FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Techniques for Improving Mobile Video Creation

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Visual Computing

eingereicht von

Dominik Schörkhuber, BSc.
Matrikelnummer 1027470

an der Fakultät für Informatik
der Technischen Universität Wien
Betreuung: Ao. Univ. Prof. Dipl.-Ing. Mag. Dr. Margrit Gelautz

Wien, 6. Dezember 2018

Dominik Schörkhuber	Margrit Gelautz

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Techniques for Improving Mobile Video Creation

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Visual Computing

by

Dominik Schörkhuber, BSc.
Registration Number 1027470

to the Faculty of Informatics
at the TU Wien
Advisor: Ao. Univ. Prof. Dipl.-Ing. Mag. Dr. Margrit Gelautz

Vienna, 6th December, 2018

Dominik Schörkhuber	Margrit Gelautz

Erklärung zur Verfassung der Arbeit

Dominik Schörkhuber, BSc.
Mühlbachstraße 21, 4451 Garsten

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 6. Dezember 2018

Dominik Schörkhuber

Acknowledgements

I would like to thank my supervisor Margrit for her great support during the development of this thesis. I want to thank my colleague Christian for the great discussions and many fun hours at the office. Most importantly, I thank my family for all the emotional and financial support to follow my education and interests.

This thesis was carried out within the research project “Intelligentes Assistenzsystem zur Videoerstellung auf Mobilgeräten (PersonalFilmAssistant)”, which is supported by the Vienna Business Agency under the Program “Users in Focus” (ID 1571390).

Kurzfassung

In dieser Diplomarbeit erforschen wir Methoden, um nicht professionelle Benutzer bei der Videoerstellung auf Mobilgeräten zu unterstützen. Die entwickelte Algorithmik ist eingebettet in ein storyboard-basiertes Anwendungskonzept. Wir stellen drei Arten von Assistenzsystemen vor, welche es einem Benutzer ohne Vorkenntnisse erlauben, kinematografische Konzepte anzuwenden und häufige Fehler bei der Aufnahme zu vermeiden. Um die Videoqualität zu verbessern, behandeln wir die Themen (a) Video Stabilisierung, (b) Shot-Typ Klassifikation und (c) Linsenverdeckung. Um ein Video zu stabilisieren, wird zunächst der Kamerapfad rekonstruiert. Wir vergleichen zwei Optimierungsansätze. Linear Programming wird eingesetzt, um den Kamerapfad stückweise zu linearisieren, und wir vergleichen diesen Ansatz mit einer lokalen Glättung des Pfades. In einem weiteren Ansatz präsentieren wir ein System zur automatischen Erkennung des Shot-Typs einer Szene. Zu dessen Erkennung extrahieren wir die Gelenkspunkte der dargestellten Akteure. Wir identifizieren den Hauptakteur und errechnen daraus eine kinematografische Beschreibung. Support Vector Maschinen zeigten in unserer Evaluierung die besten Klassifizierungsraten unter den verglichenen Ansätzen. Für Training und Evaluierung wurden mehrere Datensätze erstellt. Dabei setzen wir sowohl auf Szenen, die aus definierten Entfernungen aufgenommen wurden, als auch auf manuell annotierte Filmszenen. Das Klassifizierungsergebnis kann mit dem Storyboard verglichen werden, um korrektive Maßnahmen einzuleiten. Zuletzt behandeln wir unabsichtliche Verdeckungen der Kameralinse. Während der Videoaufnahme mit Smartphones ist es ein häufiger Fehler, die Linse unabsichtlich mit den Fingern zu verdecken. Wir formulieren das Problem als Segmentierungsaufgabe und wenden zur Lösung einen klassischen Bildverarbeitungsansatz als auch eine Deep Learning Methodik an. Die eingesetzte Deep Learning Architektur, eine Kombination aus Mobilenets und Fully Convolution Neural Network, zeigt deutlich bessere Ergebnisse.

Abstract

In this thesis, we explore methods to assist non-professional users with video creation on mobile devices. The developed algorithms are embedded into a video creation application featuring a storyboard-based workflow. We present three kinds of assistance systems which help the user avoiding mistakes commonly made by amateur users and follow cinematographic guidelines during recording. In order to improve the resulting video quality, we address the problems of (a) video stabilization, (b) shot-type classification, and (c) lens occlusion. In the context of video stabilization, the camera path is first reconstructed and then different optimization strategies are employed to improve the camera path. We use a Linear Programming approach to create a piece-wise linear path and compare it with a local smoothing method. Next, we present an approach to automatically infer the shot-type for a scene observed by a camera. Person keypoint detectors are used to extract joint information for all actors. We compute the skeletal representation of the main actor and classify it into a cinematographic description of the scene. Among the compared approaches for classification, support vector machines showed the best performance. For training and evaluation, we produce datasets based on image recordings at a set distance and manually annotated movie scenes. The result can be compared to a given storyboard in order to give feedback to the user accordingly. Finally, we address the problem of accidentally occluding the camera lens, which is a common mistake during recording with a smart phone. We formulate this task as a semantic segmentation problem and solve it with classical image processing as well as a deep learning method. The classical image processing approach is clearly outperformed by a combination of Mobilenets and Fully Convolutional Neural Networks.

Contents

CHAPTER 1

Introduction

The ubiquitous availability of smart phones with cameras enables us to capture moments of our lives by means of image and video at any given time. Recent statistical surveys show great growth in both watching and sharing video content on mobile devices, which emphasizes its current and future relevance [ST2a][ST2b]. While photography and videography tools on smartphones are straight-forward to use, it is still non-trivial to produce appealing results with them. Many precious moments like birthdays, weddings and holidays are captured on video but are often unappealing to watch when produced by an amateur user. For appealing results, a professional videographer is often hired. Camera and smartphone manufacturers are aware of this problem and have started equipping their devices with a range of assisting technologies to simplify the process of recording photo and video content. Naturally, from cameras (and phones) nowadays we expect them to provide assisting technologies like auto-focus or face detection. A more recent example is the automated generation of a bokeh effect for smartphone photography, made possible by the introduction of dual camera smartphones. While a range of assisting technologies exist for photography on modern smartphones, videography is still underrepresented in this aspect. Hence, taking high quality videos remains a challenge for non-professional users. Compared to photography, videography extends the problem of digital imagery to the temporal domain. While in photography only a still scene needs to be considered, taking video requires the user to plan the progression of a video over time. This includes both changes in the scene composition, as well as changes in camera positioning and movement. Furthermore, professional video usually does not consist of a single scene but contains a multitude of different scenes composed to a video. The diploma thesis is embedded into the research project Personal Film Assistant (PFA), which aims to ease the video creation process for amateur or novice users. The goal of the research project is to develop a smartphone application for video creation that meets the needs of a non-professional user. The concept of the PFA project is based on a storyboard approach, which guides the user through the recording process rather than letting the

user record and cut video freely. An initial user study carried out by our project partners suggests that, besides the proposed storyboard approach, further assistance for amateur users may be beneficial to improve the quality of generated videos. The main research goal of this thesis is to augment the storyboard approach with assistance systems that are capable of carrying out corrective measures on the recorded video footage, and/or give feedback and suggestions to an inexperienced user.

1.1 Background

The research project Personal Film Assistant is aiming to educate and assist users in how to take a professionally looking video. The core concept of the project is to ease the video creation process for a user, by guiding the user through the recording process and provide assistance where it is needed. Instead of relying on the user's knowledge of videography, the application encodes and applies knowledge provided by experts. This approach is implemented into a mobile application. The application is based on a storyboard workflow as depicted in Figure 1.2. While a predefined storyboard limits the users freedom, it also provides much required guidance to adhere to cinematographic rules. Once the user has chosen a storyboard matching his/her purposes, he/she is presented with a list of shots contained in the video. For each shot, the user is required to record a short video clip. Once all shots are recorded, the shots are automatically cut into a final video. In contrast to letting the user record videos freely and cut them together, this approach takes away the required planning of how to structure a video. Furthermore, the predefined structure of templates allows providing additional information for each shot in form of textual descriptions, images or videos. Further, the approach is suitable for encoding additional structured information into shots. As shown in Figure 1.2, each shot holds additional structured information, which is required to enforce and check cinematographic rules in an automated way. Information of such kind contains, but is not limited to, the length of a shot, expected camera movements or scene contents. Section 1.3 further describes the applied techniques and contributions that are presented in this thesis.

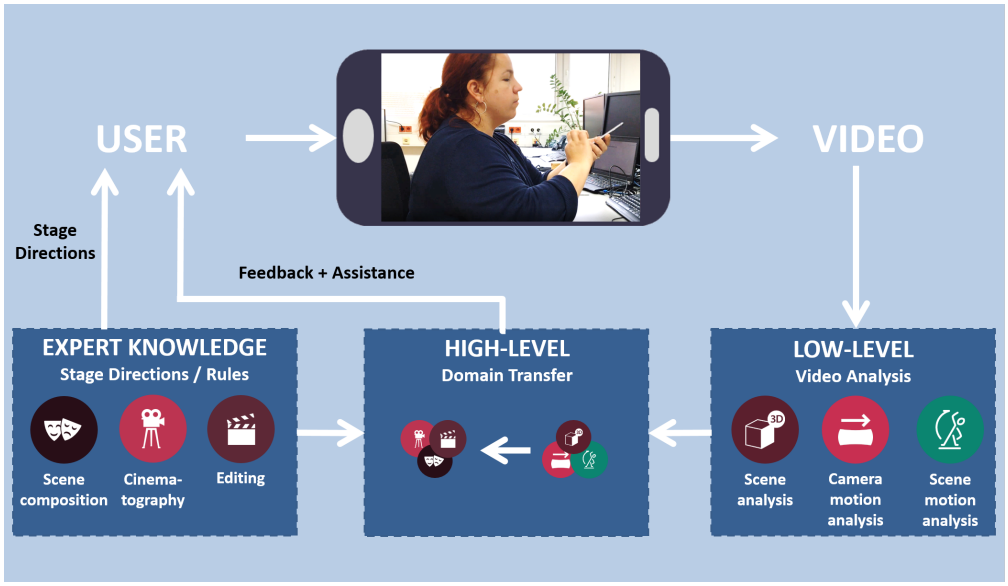


Figure 1.1: Instead of relying on a user's videography knowledge, the PFA application is guiding the users through the whole process. After and during videos are recorded video analysis algorithms are applied to extract high-level information about the scene [SSS+17].

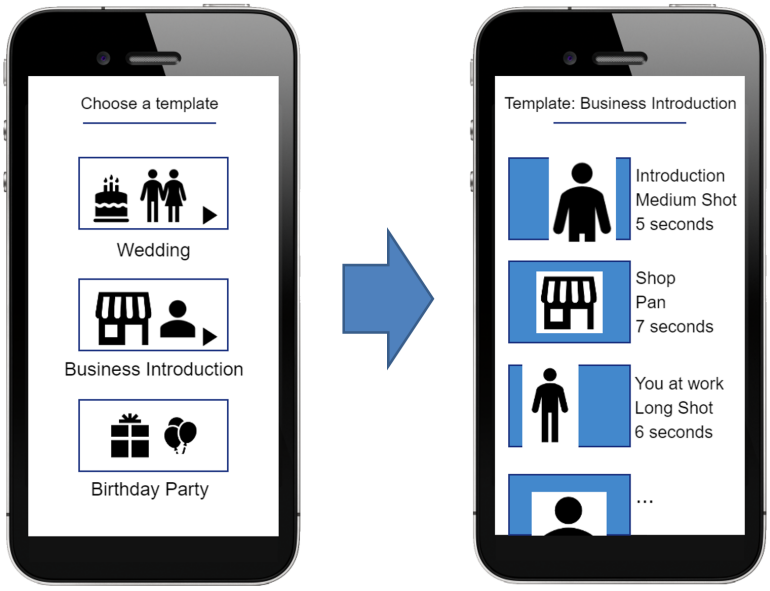


Figure 1.2: The Personal Film Assistant application is utilizing a storyboard driven approach. First, the user is choosing one of many video templates. The user then records a video for each scene in the template.

1.2 Problem Statement

Modern smartphones allow their owners to easily create videos at any time and place. Recently, the quality of image sensors, even in budget smartphones, has reached a very high level. However, while image sensors are getting closer to capture reality more faithfully, other issues in self-shot videos are still present. While professional video equipment might of course make a difference, basic knowledge about the rules of videography can increase the perceived quality a lot. In the scope of this thesis, we seek to find techniques that are suitable to assist users during use of the PFA application. After an initial literature research, we implement three assistance systems. Video, recorded with a hand-held camera often suffers from visual instability, due to shaky hands of the videographer. Different approaches can be implemented to stabilize video footage in a post-processing step after recording. Video templates in the PFA concept may be augmented with additional meta data. We assume that each shot in the template has a shot-type attached. A shot-type classifier should be created to check if the cinematographic rules defined in the template are fulfilled. Lastly, we identify that accidentally covering the lens of a smartphone camera is an easy to make mistake. A technique being able to warn the user when the lens is covered by fingers or hands is created.

1.3 Contributions

Our contributions are structured into three main topics. Each topic is showing a different approach to assist users with video creation. Below we are briefly listing our contributions in each of them.

Video Stabilization

We implement and compare two approaches to video stabilization. In the first approach, we apply Gaussian smoothing to the camera path and re-sample a video to improve stabilization. Secondly, we implement the video stabilization approach proposed in [GKE11], which globally optimizes the camera motion with Linear Programming. The approaches are evaluated on different performance metrics to quantify and compare the stabilization performance.

Shot-Size Classification

We present a novel approach to identify the shot-size of an observed scene by utilizing recent advances in person keypoint detectors. Based on the extracted skeletons we manually construct and train classifiers to determine shot-sizes used in cinematography. For training and evaluation purposes, we construct datasets based on recording keypoint motion at defined distances and by manually annotating video material.

Finger on Lens Detection

A novel approach is presented to detect fingers occluding a camera lens during photo or video creation. We formulate the problem as an image segmentation task and apply classical image processing techniques and more recent deep learning techniques for image segmentation in two different approaches. To train and evaluate the approaches we sample frames from video material and obscure them manually. The approaches are compared using different performance metrics with special attention to computation speed.

1.4 Outline of the Thesis

This first chapter gave an introduction and motivation to create assisting technologies for digital video creation. In the following, we present a brief outline of the topics of our work. The main chapters in this work follow the separation of the three assistance systems mentioned above.

Chapter 2: In the chapter **Related Work** we discuss the relevant state of the art. We start by reflecting how video recorded by novice creators is different from a professional's work. Basic principles of film and cinematography are introduced. We highlight techniques to assist users in video creation and discuss methods of computer vision to create and support such methods.

Chapter 3: Video Stabilization We briefly introduce different motion models and methods for video stabilization. We explain the algorithms behind two implemented approaches to optimize camera paths and to create a novel view on the original video material. We further discuss the implementation details for these approaches, and evaluate the methods quantitatively.

Chapter 4: Shot-Type Classification. We introduce the reader to film grammar and familiarize with different methods of describing scenes with special attention to shot-sizes. We discuss the generation of different datasets and person keypoint extraction for training and evaluation. Finally, we implement the classifiers and present our evaluation results.

Chapter 5: Finger on Lens Detection. The reader is introduced to the finger-on-lens issue and the problem is formulated as an image segmentation task. The implementation of two approaches based on classical image processing and deep learning techniques is discussed with special attention to low computational complexity. The methods are evaluated quantitatively in terms of classification performance and their real-time capability.

CHAPTER 2

Related Work

First, we describe some differences between videos recorded by an amateur and professionally produced video footage. After that, we take a look at different aspects of video production. In particular, we focus on techniques providing assistance to non-professional users for video production tasks like camera work and direction. Finally, we go into pose estimation and semantic segmentation which are required for our proposed approaches.

2.1 Professional Video

Liu et al. [NL12] studied the question "What makes a professional video?" by creating a computational approach to distinguish professional from amateur video. Image features describe the quality of each image frame separately. Noise, focus control, exposure control and the color palette are observed. For the overall video, they look at the camera motion, shot lengths and video continuity (i.e. visual smoothness).

Gleicher et al. [GL08] [GL07] analyze the difference between professional and amateur video as well. In specific, they highlight the differences between spontaneously taken home videos and professionally produced video. A main focus of their work, is to improve camera movement. While professionals use additional camera equipment like a tripod to stabilize the video footage, video recorded with a hand-held camera usually contains small jittery motion. In comparison, professional footage has a fluid, goal-directed motion. In their approach, videos are first split into several short clips, then each clip is stabilized with respect to saliency information. Grundmann et al. [GKE11] present a similar idea. In their approach to video stabilization, they aim to imitate the intentions of a professional videographer by optimizing the camera path towards a piece-wise directed motion.

In [BSA+16], Benini et al. analyze the used shot-types in movies. Similar to Mitarai and Yoshitaka [Mit12], they argue for the narrative and affective importance of camera position and framing. In their work, they analyze the movies of different directors. They create a classifier to determine the shot-type by combining different features and classify them with a support vector machine. The results are used to create distributions of shot-sizes for movies, which are used to classify the movies into different stylistic periods.

2.2 Video Stabilization

The field of Video Stabilization deals with methods to reconstruct, optimize and re-sample camera paths for image sequences. We can generally split the methods into techniques using 3D and 2D motion models. To recover scene geometry and camera position in 3D, the Voodoo Camera tracker has been used in numerous video stabilization papers [LGW+11][LGJA09][GF12]. VisualSFM [HPAP18] and CoSLAM [LLCZ16] have been used to recover scene geometry and camera positions in similar settings. While structure from motion algorithms can provide accurate results for the task, they also have high computational complexity. For that reason, many authors rather make use of 2D motion models, which can be further divided into linear and non-linear motion models. While linear motion models use a linear transformation to describe the inter-frame motion, non-linear methods usually use grid-warping. Especially when parallax motion or rolling shutter artifacts are present in the video, the motion can be only represented accurately with a non-linear approach [LYTS13][LYTS13][GKCE12]. However, the optimization of such models is more computationally expensive and can be less stable compared to linear transformations.

For path optimization, one approach is to apply low-pass filtering to the parameters of a linear motion model [MCB97]. Liu et al. [LTY+16] apply the strategy also to a non-linear motion model. They use a grid-based approach to model inter-frame motion and smooth the paths through the video on all positions of the grid. To keep neighbouring grid vertices stable, strong regularization is required. Liu et al. [LGJA09] use least squares optimization in their grid-based approach. Their objective function is split into several terms. The data term keeps corresponding points between the frames close, a similarity term avoids shearing and the temporal coherence term avoids artifacts due to unstable motion estimation. For motion optimization, Grundmann et al. [GKE11] use a Linear Programming approach and formulate the inter-frame motion by similarity transformations. Instead of just smoothing the path, this optimization approach minimizes the residual motion between the optimized video frames as well as their second and third derivatives. The result is a camera path favoring a steady camera or mostly linear motion.

2.3 Directive Assistance

In video grammar, the recording of an image sequence without interruption is called a shot. To distinguish shots, a special terminology is used among professionals. Common ways to distinguish classes of video shots are by the distance to their main subject (E.g.: Close-Up, Medium Shot), by the camera position and/or movement (E.g.: Aerial Shot, Pan), or special scene composition (E.g.: Over-the-shoulder shot).

The quality of videos is often only measurable to the point of low-level distortions, but images free from such artifacts can be visually unpleasing as well. In a cinematographically inspired approach, Hasan et al. have designed a novel motion descriptor named CAMHID [HXHX14] for videos. Their primary goal is to classify videos into shot-types (stationary, tracking, focus-in, focus-out, establishing, chaotic). Likewise, Bhattacharya et al. [BMSS14] produced a classification system for the classes aerial, bird-eye, crane, dolly, establishing, pan, tilt and zoom. While Hasan et al. [HXHX14] use videos mainly from Hollywood movies and videos shot by themselves, Bhattacharya et al. [BMSS14] use unconstrained amateur video from public sources as well. Techniques mentioned up to this point are not assistance systems per se, but may have the potential to be used as such. We have identified NudgeCam [CADB10] to be one of the earlier systems for affective assistance. It is presented as a mobile application executed on a user's Android smartphone. NudgeCam categorizes its assistance capabilities into three categories: show (present a demonstration), tell (relay instructions), and make (provide feedback). NudgeCam follows a template approach, where users first look at an example video, storyboard or other media before recording a video. Template creators may pick from a number of rules a video needs to fulfill. Possible rules are regarding the shot length, the positioning and size of faces, audio volume, camera tilt, detection of erratic motion or brightness. In a different fashion, The Director's Lens [LCRB11] provides directive assistance in a purely virtual environment. Given a virtual scene, the system is able to suggest virtual camera paths capturing the actions in the scene. Possible suggestions for camera movement are further ranked by fulfilling a number of cinematic continuity rules. Mitarai and Yoshitaka [MY11] have created a system that assists users in creating videos that communicate pre-selected emotions. While capturing a shot, the system simultaneously analyzes the result, and gives feedback to the user. Graphical and textual guidance is displayed for the user to correct his/her actions. The system features the following atmospheres: emotion, strength, weakness, tension/excitement, closeness/intimacy, loneliness and liberation. The "affective display" shows the user through icons which of the atmospheres are currently represented in the taken video. Furthermore, the "affective navi" can guide the user to reach a preferred atmosphere by applying corrective measures.

The general approach taken for automated shot-type classification is to first extract image features to subsequently train a machine learning classifier. Cherif et al. [CSP07] exploit features of the human body to classify shots based on the size of the main subject. Shots are manually annotated with measurements. For classification a decision tree for different shot-types is constructed manually. Among other features, different authors

utilize face detection as a descriptive feature for the classification of shots [CSP07] [BSA+16]. To analyze camera movement, different methods to construct temporal features based on inter-frame registration are applied. Hasan et al. [HXHX14] apply singular value decomposition to motion vector fields to extract and classify the dominant motion. Likewise, Bhattacharya et al. [BMSS14] transform inter-frame homographies into Lie space to use them as feature vectors. Tsingalis et al. [TTNP14] characterize shots with additional 3D information from depth maps. In their approach to detect different shot-sizes they also emphasize the applicability to detecting the special class of over-the-shoulder shots.

We further research methods to automatically detect and estimate poses of actors as used in [CSP07]. Recently a range of different pose estimation techniques have emerged. The goal of these techniques is to extract the pose information from images. The poses are represented as a skeletal graph structure with vertices representing the joints. Earlier approaches like [SGF+13] [SFC+11] used depth and RGB images to infer the pose. However, current approaches such as [CSWS17] [PZK+17] [FXTL17] require RGB input images only and are able to robustly detect multiple persons.

2.4 Semantic Segmentation

In recent years deep learning based approaches have superseded traditional techniques for image segmentation. These techniques can be divided into two main classes. Techniques based on region proposals and fully convolutional semantic segmentation [GLGL18]. For techniques based on region proposals, first several regions are extracted as proposals for a specific class, then the regions are classified. The classification results of the region can be further propagated to its source image. Representative work for this class of techniques has been carried out by Girshick et al. [GDDM14]. They use convolutional neural networks to extract image features from the region proposals, which are then classified by a support vector machine. FCN-based techniques extend the idea of object classification with CNNs from predicting a class for the whole image, to predicting a class for each pixel separately. Instead of fully-connected layers for the class output, Long et al. [LSD15] propose to use point-wise convolution following a de-convolution layer for bi-linear up-sampling instead. Long et al. and several following authors [RFB15, JDV+17], further suggest to combine the final prediction from several feature layers to especially improve boundaries in the segmentation. Generally such connections are called “Skip connections”, as they provide a short-cut between layers in the network. Conditional random fields (CRF) are another technique to improve the segmentation quality. CRFs have been used previously in segmentation tasks [KK11] but are very computation intensive. Chen et al. [CBP+16] use the technique successfully in the Deeplab[1] architecture to improve CNN based semantic segmentation results.

Apart from different decoding techniques to create the image segmentation output, the feature detection (encoder) part is very similar between the different publications and

[1] http://liangchiehchen.com/projects/DeepLab.html

can often be replaced with a similar architecture. In the original FCN paper, Long et al. [LSD15] used AlexNet [KSH12], VGG16 [SZ14] and GoogLeNet [SLJ+15] to evaluate their approach. Many of these architectures have been initially designed for image classification, but essentially all these networks are encoding image features which can be re-used for different purposes as well. Howard et al. [HZC+17] published an architecture called MobileNets. Their main contribution is the introduction of depth-wise separable convolutions. In this architecture, the standard convolution filters, which are the basis for CNN architectures, are replaced by a depth-wise separable convolution following a point-wise convolution layer. By sacrificing small amounts of accuracy, models using the MobileNets architecture are even scalable to smartphones.

CHAPTER 3

Video Stabilization

In this chapter, we explore techniques to improve unstable video footage. Videos recorded with a hand-held camera often contain unwanted motion due to shaky hands and other perturbations. To mitigate this effect, we discuss two computer vision based techniques to reconstruct the underlying camera path, optimize it and re-sample the original video to an optimized and more stable version. Finally, we evaluate and compare their stabilization performance using quantitative metrics.

3.1 Introduction

A common problem with mobile phone video recording is unstable video footage. As the user holds the phone in his/her hands, small shaking movements of the hands are transferred to the recording device and add distracting movement to the motion intended by the videographer. The added shakiness makes it very difficult to concentrate on details in the video, which makes it unpleasant to watch. Current smartphones use optical image stabilization, but these systems are costly to integrate and are constrained to small movements. For this technique, electromagnets move the camera lens to compensate for external movement. To stabilize video footage, professionals often use external equipment. For a steady shot, a tripod can be used. Steadycam systems also allow stable recordings of hand-held video. However, such assisting devices are expensive to buy, and the videographer needs to transport the devices. An amateur user usually has no access to such equipment. For these reasons, we address the video stabilization problem in this chapter and show the realization of two video stabilization systems purely integrated in software, without requiring external hardware.

3.2 Methodology

The purpose of video stabilization techniques is to analyze and optimize the visual quality of videos. Video recording with a hand-held camera often suffers from unwanted camera motion due to unstable hands and other motion perturbations. At the same time, the videographer may have an intended camera path he or she is trying to follow. Separation of intentional and unintentional camera motion is therefore crucial. To give an overview of the implemented video stabilization algorithms, we generally separate video stabilization techniques into three steps:

1. Motion Estimation
2. Camera Path Optimization
3. Frame Synthesizing

First, the original camera path is reconstructed. This is done by estimating the inter-frame motion and accumulating it frame by frame. The resulting path is then optimized such that the result is more appealing to the viewer. Finally, new video frames are sampled along the path to assemble a new video.

3.2.1 Motion Estimation

Feature Extraction

In the first step, the inter-frame motion is estimated. We use the Shi-Tomasi corner detection algorithm for feature extraction [ST93]. Similarly to the Harris corner detector, the Shi-Tomasi detector utilizes the eigenvalues of the Harris matrix (Equation 3.1) to determine if a pixel is a suitable feature in the image I, where I_x and I_y are the derivatives in x and y direction, respectively. If both eigenvalues $\lambda_{1,2}$ are greater than a set threshold λ_{min}, a pixel can be considered a suitable feature position.

$$A = \begin{pmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{pmatrix} \tag{3.1}$$

Feature Tracking

The extracted features are then tracked to the successive frame by the Lucas-Kanade algorithm [Bou01]. The idea of this algorithm is to assume that the brightness in a local neighborhood at time t and $t + \Delta t$ are equal (Equation 3.2).

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t) \tag{3.2}$$
$$= I(x, y, t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t \tag{3.3}$$
$$0 = \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t \tag{3.4}$$

By denoting the velocity as $\frac{\Delta x}{\Delta t} = V_x$ and $\frac{\Delta y}{\Delta t} = V_y$, we can simplify Equation 3.4 to:

$$0 = \frac{\partial I}{\partial x}V_x + \frac{\partial I}{\partial y}V_y + \frac{\partial I}{\partial t} \tag{3.5}$$

Like earlier in Equation 3.1, we again denote I_x, I_y and I_t as the partial derivatives of I.

$$\begin{aligned} I_x(p_1)V_x + I_y(p_1)V_y &= -I_t(p_1) \\ I_x(p_2)V_x + I_y(p_2)V_y &= -I_t(p_2) \\ &\vdots \\ I_x(p_n)V_x + I_y(p_n)V_y &= -I_t(p_n) \end{aligned} \tag{3.6}$$

In Equation 3.6 we setup a system of linear equations for the feature neighborhood at the positions $p_{1..n}$. This over determined system can be solved with least squares for the motion vector (V_x, V_y).

3.2.2 Gaussian Path Smoothing

In this approach to camera path optimization, we first reconstruct the inter-frame motion by estimating a homography on the matched features. The homographies are used to reconstruct the original camera path. We then smooth the reconstructed path and compute the difference between the original and smoothed path. For each frame, a homography is estimated to transform from the original to the optimized path.

Since the homographies cannot be smoothed directly, we apply the simple idea of smoothing the corners of the described image frames instead. We denote the video frames with $I_{1..N}$. For each video frame t we estimate a homography H_t from the matched features. We further define 4 points $C^{1..4}$ describing the corner positions of the image frame (Equation 3.7), where w and h describe the image width and height in pixels.

$$C = \begin{pmatrix} 0 & 0 & 1 \\ w & 0 & 1 \\ w & h & 1 \\ 0 & h & 1 \end{pmatrix} \tag{3.7}$$

CH_t computes the transformation of the corner points in homogeneous coordinates for the given frame I_t. We denote the transformation CH_t and homogenize the result by dividing by the third component in the vector.

We reconstruct the camera path P_t by accumulating the corner movement in each frame.

$$P_t = \sum_{i=1}^{t} CH_t - C \tag{3.8}$$

The corner profiles P_t now describe the movement of the image corners from frame to frame. We apply Gaussian smoothing to the corner profiles to calculate the smoothed corner profiles P'_t. The difference $P'_t - P_t$ describes the required movement of the image corners to transform each frame I_t to the corresponding frame I'_t on the smoothed camera path. We apply this transformation by a perspective warp.

3.2.3 L1 Optimal Camera Path

We denote the content of the original video as a frame sequence $I_0, I_1, ..., I_N$. The camera motion is estimated by feature matching between successive frame pairs (I_t, I_{t+1}). A linear motion model is used to describe this transformation. We denote the inter-frame transformation of I_t to I_{t-1} as F_t. Furthermore, the reconstructed camera path can be computed by concatenating the inter-frame motions. The absolute transformation of a frame along the path is computed by $C(t) = \prod_{i=0}^{t} F_i$. The stabilization approach computes a transformation for each frame from the original to the optimized camera path, $P_t = C_t B_t^{-1}$ where P_t is the optimized path and B_t^{-1} is the transformation between old and new path. The relation between reconstructed and optimized path are outlined in Figure 3.1.

To compute the novel camera path, we follow the optimization approach presented by Grundmann et al. [GKE11]. Their main idea is to minimize the residual motion between optimized frames (Equation 3.9) with Linear Programming. Equation 3.15 describes the cost function to minimize. Along with the residual motion, also the first and second derivatives are minimized to ensure a smooth target path. The weights w_1, w_2 and w_3 in Equation 3.15 are used to balance the influence of the derivatives.

$$D(P) = \sum_{t} P_{t+1} - P_t \tag{3.9}$$

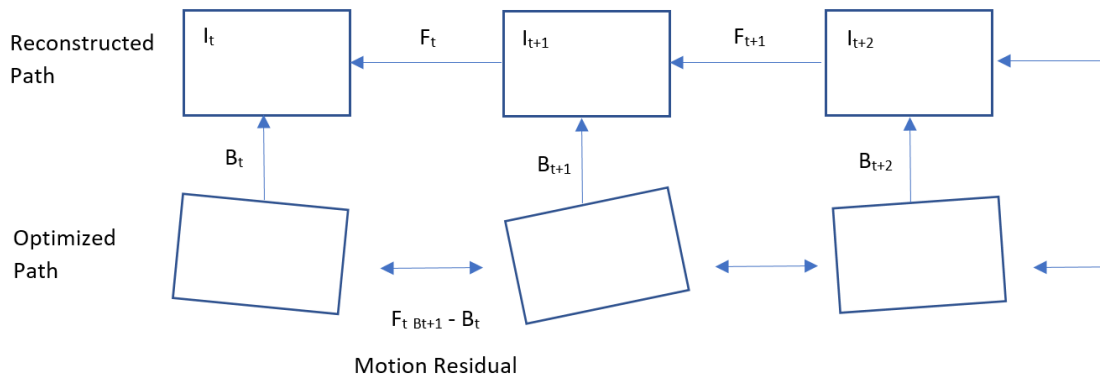


Figure 3.1: Computation schema for L1 Optimal Camera Paths

$$|D(P)| = \sum_t |P_{t+1} - P_t| \tag{3.10}$$

$$= \sum_t |C_{t+1}B_{t+1} - C_tB_t| \tag{3.11}$$

$$= \sum_t |C_tF_{t+1}B_{t+1} - C_tB_t| \tag{3.12}$$

$$\leq \sum_t |C_t||F_{t+1}B_{t+1} - B_t| \tag{3.13}$$

$$R_t = F_{t+1}B_{t+1} - B_t \tag{3.14}$$

$$\mathcal{O}(P) = w_1|D(P)|_1 + w_2|D^2(P)|_1 + w_3|D^3(P)|_1 \tag{3.15}$$

By forward differencing, the derivatives of the motion residuals are computed as shown in Equation 3.16. The motion residuals are minimized by the introduction of the slack variables e_t^1, e_t^2, e_t^3. This results in $3(N-1)$ additional variables.

$$\begin{aligned} -e_t^1 \leq & R_t & \leq e_t^1 \\ -e_t^2 \leq & R_{t+1} - R_t & \leq e_t^2 \\ -e_t^3 \leq & R_{t+2} - 2R_{t+1} + R_t & \leq e_t^3 \end{aligned} \tag{3.16}$$

The constraints in Equation 3.16 allow for a parameter-wise minimization of each parameter in the transformation matrices B_t. We further want to constrain the possible transformations. B_t should stay close to a similarity transformation, which is enforced by Equation 3.18.

$$B_t = \begin{pmatrix} a_t & c_t & 0 \\ b_t & d_t & 0 \\ x_t & y_t & 1 \end{pmatrix} \tag{3.17}$$

$$\begin{aligned} 0.95 \leq a_t, d_t &\leq 1.05 \\ -0.05 \leq b_t, c_t &\leq 0.05 \\ -0.025 \leq b_t + c_t &\leq 0.025 \\ -0.05 \leq a_t - d_t &\leq 0.05 \end{aligned} \tag{3.18}$$

After applying the optimized transformation B_t to the frame I_t, parts in the image would be missing image information. To avoid that, we define a crop window inside the image frame defined by the corners $Cr^i_{x,y}$. In Equation 3.19 the constraint is defined to keep the cropped image frame within the original image frame. This ensures that no blank areas will be revealed in the optimized video. The size of the cropping window $Cr^i_{x,y}$ steers how aggressively the video may be stabilized.

$$\begin{pmatrix} 0 & 0 \end{pmatrix} \leq \begin{pmatrix} Cr^0_x & Cr^0_y & 1 \\ Cr^1_x & Cr^1_y & 1 \\ Cr^2_x & Cr^2_y & 1 \\ Cr^3_x & Cr^3_y & 1 \end{pmatrix} B_t \leq \begin{pmatrix} w & h \end{pmatrix} \tag{3.19}$$

Finally, we summarize the minimization target and constraints in Algorithm 3.1.

3.3 Implementation

3.3.1 Motion Estimation

We implement both stabilization approaches into Python programs. For motion estimation we use the OpenCV library and the corresponding Python wrapper cv2[1]. `cv2.cvtColor` is used to convert the images to grayscale and features according to [ST93] are extracted by `cv2.goodFeaturesToTrack`. The features are then tracked with the Lucas-Kanade method described in [Bou01] and implemented in `cv2.calcOpticalFlowPyrLK`.

3.3.2 Gaussian Smoothing

With the matched features, a homography is estimated by `cv2.findHomography`. The found homography transforms features from the previous to the current frame. We

[1] `https://opencv.org/`

compute the dot-product between the frame corners and the homography to determine the relative transformation for each corner between the frames. The result is homogenized i.e. divided by the z-coordinate. To determine the relative movement of the frame corners, we subtract the transformed corner from the original image frame. Each frame, the additional relative movement is added to the absolute position of the frame to form the camera path. To filter the corner trajectories we apply Gaussian smoothing with $\sigma = 20$. We use the `gaussian_filter` implemented in the SciPy[2] library.

3.3.3 L1 Optimal Camera Path Optimization

To setup and solve the Linear Programming problem, we use the python package Cvxpy[3] [DB16]. With Cvxpy we are not required to reformulate the problem in matrix forum, but are able to directly formulate the Linear Program as stated in Algorithm 3.1. The variables $e_{1..3}$ and B_t are defined as `Variable` objects and the minimization target is computed. The additional constraints for smoothness, proximity and inclusion are added to the solver via the constraint array.

Algorithm 3.1: Summary of optimization rules for the linear program.

minimize $\sum_{t=1}^{N} w_1 e_t^1 + w_2 e_t^2 + w_3 e_t^3$

subject to:

smoothness
$$\begin{cases} -e_t^1 \leq & R_t & \leq e_t^1 \\ -e_t^2 \leq & R_{t+1} - R_t & \leq e_t^2 \\ -e_t^3 \leq & R_{t+2} - 2R_{t+1} + R_t & \leq e_t^3 \end{cases}$$

proximity
$$\begin{cases} 0.95 \leq & a_t, d_t & \leq 1.05 \\ -0.05 \leq & b_t, c_t & \leq 0.05 \\ -0.025 \leq & b_t + c_t & \leq 0.025 \\ -0.05 \leq & a_t - d_t & \leq 0.05 \end{cases}$$

inclusion $\begin{pmatrix} 0 & 0 \end{pmatrix} \leq CrB_t \leq \begin{pmatrix} w & h \end{pmatrix}$

3.3.4 Video Encoding

Due to the image warping, areas in the target frame may exist where image information is missing. To avoid blank areas in the output video, we crop the width and height of the video frames to 80%. That means we shift each each image border by 10% of the image width. While the approach based on linear programming utilizes a constraints to keep the frame border within the cropping area, the Gaussian smoothing approach does not have such a condition, which means that after strong elongation of the camera path

[2] `https://www.scipy.org/`
[3] `http://www.cvxpy.org/`

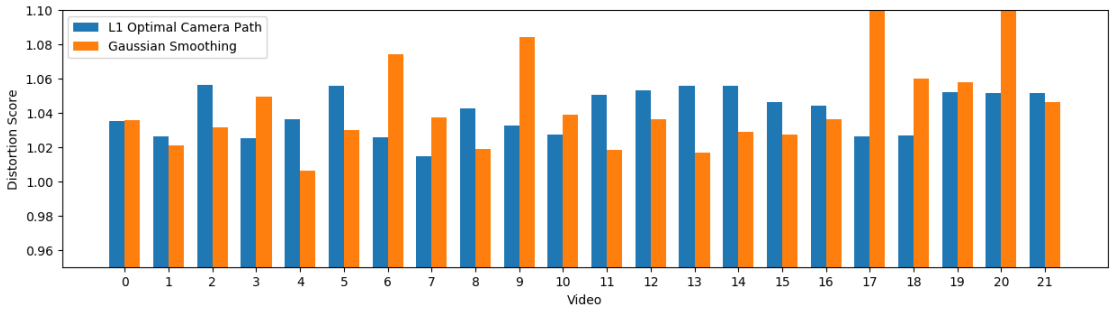


Figure 3.2: Distortion score for all videos in the category "Regular" (1 is best).

the final video may still suffer from missing image information. To mitigate the problem we replicate the image information to the border. The cropped frame is re-scaled using linear interpolation to the original frame size. Finally, the transformed image frames are encoded into a video file by the `cv2.VideoWriter` class and usage of the h264 codec.

3.4 Results and Evaluation

To evaluate and compare the two implemented approaches, we compute different scores on the results and determine the required computation times. We apply the algorithms to the video dataset[4] published by Liu et al. [LYTS13]. The dataset is split into videos of different categories, namely Regular, Quick Rotation, Zooming, Large Parallax, Driving, Crowd and Running. Each category roughly contains 20 videos each. We conduct our experiments on the category "Regular". The following evaluation is split into three parts. First, we discuss the results in terms of our subjective impressions. We then compute a metric to estimate distortion of the image frames. Further, a metric to estimate the stability of the camera paths is presented. Finally, we compare the average runtime of the employed approaches.

3.4.1 Distortion Score

The distortion score measures the distortion caused by transforming the original image frames to the optimized camera path. According to [LYTS13], we estimate the distortion by computing the ratio between the eigenvalues of the affine part of the transformation matrix B_t. Optimally, the ratio of the eigenvalues is 1, which corresponds to no distortion. Anisotropic scaling and/or shearing causes the score to deviate from 1. The score for a video is computed by taking the maximum (highest) distortion over all frames. The distortion scores for the chosen dataset are illustrated in Figure 3.2. Gaussian smoothing is superior in 12 of 22 videos, but also shows more extreme distortion in the videos 6, 7, 17 and 20 due to the more flexible motion model.

[4] `http://liushuaicheng.org/SIGGRAPH2013/database.html`

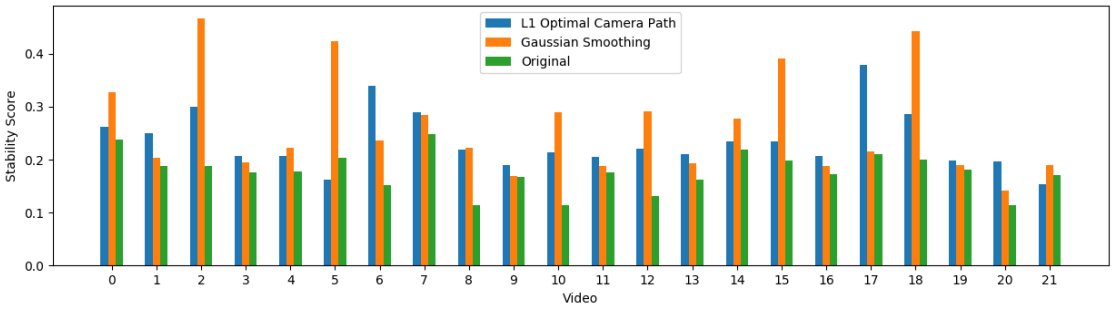
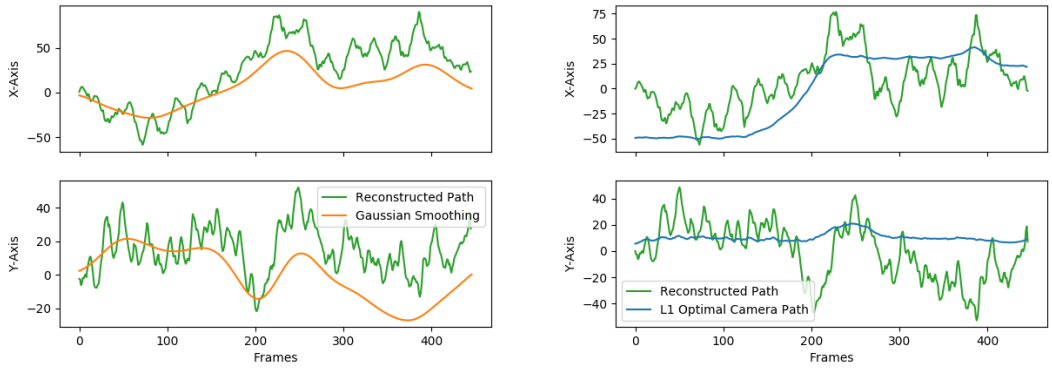


Figure 3.3: Stability score for all videos in the category “Regular” (higher is better).

3.4.2 Stability Score

For the stability score, we analyze the translational part of the original and optimized camera paths as shown in Figure 3.4. We apply a Fast Fourier Transform to the camera paths, to split the movement signal into its frequency components. In particular, we analyze the frequency spectrum of the vertical camera movement, as due to walking motion, most videos show strong elongations on the vertical axis. Figure 3.5 illustrates the spectrum of the camera movement. The intuition is that more energy in the lower frequencies suggests that a video is more stable. Like Liu et al. [LYTS13], we sum the energy of the 2nd to 6th frequency components and compute the ratio to the energy over the full spectrum. Therefore, a higher ratio indicates a more stable video. Figure 3.3 shows the results for the used dataset. The stability in the optimized camera paths is usually much higher than the unstabilized paths. Gaussian smoothing shows a higher score in 11 of 22 videos, which means there is no noticeable difference in stability. However, the average stability score of Gaussian smoothing (0.254) is slightly higher than L1 Optimal Camera Paths (0.231).



(a) Gaussian Path Smoothing (b) L1 Optimal Camera Paths

Figure 3.4: Translational parts along the horizontal (top row) and vertical (bottom row) axis of the reconstructed and optimized camera paths by Gaussian Path Smoothing (a) and L1 Optimal Camera Paths (b).

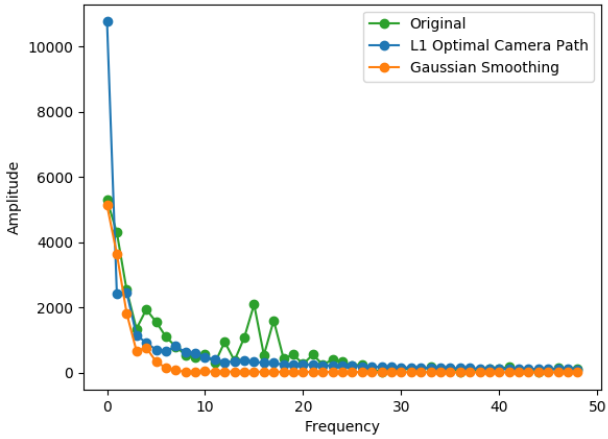


Figure 3.5: Frequency analysis of the reconstructed and optimized camera paths displayed in Figure 3.4

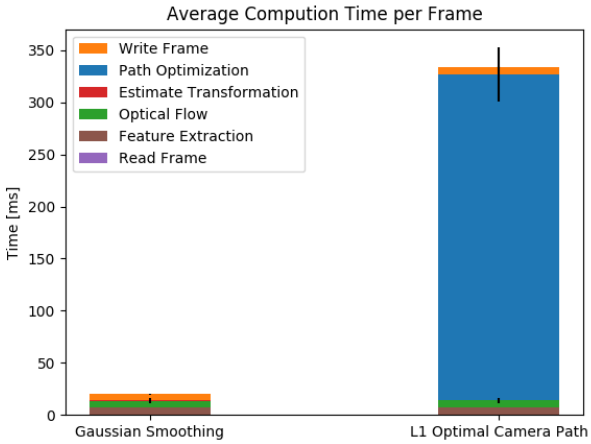


Figure 3.6: Average computation time per video frame for the employed stabilization approaches.

3.4.3 Runtime

In this experiment, we measure the computation time for both approaches. We measure the time for all stages of the algorithms separately and compute the average time for all videos in the category "Regular". The used videos have a resolution of 640x360 pixels and a framerate of 30 frames per second (FPS). We compute the results on a desktop computer system with an Intel 1230v3 processor @3.30GHz. Figure 3.6 shows average times for the stages of both approaches. For this evaluation, we measure the time of all required stages of the algorithm, including reading and writing the video to disc.

Due to its simplicity, the approach using Gaussian smoothing for path optimization is very fast. On average, this approach takes just 20ms per frame. Which means we can theoretically compute videos with up to 50FPS in realtime (on the used system). On the contrary, the L1 Optimal Camera Path approach requires an average of 333ms per frame, which means roughly 10 minutes of computation for a 1 minute video. Due to the similar results in stabilization performance according to the computed scores and our subjective impression of the stabilized videos, we believe the technique using Gaussian Smoothing should be the preferred approach.

CHAPTER 4

Shot-Type Classification

In film-making, the narrative expressiveness is greatly influenced by the way a scene was filmed. Not only the composition of a scene is important, but also the way it is filmed and what is captured in the camera frame. Variance in camera position and framing add depth and structure to a movie. These techniques enable the director to steer the viewers attention and emotions in a better way. For more in-depth information about the used concepts in cinematography, consider [Bow16] and [Bro16]. In this chapter, we are addressing a way of how the properties of a shot are described by professionals and how we can assist non-professional videographers to make use of these techniques. First, we give a brief introduction to the terminology used among professionals. In the main part of this chapter, we propose, implement and evaluate a computer vision based technique to automatically determine the shot-type of a scene.

4.1 Background and Terminology

Film makers tend to use a common language for communication between writers, directors and camera operators [Bow16]. In the following, we give a brief overview of the used terminology to describe movie content.

In order to describe and visualize the different events in a movie, professionals set up a storyboard before filming the actual video footage. In a chronological order, the storyboard describes what is happening in the movie. The storyboard is divided into different scenes and each scene may consist of multiple shots. Shots are the most atomic unit of a video. Shots are filmed in one continuous take, i.e., the shot is a series of frames recorded without interruption. In the storyboard, all shots are depicted as an image[1] of the scenery and a textual description of the depicted events. An exemplified storyboard, as used in the PFA application, is depicted in Figure 1.2.

[1] Depending on the complexity of a shot, a storyboard may describe a shot with multiple pictures. For the sake of simplicity, we are using one picture per shot.

For easier description and communication, shots are categorized into different types. Different qualitative descriptions of shots are possible. A common feature to distinguish shots is the framing of the main subject. Figure 4.1 gives an overview of different types according to this categorization. Other possibilities to characterize shots are by their duration, camera movement, camera angle and other features of scene composition. In the following, we are focusing on the distinction of shots by shot-size and describe a typically used classification scheme. However, this is not the only way to distinguish shot-types. Depending on the application and required detail of expression, it is reasonable to extend or limit the amount of classes.

- Close-Up (CU)
  The close-up shot is showing the face of the actor in great detail and may extend down to the shoulders. Due to its closeness, a Close-Up is able to magnify the emotions of the shown actor. It is therefore also often described as a very intimate type of shot.

- Medium Close-Up (MCU)
  Medium-Close-Up shots are also called waist shots, since they are framing the subject from the waist upwards. Similar to a close-up, big portions of the frame are covered with the face area. It is a preferred shot-type to choose when somebody is talking or when the focus is on the performed actions of a character.

- Medium Shot (MS)
  Often the most common shot-type in a movie is the medium-shot. Medium-Shots are depicting subjects roughly from the pelvis upwards. Roughly speaking, the distance of a Medium-Shot is similar to how we perceive a person standing right in front of us, without violating personal space.

- Medium-Long Shot (MLS)
  A Medium-Long Shot is showing about three quarters of a person. The image frame is reaching approximately from the knees upwards. It is often used in scenes where multiple actors should be visible at the same time.

- Long Shot (LS)
  Long-Shots are showing the subject from head to toe. Besides the subject, also the surroundings are visible in this type of shot. Therefore, a Long-Shots is often used to establish a scene. It may be utilized to introduce the viewer to a new scenery.

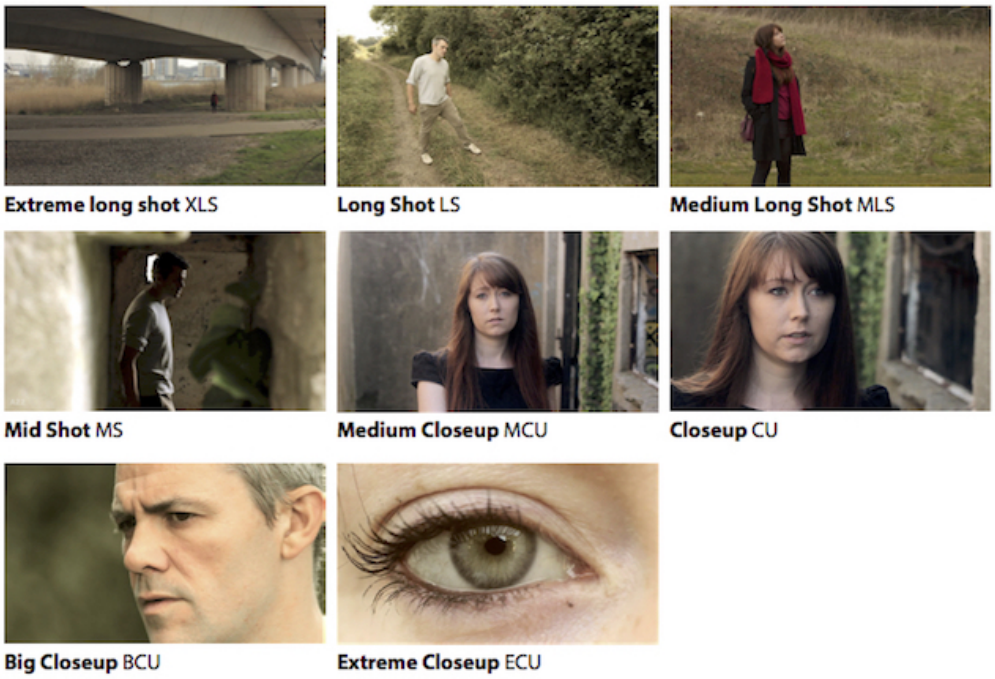


Figure 4.1: Depiction of different shot-types, based on the framing of the main actor [bar]. Terminology in this figure slightly differs from ours.

4.2 Methodology

In the following, we present our approach to an automated classification system for shot-types. The general idea of our approach is to automatically determine the shot-size of an image by first estimating the main actor's pose in the image and then classify that pose into one of several shot-size classes.

To detect the shot-type, several previous approaches [BSA+16, CBL13, XWH+11] use feature ensembles. In contrast to these approaches, we are focusing our efforts on analyzing the pose of actors. The closest previous approach is the work done by Cherif et al. [CSP07], where different measures on the human body are taken and classified by a manually constructed decision tree. However, their approach required manual annotation of said measurements. The detection of actor poses with our approach is fully automated. We can broadly describe our approach in the following steps:

1. Extract poses from scene
2. Identify the most relevant actor
3. Classify the pose into a shot-size
4. Give feedback to the user

The approach operates on a frame-to-frame basis. In the first step, a pose estimator extracts the poses of all actors shown in the video frame. The poses are represented as skeletal graph structures. In the case of multiple actors being detected, the most relevant actor is selected by its bounding box size and orientation. The poses, or skeletons respectively, are the basis for our classification. We train different classifiers to assign shot-sizes to observed poses.

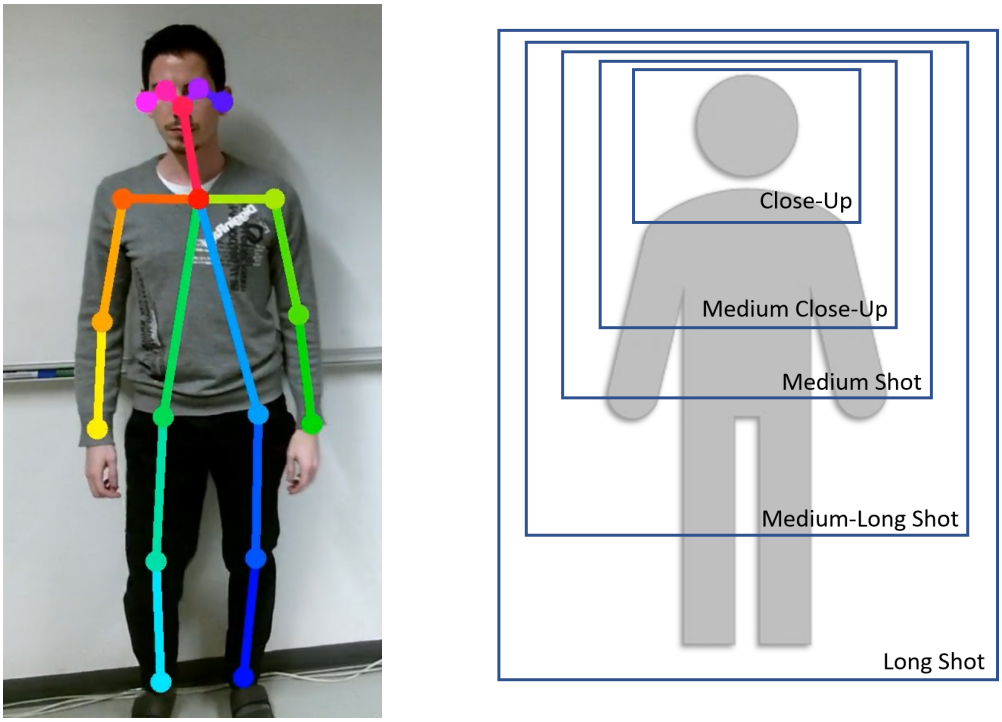
In the sense of [CSP07], our first approach is to crafted a classifier manually. Based on the appearance of different body parts in a frame rules are defined to categorize the pose. Alternatively, we present two machine-learning classifiers which are trained to distinguish shot-size classes. After the system has determined the shot-size, feedback is given to the user. In our setup, with integration into the PFA project, the shot-size classification acts as an assisting technology. As shown in Chapter 1, the videographer follows a predefined template for recording. Each shot in the template may have an expected shot-size attached, which is compared to the current state of recording to trigger actions for feedback.

To keep annotation and classification manageable, we decided to limit the set of existing classes to a small but descriptive sub-set. This simplification of the problem gives us also the opportunity to compare our approach to other shot-size classifiers like [BSA+16]. Our shot-size classification approaches are operating on two different sets of classes. Figure 4.2b gives an overview of the five shot-size classes (CU, MCU, MS, MLS, LS) in use. Besides the discrimination into five classes, classifiers are also trained on a reduced set of three classes (CU, MS, LS).

4.2.1 Extracting Actor Poses

To extract pose information, the current state of the art in pose estimation techniques has been reviewed. Recently a number of new methods for pose estimation based on CNN architectures have been published. The COCO Keypoint Challenge 2016[2] provides an expressive benchmark for pose estimation algorithms. For use in our experiments, we choose the Part Affinity Fields [CSWS17] approach by Cao et al., as it presents the currently top scoring technique in the benchmark. Furthermore, the approach has been conveniently implemented into a programming framework called 'Openpose'. Based on the input of an RGB image, the pose estimator is able to return a keypoint descriptor for each detected person in an image. Openpose, as well as other pose estimation techniques, are using a simple tree structure to represent the pose of a human body. In our approach, we utilize the COCO descriptor to define poses. The descriptor contains 18 keypoints, whereas each point is defined by its position in the image plane and the confidence of the detection. Each pose is therefore defined by $3 \cdot 18 = 54$ floating-point values.

[2]Leaderboard of the COCO Keypoint Challenge 2016 `http://cocodataset.org/`



(a) (b)

Figure 4.2: Pose detection and shot-types. a) Pose detected and displayed by the Openpose pose estimator and pose renderer. b) Shot-size classes considered for classification along with the approximate camera framing.

4.2.2 Main Actor Selection

To feed the pose information into machine learning classifiers, a fixed set of input points is desirable. To achieve that, we are aiming to select the pose of the main actor only. Usually a shot in film has one main actor, who is the center of attention while performing actions or speaking. Therefore, only the main actor is used for the selection of an appropriate shot-size. A brief analysis of our datasets has confirmed that assumption, as surrounding persons are often displayed further away or, in case of over-the-shoulder shots, even with the back to the camera. We further define the importance of actors in a shot by the smallest bounding box enclosing all keypoints of the corresponding pose descriptor. To choose the main actor, we select the actor with its bounding box having the largest bounding box area. Generally, this strategy is successful, but we noticed that this approach was under-performing especially for over-the-shoulder shots. This type of shot is often used in conversations and shows the current main actor while the camera is looking 'over the shoulder' of the character he is speaking to. Naturally, the person shot from the back appears bigger than the actual main actor in such cases. To solve this issue, we introduce an additional rule for the main actor selection. Whenever there

are multiple actors in a scene, we disregard characters who are turned away from the camera. In terms of skeletal keypoints, we can easily express this condition by analyzing the x-coordinates of the shoulder keypoints. In the case of an actor facing the camera, the left shoulder would always be further to the right than the right shoulder, and vice versa if the actor is not facing the camera.

4.2.3 Pose Classification

After choosing the most relevant pose, its keypoints are used to assign a shot-size class by a classifier. For that purpose, we apply and compare three different ways of classification. a) A manually constructed decision tree. b) Support Vector Machines. c) Fully connected neural network. The input to all three approaches consists of 18 keypoints, which are contained in the pose descriptor. A datapoint given by the pose descriptor contains the (x, y) coordinates of the keypoint as well as the corresponding confidence, describing the certainty that a keypoint was placed correctly by the pose estimator.

For the manually constructed decision tree, we follow the basic intuition of which keypoints are present when observing different types of shots. Looking at Figure 4.1, it becomes clear that in an up-right position only a Long-Shot also shows the actor's feet. Similarly, we can argue that knees would only be visible in a MLS, the pelvis is only present in a MS, shoulders are visible in a MCU and if none of the previous are present, a CU is assumed. For a binary decision on a point being present, a threshold must be introduced. Within the Openpose framework, a confidence threshold of 0.05 is used to determine the presence of a keypoint, which we found suitable for our purpose too. This classification technique is limited in comparison to the wide variety of poses that may occur in practice. As indicated in Figure 4.2, the taken assumptions are usually valid when the actor is standing up-right. While this certainly is the most common pose, the approach is inflexible to more general cases.

In order to overcome these limitations, two classification methods based on machine learning are considered. In our second approach, we have constructed a densely connected neural network with 3 hidden layers. Each layer contains 16 units. The output layer contains one neuron per class. To determine the output class, the output neuron with the largest response is chosen. Classically, back propagation and gradient descent are used to train the network. In a third approach to classification, a Support Vector Machine (SVM) is applied to the pose classification problem.

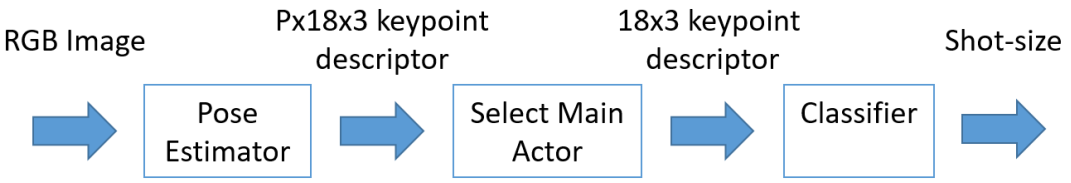


Figure 4.3: Conceptual overview of our approach to shot-size classification. First, an image is fed into a pose estimator. Poses of possibly multiple actors are extracted. Each descriptor contains 18 keypoints describing the pose, including the point coordinates and confidence. The descriptor of the main actor is selected and fed into the classifier.

4.3 Dataset

To evaluate the classification approaches, three datasets with different properties have been arranged. Each dataset consists of image files with according annotations for the shot-size class and pose of the main actor. We denote the datasets with the identifiers A,B and C. The datasets are annotated with 5 classes. Furthermore, we also derive a split into 3 classes by combining the CU and MCU as well as the MLS and LS classes. This results in a total of 6 datasets. In the following, we denote the classes by A5,B5,C5 and A3,B3,C3 to refer to the datasets containing 5 or 3 classes respectively. Table 4.1 gives an overview of the existing datasets.

Dataset A (recorded) consists of video frames, which have been recorded under controlled conditions. A camera was positioned at a set distance, to mimic camera framing corresponding to one of five shot-types. A total of 5181 frames has been collected by recording an actor moving in front of the camera. During the whole recording, the actor was freely moving in an up-right position. This way, it was possible to generate training data of high quantity in very limited time. However, this dataset may not represent the broad spectrum of poses possible in reality.

Dataset B (manually annotated) has been created by manually annotating episodes of different TV series. Hence, it represents (to some extent) the usage of shot-types by a professional director. To be suitable for our approach, the episodes are first transformed to images. Since the shot-size is defined on a per scene basis, we are extracting one image per scene. The extracted images are then annotated with the corresponding shot-type. To make the process more efficient, a simple tool for annotation has been developed. The annotation tool, visible in Figure 4.4, allows the annotator to step through the extracted images. For each image, one- or multiple shot-types can be chosen via checkboxes. After stepping to the next image, the selection is stored to disc. For this second dataset, a total of 972 frames have been annotated manually. In Table 4.1, the distribution of shot-sizes is shown. We observe a strong trend towards Medium-Shots. While the distribution of shot-types seems to be centered around Medium-Shots, we recognize the approximate shape of a normal distribution for our dataset. At this point, we would like to refer the

Figure 4.4: Our annotation tool for shot-type classes allows an annotator to quickly browse through a dataset of images and assigns one or multiple classes to it. For possible future applications and/or extensions to the current approach, we have integrated a super-set of shot-type classes into the annotation tool, as it can be seen on the right.

reader to the work done by Benini et al. [BSA+16], who have analyzed the distribution of shot-size in film as a main objective of their work. They found that the distribution of shot-types is similar in stylistic periods for the observed directors.

While Dataset B (manually annotated) better reflects the contents of directed video scenes, we have fewer data samples available than in Dataset A (recorded). A third **Dataset C (combined)** was therefore created, which combines the qualities of both previous sets. Dataset C (combined) contains all data in Dataset B (manually annotated) and is augmented by 600 randomly sampled data points from Dataset A (recorded). Especially the percentage of data points in the CU and LS classes are raised by this method. To prevent over-fitting, we are using the additionally added data for training only. The amounts of samples in Dataset A,B,C and their associated shot-types are given in Table 4.1.

To create re-producible results, we split the data into training and test sets before further evaluations are conducted. For machine learning, typical splits of 70% training and 30% test data are applied to all datasets.

	CU	MCU	MS	MLS	LS	Total
Dataset A5 (recorded)	1804	600	1526	611	640	5181
Dataset B5 (manually annotated)	13	264	455	159	81	972
Dataset C5 (combined)	112	390	678	243	141	1564

	CU	MS	LS	Total
Dataset A3 (recorded)	2404	1526	1251	5181
Dataset B3 (manually annotated)	277	455	240	972
Dataset C3 (combined)	502	678	384	1564

Table 4.1: Datasets A,B and C with associated distributions of shot-sizes.

4.4 Implementation

To demonstrate the feasibility of the proposed method, an evaluation system has been implemented for the quantitative evaluation in the scope of this thesis as well as for a qualitative user study in the PFA research project.

4.4.1 Pose Estimation

For pose estimation, the recently published Openpose[3] framework is utilized. The framework contains keypoint detection algorithms for the human body, faces and hands based on [CSWS17, SJMS17, WRKS16], respectively. The framework contains an application interface for the C++ programming language, during the development of this thesis also a Python wrapper was added. Furthermore, a binary demo is available for Microsoft Windows.

4.4.2 Datasets

To record the datasets, a combination of different tools was used. Due to Openpose being heavily in development while we were developing this part on our side, we have adapted our toolset several times during the creation of the different datasets. Firstly, for Dataset A (recorded), videos representing the different shot-size classes were recorded with the Openpose binary demo. The binary demo allows for recording pose data in xml/json format without extensive setup- or compilation time. For this first dataset we have parsed the xml formatted pose data with Matlab. For a more versatile approach, we have later implemented this approach into a C++ application. The application loops over all images in a provided folder and executes the pose estimator. The main actor for each image is determined as described in 4.2.2.

To create Dataset 2 (manually annotated), we extended the previous approach. In this dataset, episodes of TV series are used. Instead of extracting poses from all frames contained in an episode, we reduce the image data by selecting one image per scene. That

[3] `https://github.com/CMU-Perceptual-Computing-Lab/openpose`

vastly limits the amount of images which must be labeled manually. For this purpose, we use the video editing tool FFMPEG[4]. Algorithm 4.1 shows our attempt to extract one frame per scene. The described video filter computes the Sum of Absolute Differences (SAD) between each consecutive frame pair. The SAD is normalized by the number of pixels in the image. Therefore, we can specify a threshold between 0 and 1, to indicate a scene change. A threshold of 0.4 seems appropriate to detect most scene changes. For annotation, we have created a simplistic tool to navigate through the images of a folder based on Matlab GUI[5] (See Figure 4.4). In the annotation tool, classes for each image can be selected by checkboxes. When navigating to the next image, the tool creates a classes file containing the selected labels in a comma-separated format.

4.4.3 Training

Our classifiers are implemented in Python, utilizing different machine learning frameworks. The neural network based classifier is implemented on top of the Tensorflow API[6] and the Estimators API[7] in particular. This classifier consist of input-, output- and 3 hidden layers. The hidden layers are fully connected and 16 units are used in each of them. The number of input neurons, corresponds to the size of the input data. The descriptor features 18 keypoints with 2d-coordinates and a confidence value. That sums up to 54 input values and neurons respectively. The number of output neurons is corresponding to the number of shot-size classes. We are using 3 or 5 classes, depending on the classifier we are training. The training loss is optimized by a gradient descent optimizer. The support vector machine based classifier is implemented in Python as well and is based on the machine learning library Scikit-Learn[8]. The SVM implementation in particular is based on LibSVM internally [9]. For parameter optimization, we further use the included grid-search algorithm.

4.4.4 Application Interface

The implementation of this assistance system will be further evaluated within the Personal Film Assistant research project. Therefore, we have certain requirements regarding the performance of this implementation. In order to avoid degrading the user experience, the latency (i.e. time between recording and inference) needs to be as low as possible. The PFA application is running on a mobile platform, but re-implementing and scaling down the pose estimation approach [CSWS17] was out of question because of limited resources. For that reason, the evaluation system has been implemented as a client-server architecture. The PFA application acts as a video streaming device and transfers images via network connection to a workstation. The arriving images are processed on a high-end

[4] https://www.ffmpeg.org/
[5] https://de.mathworks.com/discovery/matlab-gui.html
[6] https://www.tensorflow.org/
[7] https://www.tensorflow.org/guide/estimators
[8] https://scikit-learn.org/
[9] https://www.csie.ntu.edu.tw/~cjlin/libsvm/

desktop machine and the results are then transferred back to the mobile device. While this introduces some additional transfer time for input and output data, the time and performance constraints imposed on the algorithmic part are much lower.

To utilize the shot-size classification approach in the PFA and other future projects, we expose the best performing classifier by an easy to use web-interface. The Python/Flask based service exposes two http routes. The root HTTP-GET route (accessed by a browser accessing the webservice) shows a form to select and upload a single image. By uploading an image to the service, the HTTP-POST route is triggered and the uploaded image is processed by the pose estimator and shot-size classifier. After a short processing time, the respective shot-size class is returned in textual form. To serve concurrent requests, the shot-size classifier is executed in its own thread. This is also required because otherwise the start-up time of about 2 seconds would severely slow down all requests. Synchronized queues are used for communication between the shot-size classification thread and the thread processing incoming HTTP requests.

Algorithm 4.1: Detecting scene changes with FFMPEG

```
ffmpeg -i video.mp4 -vsync 0 -vf select='gt(scene\,0.4)' -f image2 img-%04d.png
```

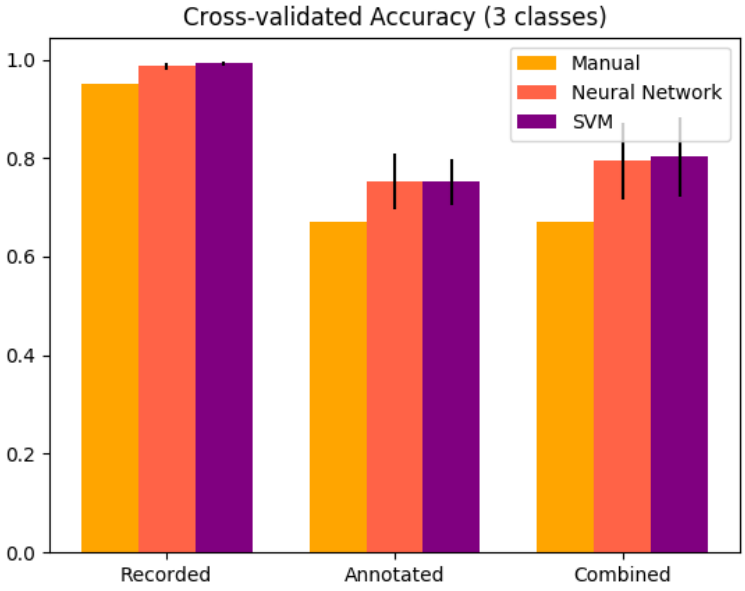
4.5 Evaluation

The implemented classifiers are evaluated on a range of datasets and performance metrics. We are evaluating the manually created decision tree, a neural network (NN) and a support vector machine (SVM). All approaches are applied to the datasets presented in Table 4.1. To show the robustness of the machine learning classifiers (SVM, NN), cross-validation is performed. For each classification method and dataset, 10-fold cross-validation is computed. Figure 4.5 presents cross-validation results for all approaches and used datasets. We can see that the SVM based approach is performing best on all datasets.

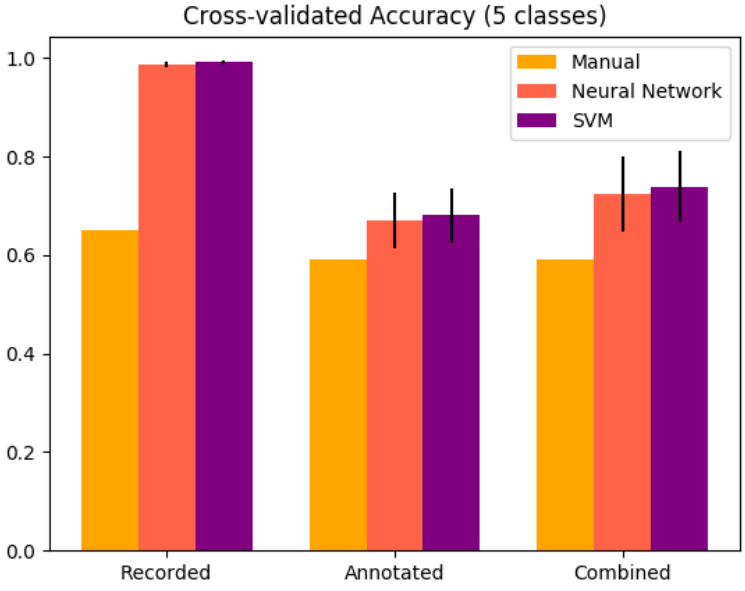
We further analyze the classification results on the presented classifiers and datasets separately by showing the resulting confidence matrices in Figures 4.6 to 4.11. The rows of the matrices contain the ground-truth values. The diagonal contains the correctly classified (true-positive) samples, values in other cells represent misclassifications. Each cell in the matrix shows the accuracy relative to its class and the absolute amount of samples. On the diagonal elements, we additionally show the total number of samples for the respective class. Our first experiments with Dataset A3 (recorded), resulted in extremely accurate classification results as shown in Figure 4.6. The accuracy is well over 90% accuracy across all classes and the machine learning classifiers even performed slightly better in this scenario. However, Dataset 3 (recorded) is covering only a small space of possible poses and may therefore not be suitable for general application. Looking at Figure 4.7 we can see the results for Dataset A5 (recorded). The downsides of the manually created decision tree become visible. Big amounts of samples are confused for neighbouring classes. For the Close-Up class even the majority of samples is misclassified.

In Figures 4.8 and 4.9, we can see the confusion matrices for Datasets B3 and B5 (manually annotated). Samples for these datasets were taken from movie scenes and provide much more diverse poses than Dataset A (recorded). As a result, the classification task is more difficult. As already observed in the previous comparison, the manually created decision tree performs considerably worse than the compared approaches. On Dataset B3 (manually annotated), the classification accuracy across all classes is still 66%. However, the Medium Shot (MS) class was classified correctly in only 46.5% of cases. Numerous confusions between Medium Shot (MS) and Close-Up (CU) samples are occurring. The neural network and support vector machine classifiers (Figure 4.8b and c) show a more consistent performance among classes. The results for Dataset B4 (manually annotated) are visible in Figure 4.9. Misclassifications are happening mainly between neighbouring classes. In all approaches, the distinction between Medium Close-Up and Medium-Shot is problematic.

For Dataset C (combined), the data in Dataset B (manually annotated) was augmented with further training data (Details in 4.3). While the testing approach remains unchanged, the training data has been extended to cover a wider variety of poses, especially in the under populated classes CU and LS. For this combined dataset, the SVM has shown the best performance, in the 3-class, as well as on the 5-class task (See Figure 4.5). As

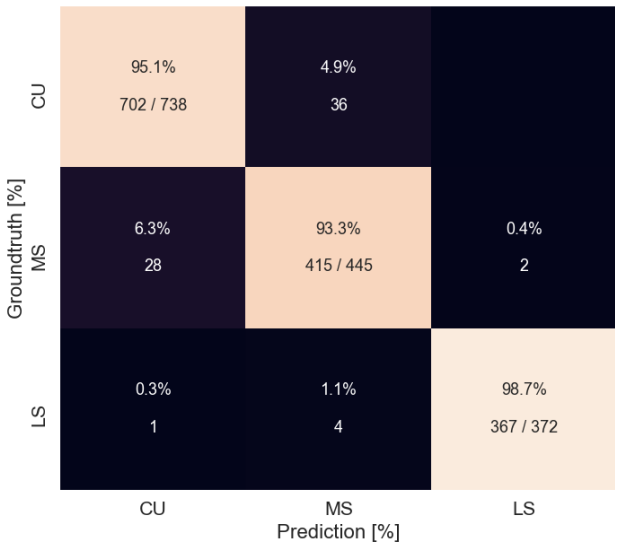


(a) Cross-validation, 3 classes

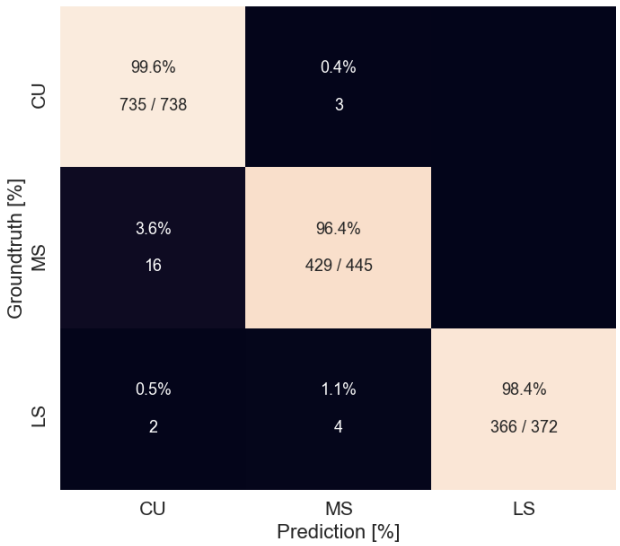


(b) Cross-validation, 5 classes

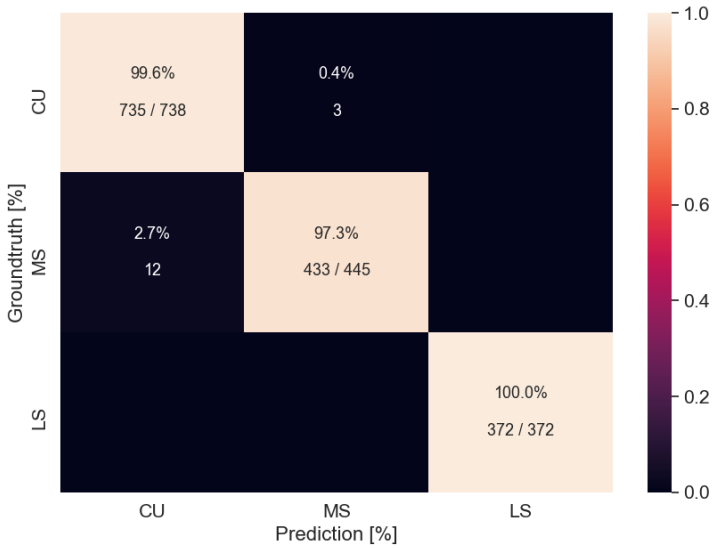
Figure 4.5: Cross-validated accuracy for all classifiers (Manual, Neural Network, SVM) applied to each dataset (Recorded, Annotated, Combined).



(a) Manually Created Decision Tree

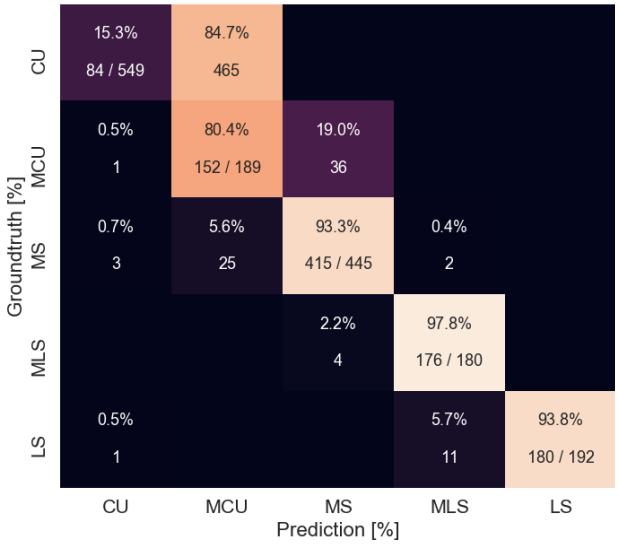


(b) Neural Network

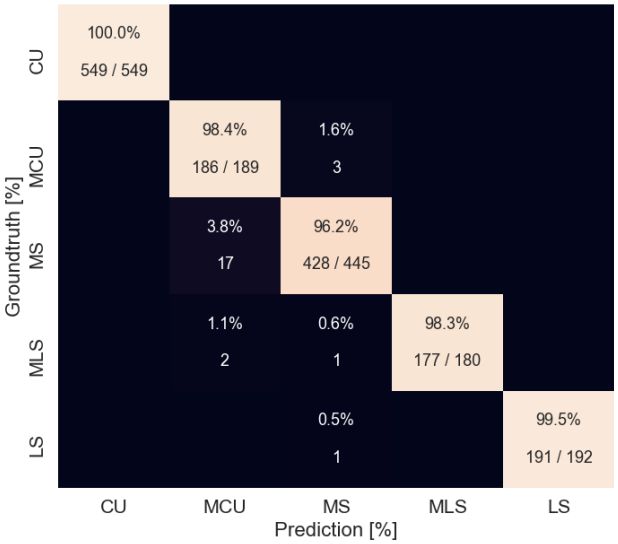


(c) Support Vector Machine

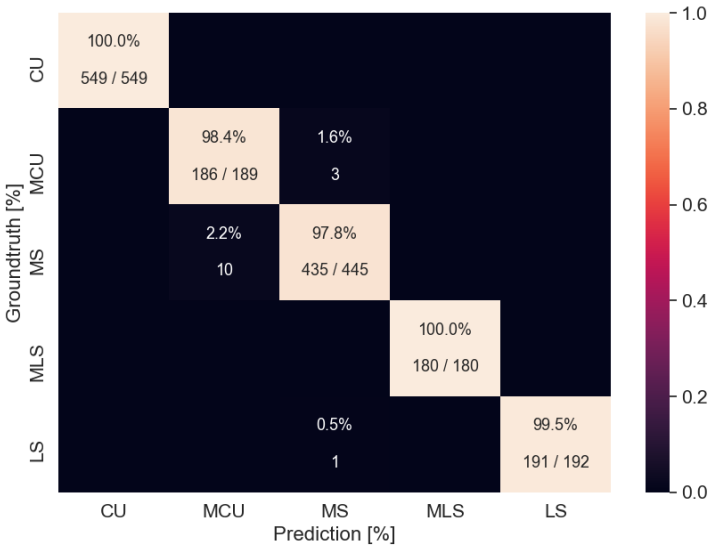
Figure 4.6: Results for Dataset A3 (recorded, 3 classes)



(a) Manually Created Decision Tree

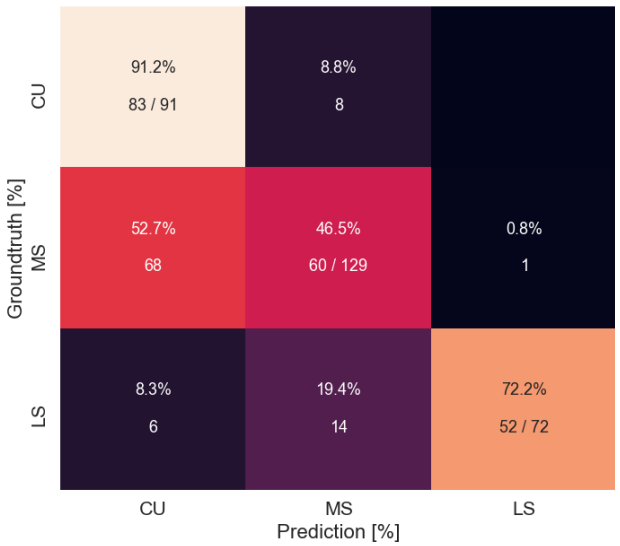


(b) Neural Network

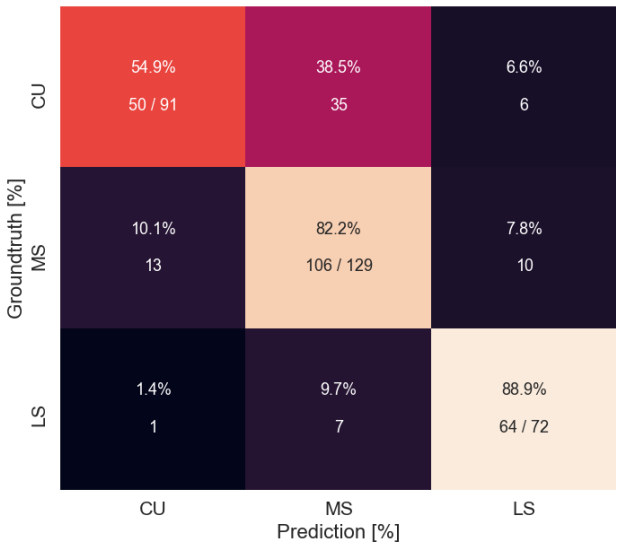


(c) Support Vector Machine

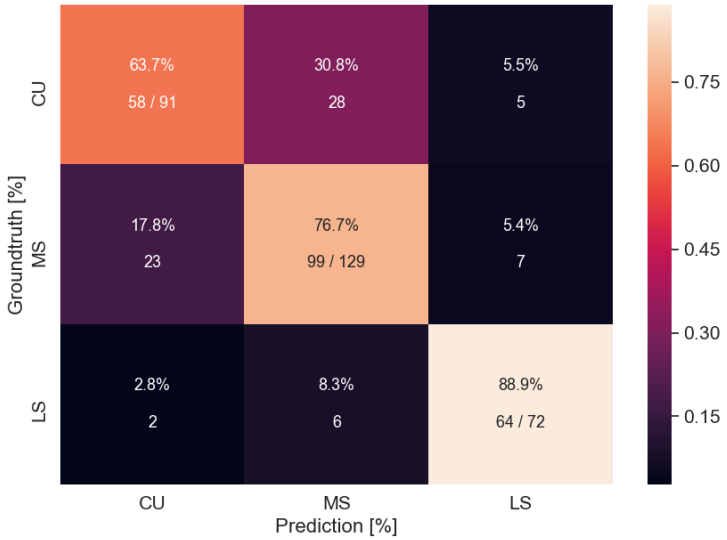
Figure 4.7: Results for Dataset A5 (recorded, 5 classes)



(a) Manually Created Decision Tree

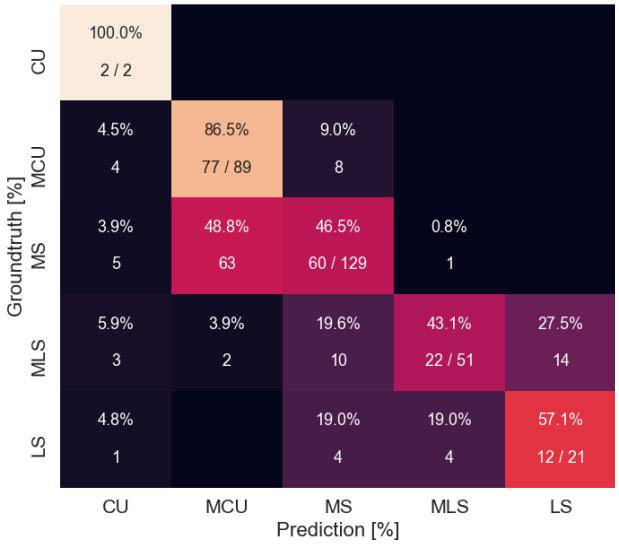


(b) Neural Network

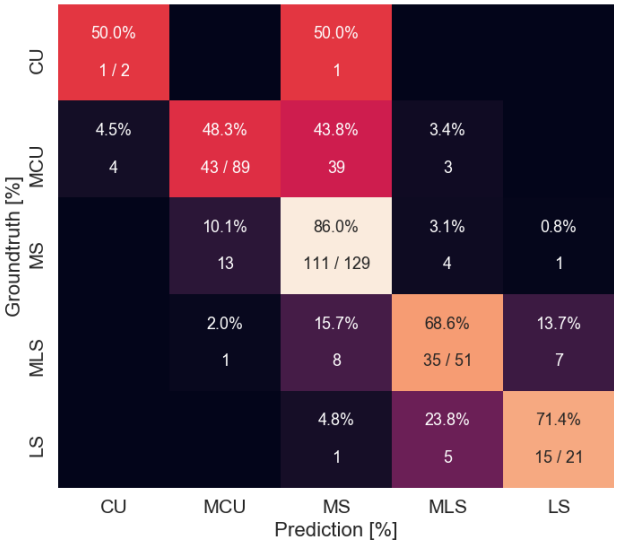


(c) Support Vector Machine

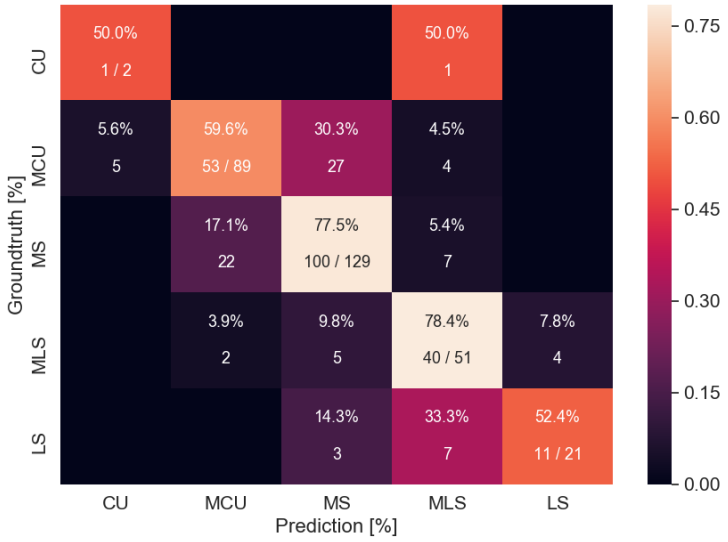
Figure 4.8: Results for Dataset B3 (manually annotated, 3 classes)



(a) Manually Created Decision Tree



(b) Neural Network



(c) Support Vector Machine

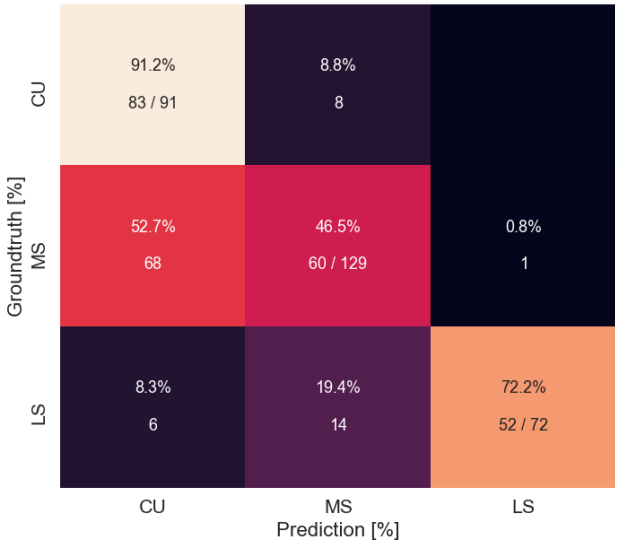
Figure 4.9: Results for Dataset B5 (manually annotated, 5 classes)

a result of the additional training data, we observe less confusion between classes in Figure 4.11b and c than in Figure 4.9b and c.

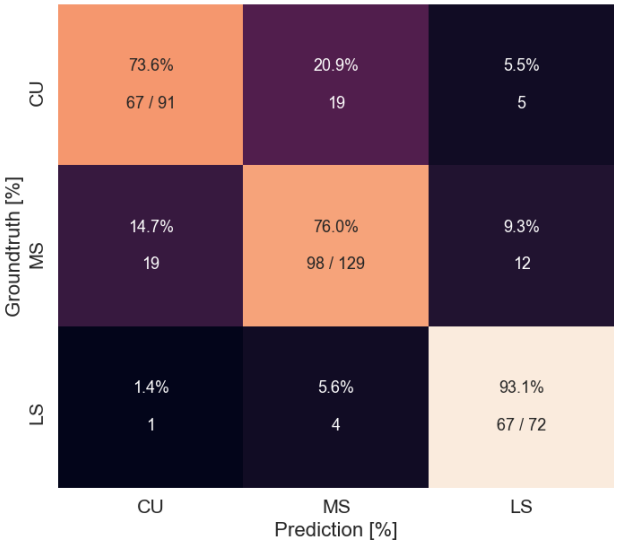
4.5.1 Inference Time

In this section, we analyze how much time is needed from the RGB input image to the final shot-size classification result. The total time is comprised of the time to estimate the pose(s) from the input image and the subsequent execution of the classification algorithm. Since the classifiers only take milliseconds to process a pose, their computation time can be disregarded for further optimization. The time required for pose estimation is, in comparison, several magnitudes larger. Previously conducted experiments in Section 4.5, are carried out with default settings for the Openpose framework, which are set to optimize the detection accuracy of the pose keypoints. However, in an interactive application we need to minimize the latency of the shot-size estimator for an optimal usage experience. A simple way to lower the inference time for Openpose is to adjust the resolution of the image input layer to the neural network. In our experiment, we adjust the network input resolution in 9 steps between 32 and 512 pixels. Figure 4.12a shows the required time to recover the pose from a single input image. We can see that the inference time greatly varies as the resolution further deviates from the default of 368 pixels[10]. For each resolution pose data for Dataset B5 (manually annotated) has been re-estimated and we re-train the best performing (SVM) classifier on the dataset with 3-fold cross validation. The evaluated accuracy measures are shown in Figure 4.12b. Surprisingly, the lowered input resolution has only little impact on the classification performance. Between 64 and 512 especially, the differences are barely noticeable, only when the input resolution is lowered to 32, performance is impaired critically. In conclusion, we are able to reduce the resolution of the network input to a height of 64 pixels, without a significant impact to the classification performance. This optimization lowers the required time for pose estimation to 20% in comparison to the default setting.

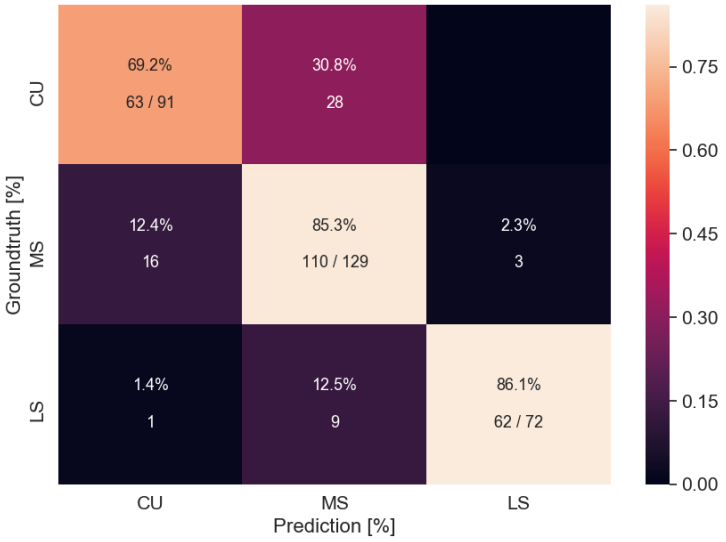
[10] In our experiment we only adjust the height of the input layer, the width is automatically adjusted according to the aspect ratio of the image.



(a) Manually Created Decision Tree

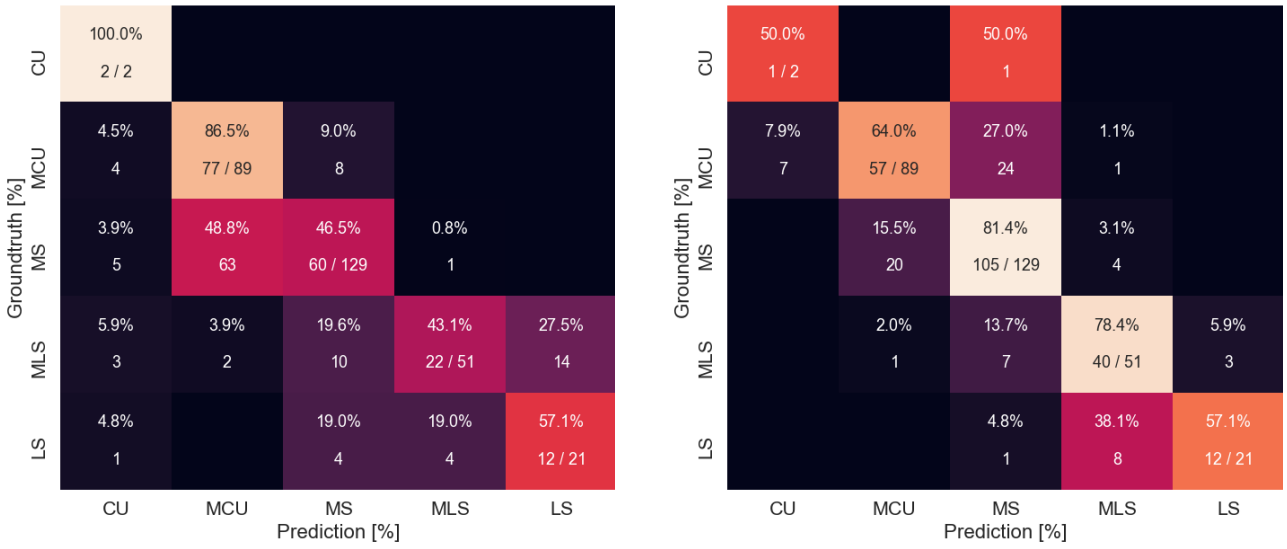


(b) Neural Network



(c) Support Vector Machine

Figure 4.10: Results for Dataset C3 (combined, 3 classes)



(a) Manually Created Decision Tree

(b) Neural Network

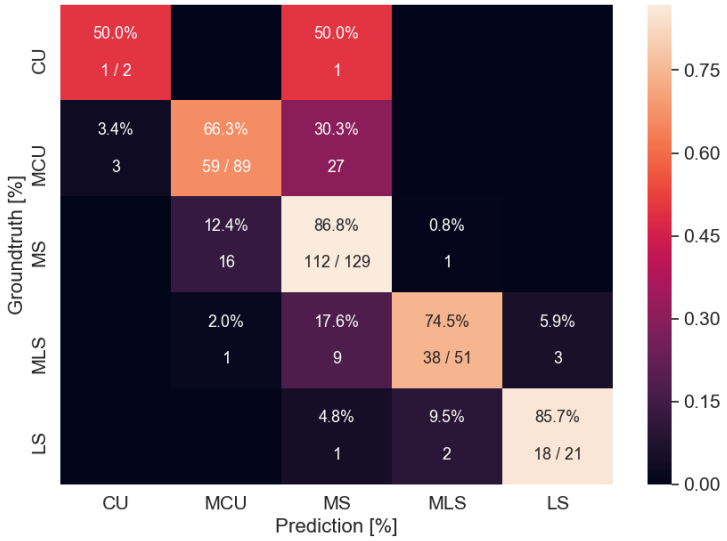
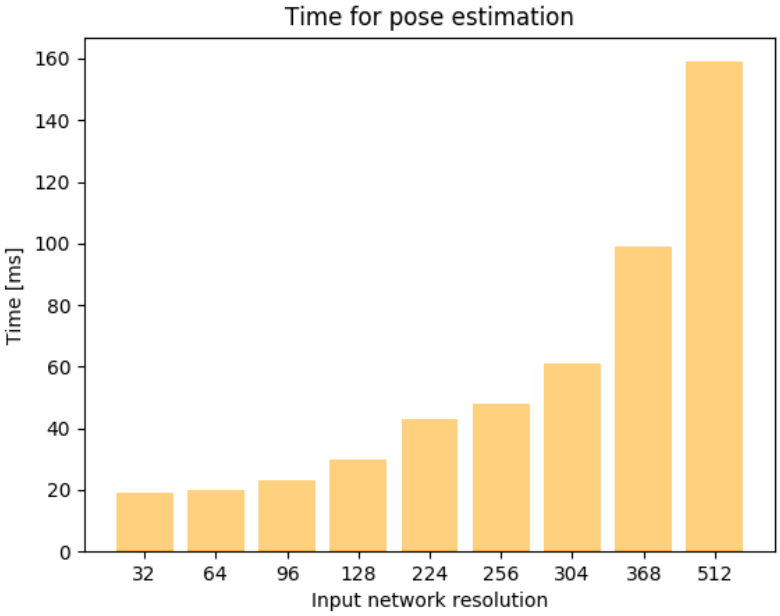
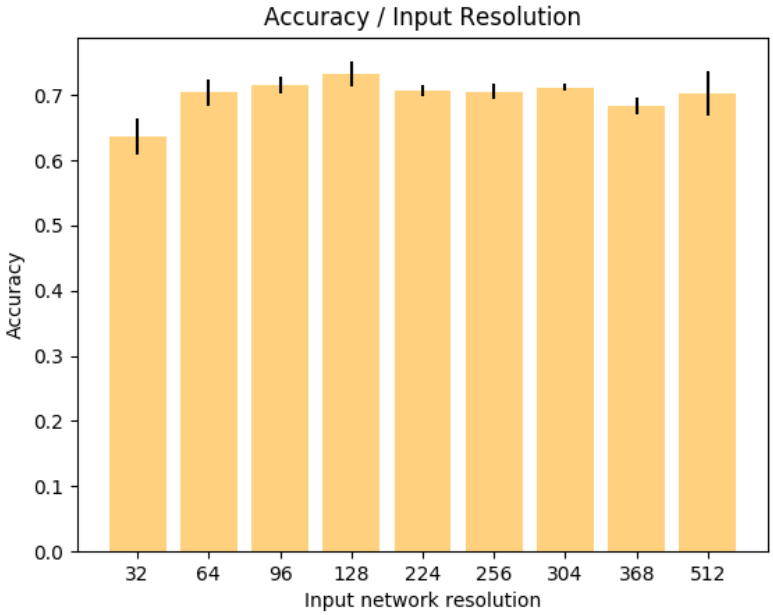


Figure 4.11: Results for Dataset C5 (combined, 5 classes)



(a)



(b)

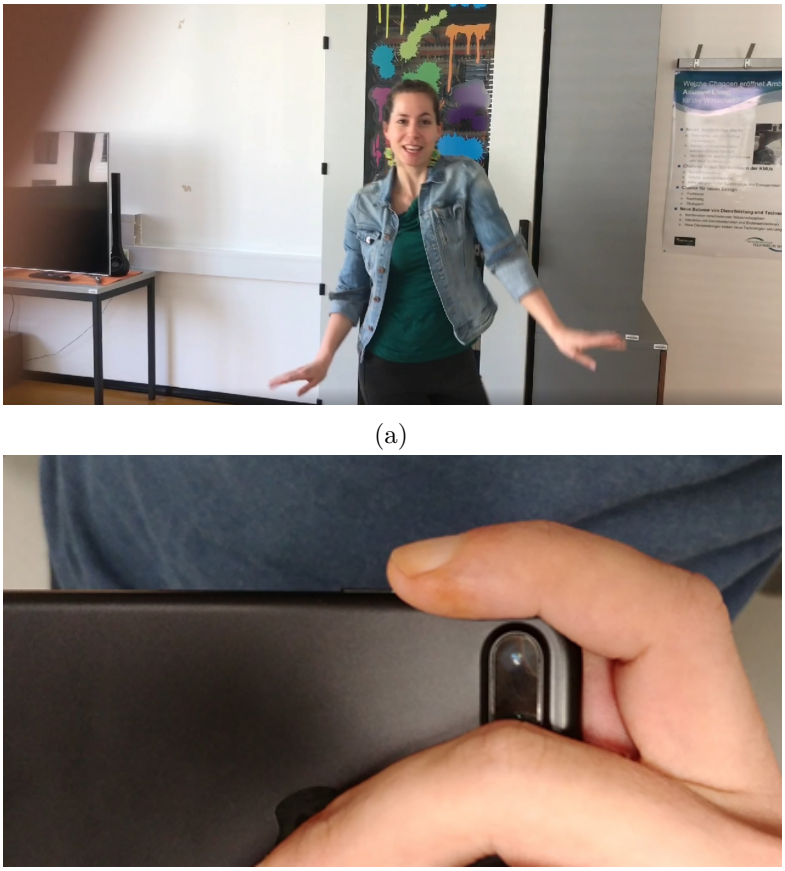
Figure 4.12: Effects of changing input resolution to the pose estimator on the runtime and shot-size classification accuracy a) Increasing the input resolution to the pose estimator increases computation time drastically. b) The input resolution to the pose estimator has seemingly only low influence on the classification accuracy for shot-size.

CHAPTER 5

Finger on Lens Detection

In this chapter, we address a prominent issue present during filming and photography with smartphones. Due to the shape and handling of these devices an easy to be made mistake is to accidentally cover the camera lens partially with hands or fingers. This results in obscured videos or images which make it necessary to retake the respective media. To avoid problems of this nature, we present an algorithm to automatically detect a covered lens. We briefly discuss why this issue arises and show different methods to address the problem based on classical image processing and deep learning.

We now take a look at the components of a modern smartphone. While we are looking at the Apple IPhone 7 Plus in specific, other modern smartphones are similarly constructed. The rear camera of this phone is placed close to the upper edge of the phone's backside, which allows to conveniently operate the phone in portrait mode. The virtually same grip to hold the phone is applied to use the camera in portrait mode, as well as for using other functions. However, as we are used to the format from film and photography operating the camera in landscape mode is a popular choice too. This often results in an unpleasant positioning of the rear camera and may produce imagery as shown by the example in Figure 5.1. Depending on the applied grip, the camera lens is easily covered with index or middle fingers in this constellation.



(a)

(b)

Figure 5.1: Example of a covered camera lens. a) The recorded video frame is partially covered by a finger in the top left corner. b) Due to the construction and handling of smartphones, the lens is easily covered accidentally.

5.1 Approach

In this section, we are presenting different approaches to detect fingers that are partially covering the camera lens. To assess the quality of the involved algorithms as well as for training purposes, a dataset is generated containing images that are affected by occlusions caused by fingers and respective annotations. We implement and evaluate two different methods for per-pixel classification. The first method is based on classical techniques for image processing, combining a skin and blur filter. In a second approach, we generate a convolutional neural network for the image segmentation task. A scaleable architecture is used to comply with resource limitations on mobile devices.

5.1.1 Dataset

For training and evaluation purposes, a suitable dataset is required. So far, the presented problem has not been analyzed in previous work, thus the first step in our work was to generate a dataset. The dataset is required to reflect both image information of filmed scenes, as well as fingers obscuring the camera's view. In other words, our dataset focuses on scenes with partial occlusions. Despite the huge amount of image and video data that are available, images with defects as we require them are hard to find. This lack of publicly available data reflects the fact that defects of this type make the affected images useless for further application. To overcome this issue, we are using un-obstructed video material and create perturbations artificially. To have great variety in background image information, we replay randomly selected internet videos at accelerated speed while taking photographs of the screen. In the recording setup, a 32-inch screen with QHD resolution (2560x1600; 94 PPI) was used. To create the required image obstructions, we took a pictures of the screen with an Apple IPhone 7 plus. While photographing, perturbations in the images are created by partially covering the lens. During recording, we repeatedly change the applied grip to the phone to increase the variety of the generated perturbations. Additionally, we alter the lighting conditions to have the photographers hand being lit from different directions. In total, we took 1138 photos.

To mark the finger positions in the images, all images are annotated manually. We are using a tool called "ImageAnnotation"[1] for that purpose. While a broad range of image annotation tools are available, we picked this one in particular as it allows for a fast annotation process by providing intuitive keyboard shortcuts. The annotations are stored for each image in a separate image file with the same name but different suffix. We only annotate the finger area with a label, the boundary area between finger and background is not annotated separately. Annotation of all 1138 photos could be carried out very efficiently within roughly two hours.

For training and evaluation purposes, the dataset is split into three subsets. The training set contains 938 images and is used for parameter optimization in the classical approach, as well as for training of the machine learning based techniques. Secondly, a validation set containing 100 images is used to assess the classification performance during training. A third set is defined as the test set containing another 100 images. The test set is used to evaluate the final performance of all classifiers, no further tweaking is done after that.

To further extend the amount of available training data, we implement methods to augment the annotated ground-truth data. Data augmentation allows for the creation of altered copies of images in the training set. The intuition is to produce slightly transformed images, which are fed into the network during training to produce a more robust classifier. We make the use of this technique optional, to be able to evaluate its effect separately. A combination of image operations is applied to each training sample. For augmentation, 0 to 2 techniques out of the following list are applied to the image.

[1] `https://lear.inrialpes.fr/people/klaeser/software_image_annotation`

According to the kind of operation, the same transformation is applied to the label image. Values below are defined in the domain [0-255].

- **Color Intensity.** The intensity for all or a single color channel is varied. A uniform probability distribution is used to sample a value between [-10,10] which is added to the color channels. With a probability of 50%, the same change is applied to all channels, otherwise different values are applied to each channel separately.
- **Contrast.** Similar to color intensity, random changes in contrast are applied. The intensity values are stretched by an amount of 50% to 200% and the resulting image is clamped to intensity values of [0,255] again.
- **Hue and Saturation.** The image is transformed to the HSV color space. A value in the domain of [-20,20] is added to hue and saturation, before the image is converted back to RGB space.
- **Flipping.** Images are flipped around the horizontal or vertical axis. With a probability of 50% an image is flipped left to right and by a 20% chance flip up to down is applied.
- **Cropping.** The images are cropped randomly by 0% to 30% in both width and height. The result is re-scaled to the original size.

5.1.2 Classical Approach

In this first approach, a detector based on conventional image processing techniques is constructed. Covering fingers have two prominent features. While the camera is focused on objects much farther away, the fingers are close and therefore out of focus. Thus, the fingers appear heavily blurred in the image. Secondly, we exploit color as a feature. The color of a finger matches the color of skin in general. However, depending on the lighting direction, the color tones can be vastly different and certainly any other persons or objects of similar color may produce false-positives. We detect and combine these features into a robust classification technique.

For separating pixels into skin and non-skin classes, we make use of the work by Rahman et al. [NKJ+06]. The color values are examined by 3 different rules, whereas each rule operates in a different color space. Equation 5.1 shows the first rule, where different thresholds are applied in RGB color space to describe skin color under flashlight and daylight respectively [KPS03]. In Equation 5.2 the YCrCb color space is analyzed. The equations describe a convex sub-space in the CrCb plane, which contains all color values representing skin. Compared to the work presented by Rahman et al., we formulate the third plane differently. While their paper proposes a slope of -4.5652 we have corrected that value to 0.92, which is also visually corresponding to Figure 6 in their paper. The third rule, described in Equation 5.3, constrains the color hues in HSV space.

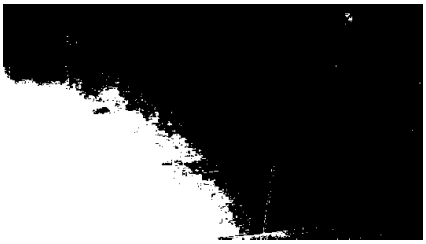
$$
\begin{aligned}
&(R > 95 \wedge G > 40 \wedge B > 20 \wedge \\
&(max(R,G,B) - min(R,G,B)) > 15 \wedge \\
&|R - G| > 15 \wedge R > G \wedge R > B) \\
&\vee \\
&(R > 220 \wedge G > 210 \wedge B > 170 \wedge \\
&|R - G| \leq 15 \wedge R > B \wedge G > B)
\end{aligned} \tag{5.1}
$$

$$
\begin{aligned}
Cr &\leq 1.58 \cdot Cb + 20 \\
Cr &\geq 0.34 \cdot Cb + 76.20 \\
Cr &\geq -0.92 \cdot Cb + 234.56 \\
Cr &\leq -1.15 \cdot Cb + 301.75 \\
Cr &\leq -2.28 \cdot Cb + 432.85
\end{aligned} \tag{5.2}
$$

$$
H < 25 \wedge H > 230 \tag{5.3}
$$

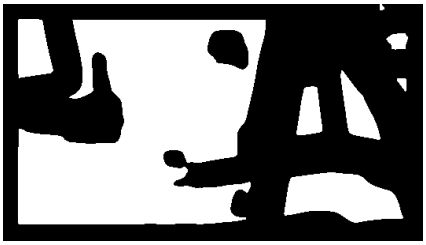
Furthermore, a metric is required to estimate image blur. A basic intention for detecting blurred image regions, is to assume that a focused image would contain more sharp image structures like edges. Affected by blur, these structures are vanishing. In a study comparing a range of algorithms for blur estimation in images, Pertuz et al. [PPG13] analyze edge detectors based on first and second derivatives. It was found that methods based on the Laplace operator tend to perform best under normal conditions (no added noise, changed contrast or saturation). We choose a method based on the variance of the Laplacian [PPCCMFV00] for its effectiveness and simple implementation. While the original approach describes a method to estimate blur of a full image, we compute the variance with respect to a local patch. A patch is computed for each pixel in the image. To compute the blurriness for a pixel, we first apply a convolution with a discrete Laplace kernel. Then the standard deviation is estimated for each pixel with respect to a local, circular area. The radius of this area is defined as a parameter that needs to be set by the user. After computing the pixel-wise standard deviation, we apply a binary threshold to produce a binary mask defining which pixels are considered blurry. Both the skin color filter and the detection of blurred image regions produce binary images. We determine a pixel to be part of an occluding finger if both conditions are met. To determine the optimal values for the hyperparameters we use a grid search. We test in an exhaustive manner for patch radii r between 3 and 55 pixels, and thresholds for the standard deviation from 0 to 0.1. At a resolution of 214x120 pixels for the input images, we found that $r = 11$ and a threshold of 0.018 to be optimal in terms of maximizing the Intersection over Union (IoU) metric on the training set. Since tests on higher image resolutions did not result in a better classification, we kept this relatively low resolution.

(a) Input image



(b) Skin filter mask

(c) Local variance map



(d) Local variance map after thresholding

(e) Combined mask of the skin-filter (b) and thresholded local variance map from (d).

Figure 5.2: a) In the lower left corner of the shown input image the shade of an obscuring finger is clearly visible. b) Is showing the color-based skin filtering. c) The local variance map shows high variances as bright pixels. Low variance/homogeneous areas appear darker. d) We threshold the local variance map to receive a binary image classifying into homogeneous and non-homogeneous areas. e) Finally, the binary masks of the skin filter and the variance map with applied threshold are combined. Pixels classified as skin and having low variance are contributing to the output mask.

5.1.3 Semantic Segmentation with Deep Learning

In a second approach to detecting fingers covering the camera lens, we utilize deep learning techniques. A neural network takes one recorded image at a time and computes a per pixel prediction of occluded and non-occluded areas. While the classical approach mostly relies on manually defined rules, this technique is based on representation learning. Before applying the network to novel data, the network is first trained on a dataset. We show the network pairs of image data and their corresponding expected output segmentation. With this information, the training procedure is able to automatically configure the parameters of the network, to solve the given image segmentation task. In the following sections, the employed neural network architecture and the training procedure are described in detail. With the chosen architecture, we particularly aim for a lightweight approach, which is suitable for the use on mobile platforms like smartphones.

Neural Network Architecture

We construct the used neural network as a typical encoder-decoder architecture. To decode the input image into image features we use Mobilenets by Howard et al. [HZC+17]. The decoding part of the network is implemented according to the work by Long et al. on Fully Convolutional Neural Networks (FCN) for image segmentation [LSD15].

Conventional CNN architectures use repeated combinations of layers with small convolution kernels, following an activation function. Differently, Mobilenets use Depthwise Separable Convolutions instead of regular convolutions. First, a depth-wise convolution is applied. While a regular 2D convolution extends the filter size over the whole depth of the tensor[2], a depth-wise convolution applies the same kernel to each depth layer. As a result, the output tensor of a depth-wise convolution has the same size as its input (assuming padding for the borders). Next, a regular 1x1 convolution is applied to the resulting tensor. The kernel extends over the depth of the input tensor, resulting in as many parameters for the kernel as the input tensor is deep. Different kernel parameters are used for each 2D position. This would result in a tensor of the same width and height, but the depth being squashed. The procedure is repeated for as many output layers as required. As reported in [HZC+17], this procedure greatly reduces the number of parameters to train and the number of required multiply-add operations respectively. Figure 5.3 (top) lists the used layers. The shown network is a decapitated version of the original Mobilenet. The architecture is identical to [HZC+17] up to the fully connected layer for class prediction, which is replaced by the following FCN in our case. The parameter D defines the output depth of the different layers and is multiplied by the hyper-parameter $\alpha = \{0.25, 0.5, 0.75, 1.0\}$ to scale the network.

Fully Convolutional Neural Networks by Long et al. extend previous CNN architectures for image classification by interpreting the fully connected layers, which are typically used for one-hot encoded class output as pixels in an image segmentation task. As the performance of fully connected layers for this purpose is underwhelming and spatial

[2] 2D in that sense refers to the dimensions of the output, not to the size of the kernel

information is disregarded. We follow the suggestions by Long et al. to replace the fully connected layer by a point-wise, i.e. 1x1, convolution with a depth corresponding to the number of classes (2 in our case) and a de-convolution [DV16] to up-sample the resulting tensor to the input image size with a depth according to the chosen number of classes. We finally determine the output class for each pixel by choosing the class with highest activation.

Training

As typically done, the network is trained by backpropagation. A loss function is used to calculate the prediction error of the network with respect to input and ground-truth data. In our case, cross-entropy is used, which is a state of the art choice for loss function in deep learning. Backpropagation is used to determine the gradients of the loss function with respect to the parameters in the network. Stochastic gradient descent (in specific ADAM [KB14]) is used to optimize the network parameters.

To be able to analyze the progression during training, we train the network in all our experiments for 100.000 iterations. The number of iterations has been determined empirically, as a number where no further training progress would be observable. In each iteration not one but a batch of images is used for training, and we set a batch size of 32 images to fit our GPU memory limitations. We introduce randomness into the training data by randomly permutating the image and annotation pairs. Batches of 32 images are drawn from the permutation until the end of the dataset is reached, then a new permuatation is generated and the process is repeated. Every 100 iterations we validate the intermediate training results. A validation set of 100 images is used to determine the descriptive quality of the network on unseen data.

Furthermore, we apply the principle of transfer learning in our training process. Transfer learning allows for the transfer of prior learned information to a new network. This approach is (at least partially) usable across different disciplines for deep neural networks. The Mobilenets architecture, as developed by Howard et al. [HZC+17], was primarily trained and used for object classification. As described previously, we apply the same architecture up to the pooling, fully-connected and softmax layers. The convolutional and depth-separable convolution layers are directly re-used in the applied image segmentation architecture. For training, we use pre-trained networks for the task of image classification trained on Imagenet [RDS+15], which contains 1.2 million training samples. Utilizing the pre-trained layers, we can reduce training time and over-fitting of our network.

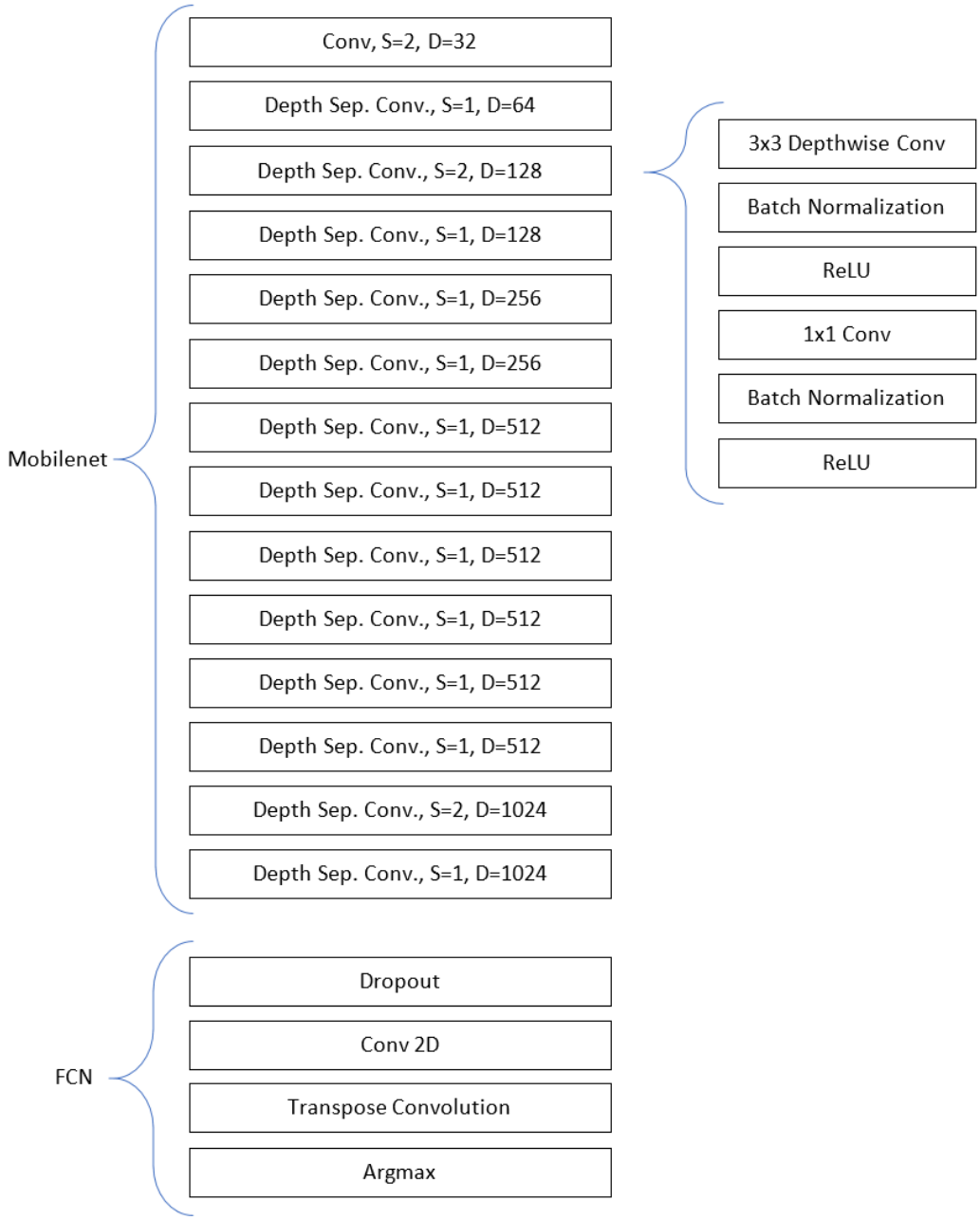


Figure 5.3: Deep learning architecture combining Mobilenet (encoder) and a Fully Convolutional Network (decoder) for semantic segmentation.

```
function mask = fol_mask(image, thres, r)
    fol_gray = rgb2gray(im2double(image));
    h = [0 -1 0; -1 4 -1; 0 -1 0];
    laplace_resp = imfilter(gpuArray(fol_gray), h, 'replicate');
    local_std = stdfilt( ...
                gpuArray(laplace_resp), ...
                strel('disk', r, 4).Neighborhood);
    vol = local_std < thres;
    mask = vol & skinfilter(image);
end
```

Listing 5.1: Matlab implementation of the Finger on Lens detection.

5.2 Implementation

In this section, we discuss the implementation details for the proposed approaches.

5.2.1 Classical Approach

The classical approach comprises of two main parts. The task of the skin filter is to detect pixels similar to human skin, while the second part of the algorithm determines the blurriness in the image. We implement the whole approach in Matlab, as it provides simple access to the required image processing tools. Instead of computing the variance of Laplacian, we compute the standard deviation of the Laplace response in Listing 5.1. First, we scale down the images to a resolution of 214x120 pixels and convert the images to grayscale. The discrete laplace kernel h is applied to the grayscale image by convolution through `imfilter`. We use a `gpuArray` to enforce the GPU support available for the `imfilter` function. This allows for a noticeable speed-up. We further use `stdfilt` to compute the standard deviation over the laplace filtered image. The effective area for `stdfilt` is defined as a disk-shaped area of radius r, which is one of the hyperparameters for this method. Finally, the threshold *thres* (second hyperparameter) is applied to the standard deviation image and the masks are combined by a binary AND operation. The computation of the skinfilter, shown in Listing 5.2, is implemented according to the rules in Equations 5.1, 5.2, 5.3. All applied operations are executed element-wise on all pixels in the masks.

```
function [ o ] = skinfilter( img )
    r1_mask = r1(img);
    r2_mask = r2(img);
    r3_mask = r3(img);

    o = r1_mask & r2_mask & r3_mask;
end

function [ o ] = r1( img )

    img_min = min(img, [], 3);
    img_max = max(img, [], 3);

    e1 = ...
        img(:,:,1) > 95 & img(:,:,2) > 40 & img(:,:,3) > 20 & ...
        (img_max - img_min) > 15 & abs(img(:,:,1)-img(:,:,2))>15 & ...
        (img(:,:,1)>img(:,:,2)) & (img(:,:,1)>img(:,:,3));

    e2 = ...
        (img(:,:,1)>220) & (img(:,:,2)>210) & (img(:,:,3)>170) & ...
        (abs(img(:,:,1)-img(:,:,2))<=15) & (img(:,:,1)>img(:,:,3)) & ...
        (img(:,:,2)>img(:,:,3));

    o = e1 | e2;
end

function masked = r2(img)
    img = double(img);
    Y = 0.299 * img(:,:,1) + 0.587 * img(:,:,2) + 0.114 * img(:,:,3);
    Cb = (img(:,:,3) - Y) * 0.564 + 128;
    Cr = (img(:,:,1) - Y) * 0.713 + 128;

    e3 = Cr <= 1.5862*Cb+20;
    e4 = Cr >= 0.3448*Cb+76.2069;
    e5 = Cr >= -0.92*Cb+234.5652;
    e6 = Cr <= -1.15*Cb+301.75;
    e7 = Cr <= -2.2857*Cb+432.85;
    masked = e3 & e4 & e5 & e6 & e7;
end

function o = r3(img)
    hsv = rgb2hsv(img) * 255;
    o = hsv(:,:,1) < 25 | hsv(:,:,1) > 230;
end
```

Listing 5.2: Matlab implementation of the skinfilter used for the Finger on Lens detection.

5.2.2 Deep Learning Approach

We implement the Mobilenets/FCN approach with help of the machine learning framework Tensorflow (v1.8) [ABC+16] and the Python (v3.5) programming language. The Mobilenet architecture is based on an implementation provided by the authors of Tensorflow[3]. Further ideas for the program structure are adapted from an available FCN implementation[4].

At the beginning of the program, the structure of the implemented neural network is set up in a Tensorflow Graph. The layers of this graph are defined according to Figure 5.3. The Tensorflow inherent methods for softmax cross entropy[5] and ADAM optimizer [6] are used to define the training operation.

The program is structured into different modes of operation for training and testing. In training mode, first the training and validation datasets are loaded and the last saved checkpoint is restored. Saving checkpoints allows us to quit and resume the training later. If no checkpoint exists yet, pre-trained parameters are loaded for the Mobilenet layers. The Tensorflow authors provide a range of pre-trained Mobilenets in differently scaled configurations [7]. We expose commandline parameters, namely `mobilenet_scale` and `mobilenet_res` to define the scaling and input resolution for the Mobilenet layers. According to the chosen parameters, the correct pre-trained model is downloaded and applied during runtime. Valid input resolutions are 128 and 192 pixels (images are required to be square) and scalings of 0.25, 0.5, 0.75 and 1.0 respectively.

To observe the training progress, different ways of monitoring have been implemented. The Tensorflow framework provides the concept of Summaries to serialize intermediate results to log files. During training, the log files can be parsed simultaneously by Tensorboard, which provides a GUI to examine the summaries and graph structure. Every 50 training iterations summaries are evaluated. The summaries store the current training loss, accuracy and IOU on the current training batch, as well as 5 image samples containing RGB input, ground truth and annotation prediction triplets. Every 100 iteration the validation step also stores the IOU over the validation set, to report the current performance on unseen data. The stored data can be visually explored during and after training within Tensorboard.

5.3 Evaluation and Results

We evaluate the implemented approaches by measuring classification performance and resource consumption on average over the test set. Furthermore, additional experiments

[3] https://github.com/tensorflow/models/tree/master/research/slim/nets

[4] https://github.com/vietdoan/fcn-mobilenet

[5] https://www.tensorflow.org/api_docs/python/tf/nn/sparse_softmax_cross_entropy_with_logits

[6] https://www.tensorflow.org/api_docs/python/tf/train/AdamOptimizer

[7] https://github.com/tensorflow/models/blob/master/research/slim/nets/mobilenet_v1.md#pre-trained-models

are conducted to optimize and compare the results. All performance tests are conducted on consumer desktop hardware. We use a system with an Intel Xeon 1231v3 processor, an Nvidia GTX 1080 graphics card and 32GB RAM. In the following, we introduce the used performance metrics and present different evaluations.

5.3.1 Evaluation Metrics

To assess the classification performance, we compare the approaches using different metrics. Besides the typical metrics Accuracy, Precision and Recall we use the Intersection over Union for our evaluations. The Intersection over Union (IOU) is a popular choice to evaluate image segmentation techniques. Intuitively the IOU can be explained by comparing box annotations for a localization task. To determine the quality of such an approach, one could compare the consensus between the annotated and the detected box. If the boxes are fully overlapping, the detection is perfect, if no overlap is occurring the algorithm does not perform well at all. To normalize the IOU, the intersection is divided by the union of the annotation and detection. This approach is a popular choice to evaluate localization tasks, especially when the object areas are small compared to the full image. Evaluation of such an approach by accuracy (Equation 5.4) yields a skewed performance towards True Negatives. While precision (Equation 5.6) is more expressive for such cases than accuracy, it still disregards False Negatives. While we are using a localization task for explanation, the same principle can be applied to a segmentation task. In the following, we use the abbreviations True Positive (TP), True Negative (TN), False Positive (FP), False Negative (FN).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.4}$$

$$Precision = \frac{TP}{TP + FP} \tag{5.5}$$

$$Recall = \frac{TP}{TP + FN} \tag{5.6}$$

$$Intersection\,over\,Union = \frac{TP}{TP + FP + FN} \tag{5.7}$$

5.3.2 Transfer Learning

In this experiment, we verify the effect of transfer learning on our application. The applied neural network architecture is combined from a Mobilenet and a Fully Convolutional Neural Network. In the following, we test our architecture with and without transfer learning for the encoder (Mobilenet) part of the network. For this evaluation we parameterize the model with a scaling factor $\alpha = 1.0$ and apply no augmentation to

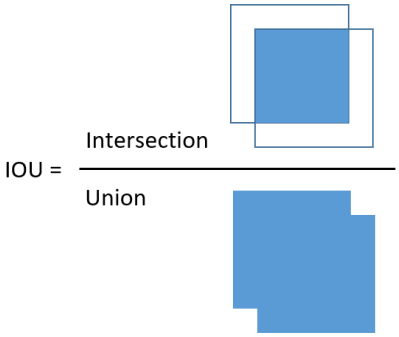


Figure 5.4: Graphical interpretation of the Intersection over Union metric. In a localization task, the IoU may be interpreted as the ratio of the overlap between prediction and annotation normalized by the union of both. The same principle is applicable to the dense prediction and annotation of image segmentation tasks.

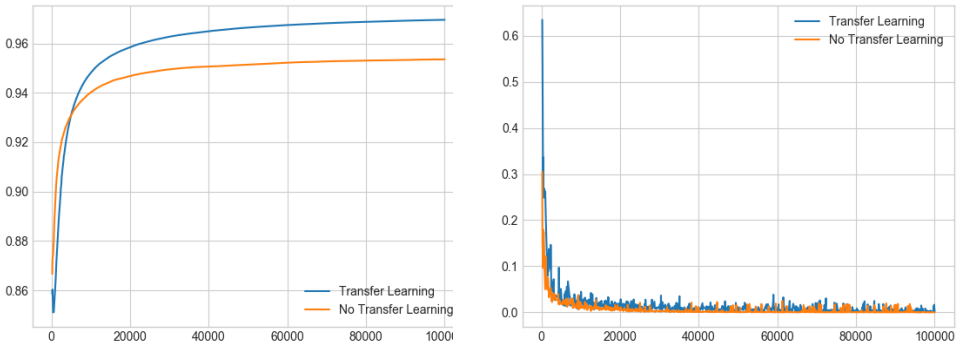
the training data. Figure 5.5 presents the differences between applying and not applying transfer learning during training.

In Figure 5.5b and c it may seem like the approach using no transfer learning is slightly superior, as the Loss and IOU computed on the current training batch appear to be lower. However, Figure 5.5a shows that the computed accuracy converges to a lower value without transfer learning. Figure 5.5d also confirms this assumption by showing a similar behavior for the IOU computed over the validation set. We can therefore conclude that transfer learning helps in our case to prevent over-fitting on the training data and create a network with better generalization capabilities towards unseen data.

5.3.3 Mobilenet/FCN Configurations

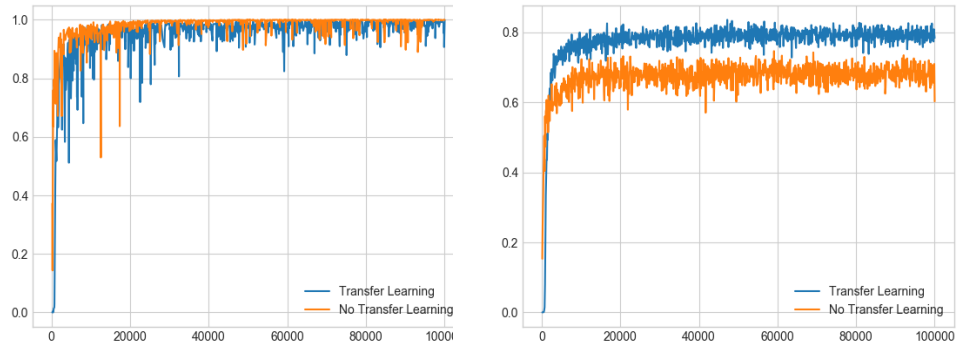
In this section, different configurations of our Mobilenet/FCN approach are compared. Mobilenet is a flexible architecture which can be scaled easily by a hyperparameter controlling the size of tensors [HZC+17]. Additional to that, we also consider removing the last convolution layer from the Mobilenet architecture to save more parameters and computation time. Furthermore, we evaluate the impact of image augmentation on the resulting networks. Table 5.1 summarizes the results. The upper part of the table shows configurations where the last Mobilenet layer was removed, the bottom part of the table uses the original Mobilenet structure. The first two columns show the IOU for training with and without augmentation respectively. Column three determines the multiplier for the number of parameters in each Mobilenet layer.

We measure the required computational effort in FLOPs (Floating point operations). This is only a theoretical estimation of the required effort as this method disregards synchronization, utilization and other factors that will slow down computation further. However, due to the high level of parallelization possible on the used desktop GPU, counting FLOPs allows for a portable estimate.



(a) Accuracy

(b) Loss



(c) IOU (Current Training Batch)

(d) IOU (Validation Set)

Figure 5.5: Impact of Transfer Learning on the training process of Mobilenet/FCN during 100.000 iterations.

The image augmentation strategy improves the result in most configurations. A slight increase of the IOU is notable for most (6 out of 8) configurations by using augmentation during training.

5.3.4 Comparison of Approaches

We compare the approaches in terms of performance and speed. Furthermore, we add a state of the art deep learning technique for image segmentation, namely Deeplab [CPK+18], to the comparison. Deeplab gives us a base-line of what is possible in a less resource constrained environment. It is available from the Tensorflow Research Repository[8]. We adapt our dataset to a compatible format, our augmentation techniques remain unused in this case.

[8] https://github.com/tensorflow/models/tree/master/research/deeplab

Without Last Layer

IOU			Inference Time		
Aug	No Aug	Scale	GPU [ms]	CPU [ms]	Flops
0.775	0.781	0.25	5.6	53	425K
0.806	0.792	0.5	5.7	87	1.24M
0.807	0.796	0.75	5.7	116	2.58M
0.814	0.797	1.0	5.6	107	4.44M

With Last Layer

IOU			Inference Time		
Aug	No Aug	Scale	GPU [ms]	CPU [ms]	Flops
0.777	0.764	0.25	6.0	50	564K
0.793	0.800	0.5	6.0	90	1.78M
0.801	0.786	0.75	6.0	119	3.78M
0.810	0.769	1.0	6.0	113	6.57M

Table 5.1: Evaluation of different configurations for the Mobilenet/FCN approach. We analyze the IOU and inference time with and without image augmentation (Aug/No Aug) during training, different Mobilenet scales and removal of the last Mobilenet layer.

	IOU	Accuracy	Precision	Recall	Inference Time	Flops
Classical	0.554	0.924	0.748	0.682	16.1ms[9]	51M
Mobilenet/FCN	0.814	0.972	0.905	0.890	6.1ms	4.44M
Deeplab	0.869	0.880	0.901	0.960	31ms	17.6G

Table 5.2: Performance and inference time (GPU) of the evaluated approaches. For the Mobilenet/FCN variant, the best performing configuration according to IOU was selected (α=1.0, augmentation=true, lastlayer=false).

The evaluation of Deeplab shows very good performance. We measure an IOU of 0.869 on the testset. However, the required computation time is roughly 6 times longer in comparison to Mobilenet/FCN. The implemented combination of Mobilenet and FCN is a much lighter alternative with similar accuracy but fewer operations required. For our application, Deeplab seems to perform only slightly better than the Mobilenet/FCN approach, while requiring roughly 4000 times more floating point operations (See Table 5.2).

In a visual comparison of the approaches, we can make observations on the qualitative results for the different techniques. The classical approach shows the least convincing results. Due to the color-based skin filter, the resulting mask can be very brittle, the detection borders are highly inaccurate and may even contain holes. Large homogeneous areas (like in Figure 5.6 line 3) are easily mistaken for a blurred area. The direction of light can make fingers close to the lens appear in quite diverse color shades, which may lead to rejection by the skinfilter (as shown in Figure 5.6 line 5).

[9]Matlab implementation; partially GPU accelerated

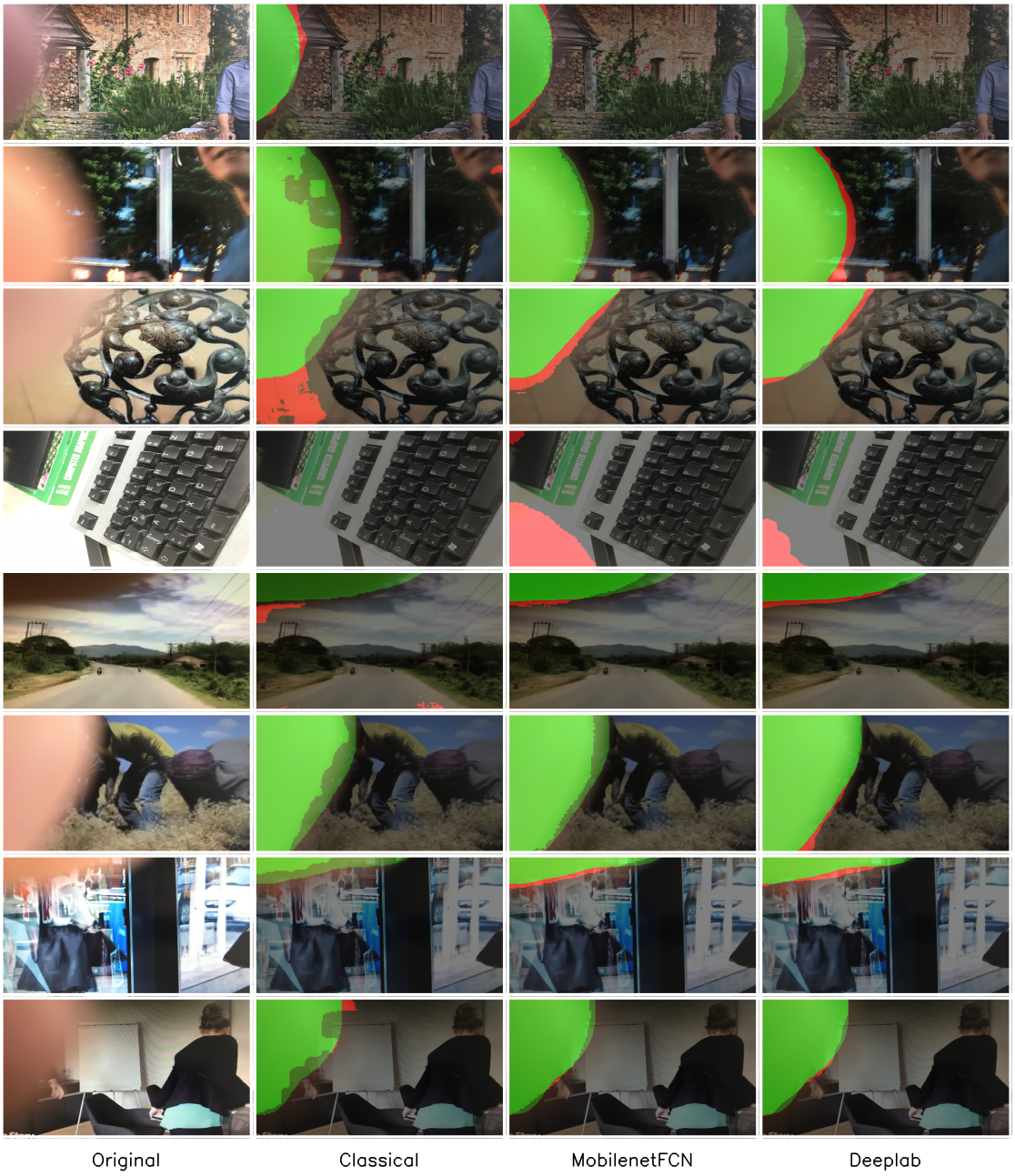


Figure 5.6: Visual comparison of the original image, classical approach, Mobilenet/FCN and Deeplab. True-Positive values are encoded in green, False-Negatives are shown in light-green. False-Negatives are colored red. True-Negatives remain uncolored.

Mobilenet/FCN is more faithful to the ground-truth annotations. The border areas between ground-truth and prediction are very similar. Complex lighting conditions seem to be handled well by both deep learning based techniques. Deeplab is, as we could expect from the required computational effort, showing the most accurate results. No singled out pixels or holes are visible in the prediction.

With respect to the whole testset, the deep learning techniques sometimes tend to produce False-Positives in areas where no true samples are present at all. An example of that can be seen in Figure 5.6 line 5. We manually identify these areas and count them in the testset (containing a total of 100 images). For deeplab, we identify 5 such cases, the results for Mobilenet/FCN show 11 vastly incorrect results. In conclusion, the Mobilenet/FCN approach is a good trade-off between computational effort and classification performance. Deeplab outperformed Mobilenet/FCN in terms of classification performance, but is vastly more complex.

CHAPTER 6

Conclusion and Future Work

In this thesis, we have shown different assistive technologies to help users with video creation on mobile platforms. The work is embedded into the concept of the Personal Film Assistant research project, which provides a storyboard-driven approach for video production. This allows us to assistively intervene the recording process of single video shots. We identify and present three approaches to help an amateur user with video creation, namely: Video Stabilization, Shot-Type Classification and Finger on Lens Detection.

The stability of a video is an important aspect in video production. Shaky hands and other externally caused motion perturbations are transferred to the camera motion and cause an unpleasant viewing experience. Low-pass filtering of the camera path is a common approach which we adpopt in our work. In the context of the encompassing video production concept, a cinematographically inspired approach is investigated as well. We have implemented and compared the two approaches based on quantitative metrics for stability and distortion of the image information. While the cinematographically inspired approach attempts to create linear and parabolic path segments, the low-pass filter is realized by Gaussian smoothing. We found that extreme camera movements are compensated well by both approaches, with the stability and distortion metrics showing a comparable result for both techniques. However, the required computation time vastly differs in favor of Gaussian smoothing.

The shot-type is a cinematographic concept to describe the distance and framing of a scene or actor. For an amateur videographer, who is not familiar with this concept, filming a scene featuring a certain shot-type may be difficult. We therefore propose an approach to automatically infer the shot-type from an image. We target specific classes of shot-sizes, which are a subset of the more general shot-type concept presented in literature. To identify the shot-size, we first estimate the poses of the actors in a scene and select the main actor. The pose keypoints are classified by a decision tree which is created manually, a support vector machine and a fully connected neural network.

We compare the approaches on different dataset configurations, whereas one part of the data contains recorded poses at fixed distances and the second part comprises manually annotated movie scenes. We found that the machine learning classifiers outperform the manually created decision tree in all dataset configurations, while the support vector machine has shown the best performance among the machine learning techniques. We have evaluated the approaches to distinguish shot-sizes of three to five different classes. When differentiating between five classes, confusions increase mostly between very close shot-types, while distinguishing three classes works very accurately with 76% accuracy by support vector machines.

As a third contribution, we have created an assistance system to detect a covered lens. When recording videos with a smartphone, a user may accidentally cover the lens with the hand or fingers. We target the issue as a semantic segmentation problem and propose two approaches to solve it. For training and evaluation, we create an image dataset and augment it with fingers partially covering the lens. In our first approach to semantic segmentation, we apply classical image processing techniques where we combine a skin-color detector and a blur estimator to identify the covered regions. Our second approach relies on a convolutional neural network. To cope with the limited computational power on mobile platforms, we use a combination of Mobilenet and a Fully Convolutional Neural Network (FCN). In our evaluation, we compare the two approaches with each other and provide further comparison with Deeplab, a state of the art image segmentation technique. While our Mobilenet/FCN combination vastly outperforms the classical image processing approach, Deeplab even performs slightly better. However, Deeplab is several magnitudes more computationally expensive than Mobilenet/FCN. Furthermore, we confirm the effectiveness of data augmentation and transfer learning for our dataset in combination with the Mobilenet/FCN architecture.

6.1 Future Work

The quality of video stabilization is difficult to assess with quantitative metrics. In the context of the Personal Film Assistant research project, a user study is planned to assess the video stabilization performance even further. We are in the process of preparing an evaluation website to present different videos combining the original video footage as well as the results of the implemented stabilization approaches.

In terms of run-time, the shot-type classification system is currently not suitable for a mobile platform. While the classifiers themselves are fast to compute, the Openpose pose estimator still requires a powerful desktop GPU. In our evaluation, we have found that the accuracy for the estimated poses are not necessarily crucial, which may render simpler pose estimation techniques feasible for future experiments.

In our approach to detecting fingers occluding the camera lens, our goal was simply to give feedback to the user. However, the per-pixel annotation of the image information enables further image processing. Instead of just notifying the user, image inpainting could be used to repair the affected areas.

List of Figures

List of Tables

Bibliography

[ABC+16] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, volume 16, pages 265–283, 2016.

[bar] Shot size. learnaboutfilm.com. `https://learnaboutfilm.com/film-language/picture/shotsize/`. Accessed: Sept. 3, 2018.

[BMSS14] Subhabrata Bhattacharya, Ramin Mehran, Rahul Sukthankar, and Mubarak Shah. Classification of cinematographic shots using lie algebra and its application to complex event recognition. *IEEE Transactions on Multimedia*, 16(3):686–696, 2014.

[Bou01] Jean-Yves Bouguet. Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm. *Intel Corporation*, 5(1-10):4, 2001.

[Bow16] Christopher Bowen. *Grammar of the shot*. Focal Press, 2016.

[Bro16] Blain Brown. *Cinematography: theory and practice: image making for cinematographers and directors*. Focal Press, 2016.

[BSA+16] Sergio Benini, Michele Svanera, Nicola Adami, Riccardo Leonardi, and András Bálint Kovács. Shot scale distribution in art films. *Multimedia Tools and Applications*, 75(23):16499–16527, 2016.

[CADB10] Scott Carter, John Adcock, John Doherty, and Stacy Branham. Nudgecam: Toward targeted, higher quality media capture. In *Proceedings of the ACM International Conference on Multimedia*, pages 615–618, 2010.

[CBL13] Luca Canini, Sergio Benini, and Riccardo Leonardi. Classifying cinematographic shot types. *Multimedia Tools and Applications*, 62(1):51–73, 2013.

[CBP+16] Liang-Chieh Chen, Jonathan T Barron, George Papandreou, Kevin Murphy, and Alan L Yuille. Semantic image segmentation with task-specific edge detection using cnns and a discriminatively trained domain transform. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4545–4554, 2016.

[CPK+18] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):834–848, 2018.

[CSP07] Ines Cherif, Vassilios Solachidis, and Ioannis Pitas. Shot type identification of movie content. In *Proceedings of the IEEE International Symposium on Signal Processing and Its Applications (ISSPA)*, pages 1–4, 2007.

[CSWS17] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1302–1310, 2017.

[DB16] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[DV16] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016.

[FXTL17] Haoshu Fang, Shuqin Xie, Yu-Wing Tai, and Cewu Lu. Rmpe: Regional multi-person pose estimation. In *Proceddings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2353–2362, 2017.

[GDDM14] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 580–587, 2014.

[GF12] Amit Goldstein and Raanan Fattal. Video stabilization using epipolar geometry. *ACM Transactions on Graphics (TOG)*, 31(5):126, 2012.

[GKCE12] Matthias Grundmann, Vivek Kwatra, Daniel Castro, and Irfan Essa. Calibration-free rolling shutter removal. In *Proceedings of the IEEE International Conference on Computational Photography (ICCP)*, pages 1–8, 2012.

[GKE11] Matthias Grundmann, Vivek Kwatra, and Irfan Essa. Auto-directed video stabilization with robust l1 optimal camera paths. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 225–232, 2011.

[GL07] Michael L Gleicher and Feng Liu. Re-cinematography: improving the camera dynamics of casual video. In *Proceedings of the ACM International Conference on Multimedia*, pages 27–36, 2007.

[GL08] Michael L Gleicher and Feng Liu. Re-cinematography: Improving the camerawork of casual video. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 5(1):2, 2008.

[GLGL18] Yanming Guo, Yu Liu, Theodoros Georgiou, and Michael S. Lew. A review of semantic segmentation using deep neural networks. *International Journal of Multimedia Information Retrieval*, 7(2):87–93, 2018.

[HPAP18] Tavi Halperin, Yair Poleg, Chetan Arora, and Shmuel Peleg. Egosampling: Wide view hyperlapse from egocentric videos. *IEEE Transactions on Circuits and Systems for Video Technology*, 28(5):1248–1259, 2018.

[HXHX14] Muhammad Abul Hasan, Min Xu, Xiangjian He, and Changsheng Xu. CAMHID: Camera motion histogram descriptor and its application to cinematographic shot classification. *IEEE Transactions on Circuits and Systems for Video Technology*, 24(10):1682–1695, 2014.

[HZC+17] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[JDV+17] Simon Jégou, Michal Drozdzal, David Vazquez, Adriana Romero, and Yoshua Bengio. The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation. In *Proceedings of the IEEE Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1175–1183, 2017.

[KB14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[KK11] Philipp Krähenbühl and Vladlen Koltun. Efficient inference in fully connected crfs with gaussian edge potentials. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, pages 109–117, 2011.

[KPS03] Jure Kovac, Peter Peer, and Franc Solina. Human skin color clustering for face detection. In *Proceedings of the IEEE EUROCON 2003. Computer as a Tool.*, volume 2, pages 144–148, 2003.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, pages 1097–1105, 2012.

[LCRB11] Christophe Lino, Marc Christie, Roberto Ranon, and William Bares. The director's lens: an intelligent assistant for virtual cinematography. In *Proceedings of the ACM International Conference on Multimedia*, pages 323–332, 2011.

[LGJA09] Feng Liu, Michael Gleicher, Hailin Jin, and Aseem Agarwala. Content-preserving warps for 3D video stabilization. *ACM Transactions on Graphics (TOG)*, 28(3):1, 2009.

[LGW+11] Feng Liu, Michael Gleicher, Jue Wang, Hailin Jin, and Aseem Agarwala. Subspace video stabilization. *ACM Transactions on Graphics (TOG)*, 30(1):4:1–4:10, 2011.

[LLCZ16] Kaimo Lin, Shuaicheng Liu, Loong-Fah Cheong, and Bing Zeng. Seamless video stitching from hand-held camera inputs. *Computer Graphics Forum*, 35(2):479–487, 2016.

[LSD15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440, 2015.

[LTY+16] Shuaicheng Liu, Ping Tan, Lu Yuan, Jian Sun, and Bing Zeng. Meshflow: Minimum latency online video stabilization. In *Proceedings of European Conference on Computer Vision (ECCV)*, pages 800–815, 2016.

[LYTS13] Shuaicheng Liu, Lu Yuan, Ping Tan, and Jian Sun. Bundled camera paths for video stabilization. *ACM Transactions on Graphics (TOG)*, 32(4):1, 2013.

[MCB97] Carlos Morimoto, Rama Chellappa, and Steve Balakirsky. Fast image stabilization and mosaicking. In *Proceedings of DARPA Image Understanding Workshop (IUW)*, 1997.

[Mit12] Hiroko Mitarai. Interaction model for emotive video production. *International Journal of Information and Electronics Engineering*, 2(5):661–666, 2012.

[MY11] Hiroko Mitarai and Atsuo Yoshitaka. Shooting assistance by recognizing user's camera manipulation for intelligible video production. In *Proceedings of the IEEE International Symposium on Multimedia (ISM)*, pages 157–164, 2011.

[NKJ+06] Anwar Nusirwan, CW Kit, S John, et al. Rgb-h-cbcr skin colour model for human face detection. In *Proceedings of the MMU International Symposium on Information & Communications Technologies*, pages 16–17, 2006.

[NL12] Yuzhen Niu and Feng Liu. What makes a professional video? a computational aesthetics approach. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(7):1037–1049, 2012.

[PPCCMFV00] José Luis Pech-Pacheco, Gabriel Cristóbal, Jesús Chamorro-Martinez, and Joaquín Fernández-Valdivia. Diatom autofocusing in brightfield microscopy: a comparative study. In *Proceedings of the IEEE International Conference on Pattern Recognition (ICPR)*, volume 3, pages 314–317. IEEE, 2000.

[PPG13] Said Pertuz, Domenec Puig, and Miguel Angel Garcia. Analysis of focus measure operators for shape-from-focus. *Pattern Recognition*, 46(5):1415–1432, 2013.

[PZK+17] George Papandreou, Tyler Zhu, Nori Kanazawa, Alexander Toshev, Jonathan Tompson, Chris Bregler, and Kevin Murphy. Towards accurate multi-person pose estimation in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[RDS+15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, pages 234–241, 2015.

[SFC+11] Jamie Shotton, Andrew Fitzgibbon, Mat Cook, Toby Sharp, Mark Finocchio, Richard Moore, Alex Kipman, and Andrew Blake. Real-time human pose recognition in parts from single depth images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1297–1304, 2011.

[SGF+13] Jamie Shotton, Ross Girshick, Andrew Fitzgibbon, Toby Sharp, Mat Cook, Mark Finocchio, Richard Moore, Pushmeet Kohli, Antonio Criminisi, Alex Kipman, and Andrew Blake. Efficient human pose estimation from single depth images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(12):2821–2840, 2013.

[SJMS17] Tomas Simon, Hanbyul Joo, Iain Matthews, and Yaser Sheikh. Hand keypoint detection in single images using multiview bootstrapping. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[SLJ+15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[SSS+17] Dominik Schörkhuber, Florian Seitner, Benedikt Salzbrunn, Margrit Gelautz, and Georg Braun. Intelligent film assistant for personalized video creation on mobile devices. In *Proceedings of the International Conference on Advances in Mobile Computing & Multimedia (MoMM)*, pages 210–215, 2017.

[ST93] Jianbo Shi and Carlo Tomasi. Good features to track. Technical report, Ithaca, NY, USA, 1993.

[ST2a] Cisco systems. (n.d.). global mobile video traffic from 2016 to 2021 (in terabytes per month). in statista - the statistics portal. `https://www.statista.com/statistics/252853/global-mobile-videotraffic-forecast/`. Accessed: Feb 16, 2018.

[ST2b] Iab. (n.d.). frequency of mobile video content sharing according to smartphone owners worldwide as of may 2015. in statista - the statistics portal. `https://www.statista.com/statistics/446453/global-frequency-of-mobile-video-content-sharing/`. Accessed: Feb 16, 2018.

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[TTNP14] Ioannis Tsingalis, Anastasios Tefas, Nikos Nikolaidis, and Ioannis Pitas. Shot type characterization in 2d and 3d video content. In *Proceedings of the IEEE International Workshop on Multimedia Signal Processing (MMSP)*, pages 1–5, 2014.

[WRKS16] Shih-En Wei, Varun Ramakrishna, Takeo Kanade, and Yaser Sheikh. Convolutional pose machines. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4724–4732, 2016.

[XWH+11] Min Xu, Jinqiao Wang, Muhammad A Hasan, Xiangjian He, Changsheng Xu, Hanqing Lu, and Jesse S. Jin. Using context saliency for movie shot

classification. In *Proceedings of the IEEE International Conference on Image Processing (ICIP)*, pages 3653–3656, 2011.