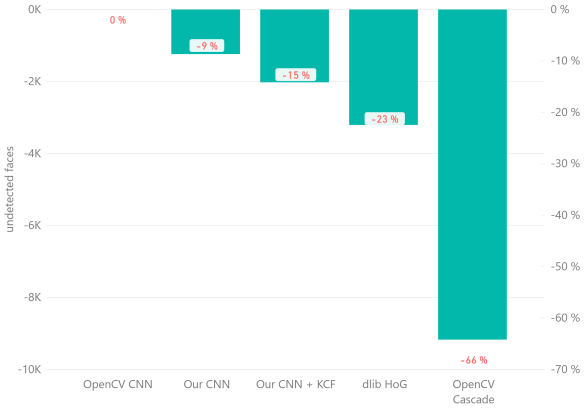

Figure 3.2.: Undetected faces by approach (more details in Evaluation)

3.1.1. Viola Jones Algorithm

The first implementation tested was the cascading classifier in OpenCV which is based on the Viola Jones Algorithm. Faces are detected in a multistep process that starts by gray scaling the image as color information is not relevant for the algorithm. After that, a Haar Cascade moves over the frame and searches for relevant features. In the last step cascading classifiers eliminate all irrelevant features so that only detected faces are left.
The algorithm has a very high performance as it works with integral images and most operations are computationally inexpensive for a CPU. As a result, the algorithm can run at about 15 frames per second on a Raspberry Pi 3.
However, the high performance also comes with significant costs in detection quality as the approach was only able to detect 34% of the faces in our benchmark video (compared to the pretrained OpenCV neural network) and even slightly tilted faces could not be detected in most cases. Bad lighting or low contrasts as well as occlusion posed an additional challenge.

3.1.2. Histogram of Oriented Gradients

Another implementation that is performant enough to run on an embedded device is the dlib HoG implementation [5]. The input stream is converted to a histogram of oriented gradients by calculating contrast deltas in the image. The conversion can utilize CPU optimizations to run as fast as the OpenCV Haar Cascade implementation however it also comes with a significant drawback. The input stream needs to be of much higher resolution for the algorithm to work with small faces. This is a must have for our use case as passengers in the backseats also need to be detected.
After fine tuning the parameters to find the optimal resolution for detecting faces up to 1.8m the algorithms performance saw a significant decrease in frames per second and was even outperformed by our convolutional neural network (0.6 FPS vs. 1.2 FPS).

3.1.3. Convolutional Neural Network

Neural networks can also be used to detect faces and objects in images and video streams. One network architecture that is often used for visual recognition tasks is a convolutional neural network. Convolutional neural network use convolutional layers that slide over the image and calculate the dot product of the RGB matrix to extract features and other relevant information. The values are then forwarded through a deep neural network before the output is generated.
The model used for our proposed system is based on the SSDLite MobileNetV2 architecture. The training process and model details are described in the next chapter. The neural network approach detects almost all faces even under occlusion and with difficult lighting conditions. The only drawback is the relatively high delay of 0.7s (1.2 FPS) on the Raspberry Pi.

3.1.4. Summary of Face Detection Approaches

Approach	# Sample Size	Duration	Resolution	FPS	Detection Quality	Range
dlib HoG	46	1.63	1000x562	0.6	77%	1.7 m
OpenCV Cascade	281	0.07	370x208	15.1	34%	1.8 m
OpenCV CNN	94	1.42	350x196	0.7	100%	1.9 m
Our CNN	67	0.72	350x196	1.22	91%	1.8 m

Table 3.1.: Face Detection Benchmark Results

Definitions:

- Sample Size: Number of measurements processed. Note that faster approaches will have a larger sample size for the same test run as they are able to process more frames in the same time.

- Duration: Average duration that the algorithm needs to process one frame on the Raspberry PI 3.
- Resolution: Resolution of the resized input stream.
- FPS: Frames per second on the Raspberry PI 3
- Detection Quality: Sum of detected faces in the benchmark video compared to the OpenCV CNN implementation (fixed at 100% for reference)
- Range: Maximum distance between the camera and the person for the face to be detected.

3.2. Comparison of Object Trackers

Two object trackers were implemented and tested for their performance and accuracy to determine their fit for our application. A qualitative assessment of both is given in the following section.

3.2.1. Minimum Output Sum of Squared Error (MOSSE)

For testing we used the OpenCV implementation of the MOSSE algorithm that is based on the paper "Visual object tracking using adaptive correlation filters". [7] The algorithm runs at about 40 frames per second for tracking one face and can track multiple faces with 25-35 FPS. As there is little occlusion and movement in our use case (mounted camera inside a car) the tracker can reliably track passenger faces with very little failures and interruptions. However, the bounding boxes produced by the algorithm are not precise enough for our use case. The algorithm has problems scaling the bounding boxes with the passengers faces as they change their distance towards the camera. Often this results in a bounding box that only covers a small fraction of the face so that the subsequent facial landmark detection would fail.

3.2.2. Kernelized Correlation Filters (KCF)

In addition to the MOSSE algorithm we also benchmarked the KCF algorithm proposed in the paper "High-Speed Tracking with Kernelized Correlation Filters". [6] For testing the OpenCV implementation was used. KCF runs with 15 FPS on the Raspberry PI and provides accurate bounding boxes. However, every additional face reduces the speed by 2-3 FPS so whilst KCF works totally fine for our use case (up to 4 people in a car) it is not recommended for tracking larger groups of people.

4. Proposed Emotion Recognition System

4.1. System Overview

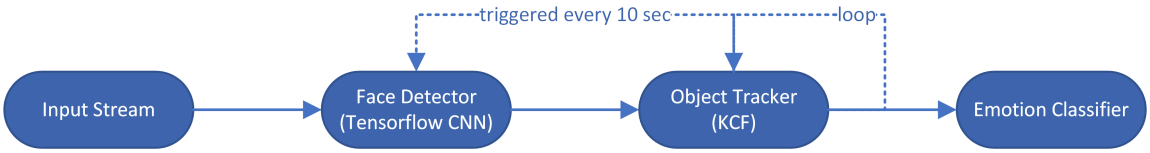

Figure 4.1.: Processing Pipeline

The proposed system as shown in Figure 4.1 consists of 4 major components. An input stream that is generated by the video capturing device, a convolutional neural network used as a face detector for detecting the bounding boxes of all faces in the image, a KCF object tracker to subsequently track all detected faces and reduce computational demand and an emotion classifier that interprets the facial landmarks. After optimizing each step, the implementation was able to run on a Raspberry Pi 3 B+ in real-time at 10-15 frames per second, depending on the amount and distance of the detected faces.

4.2. Input Stream

The input stream is generated by the Raspberry Pi camera (V2.1) and has a resolution of 1920x1080 at 30 frames per second. OpenCV is used to open and downscale the stream to 400x224. Additionally frames are buffered to increase performance.

4.3. Face Detector

We decided to use a convolutional neural network for face detection as the system is required to detect faces in all kinds of different rotations and lightning conditions. This is because our emotion detector is deployed in a car that is expected to have dynamic lighting.
The basis for the convolutional neural network implemented in our final version is the MobileNetV2 SSDLite model as found on the TensorFlow model zoo on GitHub. [9]
The data set used in the training and testing phase is the Wider Face data set. [2]
As part of the data transformation pipeline small faces and images that contain low quality faces were eliminated from the data set completely so that the final training batch contained about 30.000 faces. The training was performed on 4 Tesla K80 GPUs over 31000 iterations

with a final total loss of 2.6.
The network input resolution is 300x300 which we found to be the optimal resolution to track faces up to a distance of 1.8m (assumed maximum distance inside a car). Compared to the OpenCV pretrained model our model offers a 74% increase in frames per second whilst only detecting 9% less faces in our benchmark video.

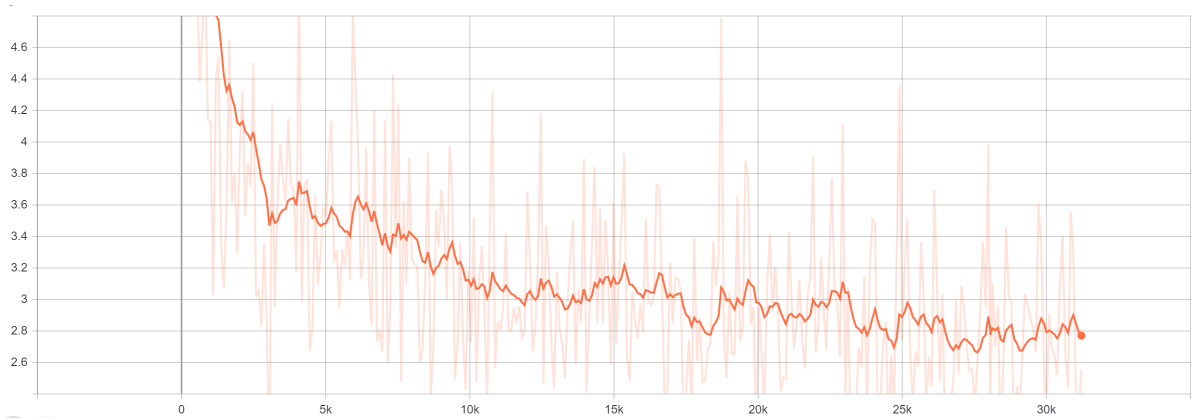

Figure 4.2.: Total loss of the trained model (y) over iterations (x)

4.4. Object Tracker

We used the KCF object tracker in combination with the Multitracker library provided by OpenCV to track the detected faces. The tracker runs with a slightly higher resolution (400x224) as this significantly improves tracking faces of passengers in the backseat whilst only having a minimal impact on performance compared to the 300x300 resolution used by the convolutional neural network.
The object tracker consumes much less resources compared to the face detection algorithm, however even after extensive fine tuning one problem remains: The sizes of the bounding boxes are not dynamically changing with the face. A way to resolve this problem is to find landmarks in the face and scale the bounding box accordingly (for instance the distance between two eyes would be a scaling factor). A first implementation of this approach showed promising results and the bounding boxes scaled correctly but a few problems still remained. In the end we decided against it as the current facial landmark detector has problems with low-light, low-contrast settings which leads to a significantly lower stability of the system in those cases.

4.5. Emotion Classifier

For emotion classification an existing emotion classifier was used as it only came with the minimal overhead of 50 ms per face and improving it was out of the scope of this thesis.
Before classifying emotions, a facial landmark detector detects 68 facial landmarks with the

dlib shape predictor and adds them to a vector. The landmark vector is then passed to a support vector machine that was trained in scikit learn on the Extended Cohn-Kanade Dataset [8]. The SVM output provides the the emotion label for each detected face.

5. Evaluation

5.1. Evaluation Methods

The following section describes the evaluation methods used to generate our benchmark results as well as the advantages and challenges imposed by each method.

5.1.1. Performance Benchmark

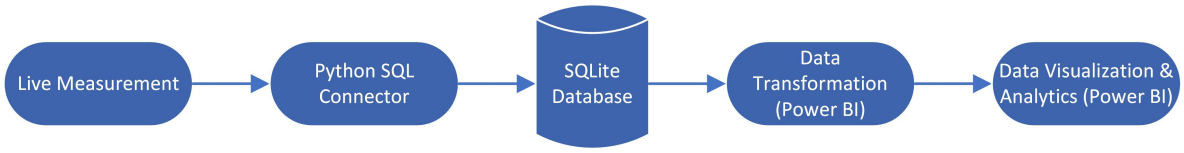

Figure 5.1.: Analytics Framework

The performance of our system was benchmarked on a Raspberry PI Version 3 B+ that was released in March 2018. The PI is powered by a Cortex-A53 quad core processor with 1024 MB of RAM and is equipped with a Raspberry PI camera V2.1.
We decided to conduct the performance benchmark with the live feed of the camera instead of using a prerecorded video to replicate our final use case as accurately as possible. In addition to that the decoding of a compressed video stream would have had a significant and measurable impact on the system's overall performance.
To produce standardized and comparable benchmark results testing guidelines were used. Every test started with one face at a distance of 0.5 m. After 15 seconds another face was added to measure the impact of tracking multiple faces. The lighting conditions remained constant for all tested approaches.
In addition to the testing performed on the PI all approaches were also benchmarked on a Windows 10 machine with an Intel Core i7-8650U and a GTX 1060 with no processes running in the background.
We created a small analytics framework to persist all measurements in a SQLite Database. The following data points were captured for every approach: Time of measurement, processing time per frame, method, platform, measurement id and number of faces detected per frame. The SQLite database was then connected to Power BI to perform further analysis and evaluate the results.

5.1.2. Qualitative Benchmark

The detection quality of the implemented system was evaluated using a mixture of different approaches. To get a first impression of each system's capabilities a subjective test was performed where different factors like lighting and face position (size and tilt) were experimented with. Those experiments were the foundation for a first qualitative assessment.
As the approach mentioned above is not quantifiable and does not produce consistent results another method was also used for benchmarking. A clip from "The Late Late Show with James Corden" [10] capturing four people in a car in Los Angeles was used to determine the number of faces each approach is able to detect in every frame. An example frame of that clip can be seen in figure 5.2 [11]. The performance metric was then created by calculating the sum of all faces detected over all frames:

$$\sum_{Frames} \sum_{Faces} 1$$

In the end both approaches were combined for the final evaluation to achieve maximum flexibility and comparability of the test results.

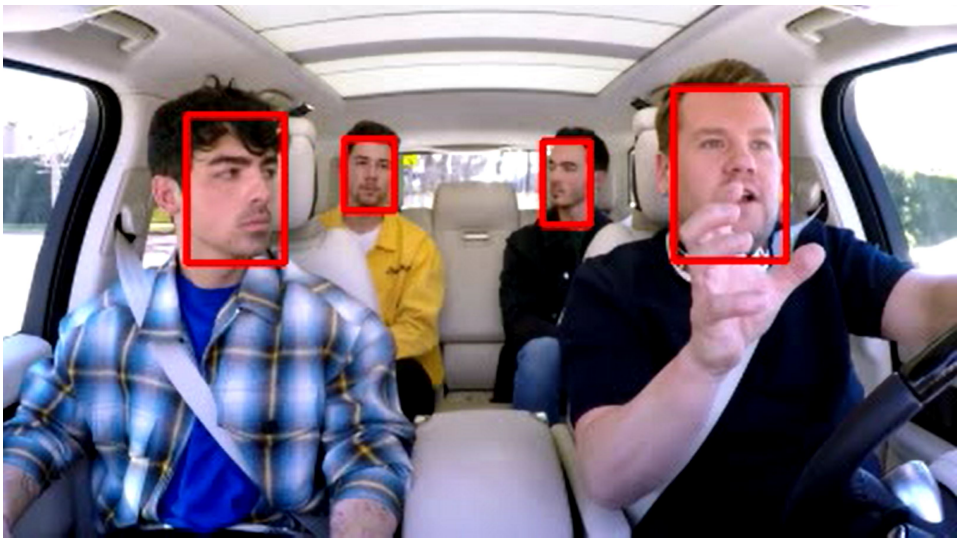

Figure 5.2.: Jonas Brothers Carpool Karaoke with face detection rectangles

5.2. Evaluation Results

In the representative benchmarking video our implementation detects 9% less faces compared to the pretrained OpenCV convolutional neural network. However, it runs at 10.9 FPS on the Raspberry PI compared to OpenCV's 0.7 FPS which makes it 16 times faster. The characteristics of our approach and our benchmark results are presented in more detail in the following section. In addition to that advantages and potential issues are highlighted in the data and discussed.

Platform	Sample Size	Median Runtime	Average Runtime	σ	FPS
Windows 10	3868	0.03	0.03	0.08	32.92
Raspberry PI	1207	0.07	0.09	0.22	10.88

Table 5.1.: Benchmark Results (Definitions in 3.1)

Figure 5.3 shows the number of faces that our proposed emotion detector was able to detect in the benchmarking video over time. Figure 5.4 plots the frames per second of the implemented system running on a Raspberry Pi over time. There are five areas of relevance which stand out from the rest and will be discussed in the following section.
Figure 5.3 A1 highlights a false positive as the system detects 5 people in a car with 4 passengers. False positives are not unusual for networks that were reduced in size and complexity in order to run on an embedded device, however the fact that the false positive remains for a few seconds is special. It is caused by the the object tracker not verifying the validity of a face which leaves the error to remain in the system. We solved this issue by running verification scans every 10 seconds so that new faces added to the frame are detected and errors are removed.
The sequence highlighted in figure 5.3 A2 shows another characteristic of our approach. The number of detected faces can remain relatively constant over time and has very little variance. This is because faces - once detected - are tracked very reliably by the KCF object tracker.
Figure 5.3 A3 shows the failure mechanism of our approach. As soon as the KCF object tracker loses track of a face (obstacles, occlusion, etc.) the face detector is reinitialized. This should not happen too often in our use case as we assume the number of passengers in the car and their visibility to remain relatively constant. However, this feature will reduce the performance of the system significantly in fast changing environments as can be seen in figure 5.4 A1.
Finally figure 5.4 A2 highlights the performance impact of tracking an additional face which is around 2-3 FPS depending on the size of the tracked area.

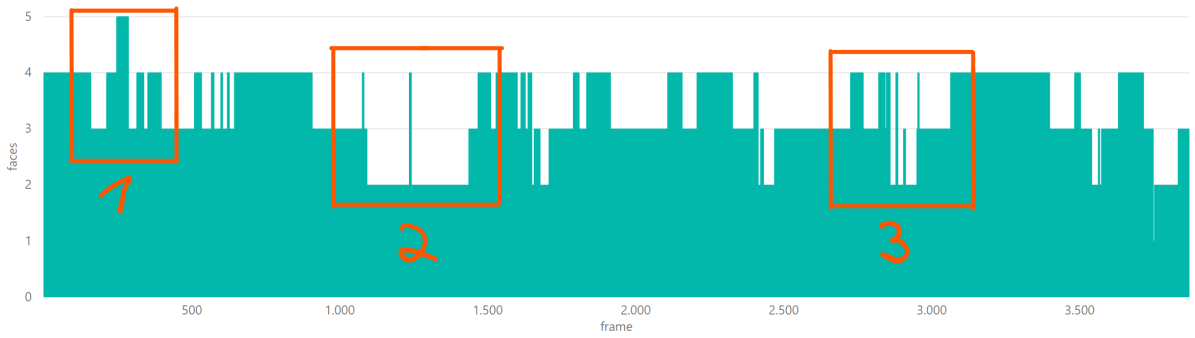

Figure 5.3.: Detected faces by the emotion recognition system in the benchmarking video

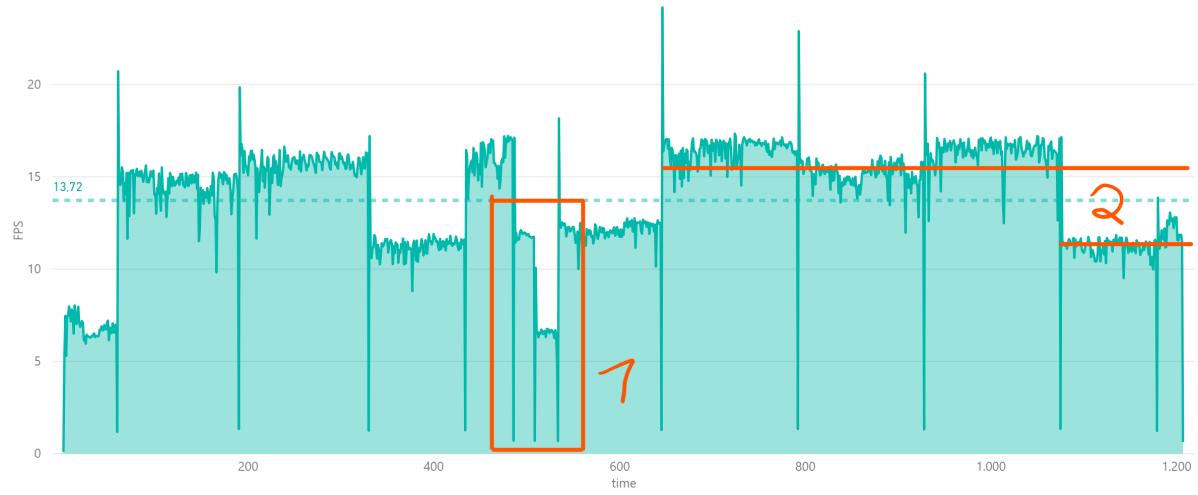

Figure 5.4.: FPS over time on the Raspberry PI

6. Future Work

As part of our research we extensively analyzed many aspects of different implementations that solve our use case. However, there is still room for improvement and there were also some new questions raised. We would like to point out some of the most relevant questions and areas for future research in the following section.
The convolutional neural network used for our implementation is based on the MobileNetV2 neural network originally designed for general object recognition tasks. Therefore it contains many additional layers with little relevance for face detection. These layers pose a significant overhead which we believe can be optimized with a more specific neural network design.
There is also a lot of room for improving the bounding boxes of the object tracker. We currently use the KCF object tracker to track the entire face. This results in a significant overhead compared to the MOSSE tracker and in some cases the facial landmarks at the edge of a face cannot be detected as the bounding boxes do not scale correctly. We believe it would make more sense to track only specific features of the face (for instance both eyes) and then use the distance and tilt between the tracked eyes to scale the bounding boxes accordingly.
The only component of our system that was not optimized is the emotion classifier as the existing implementation was accurate and fast enough for our use case. For future development it would make sense to replace the current implementation based on a support vector machine with a neural network to improve accuracy. A neural network could also be trained to recognize not just emotions but also people to enable even more use cases such as face verification.
Lastly testing our implementation on different devices presents another interesting area of research as each embedded device comes with its own characteristics and computational capabilities (like the recently released Raspberry PI 4).

7. Conclusion

As part of this thesis we looked at different implementations to enable real time multi-face detection for automotive use cases. The concrete scenario is a mounted camera inside a vehicle. We assume that no more than four people can fit in our car and that the people do not move a lot while the car is driving. The implemented system needs to be performant enough to run on a Raspberry PI 3.
Before implementing the final solution, we looked at different approaches and their performance and detection quality and conducted benchmarks as there was no previous research in this field.
Our testing showed that the OpenCV cascade implementation was the fastest (15 FPS), followed by the CNN approach (1.22 FPS) and the dlib HoG implementation at 0.6 FPS.
The most accurate algorithm is the CNN approach with 91% accuracy[1] followed by the dlib implementation at 77% and the OpenCV cascade implementation (34%).
As the convolutional neural network approach delivered the best accuracy and a fairly good performance it is the basis for our implementation. The face detector is used together with a KCF object tracker to boost the performance to more than 10 FPS.
Looking ahead we cannot wait to see how this field of research will develop in the future. New research in artificial intelligence and neural network architectures have the potential to significantly reduce the overhead of our system and improve its speed and performance.

[1] Accuracy is defined as the sum of detected faces compared to the OpenCV CNN approach in the benchmark video

A. General Addenda

A.1. Qualitative Benchmark results

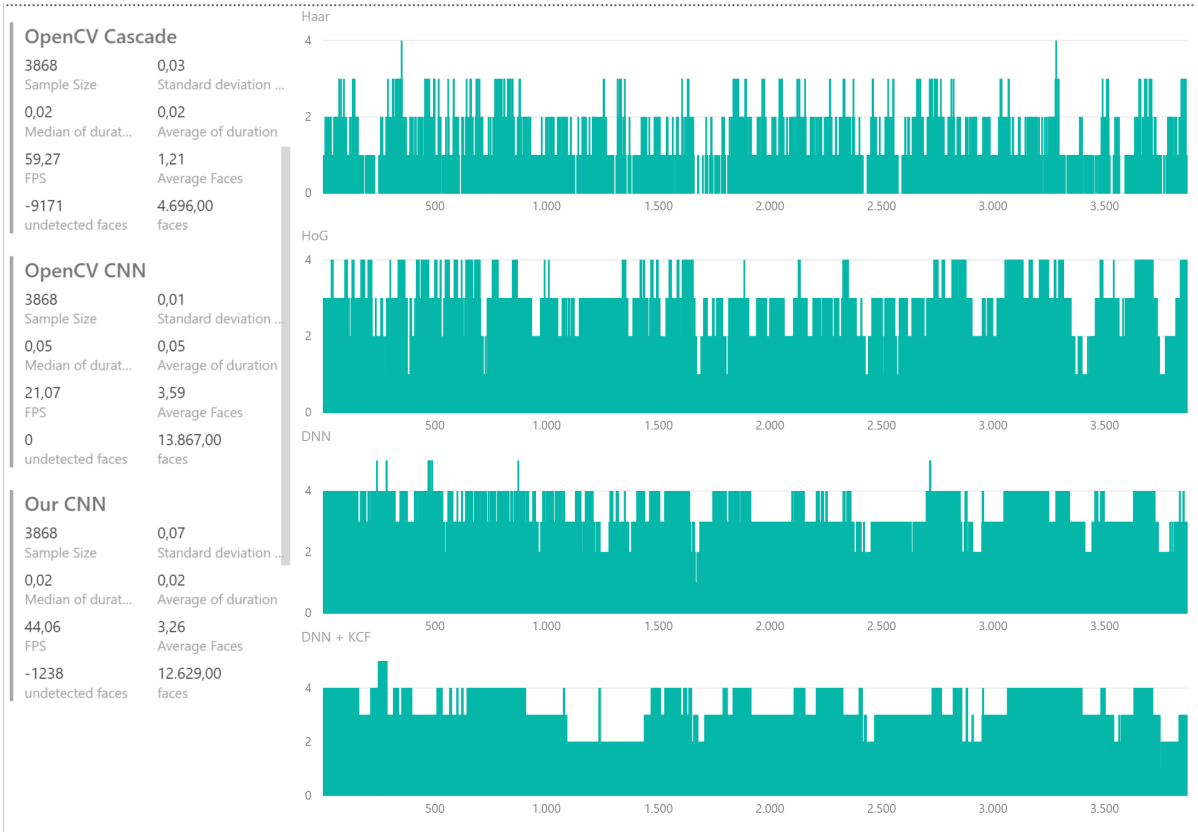

Figure A.1.: Analytics Dashboard for qualitative benchmark results

A.2. Initial Assessment of Approaches (Notes)

A.2.1. Motion Detection Approach

- The system would have a 'start-up phase' where all faces visible at that moment in the car will be detected (using a very accurate face detection method (doesn't have to be in real time, one second delay is acceptable)
- After the 'start-up phase' which will last approximately one second all detected faces

are evaluated and facial landmarks are calculated using HoG (HoG only created for the local face region to save resources)

- During runtime, the system will no longer run face detection on the entire feed, instead existing faces will be tracked using a KLT feature tracker.
- During runtime, a motion detection algorithm will detect areas in the feed that change
- When change is detected the algorithm triggers a convolutional neural network or HoG/SVM that will then try to detect a face in the limited area of change.
- If no change is detected the face detection system won't be triggered
- Since the camera will be mounted in a fixed position a motion detection approach seems to be the best way to implement the system
- Benefits:
  - If implemented correctly the approach should run in real time
  - As long as there is little change in the feed, the system can track faces extremely efficiently (usually in a car there is little change, people don't enter/exit cars every minute)
- Challenges:
  - System can be tricked if someone moves his face extremely slowly into the feed
  - System works for the specified case (mounted camera in a car), however it is not able to deal with an input feed that changes a lot
  - Bridging the gap between startup evaluation and runtime face tracking
  - Complex implementation

A.2.2. Face Detection Approach (CNN)

- The system designed would detect faces using a convolutional neural network that would run on the GPU of the Raspberry PI. Since the PI doesn't have the computational power to fully evaluate the video feed in real time some optimizations need to be performed.
- A solution would be to reduce the resolution of the feed significantly and train a model that is able to detect face like structures (schematic structures, no details).
- After detecting those structures, a local HoG needs to be computed to detect facial landmarks and verify that the object is a face.
- As soon as a face has been detected, the area of the face is no longer processed by the face detection algorithm to save resources. However the system still needs to be able to follow movements etc.. We can fulfill that constraint by using a feature tracker like KLT.

- Benefits:
  - Most accurate implementation
  - CNN runs well on GPU
- Challenges:
  - Probably won't fulfill resource constraints
  - Single components might need to be replaced by faster/less accurate ones
  - Complex implementation and model training

A.2.3. Face Detection Approach (HoG & Linear SVM)

- Detect multiple faces in real time using HoG & a linear SVM
- Will probably not run in real time so the only option would be to combine it with the previous approaches
- Benefits:
  - Fast on CPU
- Challenges:
  - Without additional optimizations the system will not run in real time

A.2.4. Face Detection Approach (Viola Jones Algorithm)

- Detecting multiple faces in real time using the Viola Jones Algorithm
- Haar Cascade -> Integral Image -> Adaboost -> Cascading Classifiers
- Benefits:
  - Easy to implement
  - Performance
- Challenges:
  - Low accuracy, probably won't be fast enough
  - Performs wore in almost every area compared to (HoG + Linear SVM)

List of Figures

List of Tables

Glossary

convolutional neural network A Convolutional Neural Network is a neural network type consisting of convolutional layers and is often used for visual recognition tasks. 2, 3, 6, 8, 9, 13, 15, 16, 18

dlib Dlib is a general purpose cross platform library implemented in C++ and accessible through python. 6, 16

GTX 1060 The Nvidia GTX 1060 is a popular graphics chip. 11

HoG A HoG (histogram of oriented gradients) is generated by calculating the contrast deltas of an image. 6

MobileNetV2 MobileNetV2 is a neural network for object detection and was developed by Google Inc. 2, 3, 6, 8, 15

OpenCV OpenCV is a library used real time computer vision. 2–9, 13, 16

Power BI Power BI is an enterprise reporting and analytics platform developed by Microsoft. 11

SQLite Free embedded SQL database. 11

support vector machine Support vector machines are a technique used in supervised learning to classify data (using regression, etc.). 10

TensorFlow TensorFlow is an open source machine learning framework developed by Google Inc. 8

Bibliography

[1] M. S. A. H. M. Zhu and A. Z. L.-C. Chen. "MobileNetV2: Inverted Residuals and Linear Bottlenecks". In: ().

[2] S. Yang, P. Luo, C. C. Loy, and X. Tang. "WIDER FACE: A Face Detection Benchmark". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[3] P. Suja, S. Tripathi, et al. "Real-time emotion recognition from facial images using Raspberry Pi II". In: *2016 3rd International Conference on Signal Processing and Integrated Networks (SPIN)*. IEEE. 2016, pp. 666–670.

[4] P. Viola, M. Jones, et al. "Rapid object detection using a boosted cascade of simple features". In: *CVPR (1)* 1.511-518 (2001), p. 3.

[5] N. Dalal and B. Triggs. "Histograms of oriented gradients for human detection". In: 2005.

[6] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista. "High-Speed Tracking with Kernelized Correlation Filters". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.3 (Mar. 2015), pp. 583–596. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2014.2345390.

[7] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui. "Visual object tracking using adaptive correlation filters". In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. June 2010, pp. 2544–2550. DOI: 10.1109/CVPR.2010.5539960.

[8] P. Lucey, J. F. Cohn, T. Kanade, J. Saragih, Z. Ambadar, and I. Matthews. "The Extended Cohn-Kanade Dataset (CK+): A complete dataset for action unit and emotion-specified expression". In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops*. June 2010, pp. 94–101. DOI: 10.1109/CVPRW.2010.5543262.

[9] *Tensorflow Model Zoo*. URL: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md (visited on 07/19/2019).

[10] *Carpool Karaoke*. URL: https://de.wikipedia.org/wiki/The_Late_Late_Show_with_James_Corden#Carpool_Karaoke (visited on 07/19/2019).

[11] *Jonas Brothers Carpool Karaoke*. URL: https://www.youtube.com/watch?v=aDcTSYZu-zY (visited on 07/19/2019).