Pontifícia Universidade Católica
do Rio de Janeiro

Rafael Antônio Pinto Pena

A robust workflow for person tracking and meta-data generation in videos

Tese de Doutorado

Thesis presented to the Programa de Pós-Graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Doutor em Ciências - Informática.

Advisor: Prof. Helio Cortes Vieira Lopes

Rio de Janeiro
February 2021

Pontifícia Universidade Católica
do Rio de Janeiro

Rafael Antônio Pinto Pena

A robust workflow for person tracking and meta-data generation in videos

Thesis presented to the Programa de Pós–graduação em Informática da PUC-Rio in partial fulfillment of the requirements for the degree of Doutor em Ciências - Informática. Approved by the Examination Committee:

Prof. Helio Cortes Vieira Lopes
Advisor
Departamento de Informática – PUC-Rio

Prof. Marcelo Gattass
Departamento de Informática – PUC-Rio

Prof. Marcus Vinicius Soledade Poggi de Aragão
Departamento de Informática – PUC-Rio

Dr. Rafael Silva Pereira
Uber do Brasil Tecnologia Ltda

Prof. Alex Laier Bordignon
UFF

Rio de Janeiro, February 11th, 2021

Rafael Antônio Pinto Pena

Graduated in Data Processing Technology at Pontifícia Universidade Católica do Rio de Janeiro - PUC-Rio in 1999. Masters in Computer Science at Pontifícia Universidade Católica do Rio de Janeiro - PUC-Rio in 2012. Experienced in Computer Science, focusing on Software Development since 1997. Software Engineer, Software Engineering Manager and Data Science Manager at Globo.com and at Ipiranga Produtos de Petróleo.

Bibliographic Data

Pena, Rafael Antônio Pinto

A robust workflow for person tracking and meta-data generation in videos / Rafael Antônio Pinto Pena; advisor: Helio Cortes Vieira Lopes. – 2021.

53 f: il. color. ; 30 cm

Tese (doutorado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2021.

Inclui bibliografia

1. visão computacional. 2. reconhecimento de faces. 3. meta-dados em vídeo. I. Cortes Vieira Lopes, Helio. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Acknowledgments

Firstly, to God for all the countless blessings in every step of life.

To my family, specially my wife Carolyna Maciel Pena and my daughter Isadora Maciel Pena for supporting me and understanding the moments when I was away to be able to dedicate all my attention to this work and my parents Fernando Pena and Nina Pena, for having invested in my education, from elementary school to graduation.

To my advisor Prof. Hélio Cortes Vieira Lopes for the continuous support of my Ph.D study and related research, for his patience and making me believe it would be possible. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor.

Besides my advisor, I would like to thank my thesis committee: Prof. Marcelo Gattass, Prof. Marcus Vinicius Soledade Poggi de Aragão, Dr. Rafael Silva Pereira and Prof. Alex Laier Bordignon for the evaluation of this work, for their availability and dedication reading and contributing to this research.

To all professors of the Department of Informatics from PUC-Rio for their knowledge and help.

To my fellow labmates and workmates, specially, Felipe A. Ferreira, Frederico Caroli and Luiz Schirmer for the stimulating discussions and software development during this work.

To Globo.com, my previous employer, that paid a better part of the expenses of this Doctoral course and gave me all resources needed for this research.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

Abstract

Pena, Rafael Antônio Pinto; Cortes Vieira Lopes, Helio (Advisor). **A robust workflow for person tracking and meta-data generation in videos**. Rio de Janeiro, 2021. 43p. Tese de Doutorado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

The amount of recorded video in the world is increasing a lot due not only to the humans interests and habits regarding this kind of media, but also the diversity of devices used to create them. However, there is a lack of information about video content because generating video meta-data is complex. It demands too much time to be performed by humans, and from the technology perspective, it is not easy to overcome obstacles regarding the huge amount and diversity of video frames. In this work we propose an automated face recognition system to detect and recognize humans within videos. It was developed to recognize characters,in order to increase video meta-data. It combines standard computer vision techniques to improved accuracy by processing existing models output data in a complementary manner. We evaluated the performance of the system in a real data set from a large media company.

Keywords

computer vision; face recognition; video meta-data;

Resumo

Pena, Rafael Antônio Pinto; Cortes Vieira Lopes, Helio. **Uma metodologia robusta para rastreamento de pessoas e geração de meta-dados em vídeos**. Rio de Janeiro, 2021. 43p. Tese de Doutorado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

A quantidade de vídeos gravados no mundo cresce muito, não somente devido aos interesses e hábitos humanos em relação a esse tipo de mídia, mas também pela diversidade de dispositivos utilizados para criação de vídeos. No entanto, faltam informações sobre conteúdos em vídeo porque a geração de metadados é complexa e requer muito tempo para ser executado por humanos. Do ponto de vista da tecnologia, não é fácil superar os obstáculos relacionados à grande quantidade e diversidade de frames de vídeo. O trabalho propõe um sistema automatizado de reconhecimento facial para detectar personagens em vídeos. Ele foi desenvolvido para reconhecer personagens, a fim de aumentar os metadados de vídeo. Ele combina técnicas padrão de visão computacional para melhorar a precisão, processando os dados de saída dos modelos existentes de maneira complementar. O modelo teve um desempenho satisfatório usando um conjunto de dados da vida real de uma grande empresa de mídia.

Palavras-chave

visão computacional; reconhecimento de faces; meta-dados em vídeo;

Table of Contents

List of Figures

List of Tables

1
Introduction

A key goal of the entertainment and media industry is to understand users preferences in order to get content production insights and provide relevant content. To this end, more information about the content potentially generates more information about the users. This work will focus on video content.

In Brazil, tens of millions of people watch soap operas almost every day. Broadcasting TV has popularized this type of content but part of its consumption is through cable TV channels, and in the last decade online consumption, whether live or on demand, has grown sharply.

Globo Group is the largest media group in Latin America and occupies a prominent position in this scenario, being the audience leader for this type of content. Either on broadcasting TV, cable TV or via the Internet. The company is the leader in the production of soap operas in Brazil. Its production started in 1965, surpasses 300 soap operas and tens of thousands of chapters. In recent years, online channels are used by approximately 2 million users every day.

Many computer vision technologies emerged in recent years and good results on specific tasks have been achieved. Face recognition (Schroff et al., 2015), scene detection (Castellano, 2020), optical flow (Kroeger et al., 2016) and pose estimation (Cao et al., 2018) are examples of existing technologies created to better understand humans appearance within videos.

The ultimate goal of this work is to create a robust model to detect humans within videos in order to enrich video meta-data. On this case study, an archive from Grupo Globo was used in order to test and validate the solution. Hundreds of different scenes from 10 different soap operas have been processed.

The task that this thesis is proposing to solve has several difficulties. For instance, the calculation of image similarity requires a high computational cost; entity match with a proprietary database is hard when using online face recognition services; and the accuracy on face recognition varies and depends on video frame aspects to get a good result (for instance, the character face angle).

From these difficulties emerge many challenging issues to create an efficient automated character detection system. Identifying the correct character

in a scene, from the very first appearance until the last frame is not a trivial task. In an attempt to identify characters presence in scenes it was necessary to test a hybrid solution based on complementary, specific techniques.

Advantages, disadvantages and results achieved using the proposed framework will be presented. Also, complementary methods from state of art and how to use it. Finally, a framework for person recognition and tracking for which a detailed report is provided.

In summary, our main contributions is the proposal of an unified framework that provides:

- Face verification (is this the same person)
- Face recognition (who is this person)
- Elimination of ambiguous annotation errors
- Tracking people in videos

An article about this framework was published in the 22nd International Conference on Enterprise Information Systems (ICEIS 2020) and received the Best Paper Award in the Area of Artificial Intelligence and Decision Support Systems.

This Thesis is organized as follow. Chapter 2 describes some previous and related work. Chapter 3 discusses the proposed method. Chapter 4 shows the results. Chapter 5 discusses the results and points out some future works. Finally, Chapter 6 concludes the work by summarizing the contributions.

2
Background

This chapter shows an analytical synthesis covering a better part of the literature on the problem. High level of conceptual linking within and across theories and techniques regarding automatic face detection, face recognition and people tracking by computers.

2.1
Face Detection

According to Dalal and Triggs (2005), the issue of human detection in pictures is a challenging assignment due to their variable appearance and the wide scope of poses that they can adopt. The main need is a robust feature set that permits the human structure to be discriminated cleanly, even in cluttered backgrounds under troublesome illumination. They study the issue feature sets for human detection, demonstrating that locally normalized Histogram of Oriented Gradient (HOG) descriptors provide excellent performance relative to other existing feature sets including wavelets (Mohan et al., 2001; Schneiderman and Kanade, 2004).

Dalal and Triggs (2005) reviewed existing edge and gradient based descriptors and demonstrated experimentally that grids of Histograms of Oriented Gradient (HOG) descriptors significantly surpass existing feature sets for human detection. They study the impact of each stage of the computation on performance, reasoning that fine-scale gradients, fine orientation binning, relatively coarse spatial binning, and high-quality local contrast normalization in overlapping descriptor blocks are immeasurably significant for good outcomes. The new methodology gives close ideal division on the original MIT pedestrian database.

Kazemi and Sullivan (2014) addressed the issue of face alignment for a single image and shows how an ensemble of regression trees can be utilized to estimate the face's landmark positions direct from a sparse subset of pixel intensities, accomplishing real-time performance with high quality predictions. The authors investigated distinct strategies and its significance to battle overfitting was additionally examined. Their paper presents an algorithm to estimate the position of facial landmarks in a computationally efficient manner.

The proposed method uses a cascade of regressors and describe the details of the form of the individual components of the cascade and how they trained the model.

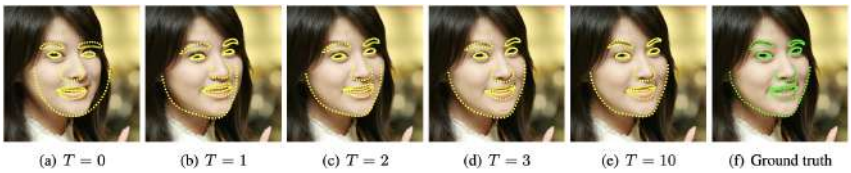

Figure 2.1: Landmark estimates at different levels of the cascade initialized with the mean shape centered at the output of a basic Viola and Jones (2001) face detector. After the first level of the cascade, the error is already greatly reduced (Kazemi and Sullivan, 2014).

To accurately benchmark the performance of Kazemi and Sullivan (2014) proposed method, which is an ensemble of regression trees, they created two baselines. The first is based on randomized ferns with random feature selection and the other is a more advanced version of this with correlation based feature selection which is their reimplementation of (Cao et al., 2012).

Ranjan et al. (2017) introduced an algorithm based on deep convolutional neural networks (CNN) focused on simultaneous face detection, landmarks localization, pose estimation and gender recognition. Their project consider the intermediate layers of a CNN utilizing a different CNN followed by a multi-task learning algorithm that works on the fused features. Their technique can detect face, localize land-marks (related to the face points), estimate the pose, which is the order of roll, pitch and yaw, and recognize the gender. They use specialized CNN layers to get each output prediction. Other than the exceptional outcomes, this methodology actually is extremely computational demanding.

2.2 Face Recognition

Zhou et al. (2018); Wang et al. (2008) and Zhou et al. (2004) realized that face recognition has been intensively studied in the literature but the attempts on visual recognition of multiple faces simultaneously in videos are rarer than single face recognition attempts. It has potential applications in practical video surveillance. Face recognition is a difficult assignment that has been attacked for over twenty years. Traditional methodologies for videos consider video sequences where each frame are analyzed separately. More exact methodologies use the temporal cues in addition to the presence of the faces in videos.

Face recognition framework based on images regularly includes task that incorporates face detection, face alignment, and face matching between

a detected face in an image and a reference dataset of faces (Zhou et al., 2018). Nonetheless considering the real world situations, facial expression, illumination condition, and occlusion represents difficult problems.

Wang et al. (2008) addressed the issue of classifying image sets, each of which contains images belonging to the same class but covering many variations in. For instance, illumination and viewpoint. The authors formulated the issue as the computation of Manifold-Manifold Distance (MMD), i.e., calculating the distance between nonlinear manifolds each representing one image set. To process MMD, likewise propose a novel manifold learning approach, which communicates a manifold by a collection of local linear models, each depicted by a subspace. MMD is then changed in order to incorporating the distances between pair of subspaces respectively from one of the involved manifolds.

The proposed MMD method is evaluated on the task of Face Recognition based on Image Set (FRIS). In FRIS, each known subject is enrolled with a set of facial images and modeled as a gallery manifold, while a testing subject is modeled as a probe manifold, which is then matched against all the gallery manifolds by MMD. Identification is achieved by seeking the minimum MMD.

Schroff et al. (2015) presented a system, called FaceNet, that map face images to a compact Euclidean space. Distances directly correspond to a measure of face similarity. These feature vectors can be used in tasks such as face recognition, verification and clustering combined with classical machine learning techniques. Their method uses a deep convolutional network trained to directly optimize the embedding itself, rather than an intermediate bottleneck layer as in previous deep learning approaches. To train, they use triplets of roughly aligned matching / non-matching face patches generated using a novel online triplet mining method. The benefit their approach is much greater representational efficiency: they achieve state-of-the-art face recognition performance using only 128-bytes per face. They also introduce the concept of harmonic embeddings, and a harmonic triplet loss, which describe different versions of face embeddings (produced by different networks) that are compatible to each other and allow for direct comparison between each other.

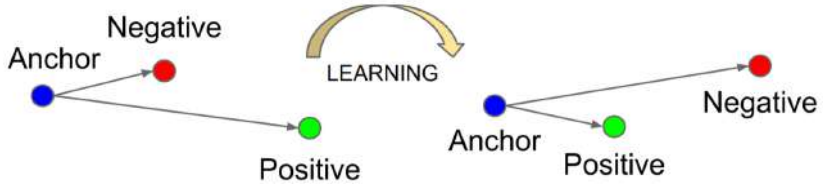

Figure 2.2: Triplet loss function intuition (Schroff et al., 2015).

FaceNet directly trains its output to be a compact 128-D embedding using a triplet based loss function based on large margin nearest neighbor (LMNN) classification (Weinberger et al., 2006). Their triplets consist of two matching face thumbnails and a non-matching face thumbnail and the loss aims to separate the positive pair from the negative by a distance margin. The thumbnails are tight crops of the face area, no 2D or 3D alignment, other than scale and translation is performed.

The triplet loss function reduce the distance between similar things and tries to increase the same between different things.

layer	**size-in**	**size-out**	**kernel**
conv1	220×220×3	110×110×64	7×7×3, 2
spool1	110×110×64	55×55×64	3×3×64, 2
rnorm1	55×55×64	55×55×64	
conv2a	55×55×64	55×55×64	1×1×64, 1
conv2	55×55×64	55×55×192	3×3×64, 1
rnorm2	55×55×192	55×55×192	
pool2	55×55×192	28×28×192	3×3×192, 2
conv3a	28×28×192	28×28×192	1×1×192, 1
conv3	28×28×192	28×28×384	3×3×192, 1
pool3	28×28×192	14×14×384	3×3×384, 2
conv4a	14×14×384	14×14×384	1×1×384, 1
conv4	14×14×384	14×14×256	3×3×384, 1
conv5a	14×14×256	14×14×256	1×1×256, 1
conv5	14×14×256	14×14×256	3×3×256, 1
conv6a	14×14×256	14×14×256	1×1×256, 1
conv6	14×14×256	14×14×256	3×3×256, 1
pool4	14×14×256	7×7×256	3×3×256, 2
concat	7×7×256	7×7×256	
fc1	7×7×256	1×32×128	maxout p=2
fc2	1×32×128	1×32×128	maxout p=2
fc7128	1×32×128	1×1×128	
L2	1×1×128	1×1×128	

Table 2.1: Structure of Schroff et al. (2015) proposed model . The input and output sizes are described in *rows x cols × #filters* . The kernel is specified as *rows × cols, stride* and the maxout [6] pooling size as $p = 2$.

In all Schroff et al. (2015) experiments, they train the CNN using Stochastic Gradient Descent (SGD) with standard backprop (LeCun et al., 1989), (Rumelhart et al., 1986) and AdaGrad (Duchi et al., 2011). In most experiments Schroff et al. (2015) started with a learning rate of 0.05 which they lower to finalize the model. The models are initialized from random, similar to (Szegedy et al., 2015), and trained on a CPU cluster for 1,000 to 2,000 hours.

The decrease in the loss (and increase in accuracy) slows down drastically after 500h of training, but additional training can still significantly improve performance. The margin is set to 0.2 and corresponds to the maximum distance between two embeddings to be considered the same face.

To deal with real video situations, Huang et al. (2015) propose a Hybrid Euclidean-and-Riemannian Metric Learning method to fuse multiple statistics of image set. They represent each image set simultaneously by mean, covariance matrix and Gaussian distribution. To test they approach, they use a public large-scale video face datasets: YouTube Celebrities, containing 1910 video clips of 47 subjects collected from the web site. Its results are impressive, although it faces some problems considering its context, such as not to re-identify people. There is no association between images frames of the same person taken from different cameras or from the same camera in different occasions.

2.3 Visual tracking

Instead of recognizing individual face independently, Zhou et al. (2018) introduces the constraints of inter-frame temporal smoothness and within-frame identity exclusivity on multiple faces in videos, and model the tasks of multiple face recognition (MFR) and multiple face tracking (MFT) jointly in an alternative optimization framework.

Masi et al. (2018) propose a method that is designed to explicitly tackle pose variations. Their Pose-Aware Models (PAM) process a face image using several pose-specific and deep convolutional neural networks (CNN). Also, in their application, a 3D rendering is used to synthesize multiple face poses from input images to both train these models and to provide additional robustness to pose variations.

Zhou et al. (2018) address the problem of recognition and tracking of multiple faces in videos involving pose variation and occlusion. They introduce constraints of inter-frame temporal smoothness and coherence on multiple faces in videos, and model the tasks of multiple face recognition and multiple face tracking jointly in an optimization framework. Also, they focus in practical problems involving surveillance cameras.

(Danelljan et al., 2014) introduced the discussion about visual object tracking in computer vision. The problem involves estimating the location of a visual target in each frame of an image sequence. Despite significant progress in recent years, the problem is still difficult due to factors such as partial occlusion, deformation, motion blur, fast motion, illumination variation, back-

Figure 2.3: A visualization of the tracking results from (Danelljan et al., 2014) approach compared with the state-of-the-art visual trackers: (Danelljan et al., 2014) approach (red), ASLA (Jia et al., 2012) (green), SCM (Zhong et al., 2012) (blue), Struck (Hare et al., 2011) (yellow) and LSHT (He et al., 2013) (orange). The image sequences pose challenging situations such as scale variations (a), partial occlusions (b) and out-of-plane rotations (c).

ground clutter and scale variations. Most existing approaches provide inferior performance when encountered with large scale variations in complex image sequences. In their paper, they tackle the challenging problem of scale estimation for visual tracking. Danelljan et al. (2014) propose an efficient method for estimating the target scale by training a classifier on a scale pyramid. This allows them to independently estimate the target scale after the optimal translation is found. They show that their approach improves the accuracy over an exhaustive scale space search method, while running at 25 times faster frame-rate. To validate their approach, they perform extensive experiments on all the 28 image sequences annotated with "Scale Variation (SV)" in the recent benchmark evaluation (Wu et al., 2013). They compare their approach with state-of-the-art trackers in literature. Despite its simplicity, their tracker achieves state-of-the-art performance, while operating at real-time. Figure 2.3 shows a comparison to state-of-the-art trackers on three benchmark sequences.

2.4 Scene Detection

Castellano (2020) created a Python library for detecting scene changes in videos named PySceneDetect. This library provides different detection methods, from simple threshold-based fade in/out detection, to advanced content aware fast-cut detection of each shot.

The content-aware scene detector works attempting to distinguish

whether a given pair of frames belongs the same scene or not. This technique discovers areas where the difference between two subsequent frames is bigger than the threshold value that is set.

The threshold-based scene detector, a more traditional scene detection method, compares the intensity/brightness of the current frame with a set threshold. When this value crosses the threshold a scene cut is executed. This method computes the value averaging the R, G, and B values for each pixel in the frame.

In (Gruzman and Kostenkova, 2014), the authors pointed that video shot boundary detection (SBD) is one of the essential pre-processing steps of semantic video analysis. It is an initial task in order to segment the sequence of video frames into shots. The article present a multi-modal visual features-based SBD framework focused on analysing the behaviors of visual representation in terms of the discontinuity signal. Previous works based on supervised learning usage on SBD systems have been proposed but the authors consider that the training process still remains the principal limitation. They adopt a candidate segment selection that uses a cumulative moving average of the discontinuity signal to identify the position of shot boundaries instead of a threshold calculation. Their method neglects the non-boundary video frames and performs transition detection identifying cut transition and a gradual transition, including fade in/out occurrence. The main objective of candidate segment selection is to reduce the processing time by eliminating many non boundary frames from the video sequences. In their proposed framework, shot limits are identified by using the intermittence signal between two video frames. The similarity signal used in shot transition detection is initially calculated based on SURF matching score, and RGB histogram, based on Cosine similarity. Speeded Up Robust Feature (SURF) is another feature that can represent the object movement effect.

3
Method

The approach of this work is to combine different methodologies to improve people recognition tasks using inferences based on established models. The method enables accuracy improvements for face recognition tasks by introducing a logical layer that organizes tasks and eliminate errors. Also, the method achieved excellent results regarding people tracking among video frames. Figure 3.1 presents a high level schema that explains how the system works. From the input video file to the final frame set containing actor annotations frame by frame. The blue dotted line rectangle corresponds to the main contribution from this work, the logical model, that uses open-source libraries and also new components developed during this research.

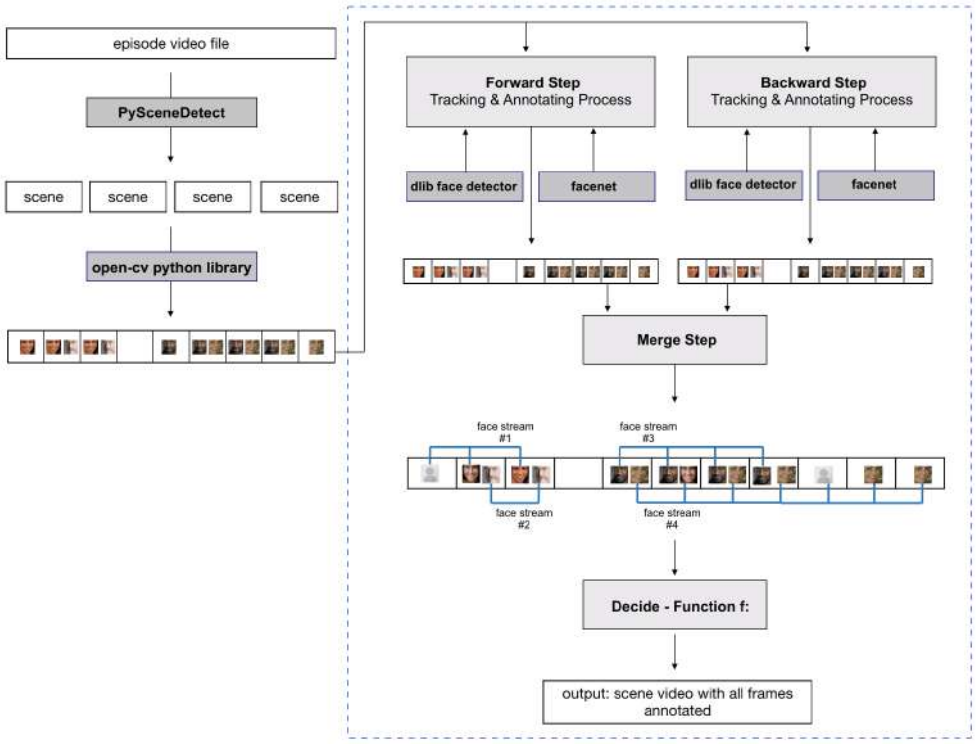

Figure 3.1: A high level schema that explains the proposed framework. White background boxes corresponds to media files, dark gray to open-source projects and light gray to new components implemented during this research. The blue dotted line rectangle corresponds to the logical model, described in 3.3.

The first step is splitting the video into multiple clips, each one cor-

responds to a different scene. PySceneDetect is a command-line application and a Python library for detecting scene changes in videos, and automatically splitting the video into separate clips (Castellano, 2020). This application was used to eliminate errors observed during the propagation of actors annotations through the frames. Another advantage is that it enables parallel processing. A ten minutes video, that represents one of the four episode blocks, has on average a hundred scenes.

Once the separate clips are available, all the further steps are taken for all different scenes, simultaneously. The next step is to identify, for each different actor that appears in the scene, the actor face stream. The face stream is the entire set of frames where a specific actor appears during a scene. It is obtained by a module developed during this work, named FaceTracker and detailed in section 3.1.

An existing system was used to overcome facial recognition challenges. FaceNet uses a deep convolutional network (Schroff et al., 2015) and this model will be discussed in section 3.2.

All the outputs from these different technologies are processed by a logical model that combines information and infers the actors for the entire frame set. This logical model will be detailed in section 3.3. In order to scale in real life situations the system was integrated with two existing components, the computer vision architecture detailed in section 3.4 and the content ontology discussed in section 3.5.

In order to clearly explain the *Python* Object-Oriented implementation from this work, this section will explain the classes, attributes and methods using pseudocode functions to describe the logic and also the open source code libraries used in this work.

3.1 Facetracker

This module was created to track actor faces through scene frames. During the tracking process, the module uses a face detector and an object tracker implemented using the function *get-frontal-face-detector* and the class *correlation-tracker* from Dlib (King, 2009) .

This face detector is made using the now classic Histogram of Oriented Gradients (HOG) feature combined with a linear classifier, an image pyramid, and sliding window detection scheme. This type of object detector is fairly general and capable of detecting many types of semi-rigid objects in addition to human faces King (2009).

To track a face among the frames the Dlib Python library provides a

specific object. This object lets you track the position of an object as it moves from frame to frame in a video sequence King (2009). Dlib is a modern C++ toolkit containing machine learning algorithms and tools for creating complex software in C++ to solve real world problems. It is used in both industry and academia in a wide range of domains including robotics, embedded devices, mobile phones, and large high performance computing environments. Dlib's open source licensing allows you to use it in any application, free of charge King (2009).

The structural SVM based training algorithm behind the easy to use object detector provided by Dlib is called Max-Margin Object Detection (MMOD). This method does not perform any sub-sampling, but instead optimizes over all sub-windows. MMOD can be used to improve any object detection method which is linear in the learned parameters, such as HOG or bag-of-visual-word models King (2015).

The system developed during this research attempts to create a face stream for each different character appearance within the scene. After the face detection, the face tracker step was implemented using dlib's implementation of the correlation tracker algorithm. To use it, you give the correlation-tracker the bounding box of the face to be tracked and a frame. This object will start tracking the face inside the bounding box in the given image. The dlib correlation tracker implementation is based on paper Danelljan et al. (2014) and used a baseline closely related to the MOSSE tracker Bolme et al. (2010). The tracker learns a discriminative correlation filter used to localize the target in a new frame Danelljan et al. (2014).

The face tracking process is started every time a face is detected by the *get-frontal-face-detector* from dlib (King, 2009). A pre-trained facial landmark detector available in this library was used to estimate the location of (x, y)-coordinates that map to facial structures on the face. There are several features that can be identified in faces. Eyes, mouth, nose and so forth. DLib algorithms enables detection of these features by providing a map of points that surround each feature. In this detector, the map is composed of 68 points. This detector, an ensemble of regression trees, is an implementation of Kazemi and Sullivan (2014).

This function is very efficient to detect a face from the frontal view. But sometimes a character first appearance in a scene is a side view. In those cases, the results obtained were not perfect because the tracker only starts to track after the frontal face detection, generating a lack of meta-data for all the frames that preceded the first frame in which a frontal face detection has occurred.

The FaceTracker class from this implementation has two main attributes:

Algorithm 1 Face object. Requires Dlib and OpenCV libraries

```
1: function FACE()
2:     initialize attributes x,y,w,h
3:     create bounding box using dlib.rectangle()
4:     initialize person_id
5:     initialize attribute annotation
6:     return face
7: end function
```

model and trackers. The model attribute corresponds to the detector model and is initialized with *dlib get-frontal-face-detector*, the trackers attribute starts with an empty dictionary and is updated along the the tracking process. Every time a new face is detected a new *Face* 1 object is created and based on this object a new tracker object is created and appended to the trackers attribute. Each tracker corresponds to a specific face in a specific frame. The algorithm 2 shows the pseudocode from this class. The complete logic will be explained in section 3.3.

Figure 3.2: Example images from the TV Globo dataset. The red boxes show the face tracking after frontal face detection.

Figure 3.2 shows that one of the actors wasn't tracked since his first appearance because *get-frontal-face-detector* couldn't detect his face in the first frame. On the second frame the same actor face was detected and after that the actor will be tracked even in frames with a side view.

To avoid this problem we created a two-step tracking process. The forward pass tracks the actor in the frames after the frame in which the frontal face detection occurred. The order is from the first frame to the last. Analogously, the backward pass tracks the actor in the frames before the frame in which the frontal face detection happened and the process starts from the last frame. At this phase the goal is to link the actor face positions among the scene frames.

During the tracking process, all frames are processed, but the detection task occurs depending on the detection rate used. Every time a face is detected, the program attempts to identify if its a new face (that didn't appear in previous frames) or not. Depending on this verification, a new random id for

Algorithm 2 Tracking. Requires Dlib and OpenCV libraries.

```
1: function CreateFaceTracker()
2:     set attribute model to dlib.get_frontal_face_detector()
3:     create an empty dictionary named trackers
4:     initialize person_id_counter = 0
5:     return face_tracker
6: end function
7:
8: function track(scene,face_tracker,detection_rate)
9:     create an empty list of faces named faces_tracked
10:    create an empty list of faces named frame_faces
11:    for each frame in scene do
12:        for each face in faces_tracked do
13:            face_in_frame = NEW_TRACKER(face, frame)
14:            frame_faces = frame_faces  face_in_frame
15:        end for
16:        if is a detection frame (based on detection rate) then
17:            faces_detected = DETECT_FACES(face_tracker, frame)
18:            for each face in faces_detected do
19:                if face not in frame_faces then
20:                    create a new Face object
21:                    add the new face to frame_faces
22:                end if
23:            end for
24:        end if
25:        update face_tracker.trackers with frame_faces
26:        set faces_tracked equal to frame_faces
27:    end for
28: end function
29:
30: function detect_faces(face_tracker,frame)
31:    convert frame to rgb using cvtColor from open-cv
32:    detect faces in frame using face_tracker.model
33:    return the faces as a list of Face objects
34: end function
35:
36: function new_tracker(face,frame)
37:    initialize object tracker with dlib.correlation_tracker()
38:    create a rectangle using dlib.rectangle() and x,y,w,h from face
39:    tracker.start_track(frame, rectangle)
40:    return tracker
41: end function
```

that face (person) will be created or an id used in a previous frame will be used again (same person from previous frame).

This tracking process occurs twice (forward and backward passes) and then merged. After the merge step, the logical model decides who is the actor frame by frame.

3.2 Facenet

As described in Schroff et al. (2015), FaceNet is an end-to-end learning technique that uses deep neural network to learn an embedding $f(x)$ from an image x in a feature space $\mathbb{R}^d$, such that the square of the distance between all faces, regardless the conditions of the image such as illumination and pose variation, of the same identity is small, whereas the square of the distances between pairs of faces of different identities images is large. To this end, a triple loss is implemented to ensure that a specific person's x_i^a *(anchor)* image is close to all x_i^p *(positive)* images of a single person is for any other image x_i^n *(negative)* of any other person. Thus, FaceNet minimizes the following loss function L for a set of N faces where α is a margin that is enforced between positive and negative pairs:

$$L = \sum_{i}^{N} [\| f(x_i^a) - f(x_i^p) \|_2^2 - \| f(x_i^a) - f(x_i^n) \|_2^2 + \alpha]_+ \tag{3-1}$$

As the resulting output, it generates a compact d-dimensional Euclidean space as feature vectors where distances directly corresponds to a measure of face similarity that can be used in tasks like face recognition, verification and clustering.

In this work, a pre-trained model that outputs 128-dimensional vectors was used for the task of recognize actors face. Another important aspect from this system that fits perfectly to the use case reported in this work is that the system depends on an initial set of faces, and from this set, for each frame processed, the system will calculate the similarity between faces from the frame and each face from the initial set. In this work, the initial set of faces is the soap opera actors faces. It is possible to assume that for each face detected in a frame, it will be one of the actors informed in the initial set (the cast). In order to obtain the cast actors faces images, a semantic query endpoint was used to query the company's content ontology. The only necessary filter is the name or the identification code of the content. Once this information is known, it is possible to use that endpoint to find out the actors that participates in a soap opera. Section 3.5 describes the existing ontology model and how it allows generic queries for various content types.

Before use FaceNet in order to generate the face embeddings, it is necessary to align the face. The alignment preprocess faces for input into a neural network. Faces are resized to the same size (such as 96x96) and transformed to make landmarks (such as the eyes and nose) appear at the same location on every image Amos et al. (2016). Figure 3.3 details the embedding generation pipeline before face recognition step.

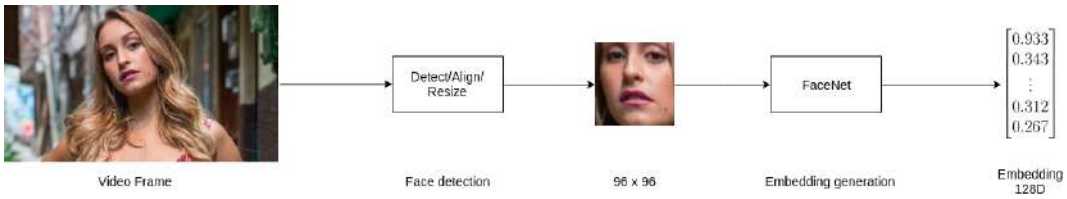

Figure 3.3: Example of 128-dimenisonal embedding generation for each face found in a frame

After all face embeddings have been generated, each of them is compared against all embeddings of all labeled faces dataset in order to recognize who is the actor that appear in the frame. Basically, the actor's facial recognition task is accomplished through simple computation of the Euclidean distance between the embedding of the frame and the embeddings of the dataset. If the distance is smaller than a certain *Threshold*, the face of the actor is recognized as being the one corresponding to the labeled embedding face from the dataset. In this work was used a *Threshold=0.2*, which is the value used by Schroff et al. (2015). The layers, inputs and outputs from the convolutional neural networks (CNN) used are described in section 2.2.

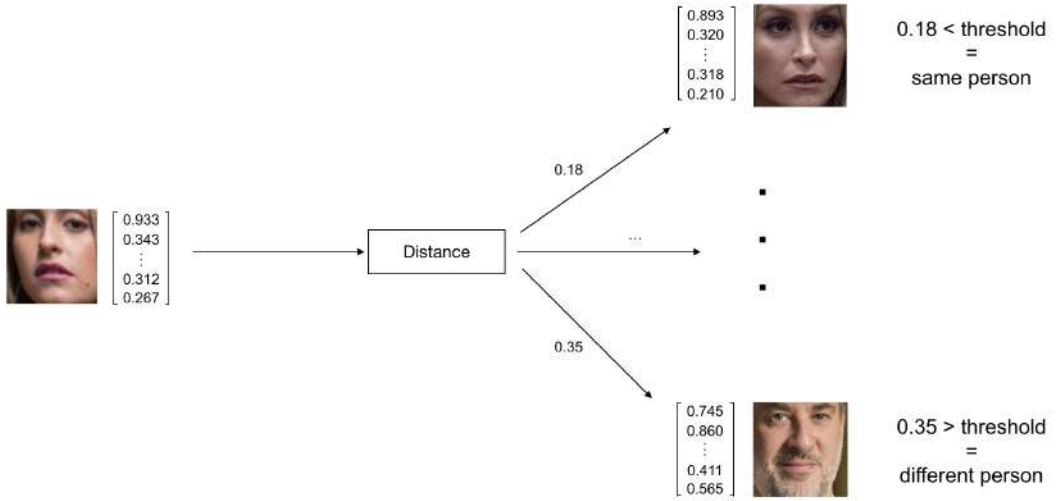

Figure 3.4: Example of face recognition step. Based on a threshold, each face embedding found in frame is compared against all labeled face embeddings dataset in order to recognize the actor face.

A class named *FaceRecognizer* was implemented to be used as an interface for using the Facenet model. This class has three important attributes: model - corresponds to the face recognition model, represented by a data file saved in the Hierarchical Data Format (HDF5), an open source file format that supports large, complex, heterogeneous data. This file contains a pretrained Facenet model obtained from (Chollet, 2015). Another important attribute from this class is *align* that is a model file designed for use with dlib's HOG face detector. It expects the bounding boxes from the face detector to be aligned a certain way. The third important attribute is *database* containing the reference database to be used, in this research this database stores face images for each actor from the soap opera cast.

Algorithm 3 Recognition. Requires Dlib, FaceNet and OpenCV libraries

```
1: function CreateFaceRecognizer()
2:     set attribute model to facenet_weights.h5 from keras
3:     set attribute align to shape_predictor_68_face_landmarks.dat.bz2
   from dlib
4:     initialize attribute database with face images reference data
5:     return face_recognizer
6: end function
7:
8: function who(image,face_recognizer,face)
9:     load image (frame)
10:    create face bounding box (bb)
11:    define landmark_indices to be used for alignment as
   AlignDlib.OUTER_EYES_AND_NOSE
12:    get an aligned face bb using face_recognizer.align + landmark_indices
13:    create embedding using the aligned face bb, face_recognizer.model +
   predict_on_batch from keras
14:    for each face in database do
15:        calculate embeddings distance (frame - database item)
16:        store minimum distance and identity
17:    end for
18:    return identity and minimum distance
19: end function
```

In order to better understand the *FaceNet* object usage, its instantiation and methods, all operations using this object will be detailed in 3.3, inside the main program course explanation.

3.3 The logical model

The logical model optimizes the frames annotation by eliminating errors and annotating frames regardless a frontal view of the face.

After splitting the original video into different scenes, this logical layer is applied to each scene. This section will cover the main code and all the functions created to manipulate the objects explained, previously.

The main course of the program consists in six steps:

- loading models - objects from classes (*FaceTracker* and *FaceRecognizer* are created and loaded in memory.
- load video - transform a video file into a frame set
- run the tracking and annotating process in the original frame order (forward)
- run the tracking and annotating process in the reversed frame order (backward)
- merge the results obtained from two previous steps
- decide the final annotation for each frame.

These steps will be explained later in this section. Also, the algorithms 1, 2, 3 and 4 can be useful to understand the logical model.

3.3.1 Loading Models

There are two important models to be loaded in memory before processing the frames. The *FaceTracker()* object is the first. As mentioned in 3.1, the attribute *model* contains the face detection model to be used in order to identify whether there is a face in a frame or not. FaceTracker object loads the default face detector from Dlib. During the tracking processes that will be explained later in this section, the attribute trackers will be updated every time a new face is detect.

FaceRecognizer() is the second object from the *loading models* step and contains the pre-trained face recognition model (FaceNet) from Chollet (2015) and also a shape predictor to detect and align face landmarks, another pre-trained model from Dlib. It helps transforming and skewing an image. The last important attribute is the reference images database.

Algorithm 4 Main Function

```
1: function MAIN()
2:     facetracker = CreateFaceTracker()
3:     facerecognizer = CreateFaceRecognizer()
4:     scene = load video using opencv packages
5:     run(face_tracker,face_recognizer,scene)                  //forward pass
6:     run(face_tracker,face_recognizer,reversed(scene)         //backward pass
7:     merge passes
8:     decide annotations
9: end function
10:
11: function RUN(face_tracker,face_recognizer,scene)
12:     create an empty list of faces named frames_faces
13:     for frame, faces in face_tacker.track(scene,face_tracker, 5) do
14:         frames_faces.append(faces)
15:         for face in faces do
16:             who = face_recognizer.who(frame, face)
17:             set face_recognizer.annotation = who
18:         end for
19:     end for
20:     return frames_faces
21: end function
```

3.3.2 Tracking and annotating process

A function was created in order to implement the face tracking and annotating task. The function *run* is detailed in algorithm 4. It requires three parameters: a *FaceTracker()* object, a *FaceRecognizer()* object and the frame set to be processed frame by frame in the same order passed to the function.

These parameters must be created in the main program and passed to the function that iterates the frame set to be processed. For each face detected using the *FaceTracker()* object, the *FaceRecognizer()* object is used to compare the detected face against the reference database and annotate it.

In algorithm 2, the function *track* runs once for an entire scene (set of frames). When iterating frame by frame, depending on the detection rate parameter set, the program will either try to detect faces or not, but always keep tracking in the next frames the faces already detected until the current frame .

After a face detection occurs, the correlation tracker algorithm, explained in section 3.1, is used to identify all subsequent frames where the same face appears and the position of the face in the image. The position enables the program to work correctly even when there are multiple faces in the frame.

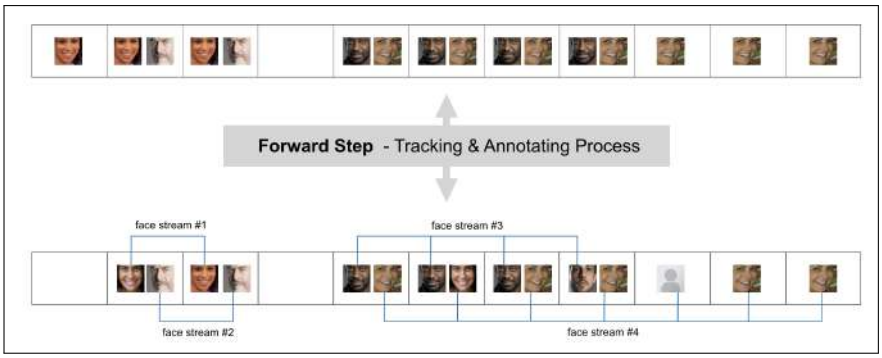

Figure 3.5: Tracking and annotating process, forward step. After processing the frame set, in the original order, the program identified 4 face streams (person appearances). The face stream #1 missed the first frame due to the lack of a frontal view of the face.

By analysing figures 3.5 and 3.6 will be clear how the complementary usage of face detection, face recognition and object tracking techniques works.

Figure 3.5 shows a high level schema containing three parts. An illustrative input frame set at the top, a gray rectangle that corresponds to the first step from the tracking and annotating process (forward) and another frame set at the bottom that corresponds to the output from this step. The input frame set is already annotated with the correct actors and will be used as a ground truth example, each face thumbnail corresponds to the correct person that appeared in the corresponding frame, rather than the face from the original image. The output frame set contains the face streams (subsets of frames linked by a line using correlation-matrix) and annotations (face thumbnails with the face recognized from *FaceNet*).

By comparing input and output frame sets, it is possible to verify that the **face stream 1** didn't include the first frame in to the actress face stream, due to a miss-detected face. This happens because the correlation tracker is triggered by a frontal face detection that didn't occur due to the face angle. Still regarding the same actress face stream, notice that even though the correlation tracker created a correct link between second and third frames for the same face, for some reason *FaceNet()*, incorrectly, recognizes two different actress in each frame. The **face stream 2** was perfectly created, it means that a frontal face detection occurred since the fist frame that the actor appeared, the correlation tracker linked all the subsequent frames that contains that face and also *FaceNet()* recognized the correct face for frames from that actor face stream. **Face streams 3 and 4** has corrects frames linked together but both face streams has wrong face recognition on specific frames. **Face stream 4** also has a miss-detected face on the ninth frame.

Similarly to figure 3.5, figure 3.6 shows the results obtained in the

backward step. Notice that **face stream 3**, created during the backward step, corresponds to the **face stream 1**, created during the forward step but linking, correctly, three frames instead of two.

After forward and backward tracking and annotation steps, the next step is to merge both outputs. The merge process in detailed in subsection 3.3.3.

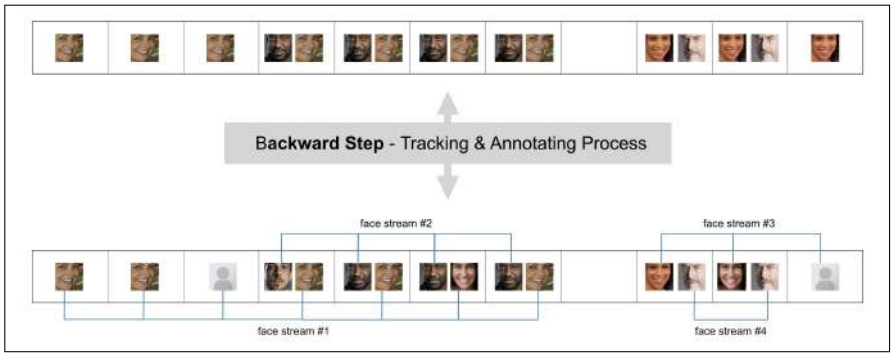

Figure 3.6: Tracking and annotating process, backward step. After processing the frame set, in the reversed order, the program identified 4 face streams. The face stream #3 corresponds to the face stream #1 from 3.5, and after this step is complete with no missing frame.

3.3.3 Merge

This is a very simple step that consists in merging outputs from the tracking and annotation process. The goal is obtain an unique set of face streams (person appearances), indexed by frame. Each face stream maps all the frames corresponding to that face stream , and also the face positions (bounding box) in each frame. After joining dictionaries (frame:faces) obtained after forward and backward steps, the output will be as many face streams as the number of different actors appearances that occurred in the scene. The main goal of the merge step is to map correctly all the frames related to the same appearance, rather than apply the best recognition for each person, task that will be performed after the merge task.

Once with the two available lists (outputs from the tracking and annotating process, forward and backward steps), the program will join the frames. The first frame in forward output corresponds to the last in backward output, the second in forward output corresponds to the penultimate in backward output and so on. After that, the program will deduplicate *Face()* objects. We use the bounding boxes positions from faces created using the *FaceTracker()* object during the tracking process.

During the faces deduplication process, the program knows the face *ids* created during forward and backward steps. For instance, when analysing

corresponding frames from both outputs, if the same face (bounding box) received *id* equals 1 during forward step and equals 99 during backward step, a new *id* will be created, and the final output will set this new *id* to all faces obtained from forward step with id equals 1, and all faces obtained from backward step with id equals 99. Something like an "outer join" by face positions.

For instance, let's consider the outputs detailed in figure 3.5 and figure 3.6 as input for the merge process and see the result on figure 3.7.

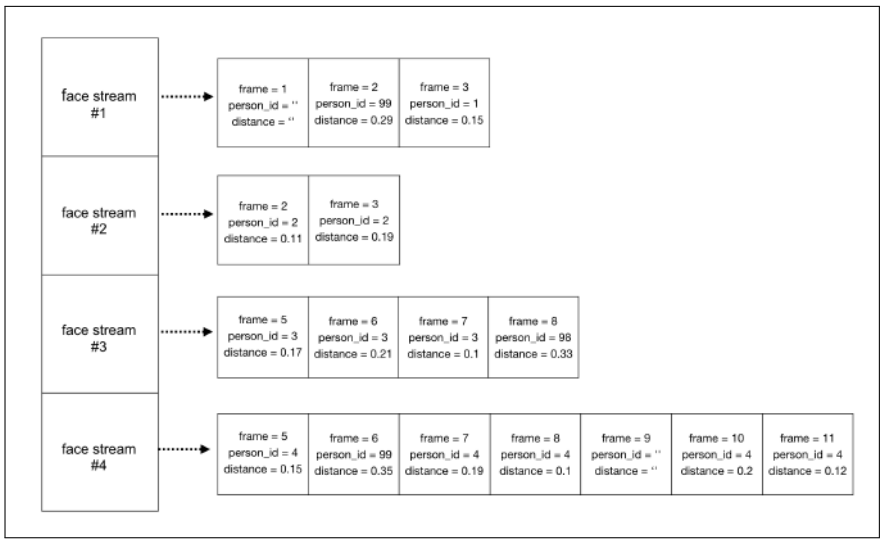

Figure 3.7: Output from merge step

After the merge step, all the face streams are available, in other words, all the actors appearances are mapped and each appearance corresponds to a continuous frame sequence based on its first and last frames. At this point, even though all face streams are defined, there is not a single annotation assigned to all frames from the same stream. Some frame may be miss annotated and also two different frames from the same stream may have different annotations.

3.3.4 Decide annotation

Once actor streams are created (output from merge step) the logic model attempts to assign the same annotation to all the frames from an actor stream. The model processes each different actor stream separately.

For each frame containing a FaceNet annotation, there is a distance metric calculated by FaceNet. The logic model uses the smallest distance actor suggested by FaceNet, all the annotations from different frames of the same face stream are considered. The implementation is prepared to accept any

function that decides the annotation to be assigned to all frames of a face stream.

3.4 Computer Vision Architecture

In Globo.com, there is an in-house computer vision architecture to support different applications. This architecture provides an asynchronous process to integrate different components from the company's video platform.

Figure 3.8 shows a high level architecture to explain how the proposed solution is integrated to the existing systems. On the top of the figure there is a component called *Scheduler* that is connected to the *Webmedia API* using the *recents* service that informs every new media published on the video platform. The *Scheduler* identifies if the media must be processed by a computer vision job and enqueues the media to be processed as soon as there is available computer resources to.

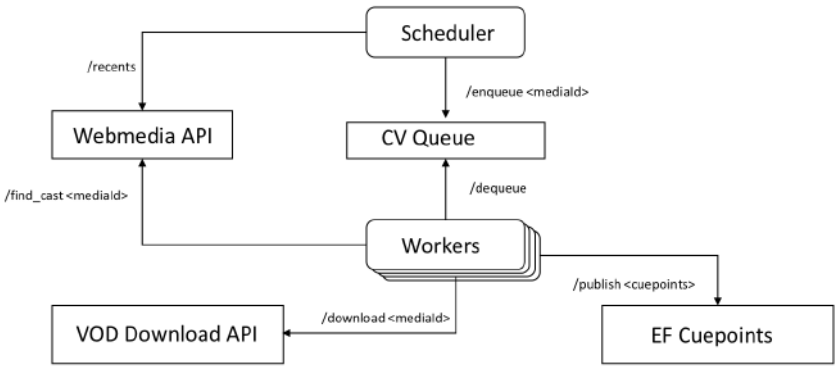

Figure 3.8: In-house architecture for computer vision applications.

3.5 The content ontology

Guizzardi and Wagner (2010) emphasizes the difference between the meanings of the term ontology in computing. On the one hand, by the Conceptual Modeling community, the term has been used according to its definition in Philosophy: A domain-independent and philosophically well-grounded formal categorization system, which can be used to spell out specific models of reality. domain. On the other hand, by the Artificial Intelligence, Software Engineering, and Semantic Web communities, the term is used as a concrete engineering artifact designed for a specific purpose without paying much attention to grounding issues.

Globo.com has previously invested in the creation of an content ontology that describes all kinds of content produced or offered by the company.

During the last 10 years this ontology has been evolved by many journalists and content producers with the focus on articles categorization to improve content search, organization of content offerings in digital products and online navigation. Nowadays this ontology has hundreds of concepts and hundreds of thousands instances that helps to describe content types, events, roles, locations, persons and objects.

The Unified Foundational Ontology (UFO) has been used to obtain a high-level abstraction that helps on generalize and organize concepts from different domains. From the technical aspect, the implementation of the ontology using Resource Description Framework (RDF - https://www.w3.org/RDF/), a W3C pattern, enabled the implementation of a semantic query endpoint that integrates concepts from the company vocabulary and other public vocabularies, like DBpedia, to enrich concepts.

Using this controlled vocabulary described by the company ontology is possible to find out all cast and crew from any creative work, just like finding any celebrity from sports, entertainment and journalism TV shows.

4
Results

To evaluate our proposed system we carried out some experiments. In the following section the results obtained during these experiments are detailed. Exact matches is the main metric that was used to evaluate the system. It corresponds to the percentage of frames for which the system correctly indicates all the actors that appear.

4.1
Facestream assertiveness compared to the state of art

Regarding the evaluation task, a ground truth dataset was created using a customized tool developed to help humans to annotate actors appearances in video frames. The dataset contains 7000 frames, randomly selected from 20 different scenes (3 different soap operas). The videos used in this work has H.264 as video coding format and a display resolution of 640 × 360 pixels. The evaluated accuracy was 92%. Two different approaches were tested in order to evaluate the proposed method: the approach 1 was based on the FaceNet recognition without any additional logic. The approach 2 was based on the FaceNet recognition together with the logic model developed during this research and detailed in section 3.3.

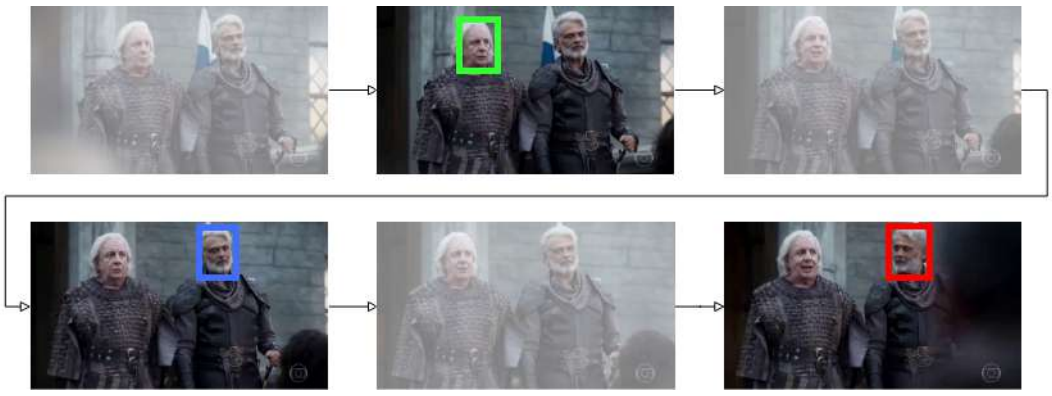

Figure 4.1: Frames sequence inconsistently annotated. The red and the blue bounding boxes corresponds to different annotations, even though are the same person.

Scene	# frames	approach 1	approach 2
Scene 1	352	182	301
Scene 2	445	290	408
Scene 3	705	558	690
Scene 4	730	300	698
Scene 5	810	507	800
Scene 6	1225	780	1180
Scene 7	1448	1154	1301
Scene 8	1604	1290	1440
Scene 9	1850	1550	1698
Scene 10	2110	1613	1891

Table 4.1: Examples of exact matches comparison between two survey approaches applied to different scenes.

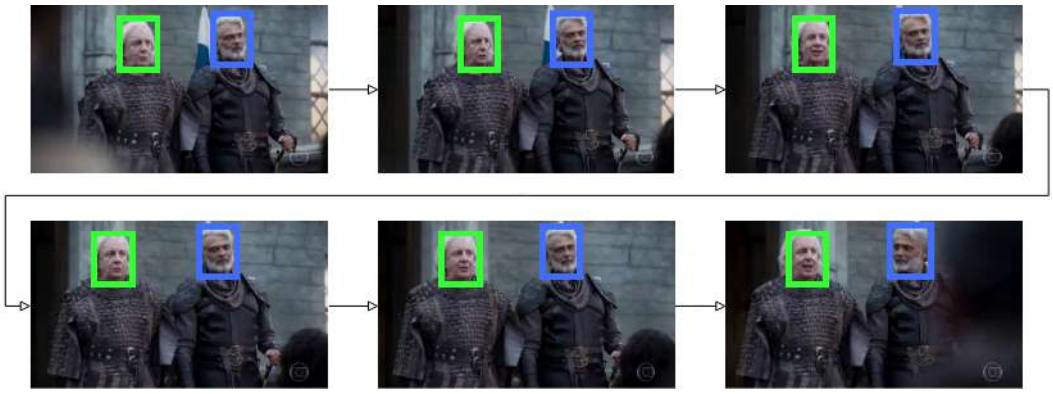

Figure 4.2: Frames sequence consistently annotated. The ambiguous annotation issue, described in 4.1, was eliminated by using the proposed framework.

Both Figures 4.1 and 4.2 shows the same frame sequence. The first one shows the initial results using *get-frontal-face-detector* from the *dlib Python* library and *FaceNet*, the second one shows results using the entire proposed framework that includes *correlation-tracker*, also from the *dlib Python* library and the Logical Model implemented during this research.

Figure 4.1 shows an example of a frame sequence containing both annotations issues. Either non annotated or wrong annotated frames can be observed. In this case, although the partial solution got some correct guesses in some frames, the assertiveness was 0% due to the evaluation metric used.

Table 4.1 shows the enormous advantage of approach 2 over approach 1 when the goal is to maximize exact matches.

During the experiments, both the accuracy and the software performance of the model were observed. Table 4.2 shows the relationship between detection rate and accuracy. It is possible to notice that if the detection rate is very high, the accuracy drops significantly. When using a detection rate equal to 30, which represents 1 frame per second, since the video was generated at 30 frames per

Detection rate	Assertiveness
1	93%
5	92%
10	89%
30	67%

Table 4.2: Assertiveness by detection rate. The detection rate means the frequency in which the detection task is performed. If equals to 1, it means that the program will execute this task in all the frames, if equals to 30, it means that the program will execute this task in 1 every 30 frames.

second, the accuracy is 67%. However, between a detection rate of 1 (all frames processed) and 5 (one for every 5 frames is processed) the difference in accuracy was not that great, 93% and 92%, respectively. At the same time, there was a significant reduction in the total processing time. Section 4.3 details the results in terms of software performance.

4.2 Qualitative analysis

Several different analyses combining metrics of assertiveness and images observation revealed specific important situations. In this section, we discuss some limitations of the method, related to the visual characteristics of the processed images. In addition, we also highlight the method's strong points, which presented excellent results in frames with challenging visual characteristics.

4.2.1 Issues to be improved

As described in 3.2, one important attribute from the *FaceNet* object is *database*. It contains the image reference database to be transformed in order to make landmarks using *AlignDlib*. During the research, it was possible to clearly observe the importance of using frontal view face photos as reference.

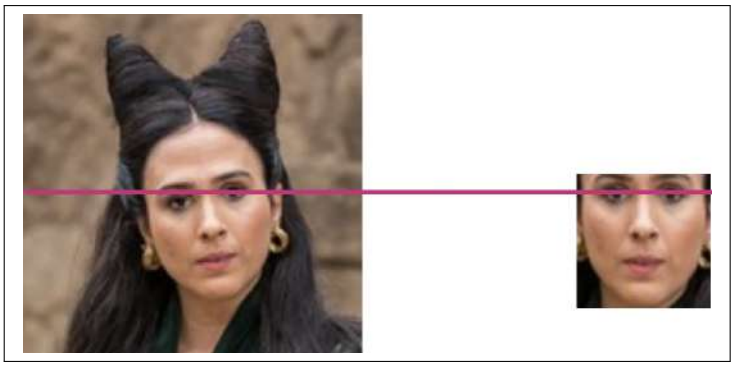

Figure 4.3: Face landmarks alignment without distortion. The face landmarks alignment using *AlignDlib*, described in 3.2, showed excellent results when applied on frontal face view images.

The left hand side of Figure 4.3 shows the original photo from an actress. Note that it is a frontal face photo and that the head is tilted to one side. The purple line helps to see that the eyes and eyebrows are misaligned (different heights). The right hand side shows the actress face thumbnail vector after being transformed to align landmarks. Note that eyes and eyebrows are aligned in the thumbnail and the image is not distorted.

Figure 4.4 shows another alignment transformation example. The left hand side shows that eyes are misaligned too. Observe that in this example the original photo is a side view and check at the right hand side of the figure that face is distorted in the thumbnail after landmarks alignment step.

These details were observed during an exploratory analysis that compared actors for whom the model had good assertiveness with those for whom the model did not have good assertiveness. In some cases, the low assertiveness of the model for a specific actor was related to the reference image that was used, as in the example in Figure 4.4.

Figure 4.4: Face landmarks alignment without distortion. The face landmarks alignment using *AlignDlib*, described in 3.2, generated distorted images when applied on side face view images.

Another limitation observed is related to the use of accessories on the face. Figure 4.5 shows a frame sequence that has an annotation problem. The first two frames are correctly annotated. At the third frame the actor face stream was interrupted because the actor turned his face left in a way that the face landmarks used by the *correlation-matrix* was not visible anymore. At the fifth frame the actor starts wearing sunglasses and at the sixth frame the model made a wrong annotation. To be precise, in this case, two different problems happened. The first was the face stream interrupted and the second was the wrong recognition due to the sunglasses. However, if the first problem didn't happen, all the frames would be correctly annotated because only one face stream would be detected and therefore all the frames from this face stream would have the same annotation, that had the best score.

Figure 4.5: Annotation error due to sunglasses usage. The system annotated an actor, correctly, in the first two frames. But he was not recognized correctly after putting on his sunglasses.

4.2.2 Model strengths

The model demonstrated consistent results even processing scenes recorded by a moving camera. It supports camera rotation and camera shift movements. Figure 4.6 shows a sequence of frames from a scene, in which the artists remained in the same place throughout the scene while the camera moved to the right, passed behind a pillar and then framed the actors again.

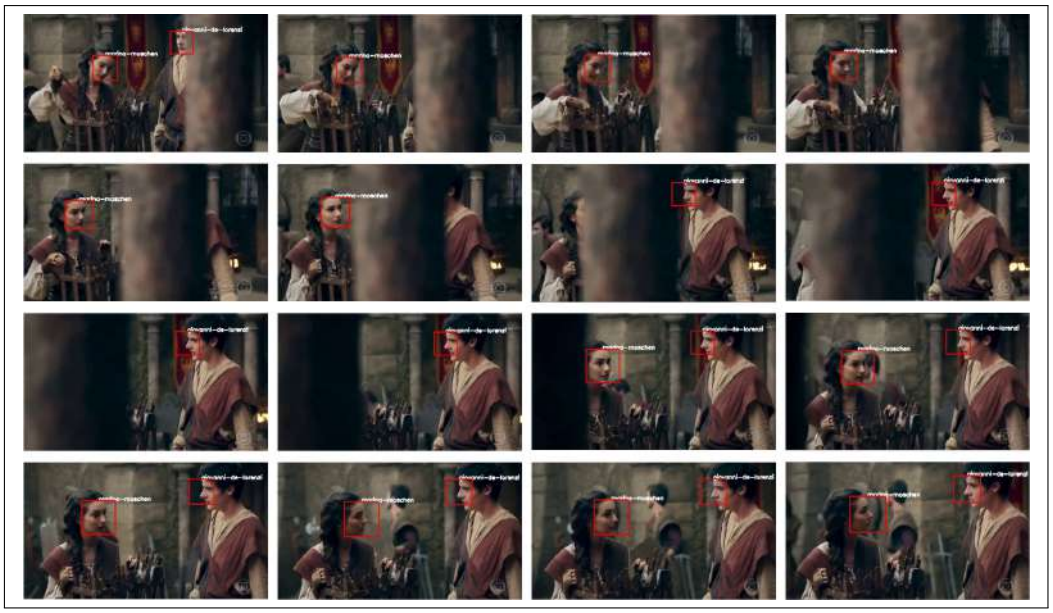

Figure 4.6: Good results in scenes with moving camera. During the scene, the actors remained in the same place, but the camera moved to the right. Even though there was an occlusion, the system did not lose tracking and did not generate any recognition error.

During the experiment, the solution was robust in situations where the camera zooms in on the actors' faces, bringing the image closer to or away from the face, without losing assertiveness in the correct recognition of the actors.

The sequence of frames shown in Figure 4.8 is an example of the observed effectiveness of the method, in zoomed scenes. Note that in the first frame of the sequence of nine the actor's face is spaced apart while in the ninth frame, after the camera has zoomed in, the face appears much closer without losing the correct annotation of the corresponding actress. In the first frame there is an interesting peculiarity, the facial recognition of the actress happened based on the image of the face reflected by the mirror. In this frame it is possible to observe on the right side of the image that the actress is facing the mirror, almost with her back to the camera.

Another strong point of the solution is that it works very well in situations where the actors are moving, that is, facial recognition is not impaired in situations where the actors move from one place to another during the scene. Several scenes with these characteristics have been correctly processed. The application maintained good results in several different movement situations, time with an actor, time with more than one actor, whether moving and facing the camera, or crossing from one side to the other of the scene.

Figure 4.7 contains a sequence of frames extracted from a correctly annotated scene video, from beginning to end. It is a scene shot inside a scenic city. In the first 3 frames the actresses were on the left side of the scene. In

Figure 4.7: Actresses moving from the left side to the right side of the scene. The system worked perfectly during movement but stopped tracking when all face landmarks were not visible.

the seventh frame, they were practically in front of the camera, in the next 2 frames they were already on the right side of the scene. It is possible to notice that in the eighth frame, one of the actresses (marina-ruy-barbosa) lost the annotation due to the impossibility of visualizing the face landmarks. In the ninth frame, both actresses lost their annotations due to the same limitation.

Regarding frames annotations quality, *FaceStream* showed excellent results, even though it was tested with different types of scenes, actors and scenarios. Whether in scenarios with high or low lighting, indoors or outdoors, actors standing still or moving, for any type of face, regardless of sex, age or race, the results were consistent. But there are specific limitations that have caused errors or lack of annotation. Usually related to the lack of visibility of the face landmarks, which serve as a trigger for detecting a *FaceStream* and also as a "key" between frames to perform tracking on subsequent frames in order to identify and annotate all the frames from an actor appearance.

The results obtained using the developed framework are surprising when analyzing the correctness of the annotations even in situations in which the actor's reference image is significantly different from the image of this actor in the video frames .

Figure 4.9 shows 25 examples of images cropped from original frames. It is possible to visually compare the reference image of the actor, always located in the lower left corner, with the image of the frame where the actor was properly recognized. Note that in many of them the reference image is quite different from the frame image, yet the annotation was done correctly.

Many of these frames would not be correctly annotated using only the state of art solutions like a frontal face detector (*Dlib*) and a face recognition system (*FaceNet*). Either due to occlusion of the face landmarks, low light or the actor pose that does not provide a frontal view of the face, these frames only received the correct annotation due to the use of the framework developed during this research that combines detection, recognition, tracking and a logic layer of decision.

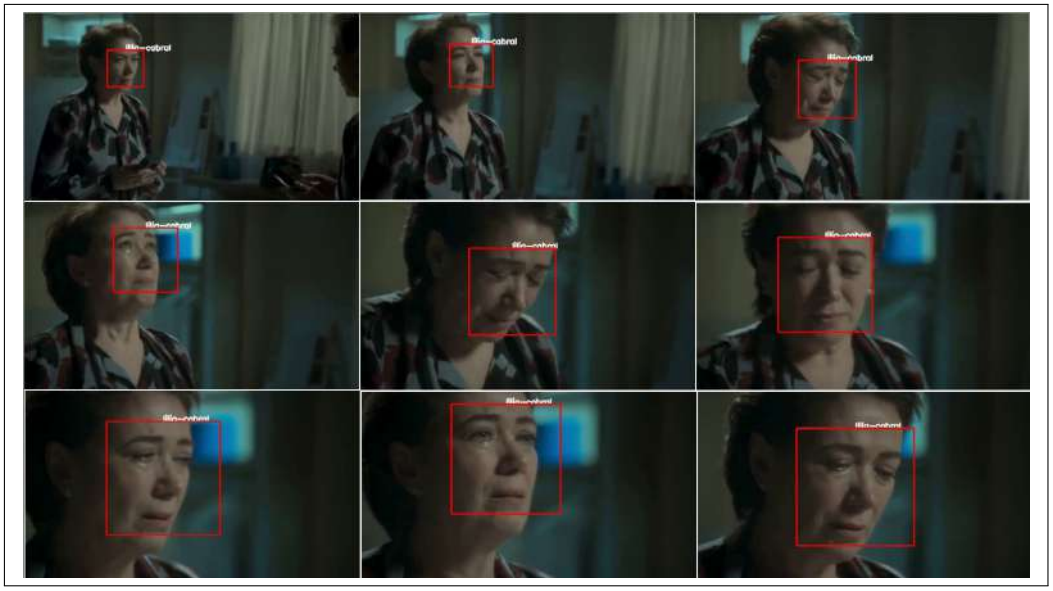

Figure 4.8: Good results filming through the mirror and zooming. On the right side of the first frame, you can see that the face image is being captured through the mirror. In the following frames, it is possible to notice the use of zoom. The system correctly annotated all frames, even with many changes in the person's face pose.

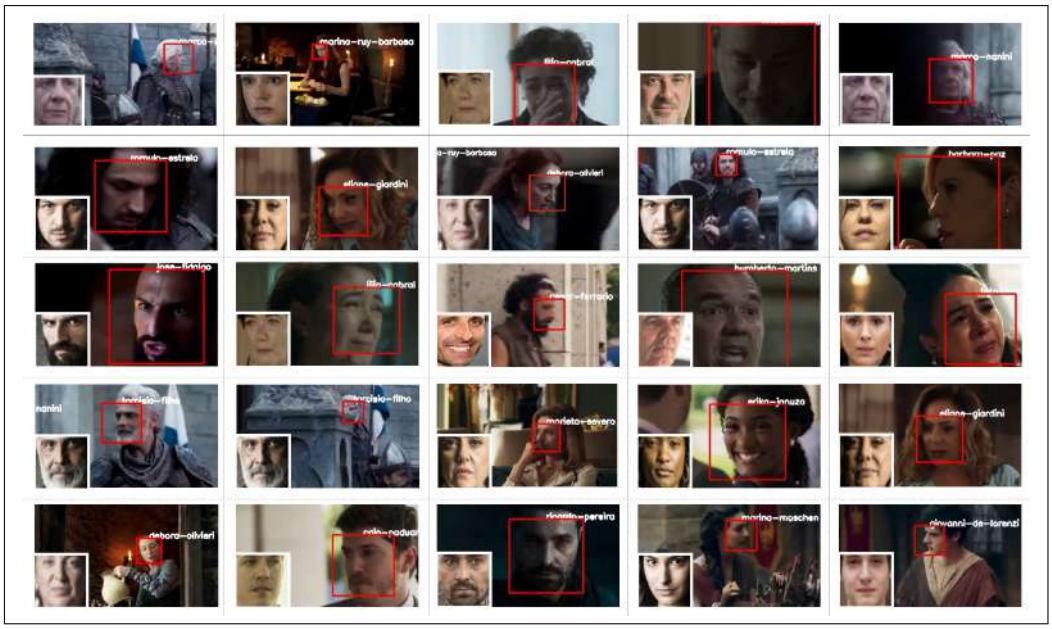

Figure 4.9: 25 comparisons between actors reference images and actors frame images properly annotated.

4.3 Software performance

This section aims to inform software performance metrics obtained during this research and provide sufficient information to usage decision making.

All the steps described in Section 3.3 were timed separately and the main results are detailed. During the software performance tests, different scenes with different amount of frames were processed. Also, in order to understand software performance optimization possibilities, the solution was tested using different detection rates. A detection rate, in this thesis, corresponds to the face detection task frequency. For example, if the detection rate is equal 1, then the logical model will process all the frames from the video (detect, track and recognize). If it's 5, then the logical model will perform the detection task in 1 out of 5 frames, instead of trying to detect faces in all frames. Therefore, this value will directly influence the total number of frames that need to be processed in order achieve acceptable results on "annotating all frames" tasks.

The first column of Table 4.3 contains the six steps detailed in section 3.3. The others columns, each one corresponds to a specific scene, and each cell from the column corresponds to the time spent on each step. Clearly, *Time Forward Pass* and *Time backward pass* are the most timing consuming steps.

The better part of the total time is regarding these steps because is during forward pass and backward pass that all the computer vision models inferences happens. Detection, tracking and recognition.

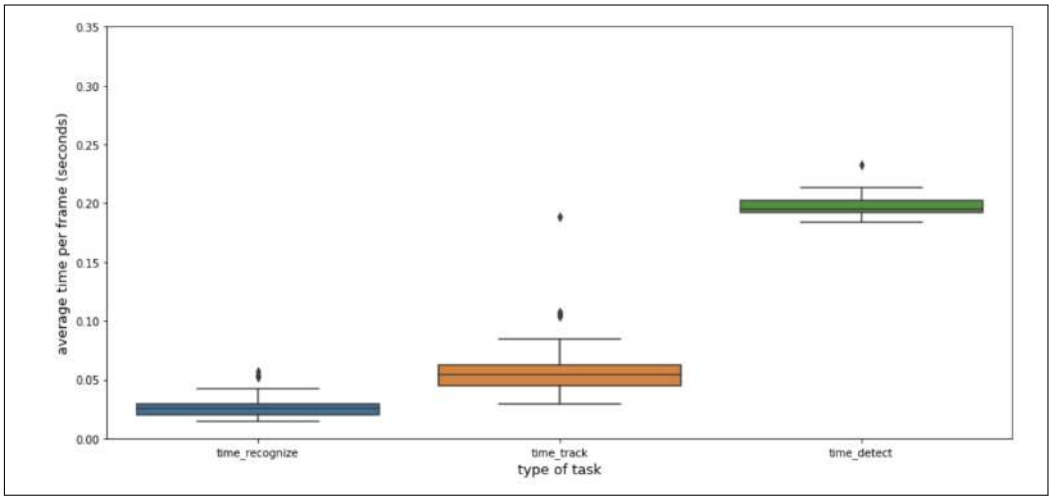

Figure 4.10: Time to detect, time to track and time to recognize. The boxplot reveals metrics obtained from 3000 frames from 7 different scenes. Low dispersion and few outliers are observed during the experiments. Detection is the most time consuming task.

Figure 4.10 shows a box plot comparing the average time per frame to perform each type of task. These metrics are based on hundreds of frames,

from several different scenes, using different detection rates, but even so it is possible to notice that they are consistent by observing the thin heights of the boxes from the plot due to the metric's low variability. In addition, it is evident that the time to detect is much longer than the time to recognize the actor and the time to track, on average, 0.201, 0.027 and 0.042 seconds, respectively.

The time to detect is directly related to the number of frames processed. In Figure 4.11 there are two graphs. The one on the left side, the y-axis corresponds to the total time spent with detection task, the x-axis corresponds to different values used as detection rates. Each line corresponds to a specific scene randomly selected from the database. By observing the legend labels from the graph is possible to know how many frames each scene has. Notice the lines order and observe that scenes with a bigger amount of frames always have a longer total time spent on detection.

The graph on the right side of Figure 4.11 is similar to the previous graph, only the y-axis that changes, in this case, to time to predict, in other words, total time spent with recognition tasks. Although the scenes are the same, in this graph the order of the lines changes. This is because the total time spent on recognition is not related exclusively to the number of frames processed but also to the number of faces detected in the frames. In this example, the scene represented by the orange line has 448 frames, the red one has 691 frames and both have a longer total time spent with recognition task than the purple line that has 985 frames. The reason is that averages faces per frame from the orange and the red scenes are, significantly, greater than the purple scene. In numbers, 2.46, 1.41 and 0.74, respectively.

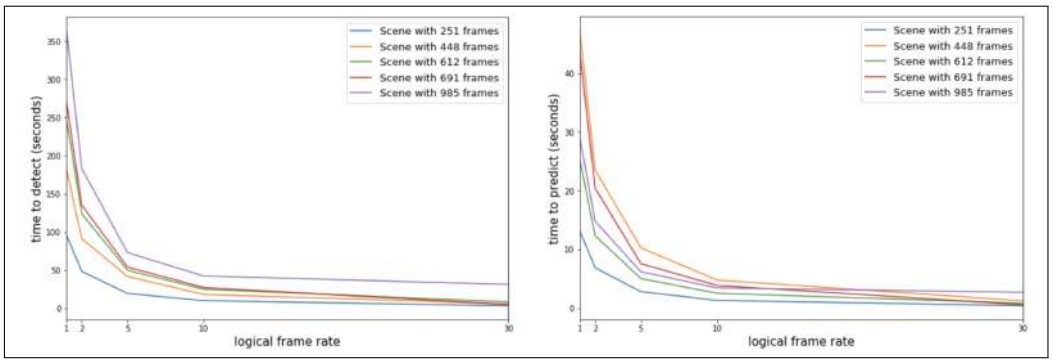

Figure 4.11: Different scenes total times per task type by detection rate. The chart on the left side shows that the total time spent on detection tasks increases according to the number of frames. On right side, the chart shows that total times spent on recognition tasks is impacted by the number of frames but also the average of humans faces per frame.

From the application performance perspective, the implementation used during this survey achieved results that enabled real life usage in the company.

	Scene 4	Scene 5	Scene 6	Scene 7	Scene 8
Duration	30	27	32	61	54
# Frames	925	810	985	1830	1628
Time loading videos	0.8610	0.8100	0.9910	1.6390	1.6090
Time loading models	10.6060	9.9700	9.8650	10.2210	10.4710
Time Forward Pass	122.4060	71.9740	71.6120	123.0910	170.2540
Time backward pass	63.8630	99.7530	88.4320	94.6830	167.3830
Time Merging passes	0.0072	0.0083	0.0028	0.0037	0.0157
Time Deciding Names	0.0166	0.0161	0.0042	0.0070	0.0108
Total Time	197.7598	182.5314	170.9070	229.6447	349.7475

Table 4.3: Total time to annotate scenes (milliseconds). Five different scenes selected from the ground truth dataset.

A totally feasible batch processing approach enabled an accuracy focus that achieved great results on an acceptable time. A parallel scene processing approach was proposed to reduce the total time spent to annotate soap opera episodes.

On average, the time spent to annotate is 5.5 times the scene duration. For instance, a 1 minute long scene will take 5 minute and 30 seconds to have all the frames annotated. Considering that each episode of a soap opera is 40 minute, divided into 4 blocks of 10 minute, the total time to annotate the entire block is 5 minutes and 30 seconds. That's because in general a 10 minute block has over 100 different scenes, and each scene is less than a minute. So, its feasible to process all scenes in parallel and spend around 5 minutes to process a 10 minutes block. This way, if there is enough computer resource in order to process all the scenes, from all the blocks in parallel, the total time is directly related to the longest scene duration. The computer used to process the videos was an Intel Xeon 2.7GHZ CPU with 128 GB RAM memory and a GPU computing processor NVIDIA Tesla P40 with 22 GB memory.

5
Discussion

Strengths and necessary improvements will be discussed in order to understand correctly the results, limitations and future works.

The workflow for person tracking and recognition and meta-data generation in videos packs the state of art Computer Vision technologies designed for object detection and recognition, adds a new logic layer based on a complementary usage of these technologies, and increases accuracy on person recognition and tracking tasks. The new approach has been tested an validated in an entertainment industry set, specifically, focused on person (actors) recognition and, in practice, used for enrich video frames meta-data annotation.

After the evaluation task it was clear that the method generated significant results regarding the amount of video frames, correctly, annotated by the model. One of the proposed method's benefit is that it enables automatic actors and actresses annotation even on frames that does not have a frontal face detection. Also, the model proved to be very efficient for disambiguate different annotations regarding the same person. The integration with the content ontology query endpoint facilitated the automatic photos retrieval from different cast. These photos were used as a reference to identify the actors in the video frames.

Even thought the outputs is a frame annotation, all the logic and the decision regarding frames annotations is based on entire actors appearances, in this article, called *face streams*. By the end, the observed results reveal that is possible to gain accuracy on frame annotations by analysing a sequences of frames where a specific frame is contained instead of analysing only the frame that needs to be annotated.

In this work, a face stream (actor appearance in a scene) depends on face landmarks to be correctly defined. In section 4.1, very good results are shown but there are issues to be improved as described in 4.2.1. An issue that caused errors in several experiments is the incorrect face stream interruption. It means that the solution wasn't able to correctly define a face stream with all the frames related to the same actor appearance. For instance, the figure 4.5 shows this kind of situation in which two different face streams was detected for the same actor appearance. This problem happens because at this moment

the solutions uses only the face landmarks in order to detect face streams and link all the frames from a face stream. But the problem is that there are some frames that does not contain any face landmark visible. Different The framework developed during this research m

A promising approach that could be tested in future works would be to evolve the solution so that the detection of a face stream and the identification of all the frames that compose it can be done based on different attributes, additionally to a vector that represents the alignment of eyes and eyebrows, how it works today. It would be like an evolution, from what is today a face stream, as described in this work, to what would be a person stream. A person stream would be created based on the attributes used in the current solution plus other new attributes of people that can be identified in an image. As examples, a lateral view of the face, the position of the head or any other human body members and also other landmarks of the face that depend on a frontal view, such as the mouth and chin positions.

One of the main challenges to be faced to make the aforementioned approach viable is the software performance. The section 4.3 provides a series of software performance metrics, details the time spent on each processing step and shows that the detection step is the slowest one. In the aforementioned approach, other types of detection would be needed, and in order not to hinder performance too much, it is important not to carry out all types of detection in all frames, during the stage of defining face streams. Due to this challenge it is important to understand the semantics of the images displayed in the frames. Either in the specific frame to be annotated or in the frames before and after it. For instance, when identifying a human head, seen from behind, in a specific frame, the effort to detect a head again or a lateral view of the face in the next frame, should take priority over the attempt to detect landmarks of the face such as eyes and eyebrows. Although this makes sense for a specific frame, it is known and it has been shown in the figures 4.11 and 4.10 that the position of the actors and cameraman differs among the frames of the same face stream. So, what type of optimal landmark should be used in order to make the best detection in the next frame, given that the current frame had the best detection based on a specific type of landmark?

Modeling this phenomenon, locating the ideal landmark type transitions between frames, identifying and understanding the transition patterns are initiatives that can increase the robustness of the method and, thus, correctly capture the entire person stream and minimize annotation problems.

6
Conclusions

When it comes to video metadata generation, the proposed method achieved important goals. The use of a content ontology was essential in automating actor photo capture. In addition, it has made it possible to use the tool in a generic and extensible way for various types of content besides soap operas, for example, TV series, movies, auditorium shows, humor shows and reality shows. Our proposed method achieved a good accuracy because it iterates the video frames in order to eliminates annotations errors and annotate frames with no frontal face detection.

The accuracy obtained with the new method is much higher when the observed metric is exact matches because it analyses the sequence of video frames that characterizes an appearance of a character in a given video instead of analyse a single frame observation. A huge archive of soap opera videos will be enriched with metadata enabling different applications improvements. New features for recommender systems and users preferences information to help advertising targeting are the first applications that will benefit from this metadata. Recently, the company developed a new pipeline for computer vision that supports different kinds of application. Nowadays, a model designed to detect intro and credits parts from TV series episodes is using this pipeline in production. The same pipeline might be used to integrate the Facestream solution to the company's video processing platform.

Bibliography

Amos, B., Ludwiczuk, B., and Satyanarayanan, M. (2016). Openface: A general-purpose face recognition library with mobile applications. Technical report, CMU-CS-16-118, CMU School of Computer Science.

Bolme, D. S., Beveridge, J. R., Draper, B. A., and Lui, Y. M. (2010). Visual object tracking using adaptive correlation filters. *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2544–2550.

Cao, X., Wei, Y., Wen, F., and Sun, J. (2012). Face alignment by explicit shape regression. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2887–2894.

Cao, Z., Hidalgo, G., Simon, T., Wei, S.-E., and Sheikh, Y. (2018). Openpose: realtime multi-person 2d pose estimation using part affinity fields. *arXiv preprint arXiv:1812.08008*.

Castellano, B. (2020). Pyscenedetect. *Last accessed*.

Chollet, F. (2015). Keras. `https://github.com/fchollet/keras`.

Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1.

Danelljan, M., Häger, G., Khan, F. S., and Felsberg, M. (2014). Accurate scale estimation for robust visual tracking. In *BMVC*.

Duchi, J. C., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159.

Gruzman, I. S. and Kostenkova, A. S. (2014). Algorithm of scene change detection in a video sequence based on the threedimensional histogram of color images. In *2014 12th International Conference on Actual Problems of Electronics Instrument Engineering (APEIE)*, pages 1–1.

Guizzardi, G. and Wagner, G. (2010). *Using the Unified Foundational Ontology (UFO) as a Foundation for General Conceptual Modeling Languages*, pages 175–196.

Hare, S., Saffari, A., and Torr, P. H. S. (2011). Struck: Structured output tracking with kernels. In *2011 International Conference on Computer Vision*, pages 263–270.

He, S., Yang, Q., Lau, R. W. H., Wang, J., and Yang, M. (2013). Visual tracking via locality sensitive histograms. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2427–2434.

Huang, Z., Wang, R., Shan, S., and Chen, X. (2015). Face recognition on large-scale video in the wild with hybrid euclidean-and-riemannian metric learning. *Pattern Recognition*, 48(10):3113–3124.

Jia, X., Lu, H., and Yang, M. (2012). Visual tracking via adaptive structural local sparse appearance model. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1822–1829.

Kazemi, V. and Sullivan, J. (2014). One millisecond face alignment with an ensemble of regression trees. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1867–1874.

King, D. E. (2009). Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, 10:1755–1758.

King, D. E. (2015). Max-Margin Object Detection. *arXiv e-prints*, page arXiv:1502.00046.

Kroeger, T., Timofte, R., Dai, D., and Van Gool, L. (2016). Fast optical flow using dense inverse search. In *European Conference on Computer Vision*, pages 471–488. Springer.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551.

Masi, I., Chang, F.-J., Choi, J., Harel, S., Kim, J., Kim, K., Leksut, J., Rawls, S., Wu, Y., Hassner, T., et al. (2018). Learning pose-aware models for pose-invariant face recognition in the wild. *IEEE transactions on pattern analysis and machine intelligence*, 41(2):379–393.

Mohan, A., Papageorgiou, C., and Poggio, T. (2001). Example-based object detection in images by components. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(4):349–361.

Ranjan, R., Patel, V. M., and Chellappa, R. (2017). Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(1):121–135.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning Representations by Back-propagating Errors. *Nature*, 323(6088):533–536.

Schneiderman, H. and Kanade, T. (2004). Object detection using the statistics of parts. *International Journal of Computer Vision*, 56:151–177.

Schroff, F., Kalenichenko, D., and Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. *CoRR*, abs/1503.03832.

Szegedy, C., Wei Liu, Yangqing Jia, Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9.

Viola, P. and Jones, M. (2001). Robust real-time face detection. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, volume 2, pages 747–747.

Wang, R., Shan, S., Chen, X., and Gao, W. (2008). Manifold-manifold distance with application to face recognition based on image set. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE.

Weinberger, K. Q., Blitzer, J., and Saul, L. (2006). Distance metric learning for large margin nearest neighbor classification. In Weiss, Y., Schölkopf, B., and Platt, J., editors, *Advances in Neural Information Processing Systems*, volume 18, pages 1473–1480. MIT Press.

Wu, Y., Lim, J., and Yang, M. (2013). Online object tracking: A benchmark. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2411–2418.

Zhong, W., Lu, H., and Yang, M. (2012). Robust object tracking via sparsity-based collaborative model. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1838–1845.

Zhou, S. K., Chellappa, R., and Moghaddam, B. (2004). Visual tracking and recognition using appearance-adaptive models in particle filters. *IEEE Transactions on Image Processing*, 13(11):1491–1506.

Zhou, X., Jin, K., Chen, Q., Xu, M., and Shang, Y. (2018). Multiple face tracking and recognition with identity-specific localized metric learning. *Pattern Recognition*, 75:41–50.