La borsa di dottorato è stata cofinanziata con risorse del
Programma Operativo Nazionale Ricerca e Innovazione 2014-2020 (CCI 2014IT16M2OP005),
Fondo Sociale Europeo, Azione I.1 “Dottorati Innovativi con caratterizzazione Industriale”

UNIONE EUROPEA
Fondo Sociale Europeo

Ministero dell'Università e della Ricerca

PON
RICERCA E INNOVAZIONE
2014 - 2020

UNIVERSITÀ DEGLI STUDI DI SALERNO

DIPARTIMENTO DI SCIENZE AZIENDALI – MANAGEMENT E INNOVATION SYSTEMS

Dottorato di Ricerca in Big Data Management

XXXIII ciclo

Multiple Object Tracking and Face-based Video Retrieval: Applications of Deep Learning to Video Analysis

Relatore:
Ch.mo Prof.
Roberto TAGLIAFERRI

Roberto Tagliaferri

Coordinatore:
Ch.mo Prof.
Valerio ANTONELLI

[illegible]

Candidato:
Gioele
CIAPARRONE
Mat. 8801500009

[illegible]

ANNO ACCADEMICO 2019/20

Dedicated to C. and F.

Contents

Chapter 1

Introduction

In recent years, deep learning (DL) has raised in popularity due to its ever-growing number of successes. It is now able to correctly classify millions of images with super-human performance [1], detect and segment objects [2], perform action recognition [3], generate images or enhance their quality [4], translate and generate text [5], synthesize speech [6], play complex games such as Go or StarCraft better than humans [7], aid in medical diagnosis [8], failure prediction [9], reducing energy consumption [10] and much more. The advent of big data and capable hardware has allowed the researchers to develop and train bigger and better models that can learn more complex and abstract features, which is the main advantage of deep learning with respect to classical machine learning methodologies, that require careful manual feature design.

In particular, the use of deep learning for video analysis is receiving increasing attention. Exploring the information contained in videos can have important repercussions: understanding what is happening in a scene, either by tracking the objects or people in it, or by understanding the actions they are taking, is an important step for tasks such as autonomous driving, video surveillance, advanced human-computer interactions, crowd behavior analysis, video retrieval, and so on.

This thesis focuses on the application of deep learning to two tasks related to video analysis: multiple object tracking (MOT) and video retrieval. In particular, I am going to present a comprehensive survey on the MOT algorithms that use deep learning, followed by a novel pipeline for face-based video retrieval (FBVR) employing DL algorithms. The main contributions of this thesis, which I will describe in greater detail in the next chapters, can be summarized as follows:

- a comprehensive survey on the use of deep learning in MOT for 2D single-camera videos;
- the identification of the four main steps characterizing a MOT algorithm, with a description of the main trackers that use deep learning in any of the four steps;
- the collection and analysis of experimental results for the presented algorithms on the most commonly used MOT datasets, in order to identify the most successful techniques and the promising future directions of research;
- a novel pipeline for the retrieval of unconstrained multi-shot videos using faces, specifically in the context of television/media content;
- the construction of an appropriate large-scale dataset for the evaluation of the face-based video retrieval pipeline;
- an extensive comparison among different models and algorithms employed in the pipeline for shot detection, face detection, face recognition and feature aggregation, with a particular focus on deep learning models, including a discussion about the advantages and disadvantages of each method. The best configuration of the pipeline obtained 97.25% Mean Average Precision on the test dataset, while performing queries over thousands of videos in less than 0.5 seconds;
- the integration of the proposed pipeline into a commercial platform, TVBridge, developed by CEDEO.

My PhD was funded by the *Italian Ministry of University and Research* (Italian: *Ministero dell'Università e della Ricerca*), under the

National Operational Program for Research and Innovation 2014-2020 (Italian: *Programma Operativo Nazionale Ricerca e Innovazione 2014-2020*) for innovative industrial doctorates. Part of the research presented in this thesis was thus performed in collaboration with the University of Granada (Spain) and with the company CEDEO, based in Turin (Italy). In particular, the research work regarding MOT was performed in Granada under the supervision of professor Francisco Herrera, while the development of the FBVR pipeline and its integration into TVBridge was performed with CEDEO under the supervision of Dr. Leonardo Chiariglione.

The rest of the thesis is organized as follows. In Chapter 2 I will define deep learning and briefly describe the main deep networks used for image classification and object detection, that will be mentioned in both the MOT and video retrieval chapters. Chapter 3 will present the survey on MOT, while Chapter 4 will focus on the proposed face-based video retrieval pipeline. Finally, Chapter 5 will summarize the findings presented in the thesis.

Chapter 2

Deep Learning

Before describing the use of deep learning in MOT or presenting the DL-based pipeline for video retrieval, I will present here a brief summary regarding the most important DL algorithms and techniques that will be widely mentioned in the next chapters, after a brief introduction about DL itself.

2.1 Deep learning basics

The term *deep learning* (DL) is used in the field of machine learning to distinguish between deep neural networks (DNN) and the classical, shallow artificial neural networks. Since classical neural networks had a single hidden layer, by convention any neural network with at least two hidden layers is considered deep, although the vast majority of DNNs has usually many more layers.

While classical neural networks (sometimes called Multi-Layer Perceptrons — MLP) mainly used fully-connected layers, which means that every neuron in a layer is connected to every neuron in the previous layer, deep neural networks take advantage of different types of layers. Arguably, one of the most important ones is the convolutional layer, used mostly (but not exclusively) for visual data, such as images. In a convolutional layer, every output neuron is only connected to a small region of the previous layer, i.e. each neuron only looks at a specific *local receptive field* of the previous layer. Moreover, the weights of the connections are shared among the output neurons, that is, the same operation is performed for each output neuron in a convolution layer, with the only difference being the input neurons it is connected to. The result emulates a discrete convolution between the input data (which can be an image in the first layer, or the intermediate features computed in the middle layers) and a *filter kernel*, i.e. the shared weights. Figure 2.1 shows an example of convolution in a deep network.

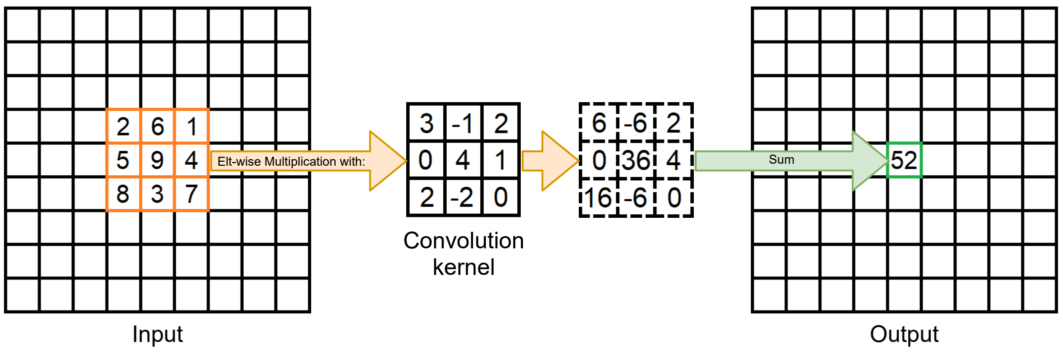


Figure 2.1: Scheme of a convolution. The 3x3 kernel in the example slides over the entire input image. For each 3x3 area in the input image a single output is obtained by multiplying each element of the input with each element of the kernel, and summing the results.

Deep neural networks that employ convolutional layers are called Convolutional Neural Networks (CNN). Usually, convolutional layers in a CNN include multiple filters, each operating independently on the input matrix and producing a different output matrix. The combined stacked features from the various filters of a layer are often called *channels*: so, we can have layers with 3, 16, 32, or even 512 channels. Since each intermediate layer outputs a 3D feature volume[1] (channels $\times$ height $\times$ width), each following convolution uses 3-dimensional kernels, so that each output neuron looks at the receptive field through the entire depth of the input volume. In this way, filters in deeper layers combine multiple features computed in the preceding layers. This results

[1]Note that if we use RGB images, the input layer is also 3-dimensional, with one channel for each of the three RGB color components.

in learning more and more complex and abstract features, the deeper we go into the network.

Another common operation used in CNNs is pooling. The pooling operation is used to reduce the spatial size (i.e. width and height) of the features in a layer. It operates with a sliding window, similar to the convolution operation, but has no parametric kernel (i.e. no learnable parameters, at least in the most common versions) and simply computes the average or the maximum of the input neurons, depending on whether we are using max or average pooling. The use of pooling is important for two reasons: first for computational efficiency, both in terms of time and space, since deeper layers of CNNs tend to have a high number of channels; second, it helps with achieving larger effective receptive fields in deep layers, i.e. a single output in the final layer of a CNN can be influenced by information present in most or all of the input image (translation invariance) — this is desirable in many tasks, such as image classification, where a decision must be made based on global image information.

Depending on the type of CNN, and on the task it is trying to solve, fully-connected layers can also be present, often in the final layers of the network. For example they can be used to output classification probabilities or regression values by looking at the features output by the convolutional modules.

For the convolutional layers, ReLU [11] is usually employed as an activation function. It is defined as $max(0, x)$, where x is the value of the neuron. ReLU was introduced to solve the vanishing gradient problem: the use of sigmoidal activations caused a progressive reduction in the magnitude of the backpropagated gradients, that tended towards zero in the first layers of the network, preventing convergence. Since the derivative of ReLU for positive values is 1, and, differently from the sigmoid, does not tend to 0 for large values of the input, it solves the problem of vanishing gradients. Other alternatives to ReLU exist, such as Leaky ReLU, Parametric ReLU (PReLU) [12] or Exponential Linear Unit (ELU) [13], which try to solve some issues with ReLU; however, the latter still remains the most used activation function in CNNs. The final classification layer of a CNN uses instead the classical softmax activation function, like shallow neural networks.

As mentioned, training deep networks is performed by backpropagation, like classical shallow neural networks, using a variety of optimization algorithms. The most used ones are variations of the mini-batch Stochastic Gradient Descent (SGD), such as SGD with Momentum [14], RMSProp[15], Adagrad[16] or Adam[17]. SGD operates similarly to classical gradient descent, with the main difference being the fact that is not applied on the entire training dataset, but on mini-batches. This helps dealing with the large datasets usually employed to train CNNs, and introduces some randomness in the gradient descent process that can help discovering new local minima. The mentioned variants of SGD aim to improve the stability and the speed of the convergence to the local minima.

The loss function depends on the problem, but multi-class cross-entropy loss is the usual choice for classification problems, while Mean Square Error (MSE) or Mean Absolute Error (MAE) are often used for regression problems, like for classical neural networks.

While early CNNs were proposed in the 80's, such as the LeNet-5 CNN [18] for hand-written digits recognition, the advent of more capable hardware and the availability of larger scale datasets popularized the use of CNNs in the first part of the 2010's. In particular, the outstanding results obtained by AlexNet [19] on the popular ImageNet image classification dataset [20] in 2012 are often cited as the spark that renewed interest in deep learning, now capable to obtain superior results with respect to classical methods. After AlexNet, many new network structures were developed, progressively improving the classification performance and solving new problems, such as object detection or instance segmentation, as we will see.

In the next sections I will describe the main networks for image classification and object detection that we will encounter in Chapters 3 and 4.

2.2 Deep learning for image classification

As explained before, AlexNet was one of the first CNNs to obtain good results on a large scale image classification dataset like ImageNet. AlexNet was composed of 5 convolutional layers, with 3 max pooling layers in between, and 3 final fully-connected layers, with the last one containing 1000 outputs, each predicting the probability that the input image belonged to one of the 1000 classes of ImageNet. AlexNet introduced the use of ReLUs for CNNs.

In 2014, the VGGNet CNNs were proposed [21], with the most famous one being VGG-16. The authors showed that increasing the depth (i.e. the number of layers) of a CNN leads to better results. VGG-16 reached state-of-the-art accuracy on ImageNet. Its structure is shown in Figure 2.2.

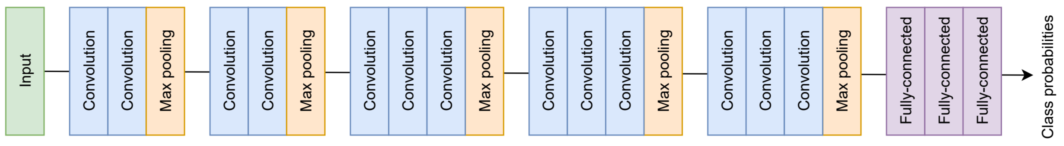


Figure 2.2: Structure of VGG-16. All convolutions have 3×3 kernel size and are followed by a ReLU. All max pooling layers have 2×2 window size and move with a stride of 2 (i.e. the sliding window moves by 2 positions for each step instead of 1). All fully-connected layers are also followed by ReLU, and softmax activation is used on top of the last one. The network has 16 learnable layers, hence the name.

Another important CNN is GoogLeNet [22] (often called Inception-v1, to distinguish it from the later versions). The main concept behind GoogLeNet is the use of the Inception module: instead of simply stacking convolutions like previous CNNs, the Inception module looks at the input features using different receptive fields, by exploiting different convolution kernel sizes. In order to reduce the number of parameters for large kernel convolutions, the authors proposed the use of 1×1 convolutions before the 3×3 and 5×5 ones. The 1×1 convolutions reduced the number of input channels to the larger convolutions, reducing the number of parameters and the computational time. The scheme of the Inception-v1 module is shown in Figure 2.3. GoogLeNet stacked 9 Inception layers, with intermediate max pooling layers and a final fully-connected layer for the classification; a total of about 100 layers were included. Before the fully-connected layer, dropout was applied during training: dropout is a regularization procedure that involves setting to 0 a fraction of randomly-chosen connections between two layers. This forces the network to learn multiple alternative pathways to produce the same classification result, making the network more robust to noise. GoogLeNet obtained better performance than VGG-16 on ImageNet. The authors later proposed improvements to the Inception module, such as Inception-v3 [23], Inception-v4 and Inception-ResNet [24], which introduced factorized convolutions, unified the number of filters in the Inception blocks, and more.

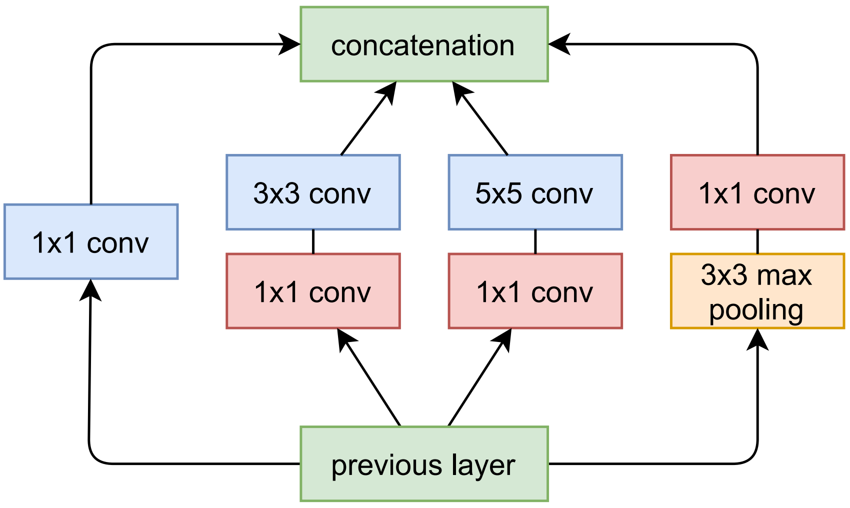


Figure 2.3: Structure of the Inception-v1 module. The red 1×1 convolutions are used to reduce the number of feature channels to speed up computation.

Adding layers to CNNs normally reaches diminishing returns, and after a certain number of layers the performance starts to degrade. To solve this problem, He et al. [1] proposed the use of skip connections in their residual networks (ResNet), as shown in Figure 2.4. Such a constructed network can learn identity functions more easily when needed, leading to faster training and better results. The authors trained various versions of the ResNets, with different number of layers, by repeating the residual blocks a varying number of times: ResNet-18, ResNet-34, ResNet-50, ResNet-101, ResNet-152, where each number represents the amount of parametric layers. Even the deepest network exhibited benefits from the greater depth: increasing the number of layers led to increasingly better results on ImageNet. The networks also used batch normalization, which helps to compensate for distribution shifts during training and stabilizes the training process, reducing the number of training steps required to reach convergence. Some variants of ResNet have been proposed, among which we can mention ResNeXt [25], that introduced grouped convolutions, and Wide ResNet (WRN) [26], that increased the number of channels in each layer.

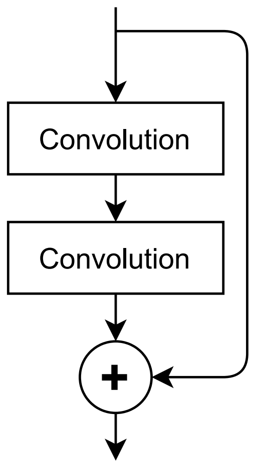


Figure 2.4: Structure of a residual block. A lateral skip connection sums the input to the features computed by the two layers. The authors claimed that this structure allowed the network to learn identity mapping more easily.

Classification CNNs trained on ImageNet have often been used as a starting base for the development of more complex networks for other tasks, such as object detection, as we will se.

2.3 Deep learning for object detection

Object detection is the task of identifying instances of one or more specific classes in an image. For example, we might have object detectors trained to detect people, cars, animals, food, everyday objects. Arguably the most famous CNN for object detection is Faster R-CNN [27]. Evolution of R-CNN [28] and Fast R-CNN [29], Faster R-CNN introduced a Region Proposal network able to produce candidate regions, i.e. regions that could contain an object. The structure of Faster R-CNN is summarized in Figure 2.5. In the original article, VGG-16 is used as a backbone feature extraction network, that is, the convolutional layers from VGG-16 are reused for feature extraction: in this way, the network weights that were pre-trained on ImageNet could be reused to ease convergence on the new task. The Region Proposal Network uses anchor boxes to produce a preliminary set of candidate regions by predicting the probability that each anchor box contains an object. These regions are then classified by the classification head, that re-uses the backbone features cropped with ROI Pooling. The ROI Pooling layer extracts a fixed size feature map from the backbone features according to the spatial location of the region proposal. ROI Pooling allows the network to take images of any size as input without changing its structure and without the need for re-training. The network head also performs bounding box regression. Faster R-CNN obtained state-of-the-art results on the COCO object detection dataset [30]. It is important to note that while the original article used VGG-16 as a backbone network, it is also possible to employ other backbones, e.g. a ResNet.

A later evolution of Faster R-CNN, Mask R-CNN [2], added a segmentation prediction branch to the classification head in order to predict segmentation masks for each of the detected boxes. This task is usually called *instance segmentation*. The authors also developed a more refined ROI Pooling layer (called ROIAlign) to crop the feature maps more accurately.

The introduction of Feature Pyramid Networks (FPN) [31] has been an important step to produce better results, especially in detecting small object.

The main idea is to add additional branches on top of the network, which merge features from earlier layers with features from the last ones. This results in higher-resolution features with the same abstraction and complexity level of the deeper layers, and it allows to use anchors at different scales, helping with the detection of small objects, which can be lost in the smaller feature maps of the last layers.

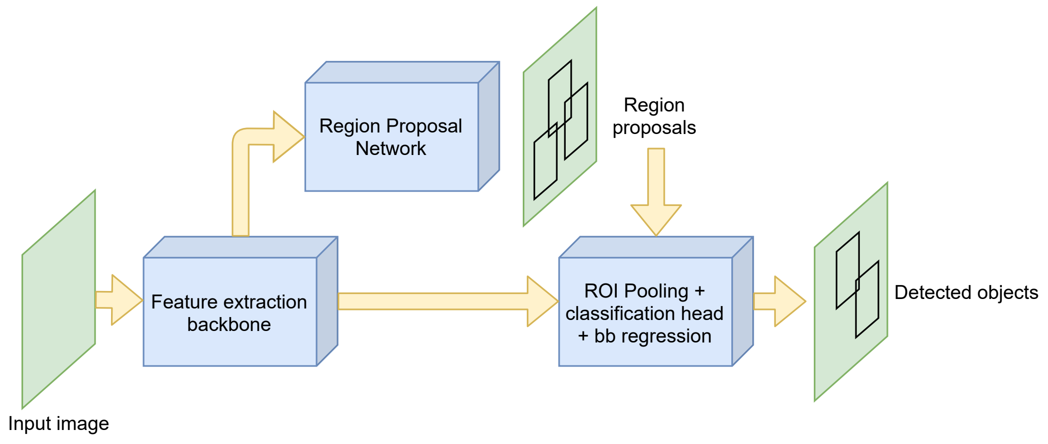

Region Proposal Network
Region proposals
Feature extraction backbone
ROI Pooling + classification head + bb regression
Detected objects
Input image

Figure 2.5: Structure of a residual block. A lateral skip connection sums the input to the features computed by the two layers. The authors claimed that this structure allowed the network to learn identity mapping more easily.

While Faster R-CNN represents an example of two-stage object detector, where the first stage is the region proposal, and the second stage in the region classification, the Single Shot MultiBox Detector (SSD) [32] is instead an example of single-stage object detector. SSD is a fully-convolutional network (i.e. it does not use fully-connected layers, but only convolutional ones), that performs region proposal and classification at the same time, by using a similar concept of anchor boxes. SSD adds additional convolutional layers on top of the VGG-16 backbone, and anchors are defined on each of these different layers, representing objects at different scales in the image. This helped the network to detect smaller objects. While two-stage detectors tend to obtain slightly better results than one-step detectors such as SSD or YOLO [33, 34, 35], the latters tend to be much faster.

Chapter 3

Deep Learning for Multiple Object Tracking

The first part of this thesis focuses on the use of deep learning for Multiple Object Tracking in videos. In particular, I will present an extensive survey on the state of the art of DL-based MOT algorithms. Part of the work described in this chapter was done in collaboration with the team lead by professor Francisco Herrera, from the Universidad de Granada, Spain. Moreover, most of content presented here has been published as a journal article on Neurocomputing [36].

The chapter is organized as follows. Section 3.1 will introduce the problem of MOT, the limitations of existing surveys and reviews, and the main contributions of this thesis. In Section 3.2 I will describe the four main steps performed by most MOT algorithms. Section 3.3 will describe the evaluation metrics usually employed in MOT, while Section 3.4 will present the main datasets used to evaluate MOT algorithms. The main part of the chapter is Section 3.5, which describes the various MOT algorithms that use DL techniques, categorized into the aforementioned four steps. I have also collected the numerical results obtained by the described algorithms on various MOTChallenge datasets: they will be presented in Section 3.6 along with their analysis and the discussion about pros and cons of various approaches. Finally, in Section 3.7 I will summarize the findings and present some future directions of research in MOT.

In addition to the survey work, I also collaborated with the Universidad de Granada in researching possible strategies to improve a MOT tracker. Since results from early experiments were not satisfactory, this work is not described in the main text. A short summary is instead provided in Appendix A.

3.1 Introduction to MOT and contributions of this thesis

Multiple Object Tracking (MOT), sometimes also called Multi-Target Tracking (MTT), is the problem of identifying and tracking multiple objects in videos, including pedestrians, cars, animals, cells, without any prior knowledge about the number of targets or their appearance. Many computer vision problems depend on MOT: video surveillance, autonomous driving, crowd behavior analysis and action recognition are just a few examples. While object detection algorithms, as we have seen in Section 2.3, output a collection of bounding boxes for each input image, without any temporal knowledge, a MOT algorithm outputs a collection of *sequences* of bounding boxes, where each sequence contains bounding boxes indicating the location of a specific object instance throughout time. Thus, a MOT algorithm, in addition to the information on location and size of the bounding boxes, also assigns an ID to each box, that is used to distinguish among different object instances of a certain class throughout the video. Figure 3.1 shows an example of the output of a MOT algorithm applied to pedestrians.

It is important to distinguish between Single Object Tracking (SOT) and MOT algorithms. In the SOT task there is a single target to track, whose appearance is usually provided as input to the algorithm; in MOT, the number and appearance of tracked targets is not known a priori, and a detection step is thus necessary to identify possible new targets entering the scene. Another difficulty of MOT is the need to avoid ID switch errors or target drift when two targets overlap or when a target is temporarily occluded

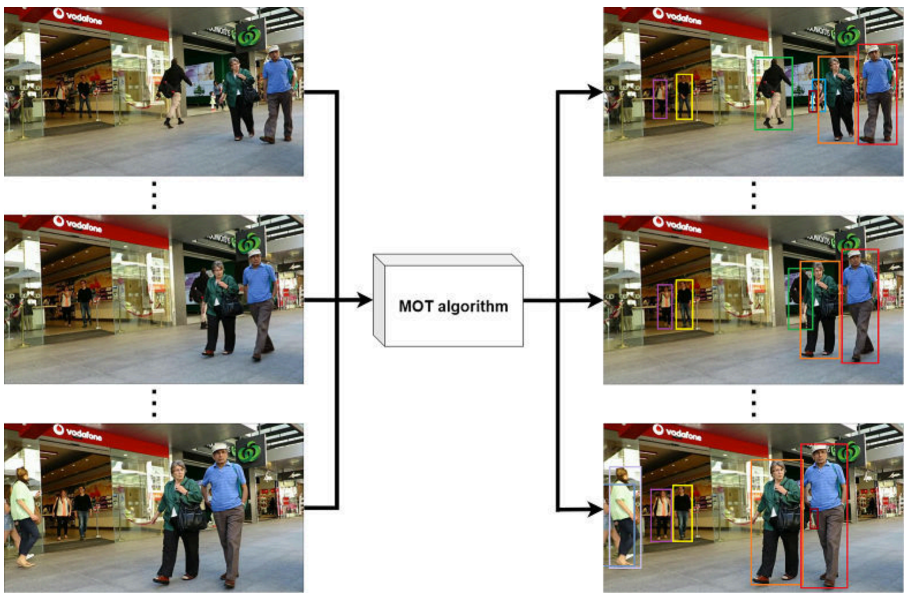


Figure 3.1: Example output of a MOT algorithm tracking pedestrians. The algorithm assigns a different ID to each separate identity, represented as different colors in the images above. Note that people can be occluded or exit the frame, and new people can enter the scene. The frames are taken from the MO17-09 video of the MOT17 dataset (see Section 3.4).

by a background object, although target drift is also a problem with SOT. The necessity to overcome these challenges implies that a naive application of Single Object Tracking algorithms for the MOT problem is not sufficient. For this reason, various competitions and datasets have been presented specifically for the MOT problem, and a lot of work has been done in recent years to develop effective and efficient MOT algorithms.

In particular, many of these new algorithms use deep learning techniques in order to extract powerful features to help with the tracking task. As we have seen in Chapter 2, DNNs are able to learn complex, abstract features which can help distinguish among different instances of the same object class, which is a fundamental aspect of MOT algorithms. For this reason, a survey on the specific use of deep learning in MOT can be useful for future researchers who want to tackle the problem.

While some works in the literature applied MOT on 3D data, for example using RGB-D data (RGB frames with depth information), this thesis will focus on MOT performed on 2D video frames. Moreover, only single-camera MOT will be considered, as opposed to multi-camera MOT, where a scene is simultaneously recorded by different points of view, and targets must be tracked across cameras.

While some surveys and reviews on MOT have been published in the literature, they present some limitations:

- Luo et al. [37] presented, to the best of my knowledge, the first comprehensive review focused on MOT, including a unified formulation of the MOT problem. The authors also described the main components of MOT systems with the related techniques used in the literature. They also presented a performance comparison between various algorithms. However, deep learning had only been used in a limited number of works at that time, so it was not the main focus of their review. The authors predicted that deep learning would be one of the promising future research directions for MOT.

- Camplani et al. [38] presented a survey on Multiple Pedestrian Tracking, but they focused on RGB-D data and did not cover deep learning based algorithms.

- Emami et al. [39] proposed a formulation of the data association step in MOT algorithms as a Multidimensional Assignment Problem (MDAP) and mostly focused on that specific step of MOT. While a few DL-based approaches were presented, it was not the main focus of their work and they did not explore the use of DL in the other steps (e.g. the object detection step) of a typical MOT algorithm.

- Leal-Taixé et al. [40] presented an analysis of the results obtained by algorithms on the datasets for the MOT15 [41] and MOT16 [42] challenges. The authors computed statistics about the results obtained on those particular datasets, and identified the main lines of research.

They found that recent algorithms focused on improving the techniques used to compute affinity between identities, as opposed to researching novel optimization methods for the object association step, which was the main goal of earlier works. They correctly predicted that future approaches would tackle this issue by using deep learning. However, this work too did not focus specifically on DL-based techniques, which were still relatively new in the MOT task at that time.

Given the limitations of existing works in the literature, this thesis will present a comprehensive survey about the recent[1] MOT algorithms that employ deep learning. In particular, the main contributions of this thesis are the following:

- the first comprehensive survey on the use of Deep Learning in Multiple Object Tracking, focusing on 2D data extracted from single-camera videos, including works that have not been covered by past surveys and reviews;
- the identification of four steps which are common to most MOT algorithms, together with the description of the various trackers in the literature that utilized DL models for each of these four steps;
- the collection of experimental results for DL-based MOT algorithms on the most common MOT datasets, along with an analysis of the most successful techniques employed in recent years;
- a discussion on the open issues of current trackers and the possible future research directions for MOT algorithms.

[1]Since I collected and analyzed the literature presented in this chapter in 2019, some very recent techniques may not be included in this thesis; however, the general categorization of MOT algorithms and most of the related observations are still valid and I hope they can be useful for researchers that want to use or improve existing trackers or develop a novel one.

3.2 Structure of a MOT algorithm

The majority of MOT algorithms follow the so-called *tracking-by-detection* approach: a set of bounding boxes (usually just called *detections* in the context of MOT), each locating a target in the input frames, are extracted from the video. These detections are then used to guide the tracking, which usually includes an association process that groups detections belonging to same person or object, and assigns them the same ID. For this reason, many MOT algorithms formulate the task as an assignment problem. Since modern detection frameworks [27, 2, 43, 32, 34] usually obtain high quality results, most researchers have focused their efforts in improving the association process, and many MOT datasets provide optional pre-computed detections that enable a fair comparison between different association algorithms.

MOT algorithms can be divided into batch and online methods. Batch algorithms look at data from the entire video to perform the detection association: for this reason, the ID of a certain detection in frame k can be determined by using information from both past and future frames. Online algorithms instead only use present and past information to determine the IDs of the detections in frame k. Since batch algorithms have access to more information with respect to online ones, they tend to perform better in general. However, many real-world scenarios require the use of an online algorithm; for example, autonomous driving requires that the tracking is done in real-time, and cannot obviously wait for all future frames to be collected. It is important to distinguish online MOT algorithm from real-time ones: while every real-time algorithm must be online, since it cannot access future information, not every online algorithm is necessarily able to run in real-time. As we will see, in fact, many online DL-based algorithms still struggle to reach real-time execution, since deep networks often require heavy computation.

While many different MOT algorithms are described in the literature, with a great variety of employed techniques, most of them follow (all or part of) these four steps (see Figure 3.2):

1. **detection stage**: an object detection algorithm analyzes each input frame to identify objects belonging to the target class(es) using bounding boxes, also known as *detections* in the context of MOT;

2. **feature extraction/motion prediction stage**: one or more feature extraction algorithms analyze the detections and/or the candidate tracklets[2] to extract appearance, motion and/or interaction features. In some cases, a motion predictor predicts the next position of each tracked target;

3. **affinity computation stage**: the computed features and the predicted trajectories are used to compute a similarity/distance score between pairs of detections and/or candidate tracklets;

4. **association stage**: the similarity/distance measures are used to associate detections and tracklets belonging to the same target by assigning them the same ID.

Batch methods tend to apply this sequence of four stages once for the entire video, running each stage on all the video frames at once; online methods instead usually iterate these four stages for each incoming frame. It is important to note that this four-stage classification is not rigid by any means, as many algorithms merge or mix some of these steps, or run them multiple times per "iteration" (e.g. two-phase algorithms with a pre-association step that gets refined in a second association stage). In addition, some algorithms do not directly associate detections, but use them to refine the trajectory predictions output by a motion model or just to manage the initialization and termination of tracks. Even in these cases, however, it is often possible to clearly identify many or all of the four described stages.

[2]The term "tracklet" is often used in the literature to indicate a short track or a fragment of a track.

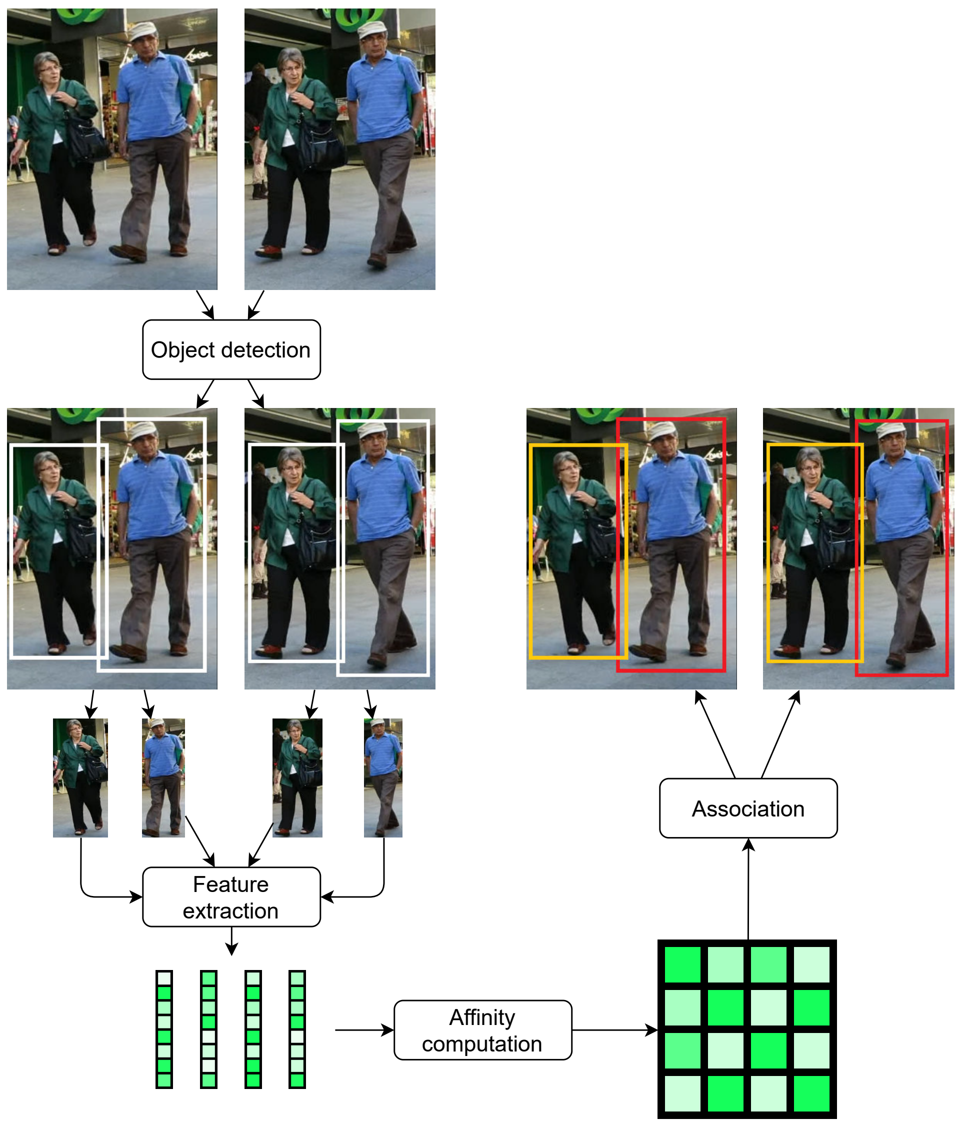


Figure 3.2: The four steps of a MOT algorithm. First, object detection is performed on the video frames, in this case to detect pedestrians. The bounding boxes are used to extract visual or motion features (represented as colored feature vectors in the figure), which are then compared to each other in order to compute some kind of affinity score (the colored distance matrix in the image). The affinities are then fed to an association algorithm that decides which detections belong to the same person, and assigns them the same ID (each ID is represented as a different bounding box color in the figure).

3.3 Evaluation metrics in MOT

In order to evaluate the performance of a MOT algorithm, a number of metrics have been proposed throughout the years. We can group the most commonly used ones in three main groups: a set of "classical" metrics [44], the CLEAR MOT metrics [45] and the ID metrics [46].

3.3.1 Classical metrics

Proposed by Wu and Nevatia [44], this set of metrics can give useful information about the type of errors made by a MOT algorithm. They are the following:

- *Mostly Tracked* (MT) trajectories: the number of ground-truth trajectories that are correctly tracked in at least 80% of the frames.
- *Mostly Lost* (ML) trajectories: the number of ground-truth trajectories that are correctly tracked in less than 20% of the frames.
- *Fragments*: number of predicted trajectories which cover at most 80% of a ground truth trajectory. Observe that a true trajectory can be covered by more than one fragment.
- *False trajectories*: the number of predicted trajectories which do not correspond to a ground truth trajectory.
- *ID switches*: the number of times a tracked object has its associated ID mistakenly changed.

3.3.2 CLEAR MOT metrics

The CLEAR MOT metrics were developed for the *Classification of Events, Activities and Relationships* (CLEAR) workshops held in 2006 [47] and 2007 [48]. They consist of the Multiple Object Tracking Accuracy (MOTA) and Multiple Object Tracking Precision (MOTP). The goal was to define two unified metrics which can summarize the performance of a MOT algorithm.

First of all, it is necessary to decide how to match track hypotheses with ground truth tracks. The most used criteria is the one proposed for the original 2015 MOTChallenge [41, 45]. The Intersection over Union (IOU) is used to match predicted and ground truth boxes. It is computed as the ratio between the area of the intersection of two bounding boxes and the area of their union.

The mapping between ground truth tracks and hypotheses follows the so-called continuity constraint: if the ground truth object o_i and the hypothesis h_j are matched in frame $t-1$, and $IoU(o_i, h_j) \geq 0.5$ in frame t, then o_i and h_j are also matched in frame t, even if there exists another hypothesis h_k such that $IoU(o_i, h_j) < IoU(o_i, h_k)$. In other words, once a predicted track and a ground truth trajectory are matched, the match is kept until their IOU falls under the 0.5 threshold. After existing tracks from previous frames have been matched, the remaining ground truth objects are matched with the remaining hypotheses, using again the 0.5 IOU threshold. At the end of the process, the ground truth bounding boxes that are not associated to any predicted box are *false negatives* (FN), while the predicted boxes that are not associated to any ground truth box are the *false positives* (FP). Every time the ID associated to a tracked ground truth object is incorrectly changed an *ID switch* (IDSW) is counted. The authors also defined the concept of fragmentations[3] (Fragm), which is the number of times the tracking of a ground truth object is interrupted for a number of frames and then resumed at a later point.

Given the previous definitions, the MOTA score obtained by an algorithm on a video is defined as follows:

$$MOTA = 1 - \frac{(FN + FP + IDSW)}{GT} \quad \in (-\infty, 1],$$

where GT is the number of ground truth boxes. If the algorithm makes no errors, that is it does not produce any false positive, false negative or ID switch, MOTA is equal to 1. It is important to note that the score can be negative, as the algorithm can commit a number of errors greater than the number of

[3]Not to be confused with the *fragments* described in Section 3.3.1.

ground truth boxes.

On the other hand, MOTP is computed as:

$$MOTP = \frac{\sum_{t,i} IOU(t,i)}{\sum_t c_t},$$

where c_t denotes the number of matches in frame t, and $IOU(t,i)$ is the IOU between the hypothesis i and its assigned ground truth object (false positive hypotheses are ignored). MOTP is thus a measure of the average localization accuracy of the tracking algorithm. In tracking-by-detection approaches, this is usually highly dependent on the quality of the object detector.

3.3.3 ID metrics

The MOTA score highligths the number of times a tracker takes an incorrect decision (e.g. an ID switch), but in some scenarios, such as surveillance, one might be more interested in rewarding a tracker that can follow an object for the longest possible time, since it is very important to avoid losing its location. Because of that, Ristani et al. [46] proposed alternative metrics, that are supposed to complement the information given by the CLEAR MOT metrics.

Instead of matching ground truth and detections frame by frame, the mapping is performed globally, and the trajectory hypothesis assigned to a given ground truth trajectory is the one that maximizes the number of frames correctly classified for the ground truth. To solve that problem, a bipartite graph is constructed, weighing the edges according to the number of false negatives and false positives the corresponding match would generate. Some "dummy" false positive and false negative nodes are also added to the graph. The minimum cost bipartite matching problem is then solved on the graph to obtain the prediction/ground truth matches. Each matched pair of nodes can be a *true positive ID* (if a ground truth trajectory node was matched with a predicted trajectory node), a *false positive ID* (if a predicted trajectory node was matched to a false positive node) or a *false negative ID* (if a ground truth trajectory node was matched to a false negative node).

Three scores are then computed:

- $IDTP$: the sum of the weights of the edges representing true positive ID matches (equivalent to the percentage of correctly assigned detections);
- $IDFP$: the sum of the weights of the false positive ID edges;
- $IDFN$: the sum of the weights of the false negative ID edges.

These serve to define the final three ID measures that are used to evaluate a MOT algorithm:

- Identification precision: $IDP = \frac{IDTP}{IDTP+IDFP}$
- Identification recall: $IDR = \frac{IDTP}{IDTP+IDFN}$
- Identification F1: $IDF1 = \frac{2}{\frac{1}{IDP}+\frac{1}{IDR}} = \frac{2IDTP}{2IDTP+IDFP+IDFN}$

Since the majority of works in the literature evaluated their results on at least one of the various MOTChallenge datasets (see Section 3.4), the most commonly used metrics are the ones that are also used in the MOTChallenge leaderboard: mainly MOTA, IDF1 and MOTP, but often accompanied by the other CLEAR MOT metrics, MT and ML, which can give some useful information that complements the aforementioned metrics.

3.4 MOT datasets

In recent years, a number of MOT datasets, usually tied to competitions, have been published.

MOTChallenge. MOTChallenge[4] is the most commonly used benchmark for MOT in the literature. Various versions of MOTChallenge datasets that focused on pedestrian tracking were published throughout the years, and are among the largest publicly available datasets for MOT. In addition to providing ground truth for the training videos, the authors also provide pre-computed detections for both training and test videos. These

[4]https://motchallenge.net/

so-called *public detections* are useful to evaluate the performance of a tracking algorithm independently from the quality of the object detector, since it usually has a major impact on the performance of a tracker. For this reason, the MOTChallenge leaderboard distinguishes between results obtained with *public detections*, for which the score mostly depends on the quality of the association procedure, and results obtained with *private detections* (i.e. detections obtained with a custom object detection algorithm), which account for the quality of the entire tracking pipeline. The leaderboard also distinguishes among online and batch algorithms, for the considerations presented in Section 3.2. All the algorithms in the leaderboard are evaluated on the test sets by submitting the tracking results to the test server, since ground truth for the test sets are not public. MOTA is the primary evaluation score for the MOTChallenge, but many other metrics are computed and shown on the leaderboard, including the ones presented in section 3.3. Since most DL-based MOT trackers in the literature focus on pedestrians, the MOTChallenge datasets are the most widely used, as they provide enough data to train deep models because of their size.

MOT15. The first MOTChallenge dataset is the 2D MOT 2015[5] [41] (often shortened to MOT15). It contains 22 videos (11 for training and 11 for testing), collected from older datasets, with a different visual properties (fixed and moving cameras, different environments and lighting conditions, and so on) in order to test the robustness of the tracking algorithms in different situations. In total, it contains 11,283 frames at various resolutions, with 1221 different identities and 101,345 boxes. The provided detections were obtained using the Aggregated Channel Features (ACF) detector [49].

MOT16/17. A new version of the dataset was presented in 2016, called MOT16[6] [42]. Differently from MOT15, the authors annotated the dataset from scratch using a consistent protocol across the videos. MOT16

[5]Dataset: `https://motchallenge.net/data/2D_MOT_2015/`, leaderboard: `https://motchallenge.net/results/2D_MOT_2015/`.

[6]Dataset: `https://motchallenge.net/data/MOT16/`, leaderboard: `https://motchallenge.net/results/MOT16/`.

contains 14 videos, 7 for training and 7 for testing, with a higher pedestrian density compared to MOT15, posing a greater challenge to MOT algorithms, which have to deal with a greater number of possible occlusions. The public detections included with MOT16 were obtained using Deformable Part-based Model (DPM) v5 [50, 51]. MOT16 includes a total of 11,235 frames with 1,342 identities and 292,733 boxes. The MOT17 dataset[7] includes the same videos as MOT16, but with more accurate ground truth and with three sets of detections for each video: one from Faster R-CNN [27], one from DPM and one from the Scale-Dependent Pooling (SDP) [52] algorithm. Using detections from multiple sources to compute the finale average MOTA score forces the trackers to be more robust to lower-quality detections in order to obtain competitive scores.

MOT20. The MOT20 dataset [53] was recently released, first as part of the CVPR 2019 Tracking Challenge[8] [54], and then as a public dataset on the MOTChallenge website[9]. It contains 8 videos (4 for training, 4 for testing) with extremely high pedestrian density, reaching up to 245 pedestrians per frame on average in the most crowded video. The dataset contains 13,410 frames with 6,869 tracks and a total of 2,259,143 boxes, much more than the previous datasets.

KITTI. Differently from MOTChallenge, the KITTI tracking benchmark[10] [55, 56] focuses on both pedestrian and vehicle tracking, which are both needed to tackle the challenging task of autonomous driving. The dataset was collected by driving a car around a city and it was released in 2012. It contains 21 training videos and 29 test videos, with a total of about 19,000 frames (32 minutes total length). Similarly to MOTChallenge, KITTI also

[7]Dataset: https://motchallenge.net/data/MOT17/, leaderboard: https://motchallenge.net/results/MOT17/.

[8]https://motchallenge.net/workshops/bmtt2019/tracking.html

[9]Dataset: https://motchallenge.net/data/MOT20/, leaderboard: https://motchallenge.net/results/MOT20/. This dataset was originally called MOT19 by the authors, and figures as MOT19 in [36].

[10]http://www.cvlibs.net/datasets/kitti/eval_tracking.php

provides pre-computed detections, that were obtained using the DPM[11] and RegionLets[12] [57] detectors. KITTI also provides stereo and laser information, but models that used this kind of data were not considered in this thesis, since the focus is on pure 2D data. Models are evaluated using the CLEAR MOT metrics, MT, ML, ID switches and fragmentations. The pedestrians and vehicle tracking tasks are considered separate challenges, and a leaderboard for each of the two is provided.

Other datasets. Before MOTChallenge, a number of other smaller tracking datasets were used. Some of the most commonly used ones are PETS2009[13] [58] (pedestrian tracking), TUD[14] [59] (pedestrian tracking) and the UA-DETRAC tracking benchmark[15] [60] (vehicle tracking in videos recorded by traffic cameras). Many of the videos in PETS2009 and TUD are integrated into MOTChallenge.

3.5 Deep learning in MOT

In this section we are going to explore the DL-based techniques employed in the literature for each of the four steps described in Section 3.2, grouping similar models together. I remind the reader that, as explained previously, the classification is not rigid, since not every algorithm strictly follows the linear order of the four steps, and some DL algorithms can play a role in multiple stages. However, the categorization can still be useful to highlight the purpose for which each DL model was used.

All of the methods described in Sections 3.5.1, 3.5.2, 3.5.3 and 3.5.4 have been summarized in a table in Appendix B.

[11] The L-SVM (latent SVM) model mentioned on the website is now known as Deformable Parts Model (DPM).

[12] http://www.xiaoyumu.com/project/detection

[13] http://www.cvg.reading.ac.uk/PETS2009/a.html

[14] https://www.d2.mpi-inf.mpg.de/node/428

[15] https://detrac-db.rit.albany.edu/Tracking

3.5.1 Deep learning in the detection step

While many MOT algorithms in the literature have used pre-computed detections from public datasets, such as MOTChallenge and KITTI, some decided instead to train a CNN in order to improve the overall accuracy of the tracker. As mentioned before, in fact, the quality of the detections impacts significantly on the accuracy of a tracking algorithm, since the reduced number of false positive and false negative detections translates into a lower number of false positive and false negative track boxes, and thus into a higher MOTA.

Faster R-CNN and derivatives

SORT. One of the first MOT algorithms to use a CNN to detect pedestrians was Simple Online and Realtime Tracking (SORT) [61]. Replacing the MOT15 public detections obtained using ACF with detections computed by Faster R-CNN increased the MOTA score by 18.9% and SORT reached state-of-the-art performance on the dataset at that time. SORT used the Kalman filter [62] to predict object motion; the predicted positions were then associated to detections in the frame by using the Hungarian algorithm [63] with a cost matrix consisting of the IOU between each predicted box and the detections.

POI. Yu et al. instead used a modified version of Faster R-CNN in their Person of Interest (POI) detector [64]. The authors added skip pooling [65] and multi-region features [66] to Faster R-CNN in order to combine features at different scales and abstraction levels, and trained the network on multiple pedestrian detection datasets, including ETHZ [67], the Caltech pedestrian dataset [68] and a self-collected surveillance dataset. The structure of the detectors is summarized in Figure 3.3. The use of this enhanced detector allowed the authors to improve the MOTA by 30% with respect to using the public detections for MOT16. The authors also noted that using more accurate detections allows to reduce the complexity of the tracking algorithm without losing accuracy. The detections computed on MOT16 by this modified

Faster R-CNN were published[16] and many other researchers used them in their algorithms [69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79].

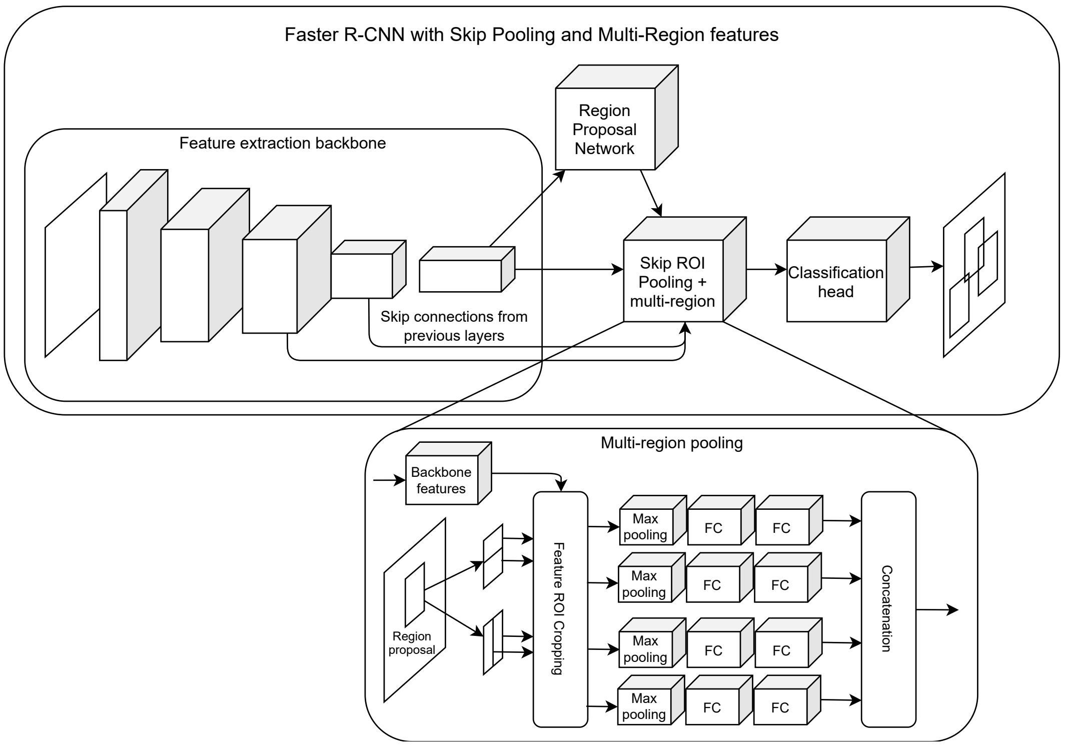

Faster R-CNN with Skip Pooling and Multi-Region features
Feature extraction backbone
Region Proposal Network
Skip ROI Pooling + multi-region
Classification head
Skip connections from previous layers
Multi-region pooling
Backbone features
Region proposal
Feature ROI Cropping
Max pooling
FC
FC
Concatenation

Figure 3.3: Architecture of the pedestrian detector used in POI. The image is simplified and does not show the actual number of layers, for lack of space. The classic structure of a Faster R-CNN, with a backbone feature extraction, a RPN with ROI Pooling and a classification head is preserved. However, two additional features are introduced: skip pooling [65], that pools features from multiple layers in order to look at different scales and abstraction levels; multi-region features [66], that are extracted from different crops of the region candidate and force the network to look at object parts and context, besides making it more sensitive to localization inaccuracies.

HOGM. Zhou et al. [80] used Mask R-CNN [2], a variant of Faster R-CNN, as explained in Section 2.3, in the detection step of their HOGM tracker. The network was used to extract masks in addition to the bounding boxes, in order to avoid the inclusion of background noise in the feature extraction step. They showed that extracting features from masks, as opposed

[16] https://drive.google.com/file/d/0B5ACiy41McAHMjczS2p0dFg3emM/view

to the entire bounding box, led to MOTA improvements in both their HOGM algorithm and DeepSORT (described in Section 3.5.2).

Other uses of Faster R-CNN. Faster R-CNN is also part of many other MOT algorithm, not only for pedestrians, but also to track athletes [81], cells [82] and pigs [83], following an appropriate re-training of the network to detect objects of the appropriate classes.

SSD

Pig tracking. Zhang et al. [83] compared the use of three different object detectors in a pig tracking pipeline: SSD, Faster R-CNN and R-FCN [43]. SSD led to better results on their dataset. This was probably due to the higher number of anchors in SSD and to its matching strategy, that allows to predict high scores even for highly-overlapping boxes. This is desirable in the pig detection task, since pigs often occlude each other in the images and their boxes overlap significantly. Moreover, SSD was also two to three times as fast as the other two CNNs. After detection, the pigs were then tracked using the ECO tracker [84], applied to small regions at the center of each pig called *tag-boxes*, in order to increase efficiency, since the animals have very similar appearances. The tracker used HOG [85] and Colour Names [86] features, and the Hungarian algorithm was used to associate the tracked tag-boxes and the actual pig detections.

Joint Detection and Tracking Kieritz et al. [87] proposed the use of SSD with a modified NMS step, which uses the affinity scores computed between tracklets and each candidate detection in order to refine the confidence score of each candidate, before NMS. In this way, the algorithm avoids to filter out low-score detection candidates which actually match an existing track. The confidence scores were re-weighted by using a multi-layer perceptron with two hidden layers.

CCF-based SSD Zhao et al. [88] used a CNN-based Correlation Filter (CCF) to predict the position of an object (a pedestrian or a vehicle) in the next frame; a box centered in this position was then cropped and fed to SSD, which was thus able to detect smaller objects in the frame. These detections were combined with the ones obtained from the full frame and then track-to-detection association was performed using the Hungarian algorithm. The latter used a cost matrix which accounted for both IOU and appearance features (Average Peak-to-Correlation Energy - APCE [89]). Appearance features were also used for object re-identification (ReID), in order to recover from occlusions. The network, trained with multi-scale augmentation, obtained results comparable to state-of-the-art online algorithms on both MOT15 and KITTI.

Other uses of SSD. Lu et al. [90] used SSD in their a_LSTM tracking algorithm (explained in more detail in Section 3.5.2), which was able to track multiple object classes, including people, animals and cars.

Other CNNs

In addition to Faster R-CNN and SSD, we can find a number of other CNNs employed in the detection stage of MOT algorithms. For example, Kim et al. [91] used YOLOv2 [34] to detect pedestrians, Sharma et al. [92] used the Recurrent Rolling Convolution (RRC) CNN [93] and SubCNN [94] to detect vehicles in videos (see section 3.5.2), Pernici et al. [95] used the Tiny CNN detector [96] in their face tracking algorithm, obtaining a better performance when compared to non-deep algorithms such as DPM [50].

Other uses of deep learning in the detection step

Instead of using a network to directly produce detections, some works used CNNs in the detection step in a different way.

Detection pruning For example, Min et al. [97] proposed a vehicle tracking algorithm in which the main detection work was performed using a modified version of the ViBe algorithm [98], which basically performs background subtraction of the input frame. Boxes were computed around the remaining foreground pixels, and were first pruned by a SVM [99], and then by the Faster-CNN based network proposed in [100]. This resulted in faster detection because the CNN was only used when the SVM did not have a high enough confidence in the prediction.

Instance segmentation Bullinger et al. [101] used instead a Multi-task Network Cascade (MNC) [102] to obtain instance-aware semantic segmentation maps, instead of bounding boxes, which were not used at all in the pipeline. This was done in order to remove background pixels from each object instance. An optical flow tracker was used [103, 104, 105], and the locations predicted by it were then matched to the detected instances in the new frame by using the Hungarian algorithm. The authors claimed that since the tracking algorithm was based on optical flow, the removal of background noise would be particularly beneficial for the quality of the results, especially in the presence of camera motion. In fact, when using MNC segmentations as detections, the tracker reached better performance than SORT on MOT15 videos with camera motion.

3.5.2 Deep learning in the feature extraction/motion prediction step

The main strength of deep learning lays in its ability to extract powerful, representative features. It should not surprise the reader that the main stage in a MOT pipeline where deep learning algorithms are employed is thus the feature extraction stage.

The decision to couple feature extraction and motion prediction as a single stage stems from the fact that the latter can be seen as the computation of motion features. Both help with the affinity computation and association processes, and thus play a similar role.

Autoencoders

One of the first MOT algorithms that used deep learning was published in 2014 by Wang et al. [106]. It used a two-layer network of autoencoders that was pre-trained on a set of natural scene videos [107]. A multi-task learning strategy was then used to update the features according to the appearance of the specific target objects. Affinity between objects was computed with a SVM, and the association was performed by solving a minimum spanning tree problem. The authors showed that fine-tuning features for each target using a multi-task approach led to improved model performance.

Apart from this early approach, subsequent works mostly employed CNNs instead of autoencoders.

CNNs for appearance feature extraction

MHT-DAM. Kim et al. proposed one of the first uses of a CNN as visual feature extractor [108]. They used a pre-trained R-CNN in order to extract 4096-dimensional feature vectors from each detection, compressed to 256 dimensions using PCA. The appearance model was updated online using Multi-output Regularized Least Squares (MORLS) [109]. The tracker was based on the Multiple Hypothesis Tracking (MHT) algorithm [110]. The use of deep features improved the MOTA score on MOT15 by 3%, and the model reached state-of-the-art results on that dataset at the time, with 32.4% MOTA.

POI. The already mentioned POI algorithm [64] (see Section 3.5.1) used a CNN similar to GoogleNet [22], pre-trained on multiple person re-identification datasets, in order to extract visual features. Affinity between detections was computed using the cosine distance, and two different association algorithms were proposed, an online method, using the Hungarian algorithm, and a batch one, using a modified version of the H2T algorithm [111]. The tracker reached top performance on MOT16 with 66.1% MOTA.

DeepSORT. Wojke et al. improved the performance of the SORT algorithm by empowering it with deep features in their DeepSORT tracker [69]. They used a Wide Residual Network (a variation of ResNet with less depth but increased width, i.e. number of channels) [26] to extract 128-dimensional features. The cosine distance between the features was used to complement the other affinity scores used in SORT. The structure of the pipeline is summarized in Figure 3.4. The use of deep features helped reducing the number of ID switches by almost a half.

Fusion with pose information. Some algorithms fused the visual features with pose information. For example, Ran et al. fused CNN visual features with pose features extracted with the AlphaPose CNN [112] in their Multi-Athlete Tracking (MAT) algorithm [81], while Tang et al. combined the body part score maps obtained by the DeepCut pose detector [113] with the cropped pedestrian images as input to a feature extraction CNN. Both these algorithms will be described in more detail in section 3.5.3.

Cell tracking. Hu et al. proposed the use of a Fast R-CNN with VGG-16 backbone in order to track cells in microscopy image sequences [82]. After separating slow and fast cells, visual features extracted from the Fast R-CNN were fused with motion features in order to track fast-moving cells more accurately. The CNN was trained for cell classification. After the main tracking process, a quality enhancement step was performed by combining partial tracklets in order to reduce the number of false positives and false negatives.

Random fern classifier. Kim et al. [91] used the features extracted by a YOLOv2 CNN object detector to build a random ferns (RF) classifier [114]. The algorithm worked in two steps. In the first step, a so-called teacher-RF was trained in order to differentiate pedestrians from non-pedestrians. After the teacher-RF was trained, for every tracked object a smaller student-RF classifier was constructed. These were specialized in distinguishing their specific tracked object from the other objects in the scene. The student models help reduce the computational complexity of the algorithm, for use in real-time scenarios.

DSGM. The Directed Sparse Graphical Model (DSGM) by Ullah et al. [115] used a Hidden Markov Model [116] to predict the position of each object in the next frame. A CNN was used to extract visual features, and the affinity computation was performed only among the pairs with a low enough distance between the predicted HMM position and the detection. Mutual information was used to compute affinity between visual features and a dynamic programming algorithm was employed to solve the association

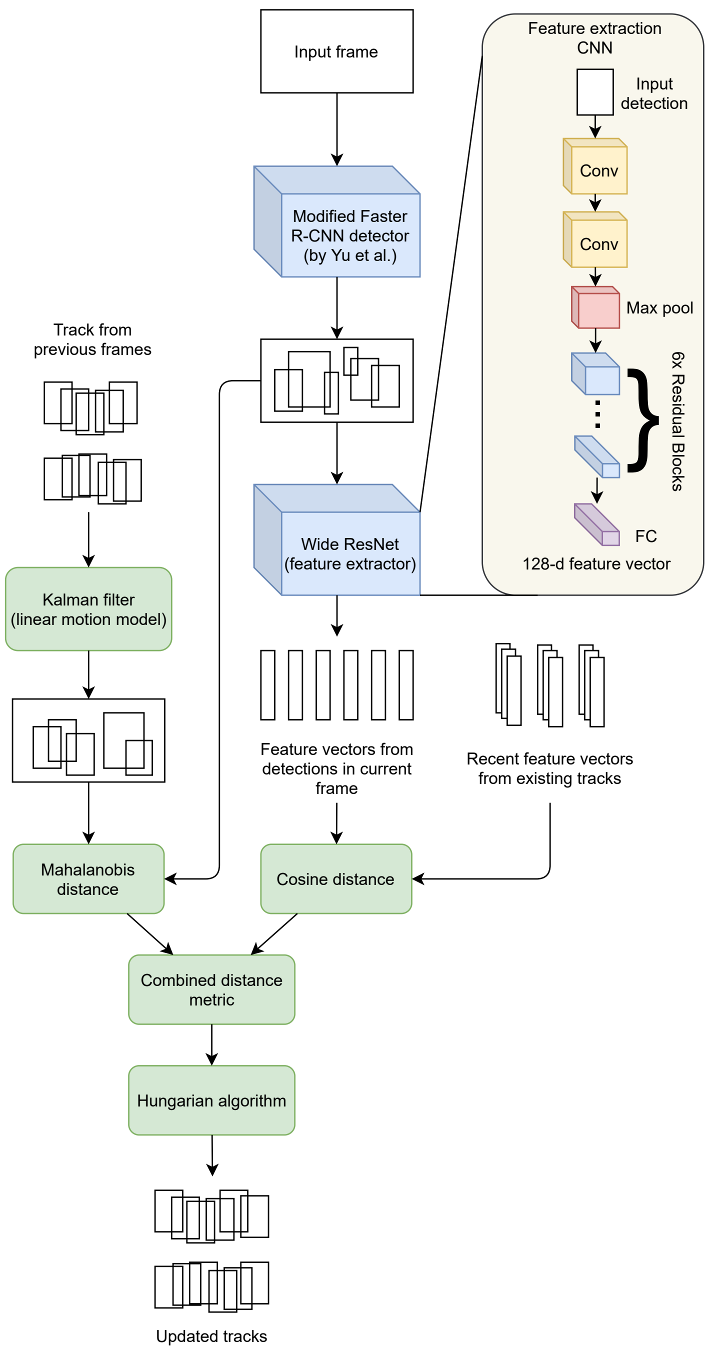

Input frame
Modified Faster R-CNN detector (by Yu et al.)
Feature extraction CNN
Input detection
Conv
Conv
Max pool
6x Residual Blocks
FC
128-d feature vector
Track from previous frames
Wide ResNet (feature extractor)
Kalman filter (linear motion model)
Feature vectors from detections in current frame
Recent feature vectors from existing tracks
Mahalanobis distance
Cosine distance
Combined distance metric
Hungarian algorithm
Updated tracks

Figure 3.4: Architecture of the DeepSORT tracking pipeline. Detections from [64] are fed to a 10-layer CNN that extracts visual features from each of them. These are compared to historical appearances of existing tracks using cosine distance. At the same time, box predictions by a Kalman filter are used to compute Mahalanobis distance with the detections. The two distances are fused into a unified metric, used to perform association.

problem.

Other uses of CNNs as visual feature extractors. Many more algorithms employed CNNs as visual feature extractors. Chen et al. used a custom CNN to extract appearance features in their MHT-based EDMT algorithm for pedestrian tracking [117], while Yang et al. used Fast R-CNN in their HybridDAT pedestrian tracker [118]. Wang et al. [119] used a CNN to extract features from fish heads, employed to track fish with the help of motion predictions obtained with a Kalman Filter. Other models combined CNN visual features with other types of features, such as CNNMTT [70], which fused them with dynamic and position features, and CDA_DDAL [120], which combined the appearance features with shape and motion models.

Pernici et al. [95] decided to reuse the features extracted by the face detection CNN in their face tracking pipeline. Matching was performed using Reverse Nearest Neighbour [121].

GoogLeNet features were used in pedestrian tracking models such as EETN [122], RAN [123], NT [124], or HAF [125].

ResNet was employed in pedestrian tracking algorithms such as MHT-bLSTM [126], an evolution of MHT-DAM employing a Bilinear LSTM to help with affinity computation (more details in Section 3.5.3), and BMH-RMNet [127], which also combined it with a LSTM, which performed bounding box regression in addition to affinity computation. The same Wide ResNet used in DeepSORT was instead employed by Fu et al. [128] in their PHD-DCM tracker, that used a discriminative correlation filter in order to compute correlation between features. This correlation was combined with a spatio-temporal relation score for use as a likelihood in a Gaussian Mixture Probability Hypothesis Density filter [129].

Siamese networks

A special kind of CNNs which are often used to compute reliable features, able to accurately discriminate between different objects, are the Siamese CNNs [130] (see Figure 3.5). Siamese CNNs are usually trained using two parallel convolutional backbones, sharing parameters, each taking a different image as input, and with a number of fully-connected layers on top that are trained to predict the probability that the two input images contain the same object instance. It is important to note that during normal operation, after they have already been trained, standard siamese networks only take a single image as input. While the output probability score can be directly used as an affinity measure, the methods described in this section instead discard the final prediction layer, and just use the feature vectors to compute object affinities using other strategies, possibly in combination with other types of features. The use of siamese networks for affinity computation is described instead in Section 3.5.3.

ESNN. Kim et al. [131] proposed the Enhanced Siamese Neural Network (ESNN) for their pedestrian tracking algorithm, trained using a contrastive loss. In addition to the two input images, the network also used the IOU between them and their area ratio. The Euclidean distance between the features was used to compute the affinity scores between boxes, together with IoU score and the area ratio between the boxes. The association step was solved using a greedy algorithm.

CNNTCM. Wang et al. [132] trained a small Siamese CNN using a margin-based loss in their CNNTCM pedestrian tracker. Affinity between detections was computed using the Mahalanobis distance, weighted with a learned weight matrix, and taking temporal constraints into account.

SymTriplet. Zhang et al. [133] proposed the use of a SymTriplet loss function for the training of their Siamese CNN. Instead of the classic two CNN branches, they used three CNNs with shared weights, taking as input an anchor image, a positive image and a negative image. The SymTriplet loss penalized a high distance between feature vectors of the positive pair (since the images belonged to the same person), while also penalizing a low distance between features of the negative pair (since the images belonged to different persons). This allowed to obtain more discriminative features, which helped with the tracking process[17]. The algorithm was tested on videos from TV shows and music videos from YouTube. Since videos included multiple shots, data association between frames in the same shot was first performed, using Euclidean distance between the feature vectors, together with temporal and kinematic information; afterwards, tracklets were merged across shots, using a Hierarchical Agglomerative Clustering algorithm that also used the appearance features to perform the tracklet merging.

Siamese CNN with stacked inputs. Leal-Taixé et al. [134] proposed the use of two stacked images as input to their Siamese CNN, in a unified CNN backbone (see Figure 3.5). Like the standard siamese networks, it output the similarity between the input images at training time, but the advantage was that CNN features were merged between the two images at earlier stages in the pipeline, leading to better performance. The feature vectors were then used as input for a Gradient Boosting model, together with contextual information, in order to get an affinity score between detections. The association step was solved using Linear Programming [135].

Quad-CNN. Son et al. [136] proposed the use of the more complex Quad-CNN for their pedestrian tracking pipeline. As the name suggests, this Siamese-like CNN took four input images: the first three images were from the same person, temporally ordered, and the last one was from a different person. The authors proposed a loss that accounted for increasing dissimilarity between features of the same person at increasing temporal distances, but that still forced a higher distance between positive and negative samples. The CNN branches were included into a more complex network, using bounding box positions to compute the distance between detections and tracklets, and included a bounding box regression branch. The entire network was trained end-to-end. A minimax label propagation algorithm was used to perform association.

[17]I will also discuss about the effectiveness of triplet losses in Chapter 4, when talking about face recognition algorithms.

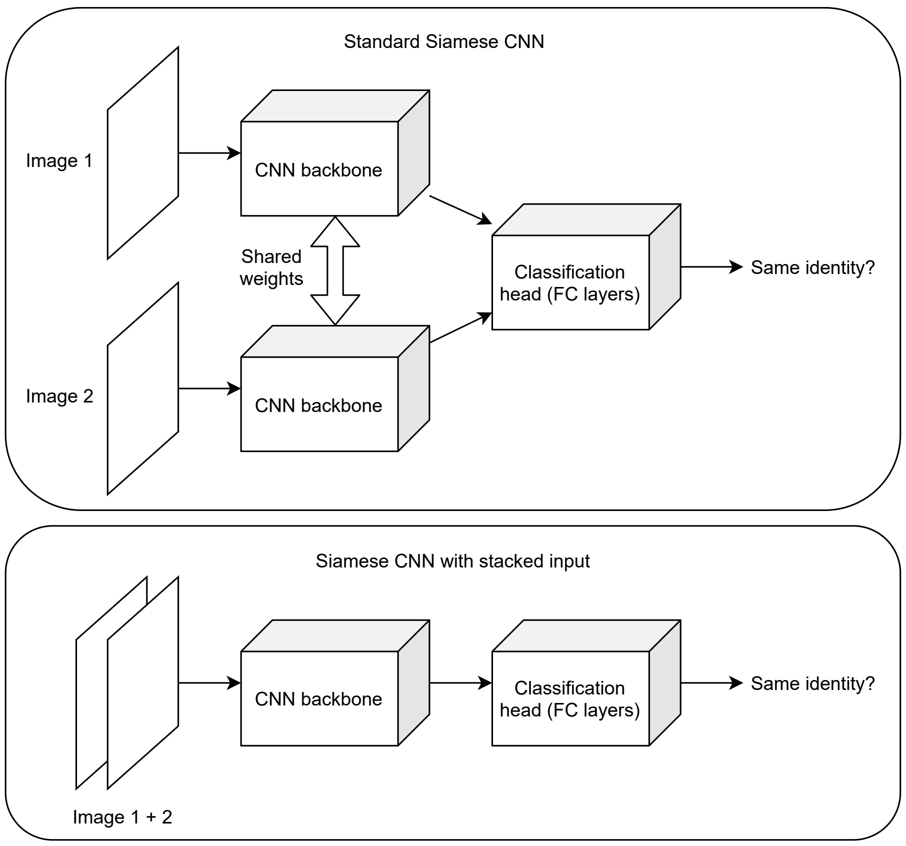


Figure 3.5: Standard Siamese CNN compared to Siamese CNN with stacked inputs. Standard Siamese CNNs run the backbone separately on the two images to be compared, and a classification head uses the features from both images to predict the probability that they contain the same object. A Siamese CNN with stacked inputs is run directly on two stacked images, allowing the convolutional layers to look at both images at the same time.

HOMG. Zhou et al. [80] used a Siamese network based on Mask R-CNN [2] in their High-Order Graph Matching (HOGM) tracker. Three masks extracted by Mask R-CNN were used to train the shallow Siamese net, with the usual anchor, positive and negative samples for the triplet loss. Cosine distance was computed on the 128-dimensional feature vectors extracted from the network, and it was combined with linear and non-linear motion models. The association problem was then solved with a power iteration over a 3D tensor of the computed similarities.

DMAN. Zhu et al. [137] proposed the use of a Spatial Attention Network (SAN), a Siamese CNN with ResNet-50 backbone, in their Dual Matching Attention Networks (DMAN) tracker. The SAN computed a Spatial Attention Map from the last convolutional layers of the model, representing a measure of the importance of different areas in the bounding box, in order to suppress background pixels and other targets from the extracted features, which were weighted by the attention map. The network was jointly trained with a classification task, in order to obtain a better performance. The affinity information was further fed to a bidirectional LSTM, the Temporal Attention Network (TAN), which was the network that actually computed the final affinity scores between pairs of detections (see Section 3.5.3).

DCCRF. Zhou et al. proposed the use of a visual displacement CNN [138] in their Deep Continuous Conditional Random Field (DCCRF) tracker. The CNN learned to predict the next position of an object by taking as input patches from consecutive frames and outputting a displacement map. The model used information about the relationship between the various objects in the scene, as well as the object's past trajectory. At the same time, the network also extracted visual information from the predicted location in the new frame and from the other detections, in order to compute a similarity score, as it will be explained in Section 3.5.3.

MOTDT. Chen et al. [139] used a GoogLeNet CNN trained with triplet loss for feature extraction in their MOTDT tracker. First, a R-FCN [43] was used to predict possible detection candidates using information from the existing tracklets. Second, those predictions were combined with the actual detections and NMS was performed. In the second step, GoogLeNet extracted visual features from the detections, and the association problem was solved with a hierarchical association algorithm. The algorithm reached top performance among online methods in the MOT16 dataset at the time of its publication.

FPSN-MOT. Lee et al. [140] combined feature pyramid networks and Siamese networks for their Feature Pyramid Siamese Networks MOT tracker

(FPSN-MOT). The authors tested the FPSN using either SqueezeNet [141] or GoogLeNet [22] as backbone network, in order to extract visual features from two different images with the usual shared parameters. Afterwards, as explained in Section 2.3, the feature pyramid network combined features at different levels in the CNN in order to produce more discriminative features at the higher granularity of lower layers. In this way, both coarse and fine-level information were used in the network. The affinity computation step is explained in Section 3.5.3.

Other trackers using Siamese CNNs. Maksai et al. [142] used the 128-dimensional feature vectors extracted by the re-identification triplet CNN proposed in [143], and combined it with other appearance-based features. In their SAS tracker, those features were further processed by a bidirectional LSTM for affinity computation (see Section 3.5.3). Ma et al. [144] trained a Siamese CNN as part of their GCRA tracker in order to extract visual features from tracked pedestrians in their model, which is also explained in detail in Section 3.5.4.

Other approaches for appearance feature extraction

Some algorithms used different CNN structures or combined CNNs with other kind of deep networks for appearance feature extraction.

a_LSTM Lu et al. [90] used an LSTM in their a_LSTM algorithm in the process to compute the association features. It used an image descriptor extracted with RoI pooling from each detection, combined with the class predicted by SSD in the detection step. Cosine distance between the LSTM features was used to perform association.

HFM The Hierarchical Feature Model (HFM), proposed by Ullah et al. [145], used the first seven layers of GoogLeNet to learn a dictionary of features of the tracked objects. The first 100 frames of the video were used to learn the dictionary. Orthogonal Matching Pursuit (OPM) [146] was used to reduce feature dimensionality. During the test phase, the OPM representation was computed for every detected object in the scene and was compared with the dictionary in order to construct a cost matrix, combining visual and motion information extracted by a Kalman filter. Association was performed using the Hungarian algorithm.

Multiple RNNs for feature extraction. Sadeghian et al. [147] used three RNNs in their tracker to compute different types of features. The first RNN extracted appearance features. The input of this RNN was a vector extracted by a VGG CNN [21], pre-trained specifically for person re-identification. The second RNN was a LSTM trained to predict the velocity vector of each object. The last RNN was trained to learn the interactions between different objects on the scene, since the position of some objects could be influenced by the behavior of surrounding elements. The affinity computation was then performed by another LSTM, using the information from the other RNNs (see Figure 3.6 in Section 3.5.3).

STAM. The Spatial-Temporal Attention Mechanism tracker (STAM), proposed by Chu et al. [148], used instead a set of stacked CNNs for feature extraction. First, a pre-trained shared CNN extracted common features for every object in the scene. This CNN was not updated online. Then, RoI pooling was applied to extract features specific for each candidate, and a new target-specific CNN was instantiated and trained online. Those CNNs extracted both the visibility map and the spatial attention map for its candidate. The probability of a candidate to belong to each track was then computed, performing association with a greedy algorithm.

Geometry and shape cues for vehicle tracking. Sharma et al. [92] combined appearance features, extracted by a CNN, with 3D shape and position features (computed from the 2D frames assuming a moving camera) in order to compute an appropriate distance metric for their vehicle tracker. In addition to those, 3D-2D distances and 3D-3D distances were computed between 3D projections of bounding boxes and 2D bounding boxes. The Hungarian algorithm was used to solve the association problem.

CNNs for motion prediction: correlation filters

Wang et al. [149] used the convolutional correlation filter (CCF) presented in [150], that computes an estimation map of the position of the object in the next frame. This was combined with optical flow affinity, computed using the Lucas-Kanade algorithm [151], a motion affinity obtained with a Kalman filter, and a bounding box scale affinity. A SVM classifier was used to remove false detections, and the response map from the CCF was also used to handle missing detections.

Zhao et al. [88] also used a correlation filter to predict the future position of the objects. The filter used CNN appearance features, compressed using PCA. The predicted position was later used to compute a similarity score, combining the IoU between prediction and detections, and the APCE score of the response map. The assignment problem was solved using the Hungarian algorithm.

Other relevant approaches

Rosello et al. [152] used a reinforcement learning framework to train a set of agents to use in the feature extraction step. The algorithm (named MARLMOT, from Multi-Agent Reinforcement Learning for MOT) only used motion cues, and did not use any visual information. An agent for each tracked object was used to manage a Kalman filter, deciding when to ignore filter predictions or when to start and stop tracks. While the authors claimed that their algorithm could solve the tracking task with good performance without using visual features, they only reported experimental results on the training set of MOT15. For this reason, the algorithm performance can hardly be compared to other models.

Babaee et al. proposed a post-processing step to deal with errors due to occlusions in pedestrian tracking algorithms [153]. This procedure can be applied on the output of any MOT model. The authors used an LSTM to predict position and size of each tracked object by exploiting position and velocity information from the previous frames. Tracks were then associated by a greedy algorithm, that used the IOU between the track boxes and the predicted positions. The authors showed that the algorithm was able to reduce the number of ID switches for many existing MOT algorithms, thus increasing the MOTA score.

3.5.3 Deep learning in the affinity computation step

While most of the algorithms presented so far used explicit distance measures to compute similarities/dissimilarities between detections and tracks, such as Euclidean or cosine distance, some works in the literature used instead a deep network to learn the affinity function.

Recurrent neural networks and LSTMs

RNN_LSTM. Milan et al. [154] proposed one of the first MOT algorithms (named RNN_LSTM in the MOTChallenge leaderboard) to use recurrent neural networks (RNN) and LSTMs. The RNN played the role of a Bayesian filter, and was composed of three blocks: a motion prediction block, a prediction refinement block (also called update block), that updated the prediction using the detections in the new frame, and a block that decided when to start or end tracks by predicting the probability of existence of the object in the new frame, given the predictions from the previous blocks. In order to update the prediction from the first block, the update block used an association vector that was computed by the LSTM. The LSTM was thus in charge of computing the probability that each detection in the new frame was part of each existing track. We can easily see this as a form of affinity computation. The LSTM used the Euclidean distance between the track and the detections in order to compute the association probabilities. Both networks were trained using syntethic sequences. While the algorithm reach better performance with respect to other appearance-less algorithms, its MOTA was still low (19%) with respect to other trackers. Nonetheless, it was faster than most other algorithms, with an average of ~ 165 frames per second (FPS) on MOT15.

LSTM feature fusion. Sadeghian et al. [147] used an LSTM with a fully-connected layer on top in order to predict an affinity score between tracks and detections. The structure of the affinity model is summarized in Figure 3.6. This LSTM used features computed by three more LSTMs (as explained in Section 3.5.2) as input. The overall tracking algorithm followed the Markov Decision Processes (MDP) based framework presented in [155], where a single object tracker was used to track targets independently until an occlusion happened; in that case, the Hungarian algorithm was used to solve the association, with the help of the association probabilites computed by the LSTM. The authors showed that using LSTMs rather than a simple FC layer would produce better results on MOT15. The algorithm reached top score on MOT15 and MOT16 at the time of publication, with 37.6% and 47.2% MOTA, respectively.

MAT. Ran et al. also used multiple LSTMs in their Multi-Athlete Tracking (MAT) algorithm [81]. A so-called Pose-based Triple Stream Network computed affinity scores by combining three base affinities computed by three LSTMs. One LSTM computed appearance similarity using CNN features and pose information computed with AlphaPose [112]. A second LSTM computed motion similarity using pose joints velocities. The third one computed an interaction similarity, using an interaction grid. The algorithm reached good performance on their proprietary Volleyball dataset for athlete tracking.

Siamese LSTMs

Some trackers used siamese LSTMs for affinity computation. Siamese LSTMs work in a similar way to Siamese CNNs, but include LSTM cells.

Liang et al. [156] used a Siamese LSTM to compute similarity scores between tracklets and detections. In order to reduce computational complexity, a pre-association step was first performed by predicting association probability between tracks and detections using an SVM. The SVM used position and velocity similarity scores computed by two LSTMs. Detections with low SVM affinity scores were discarded. The final association step was then performed

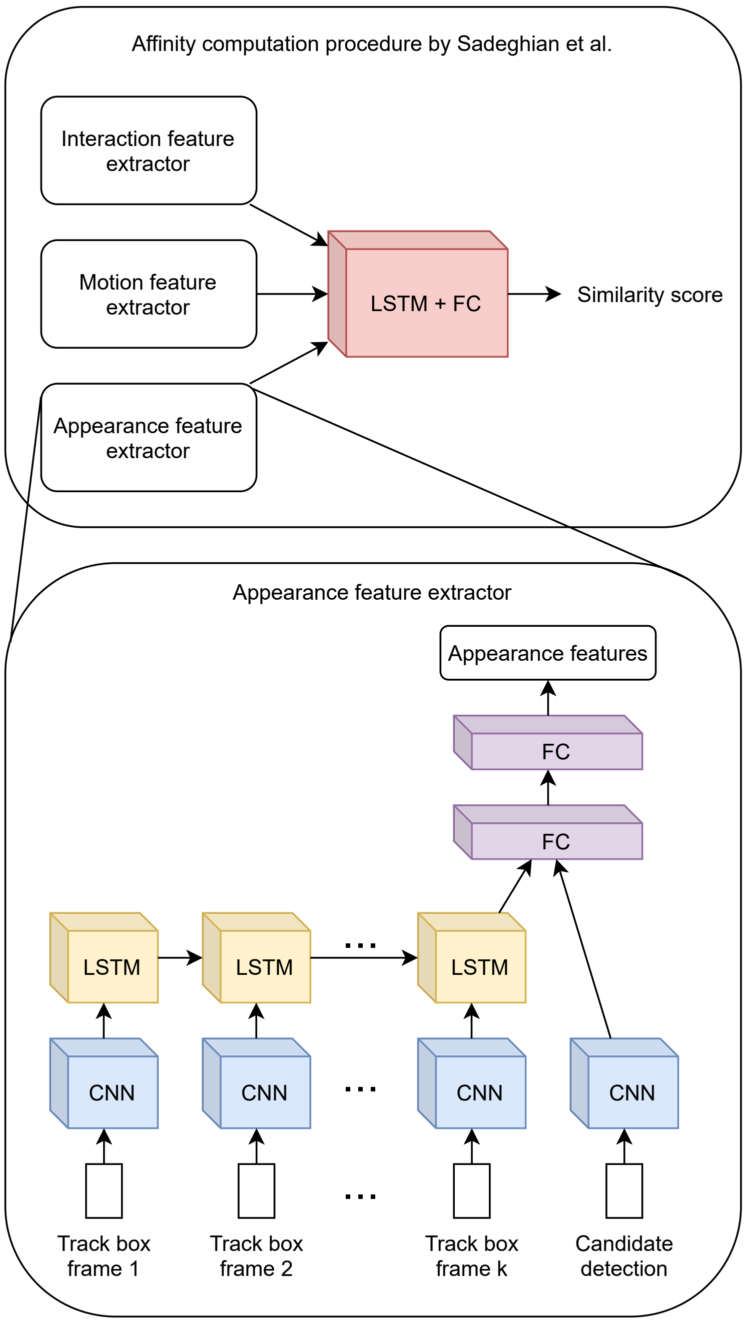


Figure 3.6: Affinity computed with LSTM feature fusion in [147]. Three LSTM-based feature extractors, one for appearance (shown in lower half of the figure), one for motion and one for interaction features, produce feature vectors that are combined by an additional LSTM-based model to produce a final affinity score.

by the Siamese LSTM, that used VGG-16 features for its predictions. A greedy association algorithm was employed. The system obtained results in line with the state-of-the-art on MOT17.

Wan et al. [71] also used a Siamese LSTM in a two-step association process. First, short reliable tracklets were built by using the Hungarian algorithm with affinity measures involving IOU and position predictions. Then, a tracklet merging step was performed, using the affinities computed by the Siamese LSTM. CNN appearance features and motion features were used by the LSTM for the affinity computation.

Bidirectional LSTMs

The DMAN algorithm [137], used a Temporal Attention Network (TAN) for the affinity computation. The TAN computed different attention coefficients, i.e. weights, for each detection in a tracklet, in order to remove the influence of noisy observations in the affinity computation step. The network consisted of a bidirectional LSTM, and used the features extracted by the Spatial Attention Network, already described in Section 3.5.2. The TAN was used as part of the occlusion recovery procedure, in case of failure of the SOT ECO algorithm [84], which was the base tracker of DMAN. DMAN obtained results in line with online state-of-the-art algorithms on MOT16 and MOT17.

Yoon et al. [157] also used a Bidirectional LSTM to compute affinities. It only used bounding box coordinates and detection confidences as features, pre-processed by a number of fully-connected layers, while no appearance feature was considered. The Hungarian algorithm was used for association. The network was evaluated on MOT15 and on the Stanford Drone Dataset (SDD) [158] and the accuracy was comparable with top algorithms that did not use visual cues, although still worse than appearance-based methods.

LSTMs in MHT frameworks

Various MOT algorithms have integrated the use of LSTMs in order to compute affinities in a Multiple Hypothesis Tracking (MHT) based framework.

MHT works by building a tree of potential track hypotheses for each candidate target. The likelihood of each track candidate is then computed and the combination of tracks with the highest likelihood is taken as the optimal tracking solution.

MHT-bLSTM. Kim et al. [126] proposed the use of a Bilinear LSTM to compute an affinity score used to prune branches of the MHT tree. The particular MHT algorithm used was MHT-DAM [108]. The LSTM cell computed a feature matrix representing the historical appearance of a tracklet, using a combination of appearance features extracted by a ResNet-50 CNN and motion features computed by a standard LSTM. The matrix was then multiplied by the vector containing the appearance features of the detection that was being compared to the tracklet. Fully-connected layers finally computed the affinity score between the tracklet and the detection. The authors claimed that this so-called Bilinear LSTM was able to store relevant appearance information for a longer time. The Bilinear LSTM and the motion LSTM were pre-trained separately and fine-tuned jointly, including localization errors and missing detections as data augmentation in order to enhance the model robustness. The algorithm was evaluated on MOT16 and MOT17 and it performed better than the original MHT-DAM when the comparison was done using high-quality detections. However, the performance was still worse than other state-of-the-art algorithms.

Eliminating Exposure Bias and Loss-Evaluation Mismatch. Maksai et al. [142] described two frequent problems that occur in training deep networks for MOT: the *loss-evaluation mismatch*, that arises when a network is trained by optimizing a loss that is not well-aligned to the evaluation metric used at inference time (e.g. classification score vs. MOTA); the *exposure bias*, i.e. lack of exposure of the model to its own errors during the training process. In their iterative variation of the MHT algorithm, the authors used a Bidirectional LSTM to score tracklets[18] in a way that

[18]While not exactly an affinity metric, it is included in this section because it plays a similar role. Like proper affinity metrics, it is used to decide whether to associate tracklets

maximizes a proxy of the IDF1 metric. This solved the loss-evaluation mismatch problem. The exposure bias was instead solved by including hard examples and model-generated tracklets at training time, so that the training distribution was more similar to the test one. The authors tested the model by using either only geometric features or both geometric and appearance features. As with other approaches, using appearance features improved the quality of the results, and the algorithm reached top IDF1 score on various MOT datasets, including MOT15, MOT17 and DukeMTMC, while not excelling in MOTA score. This was most likely due to the use of their IDF1 proxy to train the LSTM.

BMH-RMNet. Finally, we can mention a third use of recurrent networks in MHT-based trackers. Chen et al. [127] proposed an LSTM called Recurrent Metric Network (RMNet) to compute visual similarity between tracklet hypotheses and detections. RMNet used ResNet appearance features to compute the affinity and perform bounding box regression too. The affinity was used in their batch MHT-based algorithm in order to prune the hypothesis tree, like previous examples. Kalman filter was used to smooth the final trajectories. The algorithm obtained good IDF1 performance on MOT15, PETS2009[58], TUD [159] and KITTI; since the the IDF1 metric rewards the continuous tracking of targets, the good result was likely due to the use of a re-find reward inside the MHT algorithm, that encouraged track recovery after occlusions.

Other recurrent networks

Fang et al. [123] used Gated Recurrent Units (GRUs) [160] as part of their Recurrent Autoregressive Network (RAN) algorithm for pedestrian tracking. The authors used two autoregressive models for each tracked target, one for motion and one for appearance, each predicting the probability of observing a given motion or appearance based on past motion and appearance. The GRUs were used to estimate the parameters of the autoregressive

or not.

models[19]. The final association probability was computed by multiplying the motion and appearance affinities, and it was performed by solving a bipartite matching problem [155]. The RANs were trained by formulating the optimization task as a maximum likelihood estimation problem.

Kieritz et al. [87] used a recurrent MLP with two hidden layers in order to compute appearance affinity between detections and tracklets. Another MLP used such affinity, together with track and detection confidence scores, to predict the final affinity score. The Hungarian algorithm was then used to perform association using said affinity. The method reached top performance on the UA-DETRAC dataset [60], but the performance on MOT16 was not on par with other algorithms using private detections (the algorithm used a modified version of SSD, see Section 3.5.1).

CNNs for affinity computation

LMP. Tang et al. [79] formulated the tracking problem as a minimum cost lifted multicut problem (LMP) [161]. It can be thought of as a graph clustering problem, where each output cluster forms a track. The edge costs were computed using a detection similarity score that combined person re-identification confidence, deep correspondence matching [162] and spatio-temporal relations. The lifted multicut problem was solved heuristically [163]. The authors proposed the StackNetPose CNN in order to compute the person re-identification affinity. In addition to the two input images to compare, 14 body part probability maps, computed using the DeepCut body part detector [113] (see Section 3.5.2), were also concatenated to the input. The two images and the pose information were all fed to a single branch network, so that the network was able to look at both images at the same time since the first layers, similar to the stacked Siamese CNN we have seen in Figure 3.5. StackNetPose followed a similar structure to VGG-16. The algorithm reached

[19]Like [142], this work was included in this section because the probabilities predicted by the GRUs play a similar role to an affinity function, determining whether a detection is part of a tracklet or not based on their motion/appearance similarity.

top MOTA score on the MOT16 dataset at the time of publication.

STAM. Chen et al. used a Particle Filter [164] with a Spatial-Temporal Attention Mechanism (STAM) as part of their pedestrian tracker [165]. The Particle Filter predicted target motion by weighting the importance of each particle using a modified Faster R-CNN, that was trained to predict the probability that a bounding box contains a person. The Faster R-CNN was augmented with a target-specific branch, that used the target historical features in order to predict the probability that the two images belong to the same identity. The difference with most of the approaches we have explored in this section is that in this case the affinity is computed between the sampled particles and the tracked target, instead of comparing targets and detections. Despite being an online tracking algorithm, it reached top MOTA on MOT15, surpassing even the performance of batch algorithms, both with public and private detections (obtained from [166]).

DCCRF. In addition to the visual-displacement CNN, described in Section 3.5.2, the DCCRF algorithm [138] also employed a visual-similarity CNN in order to compute affinity scores between detections and tracklet boxes predicted by the Deep Continuous Conditional Random Fields. This affinity was merged with a IOU-based spatial similarity in order to perform association, with the help of the Hungarian algorithm in case of association conflicts. DCCRF reached a MOTA score on par with state-of-the-art online MOT algorithms on MOT15 and MOT16 at the time of publication.

Siamese CNNs for affinity computation

Differently from the Siamese CNNs described in Section 3.5.2, the ones presented here were used to directly compute an affinity score, instead of using hardcoded distance metrics between feature vectors.

HCC. For example, Ma et al. used a Siamese CNN to compute tracklet affinities in their two-step Hierarchical Correlation Clustering (HCC) algorithm for multiple pedestrian tracking [167]. The hierarchical clustering worked by solving two consecutive lifted multicut problems: local data association and global data association. In the first one, only temporally-close detections were merged into tracklets, with the use of the robust DeepMatching-based similarity measure presented in [168]. In the global data association step, the local tracklets were merged using affinities computed by the Siamese CNN. The network used a GoogLeNet backbone, it was trained on a re-identification dataset and fine-tuned on MOT15 and MOT16 videos. Joint verification and classification tasks were used to train the network, in order to improve the performance. At test time, the network was fine-tuned in an unsupervised manner on the test videos, in order to adapt the feature computation to the particular characteristics of each video (resolution, camera angle, illumination, and so on). This unsupervised fine-tuning was done by using the positive and negative pairs extracted from the tracklets of the local association step. The algorithm reached top performance on MOT16, with 49.3% MOTA.

FPSN-MOT. Lee et al. [140] used a Feature Pyramid Siamese Network to extract appearence features, as already mentioned in Section 3.5.2. On top of the network, three fully-connected layers merged the appearance features with a vector of motion features in order to learn an affinity metric between tracks and detections. This combined network was trained end-to-end. The association procedure was iterative, starting from high-affinity matches and stopping when the matching score reached a threshold. At the time of publication, the algorithm obtained top performance on MOT17 among the online algorithms.

3.5.4 Deep learning in the association step

A minority of works have used deep learning to improve classical association algorithms, like the Hungarian algorithm, or to manage the track status. I decided to include in this section the use of DL in track management since it is strictly related to the association process: we can think of track initialization as a prerequisite for the association stage, and of track termination as a direct consequence of the association choices made for a given frame.

Recurrent neural networks

RNN_LSTM. The algorithm presented by Milan et al. [154], already described in Section 3.5.3, used a RNN to predict the probability of existence of a track in a new frame. This was used to initialize and terminate tracks, making it one of example of the use of RNNs to manage track status.

GCRA. Ma et al. [144] used a bidirectional GRU to split tracklets in their GCRA pedestrian tracker. The algorithm consisted of three steps: first, appearance and motion affinities wesre used with the Hungarian algorithm to generate the initial candidate tracklets; second, a bidirectional GRU was employed to cleave (i.e. split) tracklets predicted to contain ID switches, in order to obtained smaller tracklets with a single identity each; finally, a Siamese bidirectional GRU was used to extract features from the tracklets in order to perform the reconnection of tracklets belonging to the same identity. A polynomial curve fitting procedure was used to fill the gaps in the newly formed tracks. The cleaving GRU used features extracted by a Wide ResNet [26] to output two feature vectors for all the detections in a tracklets, one for the forward direction and one for the backward direction of the GRU; then, for each frame, the feature vectors in the two directions were compared by using a distance metric and the tracklet was cut at the frame where the largest feature distance was found, if higher than a threshold. A large distance was in fact indicative of a great visual difference between the boxes on one side of the time arrow and the boxes on the other side. The reconnection GRU operated in a similar way, with the addition of a FC layer on top of the GRU and a temporal pooling layer, used to extract a feature vector representing the entire tracklet; the distance between the features of each tracklet pair was then used to decide whether to fuse them or not. The algorithm reached results comparable to state-of-the-art on the MOT16 dataset.

Deep Multi-Layer Perceptron

While not being a common approach, deep MLPs have also been used in the track management process. One notable example is the algorithm by

Kieritz et al. [87], previously discussed in Section 3.5.3. The authors used a MLP with two hidden layers[20] to compute track confidence scores in order to manage the termination of tracks. The algorithm only tracked a fixed number of targets through time, replacing older low-confidence tracks with new ones. In order to take its decision, the MLP exploited the track score computed in the previous step as well as the association and detection confidence scores of the last detection added to the track.

Deep Reinforcement Learning agents

MARLMOT. Rosello et al. [152] used multiple deep reinforcement learning (RL) agents in order to manage each tracked target. The agents were able to decide when to start and stop tracks, and could influence the operation of the Kalman filter, as explained in Section 3.5.3. Each agent was a MLP with three hidden layers.

C-DRL. Ren et al. [169] used multiple deep RL agents in a collaborative environment to manage the association task in their C-DRL (Collaborative Deep RL) tracking algorithm. The system consisted of two main parts: a prediction network and a decision network. The prediction network, a CNN, predicted the position of each target in the new frame by using both appearance information (extracted using MDNet [170]) from the target and the new frame, and using the recent tracklet trajectory. The decision network instead was in charge of the track management process: based on information such as the new predicted position, the nearest target and detection, detection reliability and target occlusion status, it took a decision about the examined track. It could choose to update the track and its appearance features with new information (also deciding which information to use and to ignore for the update step), to mark the track as occluded, or to delete the track. The decision network was made of a collaborative system composed of multiple deep RL agents, one per each tracked target. Each agent consisted of 3 FC layers on top of the

[20] As explained in Chapter 2, any neural network with at least two hidden layers is considered deep by convention, although DNNs usually have many more layers.

feature extraction layers of the MDNet. The authors showed that using this C-DRL system improved the performance with respect to using linear motion models and the simpler Hungarian algorithm for the association process. The system reached state-of-the-art results among online methods on MOT15 and MOT16, despite producing a relatively high number of ID switch errors.

3.5.5 Other uses of DL in MOT

In this section I will describe some uses of deep learning in MOT that do not easily fit the four stage categorization.

Q-learning for bounding box regression. One example is the approach proposed by Jiang et al. [171]. The authors used a Q-network [172], made of 3 FC layers, as a deep RL agent able to perform bounding box regression. The agent could take one of 13 possible actions, that included rescaling and repositioning the box, and determining when the regression was complete. The network used VGG-16 appearance features and a vector of the last 10 decisions made by the network itself in order to decide the next action. The authors applied this bounding box regression method to the output of a variety of MOT algorithms, as a post-processing step. The regression increased the MOTA between 2 and 7% on the MOT15 dataset, reaching top score among methods using public detections. The method also showed better performance compared to the use of a CNN regression branch, like the one used by Faster R-CNN.

MCMOT. Lee et al. [173] proposed a multi-class MOT algorithm (MCMOT) that used an ensemble of detectors, such as VGG-16 and ResNet, to compute the position likelihood maps for each target in the new frame. These likelihoods were used in a method based on Markov Chain Monte Carlo sampling in order to predict the next position of each target. The authors also used a changing point detection algorithm [174] to detect tracking drift, in order to remove unstable track segments and recombine them correctly. The results were comparable to state-of-the-art algorithms on MOT16 using private detections.

MBFILH. Hoak et al. [175] used a 5-layer CNN to compute the position likelihood of a target in the new frame. This was then used in a multi-Bernoulli filter with Interactive Likelihood (MBFILH), that is a variation of the particle filter algorithm [176] for tracking. The algorithm obtained good results on the VSPETS 2003 INMOVE soccer dataset[21] and the AFL dataset [177].

Fusion of head and body detectors. Henschel et al. [178] used head detections, extracted with a CNN [179], in addition to the usual body detections to perform pedestrian tracking. In fact, the presence or absence of a head and its position relative to the body bounding box can help determine if a bounding box is a true or a false positive. Association was performed by formulating the problem as correlation clustering on graphs, with distances computed using spatial and temporal costs that accounted for the relative positions of heads and bodies. The algorithm reached top MOTA score on MOT17 and second-best score on MOT16 at the time of publication.

CNNs for enhanced model updates. Gan et al. [180] employed a modified MDNet [170] in their online pedestrian tracking framework. The network shared 3 convolutional layers for all the targets, but also had 3 target-specific FC layers, that were updated online to capture the appearance change of each target. A set of box candidates, selected using motion models, were fed to the network, that output a confidence score for each of them, in order to determine the target position in the new frame. To reduce the number of ID switch errors, the algorithm looked for the tracklet that was most similar to the estimated box, using another appearance and motion-based affinity measure.

TripT+BF. Xiang et al. [181] proposed a so-called *MetricNet* to track pedestrians in their algorithm, called TripT+BF. It was composed of an appearance model and a motion model. A VGG-16 trained for person re-identification was used to extract appearance features and perform bounding box regression. The motion model instead included an LSTM and a so-called *BF-Net*, that used the features output by the LSTM and a candidate box

[21] ftp://ftp.cs.rdg.ac.uk/pub/VS-PETS/

to perform Bayesian filtering and output the new position of the target. MetricNet was trained using triplet loss. The algorithm obtained the highest and second-highest MOTA among online methods on MOT16 and MOT15, respectively.

Instance-aware tracker with Dynamic Model Refreshment. Chu et al. [182] used three different CNNs in their algorithm. PafNet [183], was used to separate the background from the tracked objects. PartNet [184], was employed to discriminate among the different targets. A third CNN, composed of a convolutional layer and a FC layer, was used instead to decide whether to refresh the tracking model or not. For each tracked target PafNet and PartNet computed two score maps that were used by a Kernel Correlation Filter (KCF) tracker [185] to predict the new position of the object. At certain intervals, detections were matched to the current tracks in order to determine which tracks had to be terminated. The third CNN, trained with reinforcement learning, used the PafNet maps in order to determine whether to use the predicted box or the associated detection box as the new box in the track, and the KCF model was updated accordingly. The algorithm reached top performance overall on MOT15 and top performance among online methods on MOT16.

3.6 Performance analysis and comparisons

In order to understand which approaches led to better trackers, a numerical comparison is needed. For this reason, I will include here the results obtained by many of the presented algorithms on the MOTChallenge datasets, since they are the ones on which most of the presented methods evaluated their algorithms.

In particular, I will show the results on MOT15, MOT16 and MOT17, considering only the scores computed on the respective test sets for a fair comparison. Papers that only reported their results on customized validation sets, or even on the training set, will not be included here, since they cannot be compared with the other algorithms. Furthermore, since the quality of detections is an important factor in the performance of a tracker, I will provide separate tables for methods using public detections, provided with the dataset, and “private” detections, obtained by authors using different detectors. In each table, methods will be divided into online and batch methods, since this is another major factor influencing the accuracy of a model, given the limited information available to online methods to take decisions.

For each algorithm, tables include the year of publication, the mode of operation (batch/online), and a number of metrics, following the ones provided on the public leaderboards of MOTChallenge: MOTA, MOTP, IDF1, Mostly Tracked (MT), Mostly Lost (ML), false positives (FP), false negatives (FN), ID switches (IDS), fragmentations (Frag), and average number of frames processed per second by the algorithm (Hz). Note that MOTA, MOTP, IDF1, MT and ML are expressed in percentages. For each metric, an arrow pointing up (↑) indicates that the higher score the better, while an arrow pointing down (↓) indicates the opposite. I have collected and merged the results provided in each paper with the data from the public MOTChallenge leaderboards. If multiple configurations of an algorithm have been tested on the same dataset, only the one with the highest MOTA will be shown.

Tables 3.1 and 3.2 show results on MOT15 using public and private detections respectively; Tables 3.3 and 3.4 do the same on MOT16; finally, Table 3.5 shows results on MOT17 using public detections[22]. Algorithms are sorted by ascending year of publication. For each metric, the best score obtained among batch algorithm and the best score among online methods is highlighted, the highest of the two in **bold** and the other one <u>underlined</u>.

It is important to note that the speed comparison is not always reliable, since many algorithms excluded the time taken by the detection step in

[22]At the time of writing, none of the presented algorithms has published results on MOT17 using private detections. Note that MOT16 and MOT17 provide the same videos, with the only differences being the more accurate annotations and the multiple sets of public detections in MOT17, obtained with different detectors.

reporting the performance, which is usually one of the most computationally expensive ones. This is true even for some algorithms using private detections. For this reason, many of the algorithms, that we may think as operating in real-time by just looking at reported results, might not actually able to do so yet with current hardware. Speaking of hardware, another important factor at play is the use of machines with different characteristics and performance, which makes speed comparisons even harder.

	Year	Mode	MOTA ↑	MOTP ↑	IDF1 ↑	MT ↑	ML ↓	FP ↓	FN ↓	IDS ↓	Frag ↓	Hz ↑
[108]	2015		32.4	71.8	45.3	16.0	43.8	9064	32 060	435	826	0.7
[154]	2017		19.0	71.0	17.1	5.5	45.6	11 578	36 706	1490	2081	165.2
[149]	2017		31.6	71.8		10.1	46.3			491	994	
[120]	2017		32.8	70.7	38.8	9.7	42.2	4983	35 690	614	1583	2.3
[148]	2017		34.3	70.5	48.3	11.4	43.4	5154	34 848	**348**	1463	0.5
[118]	2017		35.0	72.6	47.7	11.4	42.2	8455	31 140	358	1267	4.6
[147]	2017		37.6	71.7	46.0	15.8	**26.8**	7933	**29 397**	1026	2024	1.0
[165]	2017	Online	38.5	72.6	47.1	8.7	37.4	**4005**	33 204	586	1263	6.7
[138]	2018		33.6	70.9	39.1	10.4	37.6	5917	34 002	866	1566	0.1
[123]	2018		35.1	70.9	45.4	13.0	42.3	6771	32 717	381	1523	5.4
[169]	2018		37.1	71.0		14.0	31.3	7036	30 440			
[171]	2018		**42.3**		47.7	13.6	39.7					3.1
[157]	2019		22.5	70.9	25.9	6.4	61.9	7346	39 092	1159	1538	**172.8**
[181]	2019		37.1	72.5	**48.4**	12.6	39.7	8305	29 732	580	1193	1.0
[182]	2019		38.9	70.6	44.5	**16.6**	31.5	7321	29 501	720	1440	0.3
[134]	2016		29.0	71.2	34.3	8.5	48.4	5160	37 798	639	1316	52.8
[132]	2016		29.6	71.8	36.8	11.2	44.0	7786	34 733	712	943	1.7
[136]	2017	Batch	33.8	73.4	40.4	12.9	36.9	7898	32 061	703	1430	3.7
[142]	2018		22.2	71.1	27.2	3.1	61.6	5591	41 531	700	1240	8.9
[127]	2019		28.1	**74.3**	38.7			6733	36 952	477	**790**	16.9

Table 3.1: Results obtained by DL-based MOT algorithms on MOT15 using public detections.

	Year	Mode	MOTA ↑	MOTP ↑	IDF1 ↑	MT ↑	ML ↓	FP ↓	FN ↓	IDS ↓	Frag ↓	Hz ↑
[61]	2016		33.4	72.1	40.4	11.7	30.9	7318	32 615	1001	1764	**260.0**
[101]	2017		32.1	70.9		13.2	30.1	6551	33 473	1687	2471	
[120]	2017	Online	51.3	74.2	54.1	36.3	22.2	7110	22 271	544	**1335**	1.3
[165]	2017		53.0	**75.5**	52.2	29.1	20.2	**5159**	22 984	708	1476	6.7
[88]	2018		32.7		38.9	26.2	19.6					11.1
[123]	2018		**56.5**	73.0	**61.3**	**45.1**	**14.6**	9386	**16 921**	**428**	1364	5.1

Table 3.2: Results obtained by DL-based MOT algorithms on MOT15 using private detections.

	Year	Mode	MOTA ↑	MOTP ↑	IDF1 ↑	MT ↑	ML ↓	FP ↓	FN ↓	IDS ↓	Frag ↓	Hz ↑
[131]	2016		35.3	75.2		7.4	51.1	5592	110 778	1598	5153	7.9
[166]	2016		38.8	75.1		7.9	49.1	8114	102 452	965	1657	11.8
[120]	2017		43.9	74.7	45.1	10.7	44.4	6450	95 175	676	1795	0.5
[148]	2017		46.0	74.9	50.0	14.6	43.6	6895	91 117	473	1422	0.2
[147]	2017		47.2	75.8	46.3	14.0	41.6	**2681**	92 856	774	1675	1.0
[180]	2018		44.2	78.3		15.2	45.7	7912	93 215	560	1212	
[138]	2018	Online	44.8	75.6	39.7	14.1	42.3	5613	94 125	968	1378	0.1
[123]	2018		45.9	74.8	48.8	13.2	41.9	6871	91 173	648	1992	0.9
[137]	2018		46.1	73.8	**54.8**	17.4	42.7	7909	89 874	532	1616	0.3
[169]	2018		47.3	74.6		17.4	39.9	6375	88 543			
[139]	2018		47.6	74.8	50.9	15.2	38.3	9253	85 431	792	1858	**20.6**
[181]	2019		48.3	76.7	50.9	15.4	40.1	2706	91 047	543	896	0.5
[182]	2019		48.8	75.7	47.2	15.8	38.1	5875	86 567	906	1116	0.1
[136]	2017		44.1	76.4	38.3	14.6	44.9	6388	94 775	745	1096	1.8
[117]	2017		45.3	75.9	47.9	17.0	39.9	11 122	87 890	639	946	1.8
[79]	2017		48.8	**79.0**		18.2	40.1	6654	86 245	481	595	0.5
[126]	2018		42.1		47.8	14.9	44.4	11 637	93 172	753	1156	1.8
[153]	2018		46.9	76.4	46.8	16.1	43.2	6257	91 669	549	757	
[125]	2018	Batch	47.2	75.7	52.4	18.6	42.8	12 586	83 107	542	787	0.5
[124]	2018		47.5		43.6	**19.4**	**36.9**	13 002	**81 762**	1035	1408	0.8
[178]	2018		47.8	75.5	44.3	19.1	38.2	8886	85 487	852	1534	0.6
[144]	2018		48.2	77.5	48.6	12.9	41.1	5104	88 586	821	1117	2.8
[167]	2018		**49.3**	**79.0**	50.7	17.8	39.9	5333	86 795	**391**	**535**	0.8

Table 3.3: Results obtained by DL-based MOT algorithms on MOT16 using public detections.

	Year	Mode	MOTA ↑	MOTP ↑	IDF1 ↑	MT ↑	ML ↓	FP ↓	FN ↓	IDS ↓	Frag ↓	Hz ↑
[64]	2016	Online	66.1	79.5	65.1	34.0	20.8	5061	55 914	805	3093	9.9
[69]	2017		61.4	79.1	62.2	32.8	**18.2**	12 852	56 668	781	2008	17.4
[87]	2018		39.1			11.1	41.1	9411	99 727	1906		4.5
[72]	2018		55.0	76.7		20.4	24.5	15 766	65 297	1024	1594	16.9
[71]	2018		62.6	78.3		32.7	21.1	10 604	56 182	1389	1534	
[123]	2018		63.0	78.8	63.8	39.9	22.1	13 663	53 248	482	1251	1.6
[80]	2018		64.8	78.6	**73.5**	40.6	22.0	13 470	49 927	794	1050	**39.4**
[70]	2019		65.2	78.4	62.2	32.4	21.3	6578	55 896	946	2283	11.2
[173]	2016	Batch	62.4	78.3	51.6	31.5	24.2	9855	57 257	1394	1318	34.9
[64]	2016		68.2	79.4	60.0	41.0	19.0	11 479	45 605	933	1093	0.7
[79]	2017		**71.0**	**80.2**	70.1	**46.9**	21.9	7880	**44 564**	**434**	**587**	0.5
[153]	2018		58.1	77.2	47.4	23.1	33.3	**4883**	70 207	1624	2539	

Table 3.4: Results obtained by DL-based MOT algorithms on MOT16 using private detections.

	Year	Mode	MOTA ↑	MOTP ↑	IDF1 ↑	MT ↑	ML ↓	FP ↓	FN ↓	IDS ↓	Frag ↓	Hz ↑
[180]	2018		44.9	**78.9**		13.8	44.2	**22 085**	287 267	1537	3295	
[128]	2018		46.5	77.2		16.9	37.2	23 859	272 430	5649	9298	1.6
[137]	2018	Online	48.2	75.7	55.7	19.3	38.3	26 218	263 608	2194	5378	0.3
[139]	2018		50.9	76.6	52.7	17.5	35.7	24 069	250 768	2474	5317	**18.3**
[140]	2019		44.9	76.6	48.4	16.5	35.8	33 757	269 952	7136	14 491	10.1
[142]	2018		44.2	76.4	**57.2**	16.1	44.3	29 473	283 611	**1529**	**2644**	4.8
[126]	2018		47.5		51.9	18.2	41.7	25 981	268 042	2069	3124	1.9
[156]	2018	Batch	50.3		47.9	21.8	36.2	22 204	249 342	3243	3155	1.9
[178]	2018		51.3	77.0	47.6	21.4	**35.2**	24 101	247 921	2648	4279	0.2
[125]	2018		**51.8**	77.0	54.7	**23.4**	37.9	33 212	**236 772**	1834	2739	0.7

Table 3.5: Results obtained by DL-based MOT algorithms on MOT17 using public detections.

3.6.1 Discussion of the results

General observations

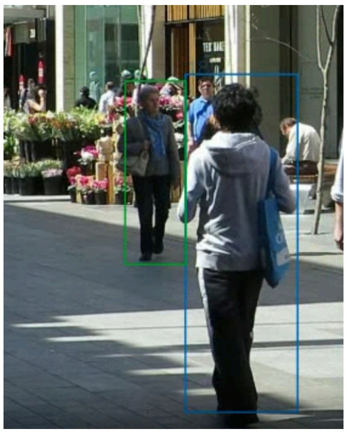
As expected, the best performing algorithms on each dataset use private detections, confirming the fact that the detection quality dominates the overall performance of the tracker: 56.5% MOTA vs. 42.3% for MOT15 and 71.0% vs. 49.3% for MOT16. Moreover, on MOT16 and MOT17 the batch algorithms slightly outperform the online ones, even though the online methods are progressively getting closer in performance. In fact, the best reported algorithm on MOT15 runs in an online fashion. However, this might be due to the fact that the research community has been focusing much more on the development of online methods, given the possible implications for developing real-time tracking systems.

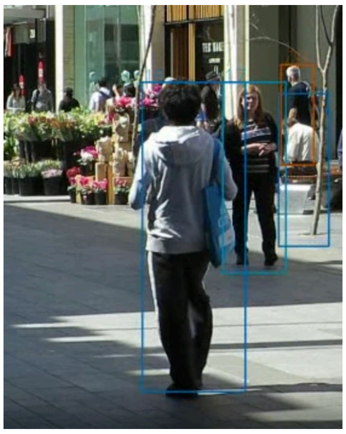
A common problem among online methods that is not reflected in the MOTA score is the higher number of fragmentations, as we can see in Table 3.6. This is probably due to the main limitation of online algorithms, that cannot look at future information in order to re-identify lost targets and reconstruct the missing parts of a trajectory, e.g. by interpolation [148, 118, 120]. In Figure 3.7 we can see an example of trajectory that is fragmented by an online method, MOTDT [139], while it is correctly tracked by a batch method, HAF [125].

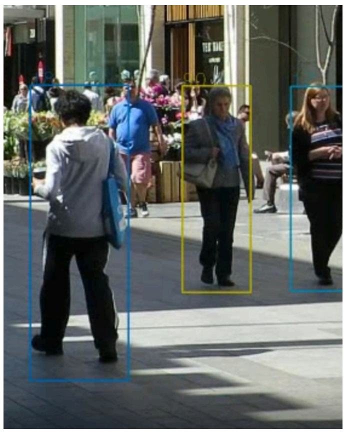
Mode	MOT15	MOT16	MOT17
Batch	1143.8	1104.9	3188.2
Online	1509.5	1820.2	7555.8

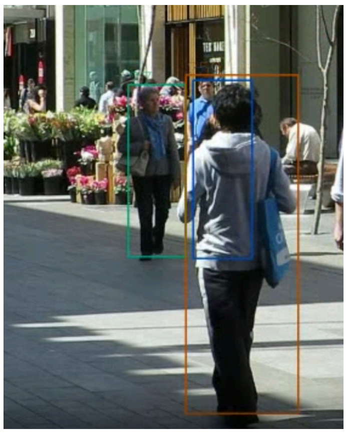
Table 3.6: Average number of fragmentations for online and batch methods in the three considered datasets.

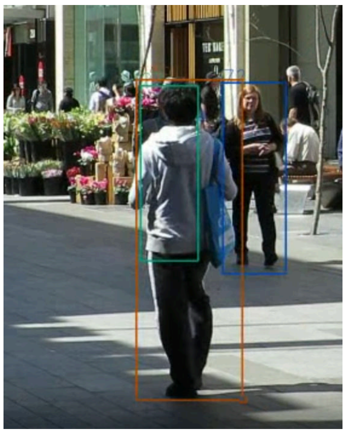
Another interesting thing to notice is that since the number of FNs dominates the MOTA score, being one or two orders of magnitude larger than the number of FPs and ID switches for most methods, the algorithms


(a) MOTDT output before occlusion


(b) MOTDT output during occlusion


(c) MOTDT output after occlusion


(d) HAF output before occlusion


(e) HAF output during occlusion

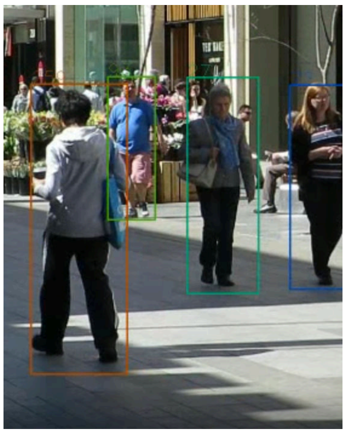
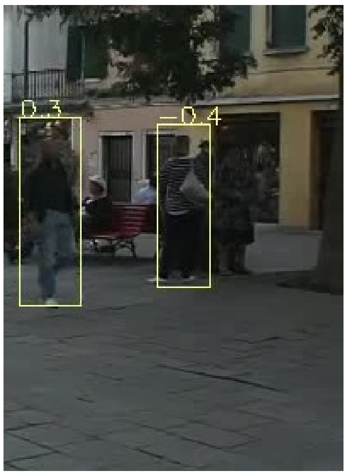

(f) HAF output after occlusion

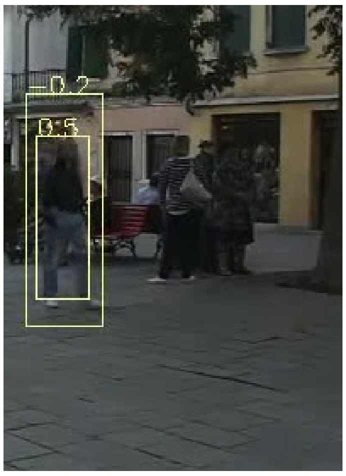
Figure 3.7: Example of fragmentation produced by an online method during occlusion. Above: tracking results for MOTDT [139], online algorithm. Below: tracking results for HAF [125], batch algorithm. From left to right, frames 145, 170 and 217 of the MOT17-08 video are shown. Images were cropped from the public MOTChallenge website and show results using the public detections. The person in 3.7a surrounded by the green box is occluded in 3.7b by the one identified by the blue box. The tracking is lost (fragmentation) until the occlusion ends in 3.7c (the person is also given a new ID, causing an ID switch). In contrast, we can see how in 3.7e the track is maintained through the occlusion, since the batch algorithm can look at the future information (e.g. frame 3.7f) and infer the position of the target during occlusions.

that manage to significantly reduce the FNs are also the ones that obtain the highest MOTA, if they manage to keep the corresponding increase of false positives relatively small. This was already observed in previous works [40]. Computing the Pearson correlation coefficient between MOTA and FNs shows that for MOT15, MOT16 and MOT17 these two measures are in fact highly correlated, with coefficients of -0.95, -0.98 and -0.95 respectively. This observation can also explain why methods using higher-quality, private detections tend to perform better, since these detectors are able to identify many challenging objects that older, simpler detectors cannot, thus reducing the final number of FNs in tracking-by-detection algorithms. In Figure 3.8 we can see how the SORT algorithm, that is particularly sensitive to missing detections, is not able to detect a target as soon as the corresponding detection is missing. When the target is detected in a later frame, the algorithm assigns it a new ID.

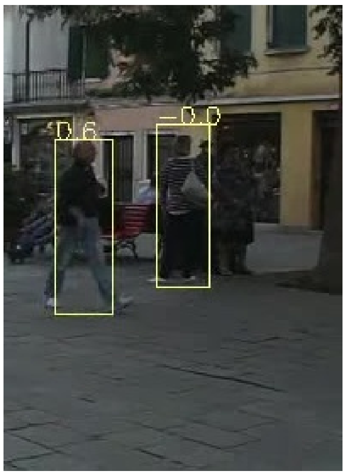
To solve this issue, new techniques have been developed. In fact, while basic approaches that perform interpolation are able to recover missing boxes during occlusions, this is still insufficient to detect targets that are not covered by even a single detection, that have been shown to be 18% of the total on MOT15 and MOT16 [40]. For example, the eHAF16 algorithm, presented by Sheng et al. [125], employed superpixel extraction to complement the publicly provided detections and was in fact able to significantly reduce the number of false negatives on MOT17, reaching top MOTA score on the dataset. MOTDT [139] instead used a R-FCN to integrate the missing detections with new candidates, and was able to reach the highest MOTA and the lowest number of false negatives among online algorithms on MOT17. The STAM algorithm [165] was also able to avoid the problems caused by missing detections by employing a Particle Filter and relying on detections only to initialize new targets and to recover lost ones. The algorithm presented in [169] reduced FNs by designing a deep network to learn the motion model of each object. Predicting the position of each object resulted in a reduction of the amount of false negatives, since the algorithm was less reliant on missing detections.



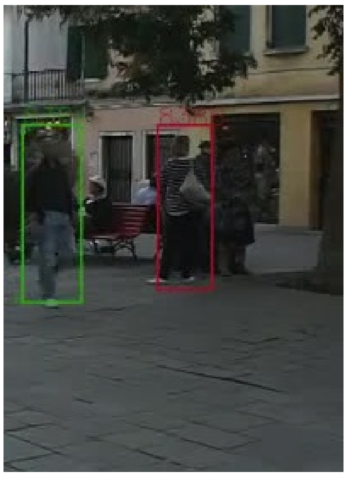
(a) DPM detections for frame 20



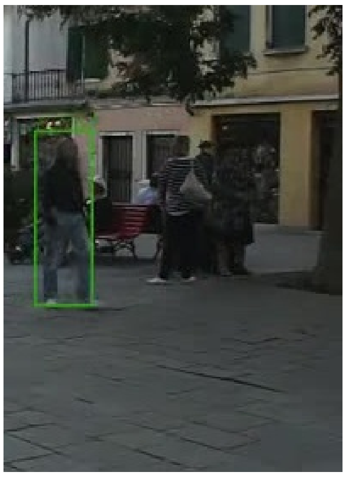
(b) DPM detections for frame 24



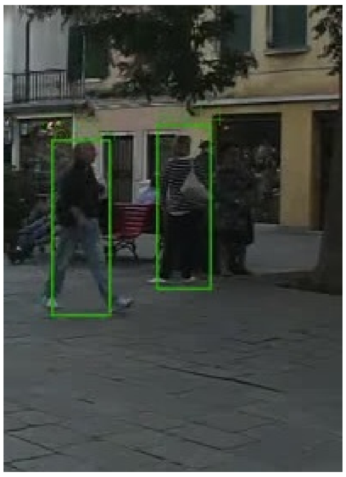
(c) DPM detections for frame 28



(d) SORT results for frame 20



(e) SORT results for frame 24



(f) SORT results for frame 28

Figure 3.8: Influence of detection failures on the SORT algorithm. The images show frames 20, 24 and 28 of the MOT17-01 video. Images were cropped from the public MOTChallenge website. Above: public DPM detections provided with MOT17. Below: tracking results using SORT [61], when applied to the DPM detections. We can see how as soon as the central target is not detected in 3.8b, SORT loses the target too (3.8e). When the detector finds it again (3.8c), tracking is restored (3.8f), but with a different ID (since SORT does not perform person re-identification).

In fact, the algorithm produced one of the lowest numbers of false negatives among online methods on MOT16.

Another interesting observation is related to the strategy used to train deep networks for affinity computation. As noted by Kim et al. [126], training a network using ground truth trajectories to predict affinities might produce suboptimal results, since the data distribution at inference time will be different, with many noisy trajectories that can include missing/wrong detections. For this reason, some of these networks have been trained using either actual detections [123] or augmented ground truth trajectories, with the artificial addition of noise and errors [126, 137]. However, some works also found that this strategy may sometimes slow the training process and not always be feasible [87].

Best approaches in the four MOT steps

Detection step. Algorithms that used private detections produced by Faster R-CNN and its variants seem to obtain the best results. In fact, the algorithm presented in [64], that used a modified Faster R-CNN, held its top ranking position among online methods on MOT16 for about 3 years, and many of the other top-performing MOT16 algorithms used the same detections.

In contrast, algorithms that employed the SSD detector, such as the ones presented in [87] and [88], tend to perform worse on average, but are able to reach faster speeds: for example, the algorithm presented by Kieritz et al. [87] can run at 4.5 FPS *including* the detection stage[23], a step in the direction of real-time performance. However, despite the efforts in developing efficient online methods, the use of deep learning techniques in a MOT pipeline still represents a computational bottleneck, limiting the application of online algorithms employing complex deep networks in real-time scenarios.

Feature extraction step. Regarding feature extraction, all the top performing methods on the three considered datasets use a CNN to extract

[23]Remember that the FPS reported by many algorithms tend to exclude the detection step, that is one of the most computationally expensive parts of a MOT algorithm.

appearance features, with GoogLeNet being the most commonly used one. Methods that do not exploit appearance (either extracted with deep or classical methods) tend to perform worse. However, visual features are not enough: many of the best algorithms also employ other types of features to compute affinity, especially motion cues. In fact, methods like LSTMs and Kalman Filters are often employed to predict the position of the targets in the next frame, which helps improve the quality of the association. Various Bayesian filters, such as particle filter and hypotheses density filter, are also used to predict target motion, and they benefit from the use of deep models [181, 165, 128]. However, even when the appearance features are used in conjunction with non-visual features, appearance still plays a major role in improving the overall performance of the algorithm [147, 181], especially by avoiding ID switches [108] or by re-identifying targets after long occlusions [69]. In the latter case, simple motion predictors do not work since the linear motion assumption is easily broken, as noted by Zhou et al. [80].

Affinity computation. While deep learning plays an important role in detection and feature extraction, the use of deep networks to learn affinity functions is less common. For now, it does not seem to have led to significant improvements over the classical techniques, that usually involve a combination of hand-crafted distance metrics on a mix of deep and non-deep features. However, some good-performing algorithm already use deep networks to learn an affinity metric [167, 165, 79, 123], mostly using Siamese CNNs or recurrent neural networks. In particular, the Siamese network variation proposed by Ma et al. [167] was able to produce reliable similarity measures that helped with person re-identification after occlusions, and allowed the algorithm to reach the highest MOTA score on MOT16. The integration of body part information was also crucial for the StackNetPose CNN proposed by Tang et al. [79], since it played the role of an attention mechanism that allowed the network to focus on the relevant parts of the input images. This, in turn, resulted in a more accurate similarity measure, that helped the algorithm reach top performance on MOT16 using private detections.

Association step. Even less works have explored the use of deep learning to guide the association process. The use of reinforcement learning [169] or recurrent neural networks [144] to manage associations and track status has showed some promising results, so this might be an interesting future direction of research.

Trends in top-performing algorithms

Some common approaches can be identified in the best performing algorithms.

Adaptation of Single Object Trackers. Many successful online MOT trackers employed appropriate variations of Single Object Trackers to manage the tracking process. For example, the Instance-Aware tracker by Chu et al. [182], DMAN by Zhu et al. [137] and the algorithm presented by Sadeghian et al. [147] all used a SOT tracker enhanced with deep learning methods in order to recover from occlusions or to refresh the target models. It is interesting to note that none of the presented SOT-based algorithms has been used yet on private high-quality detections on the MOTChallenge datasets. Since private detections increase the number of targets that are identified at least once, the use of SOT trackers, that usually only rely on a single detection to initialize a tracklet, can help cover more targets, reducing the number of false positives. In addition to that, a batch method could exploit the SOT tracker to look at past frames and recover any missed detection before the target was detected for the first time. This would lead to a higher overall performance of the tracker. For this reason, the use of SOT trackers appears to be another interesting research direction to explore.

However, SOT-based MOT trackers can sometimes still be prone to tracking drift and produce a higher number of ID switches. For example, the KCF16 algorithm [182], while reaching top MOTA score among online methods on MOT16 with public detections, it still produces a relatively high number of identity switches, due to tracking drift, as it can be seen in Figure 3.9. Moreover, since SOT-based MOT algorithms do not rely on detections as other methods, a false positive detection might jumpstart a entire spurious trajectory, so particular attention must be payed to alleviate this issue when using a SOT tracker. Current approaches [182, 137], in fact, tend to check if the predicted trajectories overlap with at least some of the detections, in order to understand if it is a true or a false positive. However, better solutions should be investigated to avoid exclusive reliance on the quality of the detections.

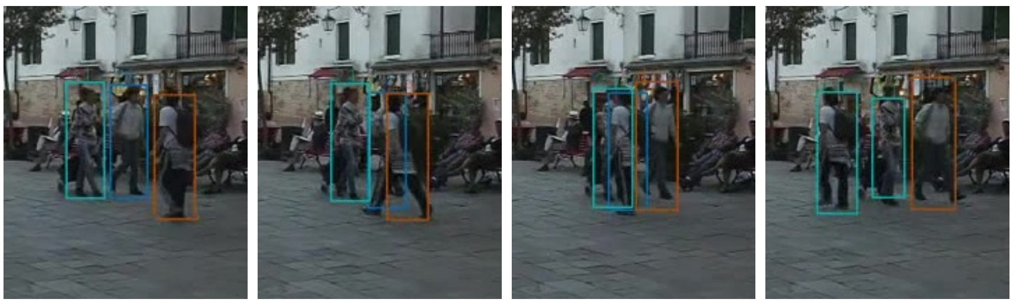

(a) Frame 134 (b) Frame 142 (c) Frame 149 (d) Frame 164

Figure 3.9: Examples of tracking drift with a SOT-based MOT algorithm (KCF [182]). The four images are cropped from the MOT16-01 video (https://motchallenge.net/method/MOT=1156&chl=5). At first (a) the three targets are tracked. After a few frames (b) the *orange* box starts drifting towards the occluded man, while the *blue* box starts drifting towards the foreground target. In (c) blue and orange have switched target, counting as two ID switches. In (d), after the blue and cyan identities overlap, the cyan one drifts to track the blue target (third ID switch), whose track is interrupted. A new identity is assigned to the woman in the background, causing a fourth ID switch.

Graph optimization techniques in batch algorithms. While both online and batch methods perform association by formulating the task as a graph optimization problem, batch methods seem to particularly benefit from it, since they are able to perform global optimization over the entire video. For example, the minimum cost lifted multicut problem has reached top performance on MOT16, helped by CNN-computed affinities [167, 79], while heterogeneous association graph fusion and correlation clustering are used in the two top MOT17 methods, [125, 178].

Bounding box refinement. Finally, we can see that the accuracy of bounding boxes can radically affect the performance of an algorithm. In fact, the top ranked tracker on MOT15 [171] obtained a relatively high MOTA score by just performing a bounding box regression step on the output of an existing state-of-the-art algorithm [118] using a deep RL agent. Developing an effective bounding box regressor to be incorporated in future MOT algorithms may be an interesting research direction that has not yet been explored thoroughly. Moreover, since relying on isolated detections might lead to regression mistakes, future approaches might be interested in using contextual information from past frames (and future ones too, for batch methods), in order to accurately regress the bounding box around the right target when different targets or objects partially overlap.

3.7 Conclusion and future directions

I have presented a comprehensive survey on the use of deep learning in MOT for single-camera videos and 2D data. I have identified and described the four major steps that characterize the majority of MOT algorithms: detection, feature extraction, affinity computation and association. For each of these steps, I have presented the deep learning techniques used to solved them in the literature, in the context of each MOT algorithm. I have also collected and presented a comparison of the results obtained by most of the described algorithms on the MOTChallenge dataset, highlighting some of the pros and cons of the most common approaches, and finding some common themes among the best performing algorithms:

- **detection quality is important**: the amount of false negatives still dominates the MOTA score. While some algorithms managed to partially compensate for missing detections, the use of higher quality detectors is still the most effective way to reduce false negatives. Thus, a careful use of deep learning in the detection step can considerably improve the performance of a tracking algorithm;

- **CNNs are essential in feature extraction**: the use of appearance features is also fundamental for a good tracker and CNNs are particularly effective at extracting them. Moreover, the best trackers tend to use them in conjunction with motion features, that can be computed using LSTMs, Kalman filter or other Bayesian filters. However, the downside of using appearance features is an increase in computational complexity, with a reduction of the overall algorithm speed;

- **SOT trackers and global graph optimization**: the adaptation of SOT trackers to the MOT task, with the help of deep learning, has recently produced good-performing online trackers; batch methods have instead benefited from the integration of deep models in global graph optimization algorithms.

Deep learning is a relatively recent introduction in the field of MOT, and thus various promising directions of research can be identified:

- **researching more strategies to mitigate detection errors**: although modern detectors are reaching better and better accuracy, they can still produce a significant number of false negatives and false positives in complex scenarios, such as dense pedestrian tracking. Some algorithms have already provided solutions to reduce the exclusive reliance on detections, for example by integrating them with information extracted from other sources (e.g. superpixels [125], R-FCN [139], Particle Filter [165]), but further strategies should be investigated;

- **applying DL to track different targets**: most of DL-based MOT algorithms have focused on pedestrian tracking. Since different types of targets pose different challenges, possible improvements in tracking vehicles, animals, or other objects with the use of deep networks should be investigated;

- **investigating the robustness of current algorithms**: while datasets such as MOTChallenge provide videos recorded with different lighting conditions and camera motion, they still show only a small part of the possible challenging conditions in which tracking might be applied. For example, while the majority of trackers focused on pedestrians, many other scenarios are possible. One possibility is tracking people in movies, which could be useful for scene understanding and automatic scene description, and poses additional challenges such as a great variety of settings, hectic scenes with high target motion, shots captured from different views and zoom levels, and so on. How do current algorithms, trained on pedestrians, fare in those new contexts and situations? In addition to that, a study on the robustness of existing algorithms to missing frames or other video artifacts, possible situations in real-world scenarios, might also be interesting;
- **applying DL to guide association**: the use of deep learning to directly guide the association process and to manage the track status has not been investigated thoroughly yet: more research is needed in this direction to understand if deep algorithms can be useful in this step too;
- **combining SOT trackers with private detections**: combining the use of SOT trackers with high-quality detectors seems a promising way to reduce the number of lost tracks, and thus decrease the number of false negatives, especially in a batch setting, where it would be possible to recover past detections that were previously missed;
- **investigating bounding box regression**: the use of bounding box regression has been shown to be a promising step in improving the MOTA score, but this has only been marginally explored. For example, it may be interesting to investigate the use of past and future information to guide the regression;
- **investigating post-tracking processing**: in batch contexts, it is

possible to apply post-processing steps on the output of a tracker to increase its accuracy. This has already been shown by Babaee et al. [153], that have applied occlusion handling on top of existing algorithms, and by Jiang et al. [171] with the aforementioned bounding box regression step. More complex post-processing procedure might help to further improve the results.

Chapter 4

Deep Learning for Face-Based Video Retrieval

The second part of this thesis focuses on the problem of face-based video retrieval. In particular, I propose a pipeline that allows to use the image of a face to search for videos containing the same person, including a list of the video shots in which the person appears.

As explained in Chapter 1, I performed part of the work described here in collaboration with the company CEDEO s.a.s. [186], based in Turin, Italy. In fact, CEDEO was interested in integrating a video retrieval system powered by deep learning inside their TVBridge platform [187].

TVBridge is an "end-to-end system that allows a broadcaster to offer television programs with enriched use of additional content relevant to specific moments of the program" [188]. It is composed of two parts: an authoring tool (AT), that allows the broadcaster to identify and mark the points of a television program that they want to be enriched with multimedia content (called *bridget*); a phone app, that identifies the program the user is currently watching on TV (using audio fingerprints) and delivers the bridgets to him at the right time.

The goal of my collaboration with CEDEO was to integrate the proposed video retrieval pipeline into TVBridge, in order to build a prototype system able to help TVBridge editors with their job of finding correlated content to show as bridgets in the TVBridge app. In fact, the use of an automated video retrieval pipeline allows an AT user to easily search for videos containing the same person appearing in the TV program he is currently editing. Moreover, the indication of the shots in which the person appears helps the editor to quickly jump to the relevant parts of each video, in order to easily find the relevant video(s) that he wants to include in the bridget. With respect to textual search methods, this kind of content-based search has the advantage of not needing any manual annotation of the videos, since the retrieval is based on the actual visual content, and not on the title or description of a video, which might not contain the name of the person of interest.

In short, the problem can be described as follows: building a system which, given an input face extracted from a video frame (for example, from the TV program that the editor is currently working on), searches a given set of videos for all the videos containing that person, with an indication of the shots in which the person appears. Moreover, the system needs to have the following characteristics:

- being able to search for arbitrary people in the videos. There is no predefined comprehensive set of all the people which might appear in a TV program, so the algorithm cannot just learn to classify a closed set of faces. Knowing the identity of the person (i.e. their name) is not necessary;

- being able to analyze effectively, end-to-end, videos which can be long, have multiple shots, include multiple people, and be of variable resolution and quality, with faces appearing in varying poses and illumination conditions;

- being reasonably efficient both in the video analysis step, which is the most computationally expensive and constrains the hardware requirements and costs, and in the query execution, since users of the system expect a fast response from the server.

The main contributions of this thesis in this regard can be summarized as follows:

- a novel modular pipeline for efficient and effective end-to-end analysis of unconstrained multi-shot, variable-length, variable-quality, videos, for the purpose of video retrieval using person faces. The focus is put on videos with similar characteristics to television content. The pipeline starts from raw, unsegmented videos and outputs features ready to be used for video and shot retrieval. The retrieval protocol is also included in the pipeline;

- the construction of an appropriate video dataset for the evaluation of the retrieval task. As we will see, currently available datasets suffer from a number of issues that make them unsuitable for a fair and realistic end-to-end evaluation of the proposed algorithm, since they are either small or their video clips are short, do not contain multiple shots or are pre-segmented, or contain only one main face. The proposed dataset is an adaptation of the VoxCeleb2 dataset for audio-visual recognition;

- an extensive comparison of the performance of various models and algorithms for shot detection, face detection, face feature extraction and feature aggregation, in the context of the proposed pipeline, in order to find the most effective combination for face-based video retrieval, highlighting pros and cons of each approach. A particular focus is put on deep learning models. The best configuration of the pipeline obtains 97.25% Mean Average Precision on the proposed dataset, while performing a query on thousands of videos in less than 0.5 seconds;

- the integration of the proposed pipeline into the commercial platform TVBridge, developed by CEDEO, to aid the platform's users with the task of searching for video content in the bridget creation process.

At the time of writing, a journal article describing the research presented in this chapter is in preparation [189].

The chapter is organized as follows. In Section 4.1 I will present a summary of the state of the art algorithms for face-based video retrieval, face recognition, face detection and shot detection, including a list of the most important face recognition and retrieval datasets in the literature. In Section 4.2 I will describe the proposed pipeline for face-based video retrieval, while the experiments and the related results will be presented in Section 4.3. In Section 4.4 I will briefly describe how the pipeline was integrated into TVBridge. Finally, in Section 4.5 I will summarize the findings and describe some future directions of research.

4.1 State of the art

4.1.1 Face-based video retrieval

Throughout this thesis I am going to refer to the task of searching for videos or video shots containing a specific person provided as a query, either via a single image or multiple images/frames, as the task of *face-based video retrieval* (FBVR). The problem I have tackled falls indeed in this category. Note that this kind of *query-by-example* problem is distinct from *query-by-keyword*, that involves the user searching for videos using textual keywords or identity names, as opposed to images and video frames [190].

Video retrieval is also distinct from the generic task of face recognition in videos, in that a retrieval algorithm has the goal of returning a list of videos containing the queried subject, while face recognition often refers to different tasks, such as face verification, i.e. determining if two face images/videos contain the same person, or face identification, i.e. assigning a specific identity to a face — either by classifying the image/video into one of N given identities or by comparing it with a series of reference templates to identify which of the templates corresponds to the given face [191].

Face recognition is in fact an important part of a FBVR system. In a sense, we might consider the problem of FBVR as the “inverse” problem of face identification in videos. Instead of finding for each input video[1] the identity it corresponds to, in FBVR the identity is given, and we want to find *all* the videos that contain the given identity. It is then easy to understand how techniques and models used for face recognition are fundamental for a successful retrieval system.

FBVR is a challenging task: the algorithm must deal with unconstrained videos, where faces belonging to the same person can vary greatly across shots and videos, with different poses, resolutions, illumination conditions and possible occlusions; a FBVR system must be able to search and match faces of identities which are not known a priori and might change over time in a real-world scenario, as video databases gradually grow in size. At the same time, an efficient and effective FBVR system can enable new multimedia fruition modalities: for example, we can imagine large-scale retrieval systems similar to Google Images but for videos; or TV show/movie streaming services with the possibility for the user to search for scenes containing a specific character, without needing manual annotations of each scene.

Various works have tackled the FBVR problem throughout the years.

Early approaches. In 2005 Arandjelović and Zisserman [192] proposed a shot retrieval system using faces. In order to obtain a so-called *signature image* of a face, invariant to pose, illumination, scale and occlusion, they proposed a series of steps for each detected face: first, Support Vector Machines (SVMs) [99, 193] were used to detect face feature locations (eyes and mouth); the face was then affine warped in order to align their features to their mean canonical locations; face segmentation was then performed by using image intensity discontinuities and the background was successively removed; band-pass filtering was used to compensate for illumination changes. To determine if two faces belonged to the same person, the L_2 norm of the difference between the two signature images was compared, ignoring pixels

[1]In the problem of face identification, each video usually contains only a single person. This is not necessarily true in video retrieval, including in my case study, as we will see.

which had a high enough probability of being occluded. The algorithm was evaluated by performing queries with each image or image set and ranking the data in order of similarity to it. When evaluated on two movies and a situation comedy episode, with a total of 8830 faces, the algorithm obtained high precision and recall scores (93% and 92% respectively).

Sivic et al. [194] improved on the previous system. Shot detection was performed on two movies, faces were detected and tracking was performed to obtain 776 face tracks over 337 shots. SIFT descriptors [195] were used to collect face features, which were aggregated for each face track. The retrieval was then performed with the following procedure: a user selected a face from a shot, the features from the corresponding track were recovered, and the face tracks with the lowest χ^2 distance from the query track were retrieved. The algorithm obtained 98.1% precision and 90.7% recall in the face matching process.

Fusion of visual features with measurement information. Herrmann et al. [196] proposed to encode face track features (which included image intensity, LBP [197], LDP [198] and SIFT [195] descriptors) together with measurement information (such as camera position and head pose/rotation). The features from all the face track frames were then processed to compute cumulative descriptors for the entire track using techniques such as Bags of visual words [199] or Fisher Vectors [200]. These descriptors were then used to form the database on which retrieval was performed. 10-fold cross-validation queries were performed on datasets of tracks such as YouTube Faces (YTF) [201] and Face In Action (FIA) [202], where a mean Average Precision (MAP) of 17.0% and 93.0% was obtained respectively.

L-QTS. Arandjelović [203] proposed a method, called Learnt Quasi-Transitivity (L-QTS), to perform retrieval of image sets. Specifically, they proposed a meta-algorithm that, starting from a baseline algorithm, used the structure of the data at hand to enhance the performance on image sets that are not correctly retrieved by the starting baseline. The algorithm exploited the quasi-transitivity on similarity scores computed between the query, target and proxy image sets using the baseline algorithm. The authors used LBP as frame features and two different similarity measures, the *maximum maximorum* cosine similarity [201] and the Locality-constrained Linear Coding (LLC) [204]. They showed that the meta-algorithm improved the performance over the baselines on frame sets from the YTF dataset.

HHSVBC. Li et al. [205, 206, 207] proposed a compact binary representation for video data, called Hierarchical Hybrid Statistic based Video Binary Code (HHSVBC). The frames were represented by Fisher Vectors computed on one of various possible local features (raw gray features, histogram equalized gray features, Dense SIFT descriptors [195], LBP and Histogram of Oriented Gradients (HOG) [85]). Variability across frames was then encoded using covariance matrices. Fisher Covariance Matrices of different sizes were used to obtain coarse and finer representations of the data. These high-dimensionality features were transformed into a low-dimensionality vector in Hamming space by means of multiple SVMs with Riemannian kernels [208] (covariance matrices are known to lie on a Riemannian manifold). The algorithm was tested on a set of around 14,000 face tracks extracted from two TV series, The Big Bang Theory and Prison Break (ICT-TV dataset [207]). It reached 91.72% and 30.35% MAP on those two TV shows, respectively [207].

HER. A variation of the previous algorithm that can deal with mixed modalities, i.e. using single images to query video data, was also presented by the same authors [209]. Since images were usually represented with vectors in Euclidean spaces, while face tracks were often represented by covariance matrices on Riemannian manifolds, the authors presented a way to learn a mapping of the two heterogeneous spaces into a common Hamming space: they embedded the two spaces into Reproducing Kernel Hilbert Spaces (RKHS) and then used SVMs to learn the hashing function based on Hamming distance. The algorithm was named “Hashing across Euclidean space and Riemannian manifold” (HER). Tests over faces extracted from The Big Bang Theory and

Buffy the Vampire Slayer episodes obtained a MAP of 55.39% and 58.77% on the two datasets respectively, higher than single modality hashing methods[2].

First uses of deep learning. Dong et al. [210] beat the scores obtained by HHSVBC by proposing the use of a deep neural network in order to learn the hash code to represent the video data. First, the AlexNet CNN was pretrained on ImageNet. A pre-trained hash function was obtained using SVMs, then a triple ranking loss was devised to train the network and the hash function jointly using gradient descent. The algorithm obtained 94.12% and 32.61% MAP on The Big Bang Theory and Prison Break datasets respectively, following the usual protocol of running 10 queries for each annotated character. A later refinement of the procedure [211], by training the network on an additional dataset, CASIA-WebFace, with the use of hard negative mining, obtained 98% MAP and 83% MAP on the two TV shows. The authors also tested cross-modal queries (image query vs. video database and vice-versa), obtaining results comparable to the video-to-video retrieval task.

Mühling et al. [212] presented a system which was able to perform video search based on textual descriptions or face images, in addition to face identification and clustering. In particular, regarding video retrieval by faces, the authors used Faster R-CNN with ZFNet backbone for face detection [213, 214] on video shots that were segmented using two shot boundary detection algorithms [215, 216]. Face overlap and RGB histogram intersection (like in [217]) were used in conjunction with LBP features in order to perform face tracking across frames, keeping only tracks with at least 5 faces. Face representations were obtained using the same network presented in [218], with 11 convolutional layers and a 320-dimensional output vector. The network was trained on the CASIA-WebFace dataset. For each face track, the feature vectors for each face image and the average feature vectors were saved. At retrieval time, the authors compared two different approaches to measure face

[2]Note that these results are not comparable to the ones from HHSVBC, since in this case queries on the videos were performed using images, while HHSVBC used video queries.

similarities: one involved simply computing the cosine similarity between the input face feature vector and the average face track feature vector; the second method instead consisted in computing the cosine similarity between the input face feature vector and all the face feature vectors contained in each track, and then averaging the similarities to produce a final score. The authors showed that the latter approach produced better results, with 67.24% MAP compared to 59.96% MAP obtained with the first approach. The test was done by querying the Movie Trailers Face Dataset [217], which contains 101 trailer videos, using face images from 6 different persons, extracted from the PubFig+10 dataset [219], as queries. Comparing each query image with all the faces in each track entailed a significant time penalty however, with a 250 ms runtime with respect to 10 ms for the single comparison approach. Moreover, the AP varied significantly among the 6 celebrities used as queries: queries on Paul Rudd obtained 92.56% AP, while queries on Jennifer Lopez only reached 2.05% AP.

DVC. Qiao et al. [220] proposed the use of temporal feature pooling to fuse information extracted by a 3-layer CNN run on each input frame. Fully-connected layers were put on top to predict the binary hash representation of each video, called Deep Video Code (DVC), and they were trained using a smooth triplet loss. The network was able to reach 99.41% and 97.88% MAP in the retrieval task on the clips from the two TV shows from the aforementioned ICT-TV dataset. They also showed that the use of temporal max/average pooling was better than using a 3D CNN.

Correlation features for video retrieval. Jing et al. [221] proposed the fusion of CNN features, extracted from an AlexNet backbone fine-tuned on CASIA-WebFace, with correlation features, that were computed on the CNN features after PCA dimensionality reduction. The correlation matrix was mapped to Euclidean space using a logarithm and Log-Euclidean distance, and it was then converted to a vector. The CNN features and the correlation vector were then concatenated and fed to a fully-connected layer, playing the role of a fusion layer. A final layer then learned the feature mapping to Hamming space in order to produce the hash code used for retrieval. The authors tested the network in a retrieval task on the ICT-TV dataset, showing that: 1) correlation features helped getting better performance and 2) extracting features from all the clip frames and averaging them led to better results. The network reached 99.24% MAP on The Big Bang Theory and 84.61% MAP on Prison Break.

Integration of image quality information. Fang et al. [222] proposed a model which used quality information such as face detection confidence in order to aggregate and denoise the face features provided in the IQIYI-VID 2019 [223] dataset. The authors then trained a Multi-Layer Perceptron (MLP) to classify faces into the 10,034 identities of the dataset. The network obtained 89.83% MAP in the retrieval task. However, the presented method had a major drawback: it could only work with a pre-defined set of identities. Adding new identities to the database would require changing the network structure and retraining it. For this reason, the method is not suitable for retrieval tasks with unknown identities.

DHH. Recently, Qiao et al. [224] proposed and end-to-end Deep Heterogeneous Hashing (DHH) method that integrates the three main stages of feature representation, video modeling and heterogeneous hashing, and learns them jointly. Differently from approaches such as the one presented by Li et al. [209], the Riemann-to-Euclidean mapping of the CNN features of the videos/images can be used for multi-modality retrieval and the hashes and features are learned simultaneously. The authors used a fast 10-layer VGG-like network specifically designed for face recognition tasks [218]. DHH obtained 61.2% MAP on YouTube Celebrities (YTC) [225], 95.63% on Prison Break and 47.36% on UMDFaces [226] in the video retrieval task with image queries. The authors also compared their binary representations with real-valued features: as expected, the compressed binary representation obtained slightly lower MAP since some of the information is inevitably lost in the compression to a lower-dimensionality representation.

HVIH. Wang et al. [227] proposed a Hybrid Video and Image Hashing (HVIH), by which both dense frame-level features and video-level aggregated features are exploited, as opposed to the usual strategy of using only the aggregated ones. A CNN is used to extract features, that are then temporally pooled to produce the video-level features. Both levels of features are transformed to Hamming space. Training was done by jointly applying both frame-wise and image set-wise supervision. The training procedure also featured a video center alignment (VCA) technique that aimed at correcting video-level feature shift with respect to the frame-level pooled features. Both the image-to-video and video-to-video retrieval tasks were evaluated on the YouTube Celebrities and UMDFaces datasets, and the algorithm reached MAP higher than 70% for video-to-video and higher than 60% for image-to-video on both datasets.

The main limitation of the vast majority of the presented algorithms is that they deal with short-length video clips which often contain a single main face for each video. In many cases, in fact, the algorithms start from already cropped videos of face tracks, which are not representative of real-world videos and cannot thus be directly employed in a real-world system, since they do not include nor evaluate the use of different shot boundary detection, face detection and face tracking algorithms. Some other works are limited to very few identities and videos, or to videos constrained to one or two TV shows, which can be insufficient to effectively evaluate the algorithm in real-world scenarios, where the data has higher variability. These are the main motivations for the development and evaluation of an end-to-end FBVR pipeline, which includes the comparison of a variety of shot detectors, face detectors, face trackers and face feature extractors, and is tested on a dataset that is suited to a fair FBVR evaluation, closer to real-world condition. For a deeper look into the limitations of existing datasets, see Section 4.3.1.

4.1.2 Face recognition

As already mentioned, face recognition algorithms play an important role in FBVR. For this reason, I present here a brief summary of the state of the art of face recognition algorithms, with a focus on deep learning methods, since they have recently reached top performance in the face verification and identification tasks [191, 228].

DeepFace. In 2014, DeepFace [229] was one of the first methods to show that deep neural networks had the potential to reach human performance in face verification. The authors proposed the use of a CNN with both convolutional and locally-connected layers[3], using 3D-aligned faces as input. The authors tested different methods to compare the features extracted by the CNN: the inner product between normalized feature vectors, χ^2 distance with SVM-learned weights, and a Siamese CNN, with the latter one obtaining better results on the Labeled Faces in the Wild (LFW) dataset [230]. The authors trained the network on a private dataset of Facebook photos. They tested both the "single" network and an ensemble of networks, each using different inputs (3D-aligned images, 2D-aligned images, greyscale images with gradient information). The network ensemble obtained 97.35% accuracy in face verification on the LFW dataset, compared to 97.53% obtained by humans [231]. The network was also tested on the videos from the YTF dataset, on which it reached 91.4% accuracy, compared to less than 80% accuracy from classical non-deep methodologies.

DeepID. Around the same time, Sun et al. proposed the DeepID algorithm [232]. 60 CNNs extracted features from 60 different face patches, obtaining a final 19,200-dimensional feature vector. The Joint Bayesian technique [233] was then used to perform face verification using those features. The network

[3]Locally-connected layers are similar to convolutional layers, in that they do not connect each output neuron to every input neuron. However, connections in locally-connected layers do not share weights.

obtained results comparable to the ones by DeepFace. The same authors later proposed an improved version of this method, named DeepID2 [234], that used AlexNet CNNs and contrasting loss for training, in order to produce feature vectors closer together for faces of the same person, and farther apart otherwise. The network reached 99.15% accuracy on LFW, training the network on the CelebFaces+ dataset [232]. A further improvement was made with the DeepID3 network [235], that used 25 networks with VGG-10-like structure mixed with Inception-like layers, trained to extract features on 25 face patches. The new network structure obtained 99.53% accuracy on LFW.

FaceNet. An even higher accuracy of 99.63% on LFW was obtained by FaceNet [236] by using a GoogleNet-like CNN trained with a triplet loss function and online semi-hard negative mining. Differently from the constrastive loss of DeepID, which only compared pairs of images, the triplet loss compares three images: an anchor with a positive (i.e. same-class) example and a negative (i.e. different-class) example. The loss enforces a margin between images belonging to the same class and images from different classes. This is similar to the loss used for the Siamese CNNs that we have seen in Chapter 3. Another difference from previous works is that the network was directly trained to learn the feature embedding, instead of using an intermediate layer to extract the features. The network was also tested for face verification on YTF, by averaging the distances between all pair of faces in the first 100 frames of each clip, and obtained 95.12% accuracy.

VGGFace. Parkhi et al. proposed the use of a VGG-16 network to learn a Euclidean embedding using triplet loss on their newly proposed VGGFace dataset [237]. The triplet loss used by the authors to learn the Euclidean embedding can be expressed as follows:

$$E(W) = \sum_{(a,p,n)\in T} \max\{0, \alpha - \|x_a - x_n\|_2^2 + \|x_a - x_p\|_2^2\}, \quad x_i = W\frac{o_i}{\|o_i\|_2},$$

where o_i is the output of the CNN on sample i, W represents the weights of the layer which projects the output of the network onto the Euclidean embedding, α is a learning margin, T is a collection of training triplets, with (a, p, n) being the anchor, positive and negative examples respectively. The network thus tries to maximize the distance between the features of the anchor and negative samples, while minimizing the distance between the anchor and positive samples. While VGGFace was the largest face image dataset publicly available at the time, it was still much smaller than Google/Facebook private datasets. However, the authors managed to reach comparable performance (98.95% verification accuracy) despite using a smaller dataset and a much simpler network architecture. The network also reached 97.3% accuracy on YTF, the highest at that time. A bigger version of the VGGFace dataset, VGGFace2, was proposed in 2018 [238], with a greater variety in pose, ethnicity and age of the faces. The authors trained a ResNet-50 and a SENet-50 to classify identities on their new dataset using standard cross-entropy loss. They showed that training CNNs on this new dataset led to better accuracy than using only the original VGGFace dataset or only MS-Celeb-1M [239], reaching state-of-the-art performance on the IJB-A dataset [240]. Pre-training the networks on the MS-Celeb-1M dataset and then fine-tuning on VGGFace2 led to the best results.

Loss functions for face recognition. Between 2016 and 2018, a lot of focus was put on the design of better loss functions to help separate faces belonging to different identities. Wen et al. [241] proposed to combine cross-entropy loss with a so-called Center Loss to train the CNN, which computes the center of the deep features for each class and minimizes the distance between the features and the class center. Later, Qi et al. proposed an improvement of this loss, Contrastive-Center Loss [242], which introduces the maximization of the distance between each feature vector and the centers of different classes. The network reached 98.68% verification accuracy on LFW.

Liu et al. [243] proposed a large-margin Softmax loss[4] (or L-Softmax)

[4]While it should technically be called “cross-entropy loss”, it is not rare to see it called “softmax loss” in the literature, since it is often used in conjunction with a softmax activation layer.

to again encourage a large margin between different-class samples and a small margin between same-class ones, with the additional advantage of preventing overfitting.

Liu et al. [244] presented the congenerous cosine (COCO) loss, which included explicit cosine similarity optimization among the feature vectors. An Inception-ResNet model trained with the COCO loss obtained state-of-the-art performance on LFW, with a record 99.86% accuracy.

Liu et al. [245] later proposed the SphereFace algorithm. Features were embedded on a hypersphere with the use of an Angular margin Softmax (A-Softmax) loss.

Qi et al. [246] proposed the use of an Additive Angular Margin (AAM) loss together with centralized coordinate learning (CCL), in order to force the features to be more dispersed in the feature space and lie on a hypersphere.

Differently from SphereFace, Wang et al. [247] used an additive instead of multiplicative margin to the softmax loss.

Zheng et al. [248] proposed the Ring Loss, a way to augment standard loss functions with the ability to gradually push the network to compute normalized feature vectors.

Wang et al. [249] presented CosFace, a ResNet trained with a Large Margin Cosine Loss (LMCL), which maximizes the margin between cosines of the angles of feature vectors.

Huang et al. [250] proposed a Cluster-based Large Margin Local Embedding (CLMLE), which consisted in training a network using a loss that can account for the presence of subclusters within larger classes in imbalanced datasets. The loss enforces large inter-cluster margins both within the same class and across different class boundaries. The CNN trained with this loss was able to reach 99.62% and 96.5% verification accuracy on LFW and YTF, respectively.

ArcFace. Deng et al. recently presented ArcFace [251], which is a further improvement over CosFace and SphereFace. It works by using the arc-cosine function to compute the angle between a feature vector and a target weight vector, which acts as a class center. An additive angular margin is then added to the angle obtained and its cosine is computed. The obtained logits are then fed to regular softmax. The loss can be expressed as follows:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{e^{s(\cos(\theta_{y_i}+m))}}{e^{s(\cos(\theta_{y_i}+m))}+\sum_{j=1,j\neq y_i}^{n}e^{s\cos\theta_j}},$$

where N is the batch size, n is the number of classes, θ_j is the angle between the normalized feature vector extracted by the network and the weight vector representing the center of class j, y_i is the ground truth class of sample i, m is the additive angular margin penalty between the ground truth weight vector W_j and the feature vectors x_i, and s is a scaling factor used to distribute the learned embedding features on a hypersphere with radius s. Note that $\cos\theta_j = \frac{W_j^T x_i}{\|W_j\|\|x_i\|}$, so it can be simply obtained by performing the inner product between the normalized feature and weight vectors. The loss encourages the network to learn a mapping that keeps intra-class feature vectors close to their class centers and forces an angular margin between vectors belonging to different classes. ArcFace obtained state-of-the-art accuracy on a number of face recognition datasets, such as YTF (98.02%), Megaface [252] (96.98% verification accuracy and 81.03% Rank-1 identification accuracy), LFW (99.83%), IJB-B [253] (94.2% verification accuracy), IJB-C [254] (95.6% verification accuracy), and more. The network was also tested on the iQIYI-VID 2018 dataset [255] where it reached 79.80% Mean Average Precision (MAP) in the video retrieval task. A three-layer fully-connected network was added in order to obtained customized feature descriptors for the challenge.

Face recognition in videos. While the methods described above mostly tackle the problem of face recognition in images, a few works have also focused on the task of face recognition in videos instead of images.

In particular, some works explicitly exploited the temporal structure of the videos using deep neural networks. For example, Rao et al. [256] proposed a Generative Adversarial Network (called Discriminative Aggregation

Network – DAN) to generate an “aggregated” face image starting from a series of consecutive input images taken from a video. Instead of extracting discriminative features from each frame, the features were extracted directly on the aggregated face image. The authors claimed that their system reduces the number of processed frames and increases the discriminative power of the extracted features. The algorithm reached state-of-the-art results compared to methods at that time, with 94.28% accuracy on YTF, 92.06% and 80.33% on the two video subsets of the Point-and-Shoot Challenge (PaSC) dataset [257], and 97.32% on the YTC dataset.

The same authors also presented an attention-aware deep reinforcement learning (ADRL) method [258], in order to selectively choose the most important frames to consider when comparing two face videos. Both CNNs and LSTMs were used to extract spatio-temporal features, used by a Frame Evaluation Network in conjunction with the raw frame images to iteratively select the most representative frames. The system reached 96.52% verification accuracy on YTF, 95.67% and 93.78% on the two video subsets of PaSC, and 97.82% accuracy on YTC.

Ding et al. proposed the Trunk-Branch Ensemble Convolutional Neural Network (TBE-CNN) [259] to tackle the pose variability in videos by adding side branches to the network, trained to extract features from specific image patches, which are less sensitive to face pose. In addition to that, they proposed to add blurred images to the training set in order to train the network to deal with blurred faces, which can often be found in video frames. Finally, they proposed a refined triplet loss in order to regularize the distance between each identity’s mean representation. The system obtained 97.80% and 96.12% verification accuracy on the two video subsets of PaSC, and 94.96% accuracy on YTF.

Recently, Zheng et al. [260] proposed an end-to-end pipeline to perform face recognition in unconstrained multi-shot videos. The pipeline is composed of the following steps: first, faces are detected in the probe videos using the SSD CNN [261] (for non-surveillance videos); facial landmarks are extracted using the All-in-One Face CNN [262], in order to align the faces; features are extracted using one of three possible CNNs described in [263]. Intra-shot faces are pre-associated by tracking the bounding boxes using the Kernelized Correlation Filter (KCF); a one-shot SVM is then used to further associate faces across multiple shots. Since the algorithm was tested on the IJB-B dataset, which indicates the person of interest for each video, the SVM could be trained to classify faces as belonging to the target person or not, based on the face features. In order to aggregate the features for each video, the authors proposed two possible techniques to map the deep features onto subspaces: "regular" subspace learning (equivalent to performing PCA) and quality-aware subspace learning (similar to a PCA, but weighting samples in the objective function according to the detection score, used as a proxy for face quality). Matching between the aggregated features is then done by one of various proposed techniques: principal angles between subspaces, a combination of principal angles and cosine distances between average features, a combination of principal angles and cosine distances between quality-based averaged features, by using a variance-aware projection metric, or by a combination of the previous methods. Combining the various metrics led to the best results on the video face identification protocols of IJB-B and IJB-S [264] datasets.

4.1.3 Large-scale datasets for unconstrained face recognition and retrieval

Large-scale face image datasets. One of the first important large-scale still image face datasets was Labeled Faces in the Wild (LFW) [230], published in 2007. It included 13,233 unconstrained face images from 5,749 identities and it was tuned for face verification. Other important publicly-available large-scale datasets published in the following years, and focusing on still image face recognition in unconstrained environment, include the CASIA-WebFace dataset [218], with 494,414 images of 10,575 celebrities, VGGFace [237], with

2.6 million images of 2,622 celebrities, annotated with bounding box and poses, the MS-Celeb dataset [239], which in its latest version included 6.8 million images of 180,000 celebrities, MegaFace [252][5], with 4.7 million images of 672,052 identities, and VGGFace2 [238], with 3.31 million images of 9,131 identities.

Large-scale face video datasets. More importantly for the goal of this thesis, a number of datasets for face recognition and retrieval in videos have also been published.

YTF. One of the first ones is the YouTube Faces (YTF) dataset [201], presented by Wolf et al. in 2011. It is composed of 3,425 videos of 1,595 celebrities, with an average of 2.1 videos per subject. It was built from YouTube videos using the Viola-Jones face detector [266] to extract clips containing each subject.

UMDFaces. UMDFaces [226, 267] included 3,735,476 annotated video frames extracted from 22,075 videos with 3,107 subjects. The dataset was focused on face verification. At the time of writing, the dataset is in maintenance, and not available for download[268].

The IJB datasets. The IARPA Janus Benchmark-C (IJB-C) [254] dataset, published in 2017, is an extension of the IARPA Janus Benchmark-A (IJB-A) [269] and IARPA Janus Benchmark-B (IJB-B) [253] datasets, and it is one of the most comprehensive face detection and recognition datasets in videos that are currently publicly available. It provides 31,334 images, and frames extracted from 11,779 videos containing 3,531 subjects. Person-centric videos (e.g. interviews) were collected from YouTube, and the dataset thus presents a large variation in face pose, illumination conditions, and other visual properties. The challenge provides many evaluation protocols, including

[5]Both MS-Celeb and MegaFace are not available anymore for download. The first was retired for ethical concerns, the second one because of administration costs[265].

face detection, 1:1 face verification (determining if two templates, made of single or multiple images, belong to the same person), 1:N face identification (determining for each probe template the corresponding gallery template, if any). In addition to those, the benchmark includes end-to-end protocols, which require the algorithm to first detect the faces in the input image/video, and then match them against the gallery templates.

iQIYI-VID. The iQIYI-VID-2019 dataset [223] is a large-scale video retrieval dataset presented by iQIYI for the 2019 iQIYI Celebrity Video Identification Challenge, in conjunction with the 2019 ACM International Conference on Multimedia. It is an extension of the iQIYI-VID-2018 dataset [255], published in 2018 for the Multi-modal Video-based Person Identification Challenge as part of the Chinese Conference on Pattern Recognition and Computer Vision (PRCV 2018). The 2019 version contains 211,490 video clips from 10,034 identities. The goal of the challenge is to search for videos in the test set for each of the identities included in the training set. The models can be trained to recognize each identity by using the training videos, including audio, but external data can also be used. The test set also includes distractor videos.

VoxCeleb. Recently, the VoxCeleb datasets have been published, being the first large-scale datasets for audio-visual person recognition in unconstrained videos. VoxCeleb consists “of short clips of human speech, extracted from interview videos uploaded to YouTube” [270]. It was published in two versions: VoxCeleb1 [271] and VoxCeleb2 [272]. The main goal of the dataset is to help develop and evaluate audio-visual models for speaker recognition. They both provide YouTube URLs to the original videos, face detections and tracks, the audio files with the utterances of each clip, cropped face videos and speaker metadata. The main difference between VoxCeleb2 and VoxCeleb1 is in the size of the dataset and the heterogeneity of the identities, both improved in the second version of the dataset. VoxCeleb2 contains 1,128,246 utterances extracted from 150,480 videos and 6,112 speakers.

Movies and TV shows. In addition to the aforementioned datasets, which mainly focus on interviews and YouTube videos, there exist datasets for face recognition in movies and TV series, albeit usually limited in size. For example, there are datasets with face annotations for *Buffy the Vampire Slayer* [273], *The Big Bang Theory* [274] and *Sherlock* [275], each limited to the small set of characters included in a TV series. They also do not provide the original video material, for copyright issues. The Cast Search in Movies (CSM) dataset [276] instead provides a larger number of identities (1,218) included in 127,000 tracklets from 192 movies. The authors tackle the problem of searching tracklets with a single query image. The tracklets are provided with the dataset, but not the entire videos, again for copyright reasons.

4.1.4 Face detection

Another important component of a FBVR system is face detection: in real-world scenarios, face bounding boxes are of course not included with the video data, and must be extracted in order to identify the people in videos.

Deep learning algorithms have obtained excellent results in the face detection task, compared to classical ones like the Viola-Jones algorithm [266], although some older techniques still proved useful, such as the use of detection cascades to reduce the computational overhead [277].

For example, Li et al. proposed in 2015 one of the first uses of a CNN for face detection [278]. They proposed the use of a cascade of 6 CNNs, 3 for face detection and 3 for face bounding box calibration. They built an image pyramid in order to recognize faces at different scales. The first network was executed on a sliding window over each image in the pyramid. A calibration network refined the boxes and Non-Maximum Suppression (NMS) was applied. The remaining boxes were then re-classified in a second classification CNN and re-calibrated by another calibration CNN, including NMS at the end. The same steps were finally performed a third time, returning the final set of face bounding boxes as a result.

MTCNN. Zhang et al. proposed the widely-known Multi-task Cascaded Convolutional Network (MTCNN) [279]. A network cascade of three CNNs was employed, called P-Net (Proposal Network), R-Net (Refinement Network) and O-Net (Output Network), using an image pyramid similar to [278], but this time the bounding box regression was included as an output branch of each CNN, without needing an entire separate CNN. A NMS step was also performed on the output of each CNN. MTCNN is capable of detecting 5 facial landmarks (eyes, nose, mouth corners). The network structure is shown in Figure 4.1. The networks were jointly trained on the three tasks (hence the "Multi-task" in the name): face detection, bounding box regression and landmark detection. Online hard negative sample mining was used during training. The network was able to run at 16 frames per second (FPS) on a 2.6 GHz CPU and 99 FPS on a GPU, while obtaining state-of-the-art results on the Face Detection Data Set and Benchmark (FDDB) [280], WIDER FACE [281] and Annotated Facial Landmarks in the Wild (AFLW) datasets.

HyperFace. Ranjan et al. also proposed a multi-task network, called HyperFace, to perform face detection, landmark localization, pose and gender prediction [284]. They used Selective Search [285] to extract region proposals, similarly to R-CNN [28]. They proposed two version of the network, a faster one using AlexNet, and a slower but more accurate one using ResNet as backbone. The network fused features at different levels in order to use both low-level and high-level features effectively. They also proposed to refine the regions obtained by Selective Search using an Iterative Region Proposals (IRP) algorithm. A novel Landmarks-based Non-Maximum Suppression algorithm was also proposed in order to perform NMS by taking into account landmark coordinates. The multi-task training was once again shown to be more effective than learning each task separately, and HyperFace reached state-of-the-art performance for the various tasks on a number of datasets, including CelebA, Annotated Facial Landmarks in the Wild (AFLW) [286], Annotated Faces in the Wild (AFW) [287] and others.

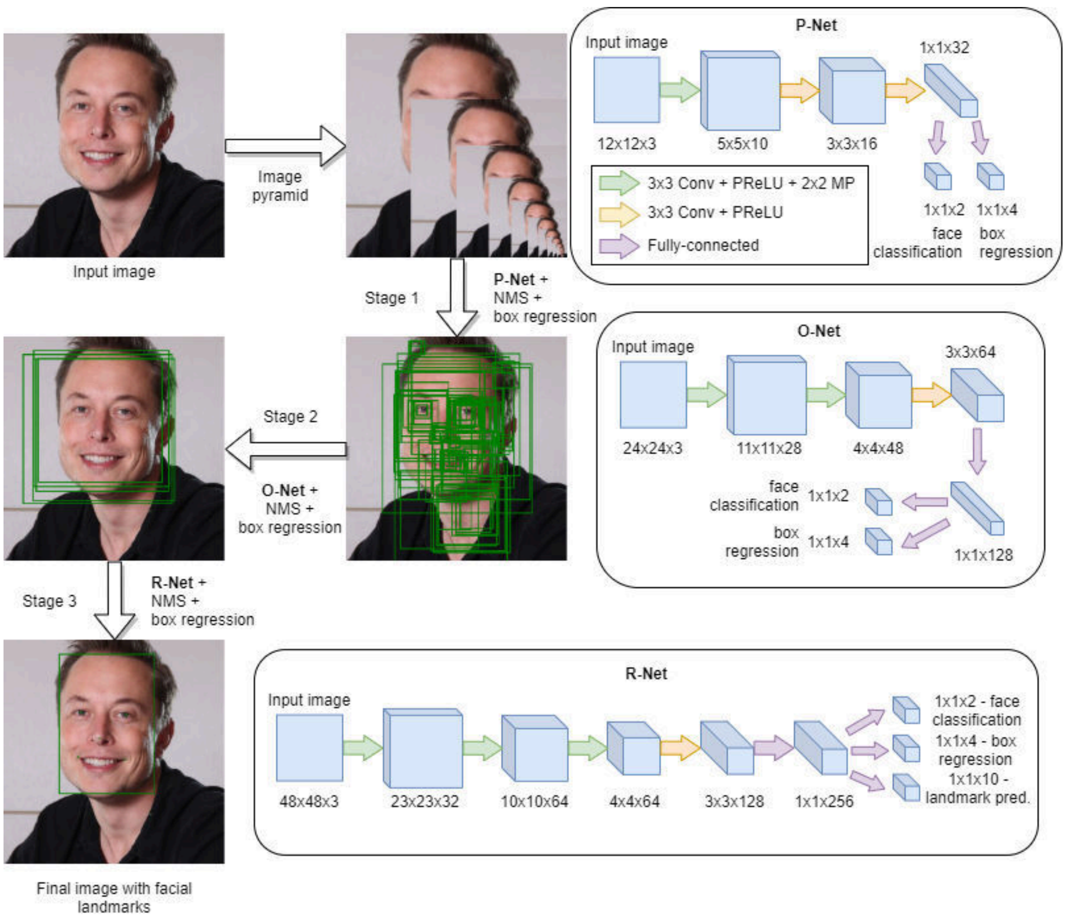


Figure 4.1: Architecture of the MTCNN face detection algorithm. Example detection on a face image is shown. First, an image pyramid is created. Then the image is fed to P-Net, which predicts candidate boxes, refined with bounding box regression and Non-Maximum Suppression (NMS). The candidates are fed to O-Net, which operates similarly, and its candidates are finally fed to R-Net, which also predicts face landmarks. Note that the network hyperparameters shown here follow the ones from the official GitHub repository [282] and not the ones in [279]. *Conv* stands for convolution, *MP* stands for Max Pooling, *PReLU* is the Parametric Rectified Linear Unit activation function [283].

All-In-One CNN. Ranjan et al. also proposed a similar network to HyperFace, called All-In-One network [262], with the addition of the face recognition, face age estimation and smile detection tasks. Differently from HyperFace, the network was initialized using weights from a face recognition task, and it was trained on six datasets performing domain-based regularization. The network obtained accuracy improvements over HyperFace.

Supervised Transformer Network. Chen et al. [288] proposed a network similar to Faster R-CNN in structure, but replacing the bounding box regression branch with a landmark localization branch. They added a Supervised Transformer layer (which gives the name to their network: Supervised Transformer Network), which learns the canonical position of faces in an end-to-end manner, in order to compensate for pose variations. They also replaced NMS with a non-top-K suppression strategy over the regions proposed by the Region Proposal Network.

Detecting faces at multiple scales. A variety of works focused on detecting faces at different scales. Hao et al. [289] proposed the use of a Scale Proposal Network (SPN) in order to choose which scales of the image pyramid need to be scanned by the RPN for region proposals. This was done to reduce computational time, since the RPN was not run anymore on every image in the pyramid.

Hu and Ramanan [96] proposed the use of a coarse image pyramid combined with two different types of face box templates, each tuned for either bigger or smaller images, in their “Tiny Faces” CNN. They also showed that context can help detecting tiny faces more easily.

Yang et al. [290] dropped the use of image pyramids, employing instead three networks with different structures to detect faces of different scales in their ScaleFace CNN. They claimed that smaller CNNs can detect smaller faces better. The three CNNs were fused into a single-backbone CNN, thereby increasing the computational speed of the model.

Zhang et al. [291] proposed a single-shot face detector, S^3FD. The network structure was similar in concept to SSD [32], but with the use of a wider range of anchor-associated layers, in order to predict faces across different scales more effectively.

SSH. Najibi et al. [292] presented the single-shot “headless” CNN (SSH), a fully-convolutional CNN. The removal of the fully-connected layers helped reduce the number of parameters and the computational time. Three branches on top of different layers of the VGG-16 backbone were added to perform detections of faces at different scales. Each detector module in each branch contains a so-called context module, which performs consecutive 3x3 convolutions in order to increase the receptive field and to consider context in the detection process. SSH regresses anchor boxes to produce the final detection boxes, in a similar way to the RPN in Faster R-CNN. The network structure is shown in Figure 4.2. The authors showed that a headless VGG-16 could beat state-of-the-art ResNet-101 detectors on the WIDER dataset.

Detection of occluded and “hard” faces. Wang et al. [293] focused on detecting occluded faces by proposing the Face Attention Network (FAN). The network included an anchor-level attention mechanism, able to highlight the features in face regions and suppress features from non-face regions. The network followed a feature pyramid structure like in RetinaNet [294], and the use of a larger number of occluded faces in the training phase allowed to reach state-of-the-art performance on the WIDER FACE dataset.

Shi et al. [295] proposed the use of Progressive Calibration Networks (PCN) in order to detect rotated faces. The system used a cascade of three CNN, which, besides predicting the face bounding boxes, progressively predicted the rotations to apply to region candidates in order to obtain upright faces, thus producing more reliable detections of rotated faces.

Yang et al. [296] approached the problem of unconstrained and occluded faces by the use of attribute-aware CNNs, which detected various facial features such as hair, eyes, nose, mouth and beard. The locations of the detected features helped determine a number of region proposals, analyzed and classified

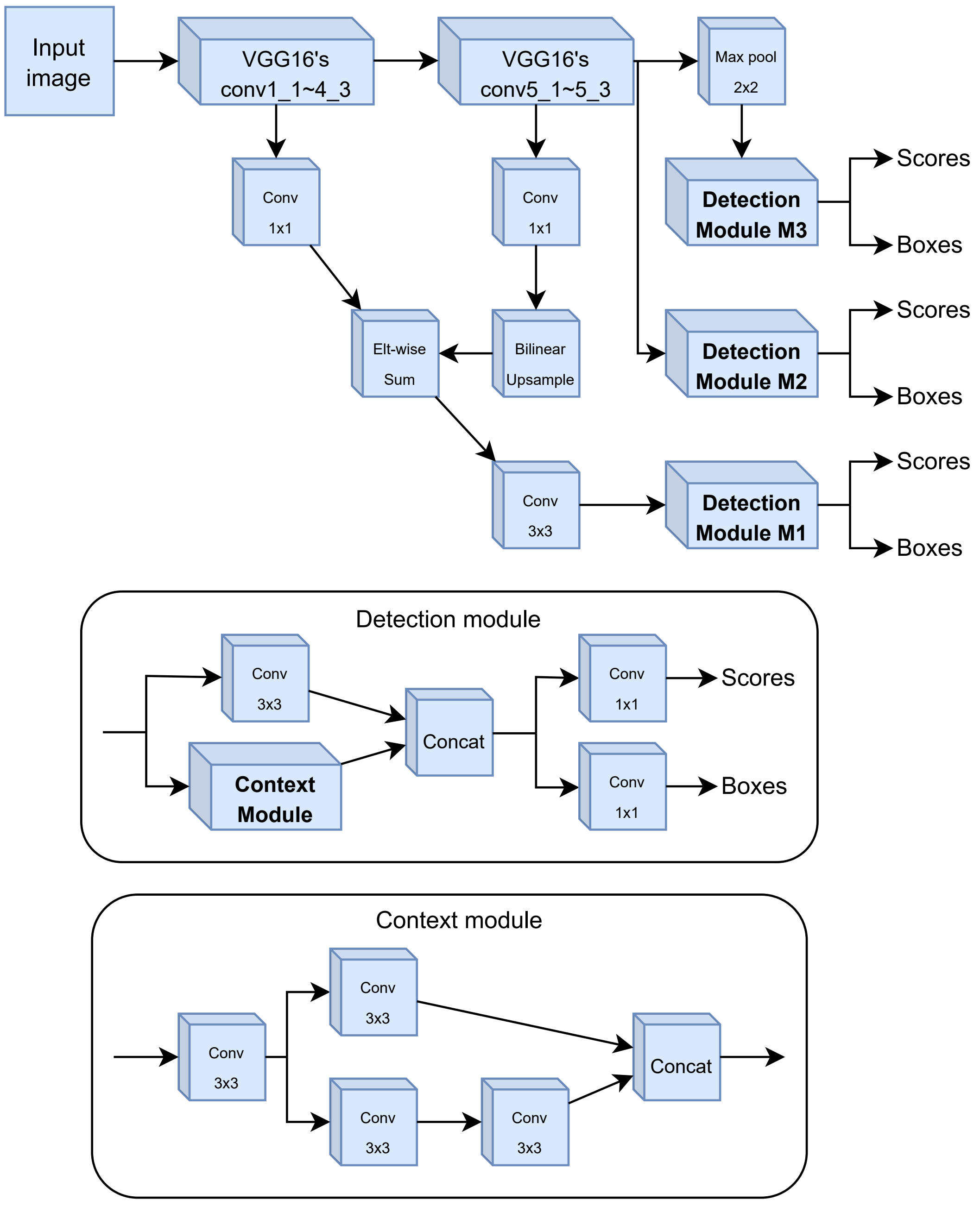


Figure 4.2: Architecture of the SSH face detection algorithm. Each block represents a layer or set of layers of the network. Each of the Detection modules M1, M2 and M3 looks at features at different downscaling factors (8, 16 and 32), thus detecting small, medium and large faces, respectively. Since SSH is fully-convolutional, the image input size is not fixed.

by an additional CNN. The entire pipeline was called Faceness-Net.

Tang et al. [297] proposed the single-shot PyramidBox detector. Its structure was based on the S^3FD detector, but with the addition of the Low-level Feature Pyramid Network (LFPN), which helped merge facial and contextual features in order to detect hard faces (e.g. occluded and small faces). The authors also changed the training sampling strategy in order to put more emphasis on smaller faces.

Zhang et al. [298] proposed a refined Faster R-CNN for face detection, called FDNet 1.0. They reduced the computational complexity of the predition head, introduced a deformable layer before ROI pooling to better exploit image context, and multi-scale training and testing was performed. The network reached state-of-the-art performance on WIDER FACE.

Chi et al. [299] introduced two-step classification and regression modules inside a one-shot face detector in order to reduce false positives and improve the location accuracy of faces. Their Selective Refinement Network (SRN) also included a Receptive Field Enhancement module, in order to improve detection of non-square faces.

RetinaFace. Recently, Deng et al. [300] proposed RetinaFace, a lightweight face detector that benefits from an extensive multi-task strategy. Besides the classic face detection and bounding box regression branches, the network benefits from landmark prediction, self-supervised 3D mesh prediction and camera pose prediction. The network uses Feature Pyramid Networks and the concept of a context module similar to the one from SSH, and the authors evaluated the use of two different backbones: ResNet-152 for maximum accuracy and MobileNet-0.25 [301], capable of real-time detection on 4K images. MobileNet was in fact designed to run on mobile devices. It consists of 28 layers and exploits depth-wise separable convolutions[6] in order to reduce

[6]Depth-wise convolutions can be seen as a two-step version of the classical convolution. First, channel-wise kernels are convolved with the input volume, then a 1x1 convolutional layer is added on top. The depth-wise convolution results in much faster computation and less parameters.

the number of parameters and the computational time. The network structure is shown in Figure 4.3. The added supervised and self-supervised tasks allowed RetinaFace to reach state-of-the-art performance on WIDER FACE, with 91.4% AP, and the use of detections obtained by RetinaFace improved ArcFace recognition performance on the IJB-C dataset.

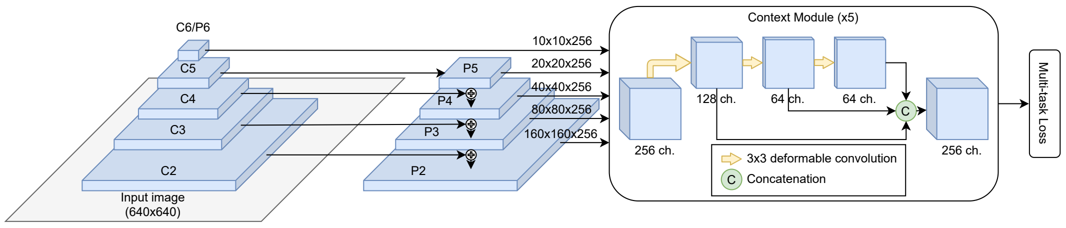


Figure 4.3: Architecture of the RetinaFace face detection network with ResNet backbone. Each block represents a layer or set of layers of the network. C2, C3, C4 and C5 represent the stages of ResNet. C6 is a new stage added on top of the backbone, output of a 3x3 convolutional layer with stride 2. P2, P3, P4, P5 and P6 represent the layers from the Feature Pyramid Network (FPN), similar to [31] but with deformable convolutions [302] instead of standard ones. Each of the 5 stages of the FPN is fed to a different Context Module. *ch.* stands for channels. Note that RetinaFace is fully-convolutional, so the image input size is not fixed: the feature map sizes shown in the figure are for a 640x640 input image. The network structure is analogous for the MobileNet backbone.

4.1.5 Shot boundary detection

Since an important component of the pipeline proposed in the Section 4.2 is a shot detector, a summary of the state of the art of shot detectors is presented here.

Shot detection (also called shot transition detection or shot boundary detection) is the process of automatically identifying shot transitions in videos. A shot is a series of interrelated consecutive frames that represent a continuous action in time and space [303]. Frames belonging to a single shot have been recorded contiguously by a single camera. Shot transitions are usually divided into *cut* and *gradual* transitions: cut transitions are instantaneous, while gradual ones span a number of transition frames that include visual content from both the old and new video shots. Detecting gradual transitions is usually harder and a lot of effort has been spent in the recent years to tackle the problem of correctly identifying long gradual transitions.

Early approaches. Shot detection has been studied long before the advent of deep learning. One of the first approaches [304] used color histogram differences between frames, along with two different thresholds in order to properly detect gradual transitions without increasing the number of false positives. Other classical approaches used Color Coherence Vectors [305], to exploit the color structures present in the frames, which are lost in the color histograms; SIFT descriptors combined with SVM for keypoint matching, with different strategies to detect different types of transitions [306]; SURF descriptors combined with entropy for cut transitions detection [307]; HSV color histograms and edge histograms with SVM [308]; a combination of HSV color histograms and HOG features [309]; HSV color histograms combined with SURF descriptors [310].

ILSD. One of the best-performing algorithms proposed before deep learning methods emerged is the ImageLab Shot Detector (ILSD), presented by Baraldi et al. [311]. It is based on the difference between features computed in two frames (denoted in the article as M_w^n, where n is the central frame of the window and $2w$ is the window size). The authors chose to use a linear combination of the sum of squared differences of pixels of the two frames and of the χ^2 distance of their color histograms. Both measures were normalized by the number of pixels in a frame. The two features have the property to be almost constant immediately before and after a transition, and to have a constant derivative during a linear transition. [311]. The differences between the features were computed between frames that were increasingly far apart, in order to help detect longer gradual transitions. In order to reduce the number of false positives, the maximum value of M_w^n within the candidate transition was compared to the lowest of the two M_w^n values that were computed $2w$ frames before and after the start and end of the transition respectively. This ensured that the difference scores during a transition reached a high enough peak with respect to the surrounding shots, reducing the number of false alarms. ILSD was evaluated on the RAI Dataset [311], reaching 0.84 F_1 score.

First use of a CNN for shot detection. One of the first algorithms to use a CNN to perform shot detection was the one proposed by Xu et al. [312]. A segment selection step based on pixel intensities was performed to select a set of candidate segments which might contain a transition. AlexNet CNN trained on ImageNet was then used extract features and similarity between frames was computed using the cosine distance among those features. Various thresholding strategies were then used to determine whether a frame o a sequence of frames was a cut transition or a gradual transition, respectively.

3D CNNs for shot detection. Gygli et al. [313] presented one of the first end-to-end CNNs for shot detection. The network used convolution through time to avoid processing each frame multiple times, and it was fully convolutional, thus able to take as input a variable number of frames. For these reasons it was able to reach 120 times the real time speed, while still being more accurate than classical algorithms on the RAI Dataset, with 0.88 F_1 score.

Hassanien et al. [314] proposed the use of a spatio-temporal CNN (DeepSBD) similar to the C3D network presented by Tran et al. [315]. Each video was split into overlapping segments, fed to the CNN; a SVM used the extracted features to predict the probability that each frame belonged to a cut transition, gradual transition or did not belong to a transition. A final post-processing steps using color histograms and Bhattacharyya distance was used to reduce the false positives. The authors merged a number of datasets in order to obtain a large-scale shot detection dataset appropriate for training the network; they also employed synthetic transitions. The algorithm reached 0.94 F_1 score on the RAI Dataset and scores comparable to the state-of-the-art algorithms on various other datasets that were part of the TRECVID challenges. The algorithm runs between 11 and 19 times faster than real-time, depending on the batch size.

DSM. Tang et al. [316] proposed the use of two separate detectors for cut and gradual transitions, in their Deep Structured Models pipeline (DSM). First, features were extracted from frames using SqueezeNet [141] trained on ImageNet. The cosine distance among the features computed with different window sizes was used to select frame candidates which could be part of transitions. Then cut transition detection was performed by using a ResNet-50 CNN which took as input 6 concatenated images. The left candidates were fed to an SSD-like CNN that included a C3D CNN, to extract the features from the frames, and two fully-connected "sub-networks" on top. These performed classification and regression of the so-called "default segments", which played a function similar to the anchors in the SSD object detector. The system was able to run at 700 frames per second, and obtained 0.94 F_1 score on the RAI dataset, state-of-the-art performance on TRECVID 07 with 0.841 F_1 score, and 0.848 and 0.870 F_1 score on cut and gradual transitions respectively on their newly proposed large-scale ClipShots dataset. The introduction of ClipShots was necessary to perform training on a larger, more varied and more challenging set of videos, taken from YouTube and Weibo. The dataset was manually annotated for cut and gradual transitions.

TransNet. Souček et al. [317] proposed a different CNN to quickly and reliably segment videos, called TransNet. The network takes a sequence of 100 resized video frames as input, applies a series of 3D dilated convolutions to it and produces a list of 100 probability scores, each representing the probability that the corresponding input frame belongs to a transition. The network uses multiple dilated convolution blocks [318] run in parallel for each layer of the network, in order to have a larger receptive field with less trainable weights. Since only the 50 central outputs of the network have a large enough receptive field, the videos are split into overlapping 100-frame windows, so that predictions for each video frame are all taken from the 50 central outputs. The network structure is shown in Figure 4.4. The network was trained on videos from the TRECVID IACC.3 dataset [319] and obtained an F_1 score of 0.94 on the RAI Dataset.

TransNetV2. TransNetV2 [320], by the same authors, is an improvement over the original TransNet with a series of additions and architectural changes. First, residual connections [1] and batch normalization were added. 3D convolutions were factorized into 2D ones in order to reduce the number of trainable parameters and to separate the spatial dimension from the temporal one. The authors also added a frame similarity layer and a RGB histogram similarity layer, producing similarity vectors that are fused with the other CNN features before the final dense layers. Finally, the network uses two classification heads, one trained to predict only the central frame of a transition as positive, while the other is trained to predict every transition frame as positive; the latter is used to help with the network training, but is not used by the authors at evaluation time. The network structure is shown in Figure 4.5. TransNetV2 was trained on a mix of real shot transitions from the ClipShots dataset [316] and synthetic transitions obtained from both IACC.3 and ClipShots. The network outperformed other shot detectors on ClipShots and BBC [321] datasets, while having competitive performance (0.94 F_1 score) on the RAI Dataset. The authors also showed that mixing real and synthetic transitions helped improve the model accuracy.

4.2 The proposed pipeline

The pipeline includes two distinct processes: the creation of a face feature database and the execution of queries on that database. They are summarized in Figures 4.6 and 4.7, respectively.

When a new video is added to the database, the following steps are always

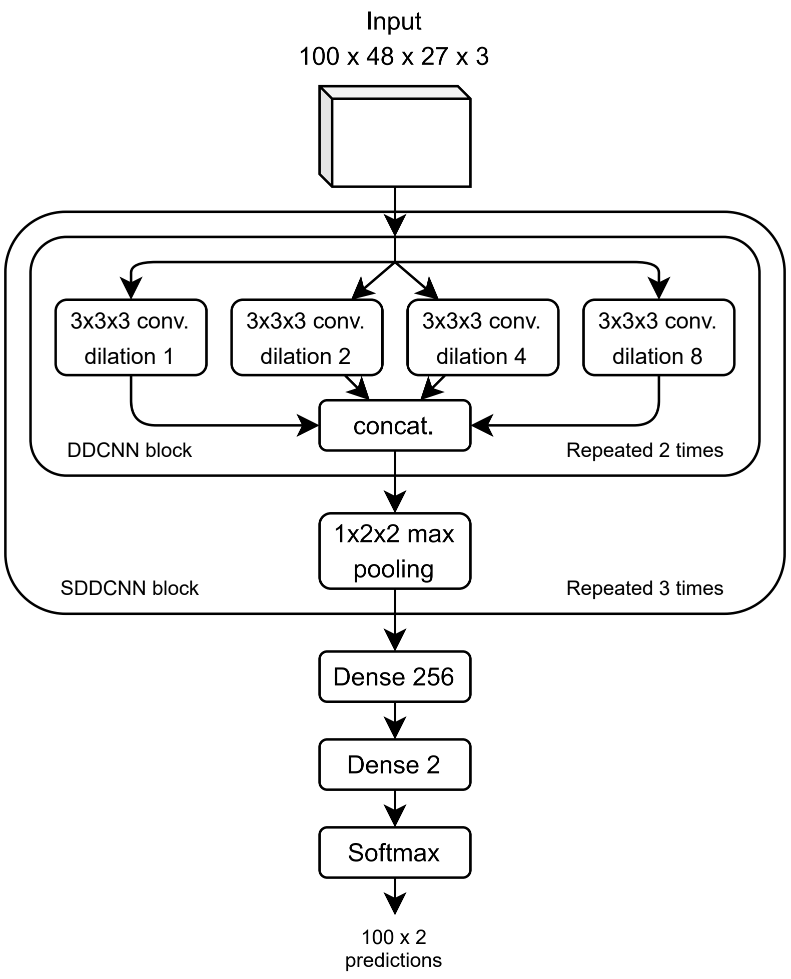


Figure 4.4: Architecture of the TransNet shot detection CNN. The network takes a set of 100 frames of size 48x27 as input. The backbone of the network is composed of 3 Stacked Dilated Deep CNN (SDDCNN) blocks, each containing a series of 2 Dilated Deep CNN (DDCNN) blocks. Each DDCNN performs 4 parallel dilated 3D convolutions with different dilation factors. The outputs of the 4 parallel convolutions are concatenated before the next DDCNN block. After the DDCNN blocks, each SDDCNN blocks performs max pooling before the next SDDCNN blocks. Each dilated convolution has 2^{i+3} filters, where i is the index of the current SDDCNN block, starting from 1. The final prediction is done using two fully-connected (Dense) layers, the first with 256 neurons per frame and the last with 2 neurons per frame, representing the probability that the frame belongs or not to a transition.

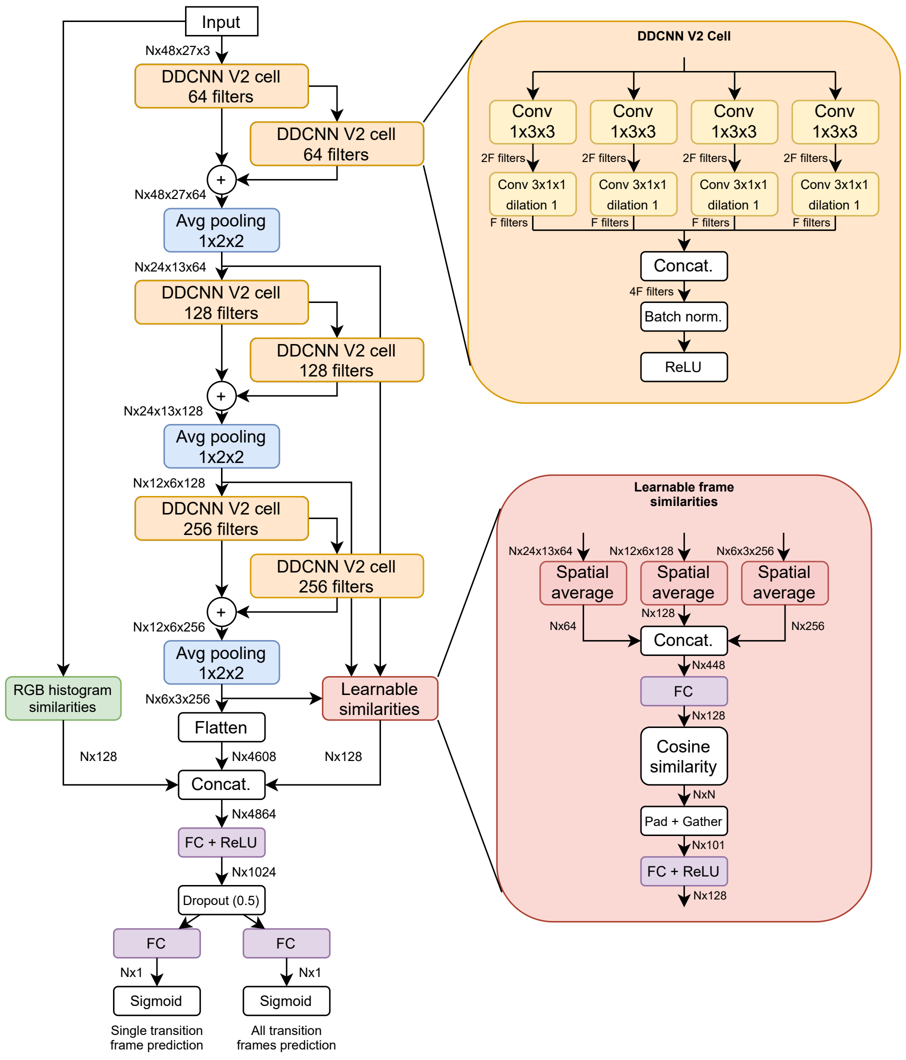


Figure 4.5: Architecture of the TransNetV2 shot detection CNN. The network takes a set of N frames of size 48x27 as input (N is set to 100 in the paper, but can be changed). TransNetV2 uses residual connections between the DDCNN blocks. The DDCNN V2 blocks factorize the 3x3x3 convolutions of the original TransNet into two consecutive 2D convolutions, reducing model and computational complexity. RGB histogram similarities between frames are added, as well as a learnable module which computes frame-to-frame similarities. *FC* stands for fully-connected. The *Pad + Gather* operation transforms the cosine similarity matrix so that each row includes the similarities between the corresponding frame and the 50 previous and subsequent frames (50+1+50 = 101, with the additional 1 being the similarity between the frame and itself).

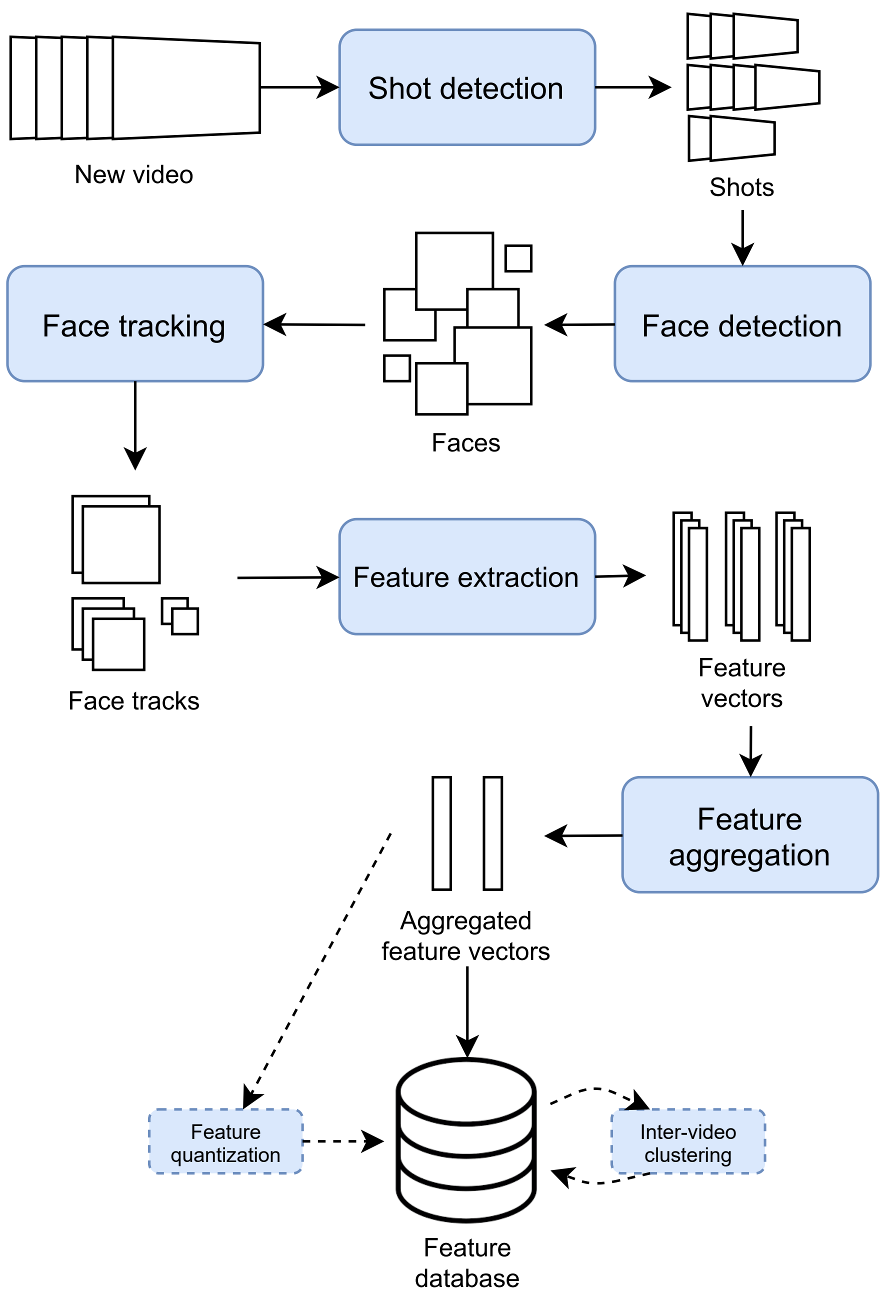


Figure 4.6: Creation of the face feature database. When a new video is added to the database, shot detection is first performed. The face detector then extracts faces from a subset of the frames not belonging to shot transitions. Tracking is performed for the detected faces, within each shot. For each track, features are extracted for a subset of its faces, and are then aggregated before being saved in the feature database. Optionally, features can be quantized before saving them in the database and/or inter-video clustering can be performed on all the features in the database.

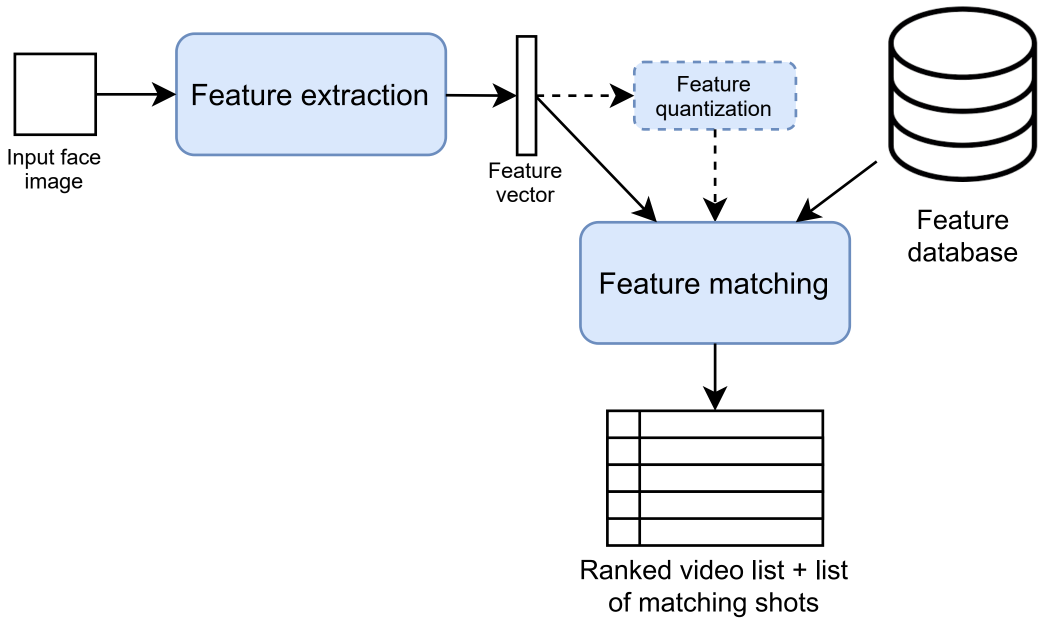


Figure 4.7: Querying videos in the face feature database. The input face image is fed to the face feature extraction network to obtain a feature vector. This is compared to the other feature vectors in the database, and the videos are ranked based on the matching score. For each returned video, the matching shots are also returned. If feature quantization is enabled, the feature vector is quantized before comparing it to the database features.

performed:

1. a shot detector is used to split the video into coherent non-overlapping shots. Frames belonging to transitions are ignored in the rest of the pipeline;
2. for each shot, at regular intervals, a face detector is used to extract all the image patches containing faces in each of the selected frames;
3. a fast tracking algorithm is employed to group faces belonging to the same person throughout each shot;
4. for each obtained tracklet, face features are extracted for a small subset of the faces that are part of the tracklet;
5. a feature aggregation step is finally performed: the feature vectors are

clustered across the different shots and fused in order to obtain a single feature vector for each distinct face/person in the video. Information about the video shots in which each person appears is retained.

Besides the aforementioned steps, I also investigated two additional optional steps (shown as dashed boxes in Figures 4.6 and 4.7). First, the possible use of a quantization algorithm in order to reduce feature size, both on disk and in memory; second, performing a global clustering step in order to aggregate feature vectors across all the videos in the database, with the aim of reducing the number of feature vectors and increasing accuracy by "stabilizing" the features for each person across different visual contexts.

Most of the described steps are performed using a deep neural network. Each step is described in a greater detail in Section 4.2.1.

The query process is relatively simple and composed of two main steps:

1. a feature vector is extracted from the query image using the same feature extraction algorithm employed in the database creation step;
2. the feature vector is compared to the feature vectors in the database and a ranked list of matching videos (with the corresponding matching shots) is returned as a result.

If feature quantization has been used in the feature extraction step, then it must be also performed at query time in order to be able to compare the feature vectors appropriately. A more in-depth description of the query algorithm is presented in Section 4.2.2.

4.2.1 Creation of the face feature database

4.2.1.1 Shot detection

The use of a shot detector in the proposed pipeline is justified by three main reasons:

- the use of a fast and efficient shot detector can drastically reduce the complexity needed for the face tracker. Since MOT algorithms cannot deal effectively with shot/scene changes without resorting to the extraction of computationally expensive appearance features, the use of a shot detector allows to track faces separately for each camera shot, removing the need for expensive algorithms;

- detecting and discarding transition frames can avoid extracting faces and face features in the middle of a transition, thus avoiding the issue of including faces blended with other faces or with the background, which would lead to the extraction of unreliable features;
- from a usability perspective, providing the user of a FBVR system with a list of shots in which a person appears can help in two different ways: first, he can easily see if each returned video is of interest (especially for long videos, where it can be hard to quickly find the segment in which the person of interest appears); second, the user can easily discard a video if the algorithm has made a mistake by matching the wrong person, by quickly jumping to the various shots in the video and checking the video content.

I tested three possible shot detection algorithms for this stage of the pipeline: the ImageLab Shot Detector (ILSD) [311], TransNet [317] and TransNetV2 [320].

I chose ILSD in order to compare the performance, both in speed and accuracy, between a "classical" non-deep shot detector and a deep one, since the use of CNNs is still quite recent in the field of shot detection. My choice for the deep shot detectors was instead motivated by their good accuracy on the RAI Dataset, as compared to previous networks, combined with their execution speed, which is important since shot detection is one of the three main time-consuming steps in the feature extraction pipeline (together with face detection and feature extraction).

4.2.1.2 Face detection

The next step in the proposed pipeline is the face detection step. Consecutive frames belonging to the same shot are likely to contain very similar image content, including face location and poses; for this reason, extracting faces from every frame would lead to high computational times without any retrieval accuracy benefit, since no significant new information would be acquired about the faces. Moreover, the benefit of using each frame would also be minimal for the tracking algorithm, since in most situations (except when fast camera/person movement is present) face location does not change significantly when skipping a few frames, and the pipeline is robust to a small number of tracking failures anyway, thanks to the clustering algorithm employed in the feature aggregation step. For this reason, faces are only extracted every k seconds, with k chosen experimentally as explained in Section 4.3. At least one frame is analyzed for each shot.

I decided to test three possible face detectors: MTCNN [279], SSH [292] and RetinaFace [300]. MTCNN is still widely-used today despite being one of the oldest face detectors using CNNs. It reached good accuracy, while being able to run in real-time. SSH uses a single deeper CNN as a backbone (VGG-16), thus it is more accurate, but slower. RetinaFace is one of the most recent CNNs for face detection, it reaches state-of-the-art performance with the ResNet-152 backbone and it can run in real-time while still performing reasonably well with the MobileNet-0.25 backbone.

4.2.1.3 Face tracking

At this stage of the pipeline we have a list of shots, and for each of them we have a number of faces extracted from a subset of its frames. We need to group together the faces belonging to the same identity for two main reasons:

- grouping faces belonging to the same person allows to extract less features for each person, since we expect that faces belonging to the same person will have similar features, thus reducing computational time. It also reduces disk usage, because it allows to merge features extracted from multiple faces of the same person;

- performing intra-shot face association can reduce inter-shot association mistakes in the feature aggregation stage: averaging the features of faces that we know belong to the same person can produce more robust features for the inter-shot aggregation step, by reducing the variability due to random noise in the data (see Section 4.2.1.5).

With this in mind, it is important to note that a state-of-the-art tracker is not needed if it is also going to substantially increase the computational time. The pipeline is robust to tracking mistakes, since the feature aggregation stage will compensate for tracks that were mistakenly split. The inter-shot aggregation stage also removes the need for an expensive re-identification CNN, since the tracking algorithm now only has to deal with intra-shot frames, without sudden changes in face position and size due to shot transitions.

I thus chose to use the fast IOU tracker [322], which can perform tracking on thousands of frames per second. The IOU tracker simply considers the Intersection Over Union (IOU) score between two bounding boxes in consecutive frames in order to decide whether to assign or not the same ID to the two boxes. Each box is assigned the same ID as a box of the previous frame if it is the box in the current frame that overlaps the most with the previous one and if the IOU is above a certain threshold. The algorithm does not use any visual information, and it is thus computationally efficient. Optionally, the input faces can be discarded based on the detection confidence score and tracklets can be filtered out based on their length and on the highest detection score of the faces belonging to each tracklet. Despite its simplicity, this algorithm reached good performance on the UA-DETRAC [60], MOT16 and MOT17 [42] challenge datasets.

The output of this stage consists of a list of face tracks that do not cross shot boundaries.

4.2.1.4 Face feature extraction

In order to perform face queries, we need to perform face recognition, i.e. we need a way to compare two faces and determine if they belong to the same person (face verification). One possibility is employing a deep neural network to extract features for each of the two faces, and then use a distance measure to compute the semantic difference between the two faces: faces with a high distance likely belong to a different person, while faces with a low distance are likely to be of the same person.

For this purpose I chose to use the networks presented in the VGGFace [237] and VGGFace2 datasets papers [238], together with the more recent ArcFace [251]. The VGGFace networks[7] are a set of VGG-like [21] CNNs trained with a triplet loss in order to force a higher distance between faces of different persons, and a lower distance between faces of the same person. The authors used the Euclidean distance in order to determine if two faces belong to the same person. The networks presented in the VGGFace2 paper are a ResNet-50 [1] and a SENet-50 [323], both trained on their newly proposed dataset. These networks too can be used to extract feature vectors, which can be compared using the cosine distance. Moreover, the authors aggregated feature vectors by computing their average. This will prove useful in the feature aggregation stage (see Section 4.2.1.5). ArcFace uses one of various ResNet backbones and, as we have seen in Section 4.1, a special Additive Angular Margin Loss to enforce a low/high angular margin between feature vectors belonging to the same or different persons, respectively. It is the most recent of the three networks, and it is the one that has obtained the best performance on a number of face recognition datasets.

Since there can be very long face tracks (e.g. in a long TV interview without camera changes), repeatedly extracting features from the same person

[7]Note that while VGGFace and VGGFace2 are the names of the datasets on which the networks were trained, the networks themselves are often referred to with the same names in the literature. For the sake of brevity, I will also refer to the networks as VGGFace and VGGFace2, and I will note otherwise if it is not clear from the context.

can lead to unnecessary computation, since most of the information will be redundant. For this reason, I chose to extract a fixed number of faces (see Section 4.3) within each face track, avoiding the initial and final parts of each track, which can be noisier[8]. With this optimization, the face feature extraction stage is substantially faster than the shot detection or face detection stages, since the network only needs to examine a limited number of faces for each video.

The output of this stage is then a set of feature vectors for each detected intra-shot face track.

4.2.1.5 Feature aggregation

While in theory it would be possible to just save all the feature vectors we have got from the previous stage and perform queries using them all, this would lead to substantial disk usage, since we would be saving a lot of redundant information, but also to a longer computational time at query phase, since the query feature vector would have to be compared with a higher number of stored vectors. Moreover, some challenging face poses can produce noisy feature vectors that could lead to errors at query time. A strategy to aggregate and stabilize these feature vectors is thus necessary.

For this reason, I decided to implement the following steps:

- first, averaging the feature vectors for the faces belonging to each track, since we know they likely belong to the same person. Feature vector averaging has already been used in the literature, for example with the VGGFace2 features [238]. Feature vectors are normalized to have unitary norm both before and after the averaging process;
- after that, we can merge feature vectors from tracks that contain the

[8]For example, if the track starts at the start of a shot and the shot detector has not perfectly identified all the frames belonging to the transition, some faces might be blended with the background or other faces. Another example is the case in which the face track is “new” after a tracking interruption caused by fast motion of the camera or of the subject: in this case we might want to wait a few frames for the face to stabilize.

same person that are currently split from each other because they are either part of different shots, or because of a tracking error. So, feature vectors are clustered both within and across shots;

- the new grouped feature vectors are then again merged by averaging, followed by normalization.

For the track clustering I chose to use a hierarchical agglomerative clustering (HAC) algorithm. Briefly, the algorithm starts with a number of clusters equal to the number of input objects (that in our case are the face tracks, represented each by a feature vector), with each starting cluster only containing a single object; it then computes a distance measure between each of the clusters and then merges the two clusters with the lowest distance. After that, it recomputes the distances between the new and the old clusters and repeats the previous steps until a certain distance threshold is reached. In this way, a clustering tree is built, which is then cut to produce the final clusters.

Various strategies for the choice of the point at which to cut the tree exist. I decided to use a distance threshold since both the cosine and euclidean distances are bounded for normalized unit vectors[9], and so various thresholds can be easily tested in order to choose the best performing one.

The choice of using a simple unsupervised algorithm, without forcing any temporal constraint on what tracks can be merged together, has the aim to make the algorithm robust to tracking and detection errors. Duplicate detections might exist and some tracks could have been mistakenly interrupted because of a detection failure or of object motion/occlusion. For this reason, the algorithm is allowed to group together tracks that belong to the same shot and even tracks that overlap in some frames, since we are not sure that the tracks actually represent different persons, and it is unlikely that two faces, of two different identities, appearing in the same shot or frame also have features

[9]The cosine distance is defined as one minus the cosine similarity (which has values between -1 and 1), thus it always take values in the interval $[0, 2]$. The euclidean distance between unitary vectors also takes values in the interval $[0, 2]$.

so similar that they result in a merge mistake[10].

In addition to the clustering threshold, I also tested different linkage methods. The linkage method defines how we compute the distance between each pair of clusters during the execution of the HAC algorithm. I tested simple linkage, which uses the smallest distance between any pair of feature vectors belonging to the two clusters we are comparing, complete linkage, which uses instead the highest distance between all the possible pairs of feature vectors, and average linkage, which computes the average of the distances of all the pair of feature vectors in the two clusters.

The distance measure employed (cosine or euclidean) was chosen according to the specific feature extraction network employed. VGGFace2 was trained to use cosine distance between vectors, while VGGFace1 and ArcFace used euclidean distances in the original papers.

At the end of this stage we finally have a list of feature vectors, each ideally representing a different person in our video, and each with an associated list of shots in which the person appears. This data is saved to the disk, where it constitutes the feature database that is used to perform queries.

4.2.1.6 Feature quantization

In addition to the previous steps, I also briefly investigated the possibility to employ a quantization algorithm in order to reduce the size of the feature vectors. Depending on the network used for feature extraction, each vector can contain between 512 and 2048 floating point values. On large scale databases, with thousands of videos, the pipeline can extract 300-400 MB of features, if not more. This can easily grow and become a problem both for storage space and for memory consumption, especially when more processes are run in parallel to execute feature extraction and queries.

[10]One obvious caveat would be for identical twins appearing in the same frame. But first, the two people would be hardly distinguishable anyway even without merging their features, and second, it is a relatively rare occurrence that we can ignore without any significant accuracy penalty.

It is thus possible to employ a quantization strategy in order to compress the features without excessively reducing the recognition accuracy. While this is not the main focus of my thesis, I studied the feasibility of using a quantization algorithm before saving the features on disk. I chose to use the Angular Quantization-based Binary Coding (AQBC) [324]. The reason is that this algorithm does not require any training step, so it is agnostic to the choice of a feature extraction network, and it takes into account the angular distance between feature vectors, that is suitable to the features I am using for face recognition. The only major restriction is that the feature vectors must only have positive values (which is not always true depending on the network used) and must be distributed on a hypersphere. Briefly, the quantization algorithm works by projecting the 2^d vertices of the binary hypercube $\{0, 1\}^d$ onto the unit hypersphere, where d is the dimensionality of the original feature vectors. The feature vectors are then quantized to a $d-$bit vector, by assigning the closest projected vertex to each original vector.

In principle, this allows to reduce the size of each value in the feature vector from 32 or 64 bits to a single bit, thus reducing the disk space (ignoring compression) by 32 or 64 times. In practice, some efficiency limitations arise using NumPy and Python, both in storing bit vectors and in performing an efficient cosine distance computation over bit vectors, since some functions are not natively implemented in the language. A more detailed discussion of the implementation issues can be found in Section 4.3.

4.2.1.7 Inter-video clustering

A second optional step I investigated is the use of inter-video clustering, i.e. applying a global clustering algorithm over the features extracted from all the videos, in order to further reduce the number of feature vectors in the database and to reduce feature variability for people that appear in more than one video, whose features might then be affected by illumination conditions or poses that are specific for each video he appears in.

For simplicity and consistency, I used the same HAC algorithm used in the intra-video feature aggregation stage. Despite inter-video clustering was in fact able to increase the retrieval performance for some pipeline configurations by as much as 1.5% AP, I consider this step optional for a number of reasons: first, it can be extremely memory consuming, at least using the standard implementation in the main Python machine learning libraries such as Scikit-learn — in some cases, where more than 100,000 feature vectors had to be clustered, I was not able to perform clustering in a single step using all the vectors, since the distance matrix would require more than the 64 GB of RAM available on the machine; the second reason is that since HAC is an offline algorithm, every time a new feature vector is added to the database, clustering has to be performed again from scratch, and can require significant computational time, especially if not carefully optimized. Moreover, for some feature extraction networks, the additional inter-video clustering step did not have any additional benefit: since this step adds computational time and does not reduce the feature size on disk, it is better to avoid it in these cases. The reason why the additional clustering does not reduce disk size is that in order to re-execute the clustering after new feature vectors are added, the original feature vectors are needed to run the HAC from scratch, so it is not possible to just discard them. Some online approximations of the HAC algorithm exist [325, 326], but none seems to solve efficiently all of the problems described above, for a possibly even minor impact on retrieval accuracy.

4.2.2 Querying the database

In order to query videos in the database it is sufficient to compute a feature vector representing a single face[11] and then compare it using cosine/euclidean similarity with all the person feature vectors stored in the feature database. If feature quantization was enabled in the extraction phase

[11]The algorithm is easily applicable to a sequence or set of query faces too (for example in order to use video queries). It is in fact sufficient to average the feature vectors for all (or part of) the faces in the frame sequence, and then compare the result with the database features.

(see Section 4.2.1.6), then the feature vector extracted here must also be quantized in order to be comparable with the database feature vectors.

For each video, the person that matches with the query image with the highest similarity score is considered the matching identity for that video, and that score becomes the video's matching score. A ranked list of videos is then returned, sorted by matching score. A threshold can be used to eliminate poorly matching videos and only present good matching ones to the final user. For each returned video, the information about the shots in which the matching person appears is also provided.

4.3 Experiments and results

4.3.1 Dataset

In order to properly evaluate the performance of the pipeline and to ensure it will work in real-world systems in the TV context, like TVBridge, an appropriate dataset is needed, ideally with the following properties:

- a large number of videos recorded in the TV setting or similar, and with a sufficient number of identities. The characteristics of television shows is different from, for example, surveillance videos, and we need a dataset as close as possible to the use case of the system for a proper evaluation of the performance. Having a large dataset ensures that the pipeline is robust enough to both the presence of a high number of identities, and to a higher variability in video content;

- videos should have varying resolutions. This is important in order to evaluate the robustness to low-quality videos, but also to check that the analysis of HD videos is feasible in terms of time and memory constraints;

- most videos should contain multiple identities. Some datasets in the literature only provide cropped clips containing a single face. This is not representative of real-world conditions, and the presence of multiple people helps evaluate various parts of the pipeline, including face detectors, face trackers and feature extraction networks for face recognition;

- since the vast majority of TV-like videos contains multiple shots and have variable length, the dataset should include multi-shot videos in order to match real-world conditions and to evaluate the impact of the different shot detectors;
- faces in the videos must be unconstrained, i.e. they should appear in a wide variety of poses and orientations, to simulate real-world conditions;
- optionally, the dataset should provide a video retrieval evaluation protocol with the related ground truth.

However, to the best of my knowledge, no existing dataset matches all of the described criteria[12]:

- **YTF**: while it has the advantage of containing many identities and including complete face track annotations, it has two significant drawbacks which make it unsuitable to test the proposed pipeline effectively: it does not include multi-shot videos, since the clips only contain the consecutive frames in which the subject of interest appears, in many cases without other people in the background; moreover, since it was collected in 2011, it only contains low-resolution videos. In addition to that, the dataset does not contain still images to use as queries, posing the problem of having to create the probe set either via web searches (which can introduce significant noise without time-consuming manual inspection for the 1,595 subjects) or extracting the faces from the provided videos, but with the risk of introducing reliability issues at evaluation phase for subjects with only a few videos in the dataset, given the low average number of clips per subject;

[12]For more information about the listed datasets, see Section 4.1.3.

- **IJB-C**: while the dataset provides a great variety of unconstrained videos (although mainly restricted to interviews), like YTF this dataset is also not built for video retrieval using face images, but was instead conceived for face identification and verification, since the galleries only contain images. However, a video retrieval protocol might be built by “inverting” the role of the gallery still images and of the video probes, and by using an appropriate accuracy measure, since we are not matching the images to a single video, but we want to retrieve all the videos containing that person;

- **iQIYI-VID-2019**: while this dataset was conceived for video retrieval, it still presents some issues: the clips are of short duration (4 seconds on average), do not contain shot transitions, and mostly only show a single person each. Since there is no way to obtain the original videos from which the clips were extracted, this makes the dataset unsuitable for the end-to-end evaluation of the proposed pipeline, since it does not allow to appropriately test the performance of components like the shot detector or the tracking algorithm;

- **VoxCeleb2**: while the dataset was not originally meant for video retrieval, it can be used as a starting point to build an appropriate FBVR dataset. Since the URLs for the whole videos are included, and these videos include shot transitions, multiple persons per frame and unconstrained faces (with varying poses and occlusions). Moreover, videos vary in resolution and quality (from old low-resolution videos to newer HD ones) and the dataset provides the list of videos in which each identity appears. Another advantage of VoxCeleb2 is that face tracks are included in the dataset, so it is possible to easily extract and crop faces from video frames to use as query inputs, resembling more closely the use case of TVBridge (remember that users will select faces from a video frame to perform the video search);

- **TV series and movie datasets**: while videos from TV series and movies would be useful to integrate interviews and YouTube videos, which have different characteristics, the *Buffy the Vampire Slayer* [273], *The Big Bang Theory* [274] and *Sherlock* [275] datasets are each limited to a single TV show with a very low number of identities. More importantly, all the videos from these datasets are not publicly available because of issues related to the use of copyrighted material. The Cast Search in Movies (CSM) dataset [276] only contains cropped tracklets for each person, posing the same problems described for some of the other considered datasets.

Since there was no suitable FBVR dataset in the literature that I could use to fairly evaluate the proposed pipeline, I chose to build a new dataset starting from VoxCeleb2. I chose VoxCeleb2 because it was the easiest dataset to adapt for my use case, for the reason discussed previously. IJB-C was also a good candidate, however, the face tracks provided with VoxCeleb2 (and not included in IJB-C) opened the possibility to extract face images directly from the provided videos for use in the query evaluation process, as we will see. This removed the need to manually extract images from other online sources, and matched more closely the pipeline use case in TVBridge, where queries are performed using face images extracted from videos.

VoxCeleb2, however, still has two main limitations. First, the videos mainly represent interviews, so while the dataset might be a good fit to evaluate the face recognition pipeline in talk shows and similar TV programs, it might not adequately represent other common TV programs, such as movies or TV shows, in which the subjects might have a larger motion and the lighting and environmental conditions are more varied. Nonetheless, some of the interviews were recorded in outside environments and contain a high number of people in the background, which can help evaluate the robustness of the pipeline in the presence of a high number of distractors. The second limitation is that since the dataset only provides cropped clips instead of complete videos, the whole videos must be downloaded from YouTube. Not every video was still available on YouTube when I built the dataset, slightly reducing the dataset size. Moreover, some videos were edited or re-processed by YouTube after the original dataset had been built, and this introduced some noise and inaccuracies in the face tracks provided by the dataset, as we will see.

4.3.1.1 The proposed dataset

The proposed dataset uses annotations and video URLs provided with the VoxCeleb2 dataset [272]. Since the test set contains 118 identities, many of which overlap with training identities in the VGGFace/VGGFace2 face extraction networks, I used the VoxCeleb2 identities from the test set as a validation set for the FBVR pipeline, in order to choose the best hyperparameters. Please note that while some identities in the validation set were also used to train some of the feature extraction networks, the images in VoxCeleb2 are extracted from video frames, while the networks were trained on still images; images extracted from videos have different characteristics from still pictures, since the first are subject to motion blur and other video artifacts.

However, in order to ensure that the hyperparameter choice procedure did not lead to overfitting, I also selected a specific subset of subjects from the VoxCeleb2 development dataset, which does not overlap with the subjects used to train either the VGGFace2 CNNs or ArcFace. The videos of these subjects formed an independent test set. I did not consider the subjects used to train the third model, the original VGGFace, since it obtained relatively poor results on the validation set, and thus I decided not to evaluate the pipeline with this model on the test set.

Test set identity selection. In order to determine the set of unseen subjects to be included in the test set, I proceeded as follows. From the set of identities in the VoxCeleb2 development set, I had to remove every identity that was also included in the VGGFace2 and MS-Celeb-1M [239] training datasets, since I used VGGFace2 weights that were pre-trained on MS-Celeb-1M dataset and fine-tuned on VGGFace2, and ArcFace weights that were learned on MS-Celeb-1M. VoxCeleb2 provides a CSV file that allows to match its identities to the VGGFace2 ones. For MS-Celeb-1M I was not able to find the list of names of the identities, but only their IDs as provided by the Google Knowledge Graph API. For this reason, I first removed the VGGFace2 training identities from the 5994 VoxCeleb development set identities; I then used the Google Knowledge Graph API to retrieve the IDs of the remaining identities by querying them by name; finally, I was able to remove the identities that appeared in MS-Celeb-1M. 78 identities were left.

Dataset construction process. The dataset construction, for both the validation and test sets, was done as follows. First, the YouTube videos were downloaded using the youtube-dl Python library. For each video, the highest resolution version not superior to 720p was downloaded. I excluded video streams encoded with the recent AV1 codec, since the versions of OpenCV and ffmpeg that I had available were not able to decode it.

Since some videos of the VoxCeleb2 dataset were not available anymore on YouTube, I was able to download 4260 of the 4910 videos for the validation set[13], and 1375 of the 1536 videos portraying the 78 non-overlapping identities of the test set.

On average, there are about 36 videos for each identity in the validation set and about 18 in the test set. The validation videos contain a total of 40,153,695 frames, with 396.4 hours of footage and an average of 28.1 FPS. The test videos contain a total of 13,862,357 frames, with 146 hours of footage and an average of 26.4 FPS. Videos are also quite varied in resolution, ranging from 160x120 pixels up to 1280x720, with an average of 1007x585 pixels for the validation videos and 956x563 for the test videos.

Figure 4.8 shows example frames from videos in the test set.

Query generation. After downloading the videos, a set of queries must be created in order to evaluate the pipeline. For this reason, I extracted faces

[13]The website says VoxCeleb2 test set consists of 4911 videos, but actually one of the videos appears twice for two different identities and it is counted twice.

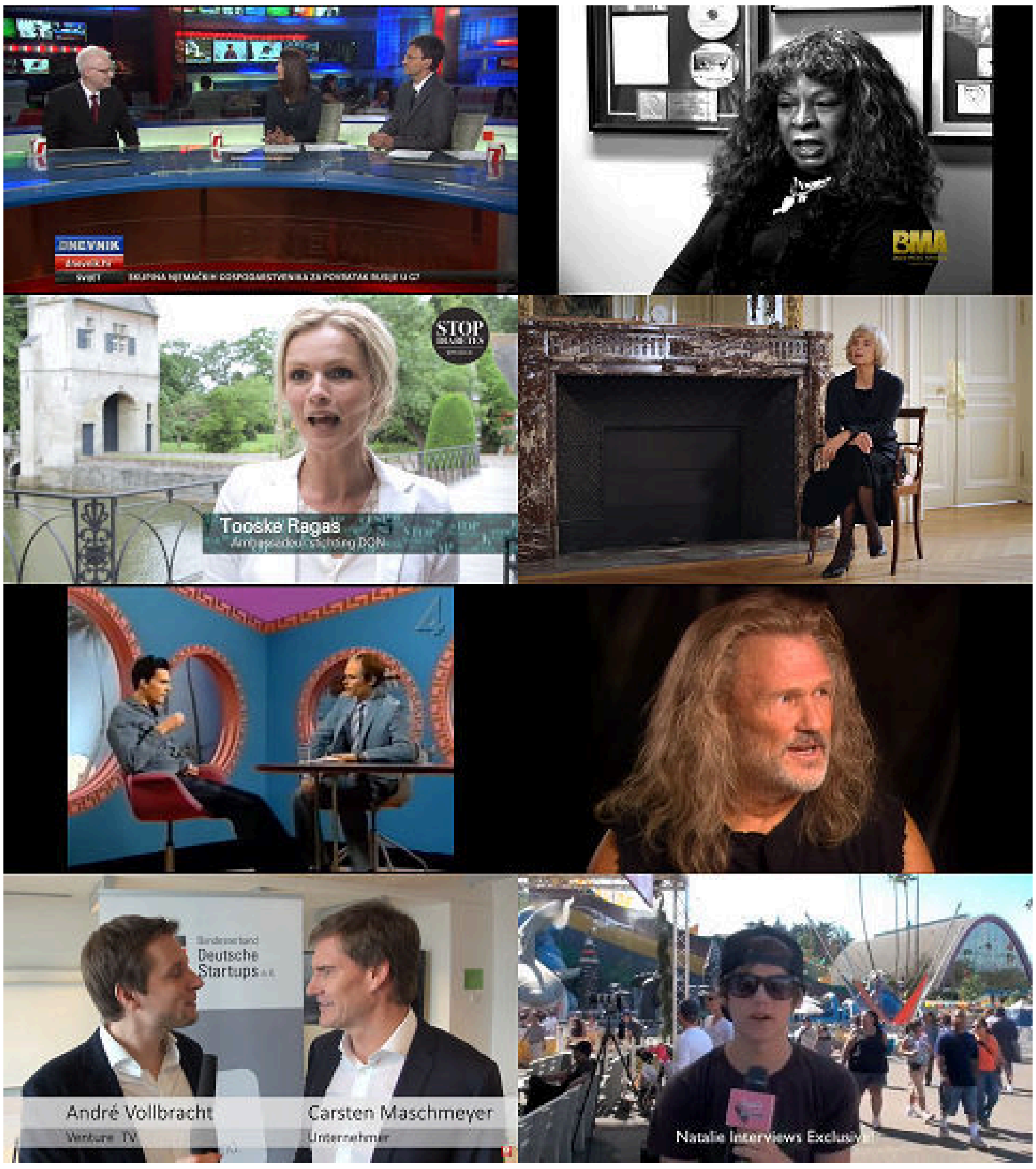


Figure 4.8: Example frames extracted from videos in the test dataset. Note the variety of locations and image quality.

from the videos, used as query inputs. VoxCeleb2 annotations include face tracks for each time a person of interest speaks in a video. For each of those tracks I extracted 3 face images, one from the frame at 25% of the track length, one from the middle frame of the track, and one from the frame at 75% of the track length. I avoided starting and ending frames in order to reduce possible noise from initial and final frames in the tracks, similar to the strategy described in Section 4.2.1.4. This has resulted in 94,233 query faces for the validation set, and 29,652 query faces for the test set. The high number of faces per identity allows to evaluate the robustness of the retrieval performance to pose/illumination changes, that can be present even within a single video.

While I have run all the queries on both the validation and test sets on the faces extracted from the downloaded videos, I have also tested queries on the validation set using faces extracted from the clips provided with the original dataset. As we will see in Section 4.3.4, the scores obtained with those were lower, since the clip quality was worse than the YouTube version of the videos: in VoxCeleb2 the faces were resized to 224x224 for the entire track for all tracks, and were recompiled as a video, thus adding another compression step on the top; in contrast, the average size of the faces extracted directly from the original YouTube videos is about 286x284, and they were subject to one less compression step. For this reason I decided to only use the YouTube-extracted faces for the rest of the tests.

For each face image used as query, a list of ground truth matching videos is also saved. For each face, the matching videos are all the videos that in VoxCeleb2 are annotated as containing the same person as the query image, excluding the video from which the face was extracted. This is because retrieving the same video that was used to extract the query face is easy, since the face appears in it with the same lighting conditions and pose. For this reason, every evaluation has been performed by ignoring the video containing the query image.

Figure 4.9 shows examples of query faces from the test set.

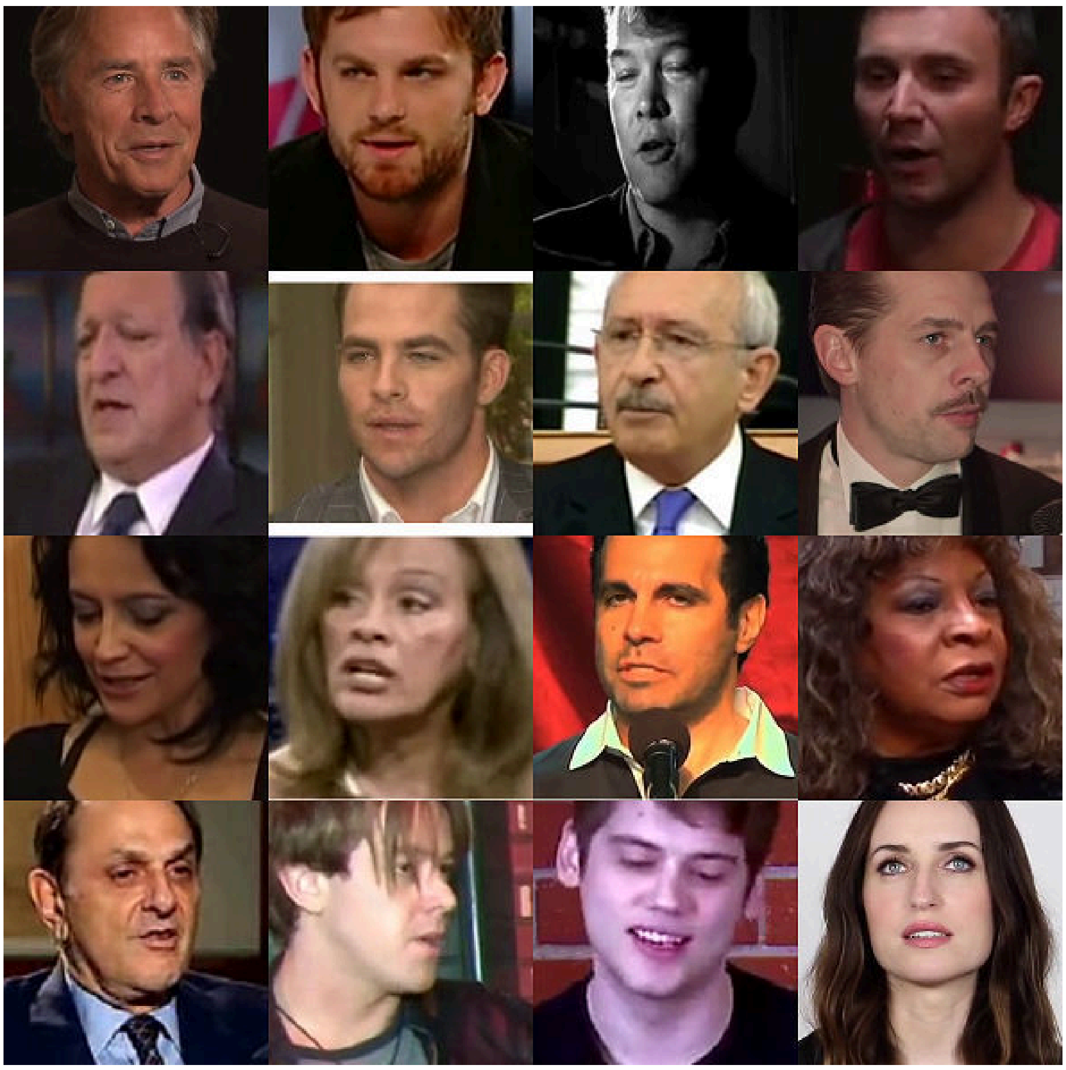


Figure 4.9: Examples of query face images from the test dataset. Images were resized to fit the grid in this figure, but originally ranged from 108x108 pixels to 394x394 pixels.

Dataset cleaning. The annotations of the constructed dataset can be subject to noise and mistakes.

First, there is no guarantee that a person only appears into the set of videos indicated in the VoxCeleb2 annotations, since the VoxCeleb2 dataset (like most large-scale video datasets) was built using automated tools. While this error might slightly lower the evaluation score, since the pipeline might correctly retrieve a video that is mistakenly marked as wrong in the ground truth, this situation is unlikely and in any case it would provide us with a conservative value of the Average Precision (AP).

A possible second issue, also related to the partially automated construction process of VoxCeleb2, is that some face tracks might not be precise or may contain non-face objects.

In fact, early tests on the test set showed an anomalous number of queries with very low scores compared to the validation set (see Figure 4.10). Performing manual inspection of the dataset I found that various videos in the test set did not actually portray the identity they were supposed to contain, while for many other videos the box annotations provided with VoxCeleb2 were either very inaccurate or surrounded the wrong person in the video. Since the test set was relatively smaller than the validation set, and since having a correct performance measure on the test set is arguably more important (the validation set is used to choose hyperparameters, so relative changes in the mean AP are more important than absolute scores), I performed a manual cleaning step on the test set only.

For each video in the test set I computed the mean AP for the queries performed using images from that video. Then I checked which videos obtained non-perfect scores ($< 95\%$ mean AP), and I manually checked all the videos from that identity. I did not examine videos for which all the queries obtained near-100% AP, because it is very unlikely for the pipeline to obtain an almost perfect score if those videos contain significant box annotation mistakes. This helped speeding up the cleaning process by reducing the number of images to check.

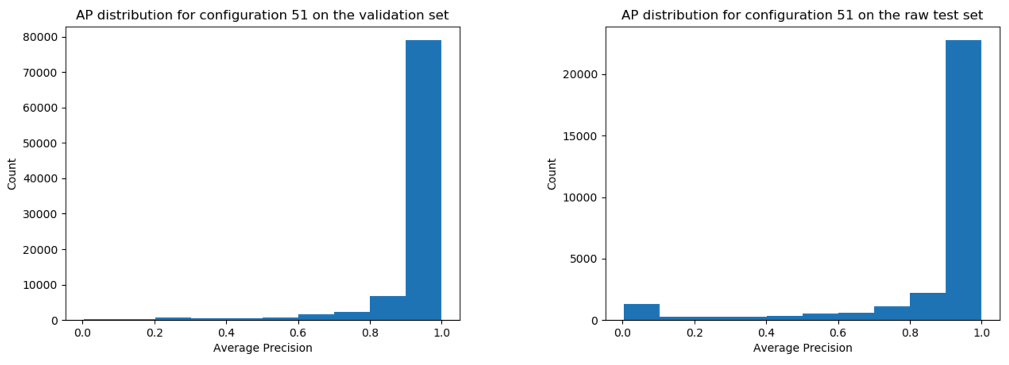
In order to understand if the errors were caused by the feature extraction networks or by annotation mistakes, such as wrong boxes or videos not containing the person they were supposed to contain, I performed this process for both of the main face feature extraction networks (VGGFace2 and ArcFace) to ensure that I did not miss something related to model-specific issues. The vast majority of score anomalies was in fact produced by annotation errors, and the remaining ones were mostly hard samples (low-quality images, old footage, glasses, extreme face poses, etc.).

I discovered that at least 96 of the 1375 test videos did not portray the person of interest, but somebody related in some way (similar name, a sport teammate, a fellow band member, relatives, etc.). I thus removed those videos from the ground truth of the queries for that identity, but I still kept the videos in the dataset to act as distractors.

Moreover, I found that at least 195 more videos had a significant number (ranging from 15% to 100%) of wrongly annotated boxes, making the pipeline perform queries with a wrong face image. I thus excluded the images extracted from those videos from the query list, while still keeping the videos as query ground truth results, since differently from the previous set of videos, these still contained the person of interest.

The final query set of the test split contains 23,208 queries from 1,077 videos and 74 identities. 4 identities were removed because less than 2 valid videos containing them were left in the dataset after the cleaning procedure. However, note that the pipeline built the feature database for each experiment on the entire set of 1,375 videos, since videos were only removed from query ground truth of from query inputs, but were always kept in the dataset to act as distractors.

The cleaned test set had an AP distribution much more similar to the one in the validation set, without the anomalous peak in low-score queries, as it is shown in Figure 4.11. While all the figures show the AP distribution for configuration 51, which uses VGGFace2 (see Section 4.3.4), the test was also performed using configuration 60, with ArcFace, where a very similar anomaly was present. The cleaning also had an analogous positive effect, so the graphs are not shown here.



(a) Validation set (b) Raw test set

Figure 4.10: Distribution of the AP for all the queries in the validation set and in the raw test set. Note how in the test set there is a peak in the distribution for AP < 0.1. The test was done using configuration 51 of the pipeline, described in Section 4.3.4.

While analysing the results on the validation set, I also found out that a consistent fraction of the videos that were supposed to contain one of the identities of the validation set, Patrick Monahan, singer, actually contained a different Patrick Monahan (a Irish-Iranian comedian). This was probably a result of the VoxCeleb2 dataset construction protocol not being able to distinguish between homonyms. For this reason, queries containing faces of this identity were ignored at evaluation time; the features from people detected in those 27 videos were still kept in the database to increase the number of distractors at query time, similarly to what was done on the test set.

Other than this, I did not perform a thorough cleaning of the validation set, since it would be way more time consuming and it would probably have less benefits (you can see the validation has very few queries with less than 10% AP, indicating a much lower number of important annotation mistakes). As previously stated, it is more important to look at relative differences in AP on the validation set in order to evaluate different pipeline hyperparameters, rather than at absolute accuracy values. The latter might not be representative

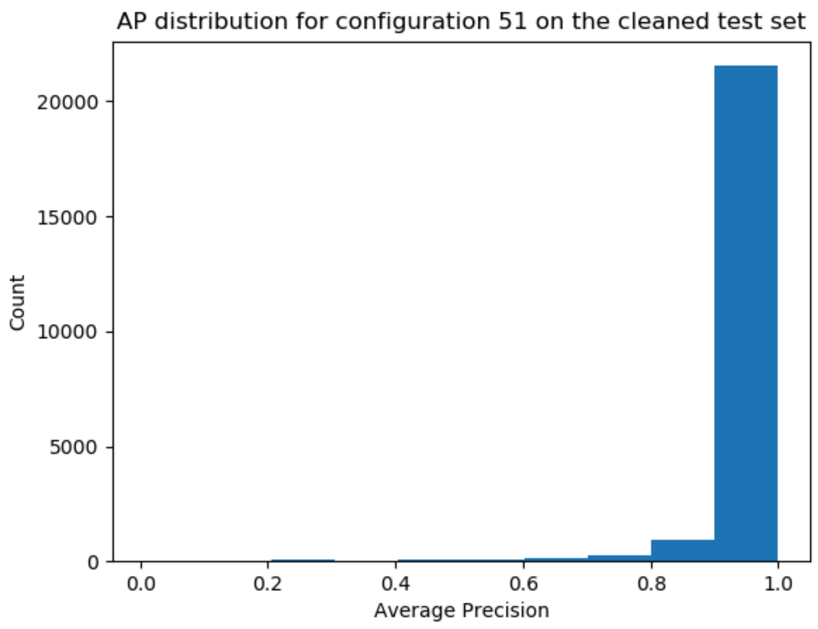


Figure 4.11: Distribution of the AP for all the queries in the cleaned test set. The test was done using configuration 51 of the pipeline, described in Section 4.3.4.

of the performance of the network, since most validation identities were also used to train the feature extraction networks.

4.3.2 Implementation details

I implemented the main code of the pipeline in Python 3, specifically Python 3.7. The ImageLab Shot Detector was originally implemented for Windows in C++; I ported it to Linux and added a Python interface on top. Since many of the networks I tested already had an open source implementation in the PyTorch deep learning framework, I decided to use PyTorch implementations for almost every CNN employed, except for TransNetV2. TransNet and TransNetV2 were only available in TensorFlow. Since PyTorch tends to use less GPU memory than TensorFlow, which was important in order to run multiple parallel processes for feature extraction, I ported TransNet to PyTorch. TransNetV2 was instead kept in TensorFlow, since the code was more complex to port to PyTorch, and the network didn't seem to bring any accuracy benefit to the pipeline anyway, so I mostly used TransNet in the various tests described in Section 4.3.4. I used PyTorch 1.4 and TensorFlow 2.3, both with CUDA 10 and cuDNN 7.

I ran the tests on a machine with the following hardware: Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz, GPU NVIDIA GeForce GTX 1080 (8GB VRAM), 32 GB RAM.

4.3.3 Evaluation metrics

For every test I report results using the Mean Average Precision (MAP) metric, commonly used in information retrieval. It is defined as follows:

$$MAP = \frac{1}{n_q} \sum_{i=1}^{n_q} AP_i,$$

where n_q is the total number of queries performed, and AP_i is the Average Precision (AP) of the i-th query. The Average Precision approximates the area under the precision-recall curve, and various formulations exist. I used the scikit-learn implementation [327], which can be formulated as follows:

$$AP = \sum_k (R_k - R_{k-1}) P_k,$$

where P_k and R_k are the precision and recall, respectively, computed considering only the top k retrieved items (videos in our case), sorted by descending matching score. The AP metric implicitly assigns a higher weight to errors in the top results with respect to errors ranked lower in the list (i.e. with lower matching scores). This is a desirable property, since in a retrieval system the user mainly focuses on the top retrieved results, and expects the first results to match better with the query with respect to the ones that are ranked lower.

4.3.4 Results and discussion

4.3.4.1 Hyperparameter choice and pipeline validation

Since the pipeline has a large number of hyperparameters, and each different configuration requires between 1 and 3.5 days to extract all the features and run the queries on the validation set, testing every possible combination of hyperparameters (e.g. using a grid search) was not feasible. For this reason, I started from a base configuration and gradually evaluated changes in one or two hyperparameters at a time in order to find the locally best configuration.

The starting configuration

The starting configuration was the following:

- shot detector: ILSD, with default maximum window radius W equal to 3, following the original paper;
- face detector: MTCNN, with default parameters from the original implementation (minimum face size: 32 pixels per side, score thresholds for the 3 stages: 0.6, 0.7, 0.8, NMS threshold: 0.7, scale factor for the image pyramid: 0.707). For this starting configuration, faces were extracted every 10 frames, regardless of the FPS rate of each video. MTCNN (like many other face detectors) tends to extract tight boxes around the faces, often cutting off some contextual features such as the hair, due to the dataset annotations used for training. Since such features are useful to obtain better results, and are in fact included in the face recognition datasets on which VGGFace/ArcFace were trained on, I decided to expand the boxes computed by MTCNN and extract face patches that were centered on the MTCNN boxes and enlarged to be 1.2 times the length and height of the predicted boxes;
- face tracker: IOU tracker. σ_{IOU}, the minimum IOU to consider two boxes in consecutive frames as matching, was set to 0.5. Tracklets with less than $t_{min} = 4$ detections were discarded. In this way, only tracks which covered at least a 30-frame interval between the first and last detection were considered;
- face feature extractor: VGGFace2 with SENet-50 (also known as SE-ResNet-50) backbone. Only 3 samples were extracted for each track,

avoiding the initial and final sections of the tracks, as explained in Section 4.2.1.4;

- feature aggregation: features were aggregated by averaging the feature vectors and applying L_2 normalization before and after the averaging. The inter-shot clustering was performed by using Hierarchical Agglomerative Clustering with average linkage, cosine distance, and 0.5 maximum distance threshold.

I tested this starting configuration both on images extracted from the clipped videos provided with VoxCeleb2 (denoted as LQ) and on images extracted from the higher quality version of the YouTube videos I downloaded (denoted as HQ), as explained in Section 4.3.1.1. An example of the same face from the two different image sets is shown in Figure 4.12. Image 4.12a is clearly more blurry than image 4.12b, since it has a lower resolution: all faces in the clips provided with VoxCeleb2 were resized to 224x224 pixels, while in the example the high quality face is 410x410[14]). Moreover, since the extracted faces in VoxCeleb2 were recompiled as an mp4 video, the image on the left experiences one additional video compression round with respect to the image on the right, and thus presents more compression artifacts.

The evaluation results on the two image sets are shown in Table 4.1. As we can see, the higher quality images resulted in an improvement of more than 2% MAP. For this reason, all the subsequent experiments were done on the high quality images. Note that an end user in TVBridge would also run queries using face images extracted from original videos, without additional processing steps; so, the use of the higher quality images for evaluation more closely reflects the real-world usage conditions of the system.

Initial hyperparameter exploration

[14]Note that the VGGFace2 network preprocessing step resizes the input image to 256x256 pixels and then crops the middle square of size 224x224, which gets fed to the network. For this reason, any face image smaller than 256x256 pixels is not optimal, since it needs to be upscaled first.

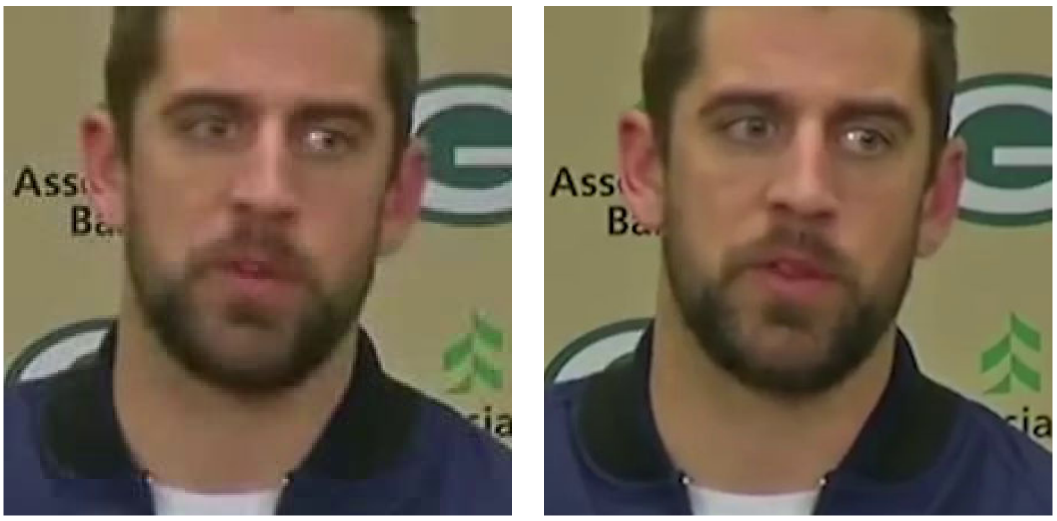


Figure 4.12: The same face from the same frame extracted from the low-quality VoxCeleb2 clips and from the original YouTube videos.

Query images	MAP
LQ	90.60
HQ	**92.95**

Table 4.1: Mean Average Precision (percentage) for the two considered query datasets (validation set). Features were extracted using the base pipeline configuration described in the main text.

The next step I took was modifying some of the hyperparameters of the pipeline without changing the algorithms employed for each step. I tested a different clustering distance threshold σ_{cl}, various frame intervals I_{fd} for the face detection (specified in seconds, instead of the raw number of frames, in order to be independent from the video frame rate), I changed the aggregation function f_a to median instead of average, the tracking threshold t_{min}, the VGGFace2 backbone to ResNet-50 (as opposed to SENet-50), and the minimum size of the detected faces s_{min}. The results for all these experiments are shown in Table 4.2.

ID	σ_{cl}	I_{fd}	f_a	t_{min}	VF2 backbone	s_{min}	MAP
0	0.5	10 frames	mean	4	SENet-50	32	92.95
1	0.4	10 frames	mean	4	SENet-50	32	93.06
2	0.4	1 s	mean	4	SENet-50	32	**93.71**
3	0.4	2 s	mean	4	SENet-50	32	91.36
4	0.4	2 s	mean	2	SENet-50	32	**93.68**
5	0.4	1 s	median	4	SENet-50	32	93.39
6	0.4	1 s	mean	4	ResNet-50	32	93.40
7	0.4	0.3 s	mean	4	ResNet-50	32	92.70
8	0.4	2 s	mean	2	ResNet-50	32	93.37
9	0.4	1 s	mean	4	SENet-50	20	**93.69**

Table 4.2: MAP (percentage) on the validation set for various configurations of the pipeline. Bold values highlight the 3 best configurations, which have a very close performance. The first row indicates the starting configuration, the same considered in Table 4.1. The ID represents each configuration for easy reference, σ_{cl} is the clustering distance threshold, I_{fd} is the frame sampling interval for face detection, f_a is the feature vector aggregation function, t_{min} is the minimum track length (number of faces), VF2 backbone is the VGGFace2 CNN backbone, s_{min} is the minimum face size threshold in pixels.

First of all, decreasing the clustering threshold to 0.4 slightly increased the MAP by around 0.1%. Since the improvement was small, I did not test other thresholds and fixed it at 0.4 for the subsequent experiments. Reducing the frequency of frame sampling for face detection from 1 every 10 frames (that converts roughly to one frame every 0.17-0.42 seconds, depending on video frame rate) to 1 every second, the MAP increased by more than 0.6%. Pushing it further to every 2 seconds did decrease the accuracy by 2.35% though; the track length threshold might have played a role, since now only tracks longer than 6 seconds were kept, and reducing t_{min} to 2 brought the performance back to 93.68%. Changing the aggregation function to median reduced the MAP by roughly 0.3%, and thus this change was discarded. Replacing the VGGFace2 backbone to ResNet-50 also worsened the performance by a similar amount, and playing with the sampling frequency did not help. Finally, I reduced the minimum face size from 32x32 pixels to 20x20, in order to include some smaller faces; this did not have a significant impact in performance though, probably because smaller faces are unlikely to represent foreground people, especially in interviews. Moreover, small faces can be hard to distinguish and the feature extraction network might not perform well on them. In fact, lowering the threshold might have introduced a few small, low resolution faces which might have acted as distractors and slightly influenced the performance, albeit in a negligible way.

Changing the shot detector

After the first hyperparameter search, I proceeded to test different shot detectors. I replaced ILSD with TransNet and TransNetV2. Regarding TransNet, I also performed some fine-tuning experiments on the ClipShots dataset and tested a double threshold strategy in order to try to improve the performance. Since these experiments were not particularly successful, they are briefly described in Appendix C.

The results are shown in Table 4.3. As we can see, changing shot detector does not have any significant impact on the performance. However, the time/memory efficiency of the three algorithms is different. Running the pipeline with ILSD and TransNet in “serial” (using parallel processes would interfere with a clean measurement of running time) showed that TransNet makes the pipeline much faster: extracting features on the 4260 videos of the validation set took 3189 minutes (2.21 days) with TransNet, compared to 3773 minutes (2.62 days) using ILSD. That means that changing the shot detector alone made the pipeline about 16% faster. Note that this is after I optimized the running time performance of the original implementation of ILSD by implementing parallel video decoding, so that the shot detection process did not have to wait for the entire video to be decoded.

TransNetV2 had a similar running time to TransNet[15], but used much more GPU memory (probably because of the use of TensorFlow, which is often more memory-intensive). Since TransNetV2 got no better results than the original TransNet, I used the latter for all the subsequent experiments. I also used the original TransNet weights, without the ClipShots fine-tuning or the double threshold, since these changes had no significant impact on accuracy. This might be due either to the fact that in general small changes to the quality of the shot detector do not influence the retrieval performance in a significant way, or to the fact that the distribution of transitions in the ClipShots dataset does not reflect the one in the VoxCeleb2 dataset. It is not possible to easily check if the second hypothesis is the actual problem, since there is no ground truth for shot detection in VoxCeleb2 and it is thus hard to compare transitions statistics of the two datasets.

Exploring hyperparameters related to face recognition

Before testing different face feature extraction networks, I decided to experiment first with some related hyperparameters. Specifically, the number

[15]In the original TensorFlow implementation of the networks. Porting TransNet "V1" to pyTorch at first made the network slower than TransNetV2, but I changed the implementation so that the video decoding was done on a separate thread, in parallel with the shot detector. This actually made the shot detection process using TransNet "V1" slightly faster than using the TensorFlow version of TransNetV2. Since TransNetV2 did not improve the pipeline retrieval performance, I did not port it to pyTorch, nor I optimized the video decoding step.

ID	Base conf. ID	Shot detector	MAP
2	-	ILSD	93.71
9	-	ILSD	93.69
10	9	TransNet	93.66
11	2	TransNet	93.71
12	9	TransNet (ClipShots)	93.61
13	9	TransNet (double thr.)	93.67
14	9	TransNetV2	93.61

Table 4.3: MAP (percentage) on the validation set for various configurations of the pipeline, using different shot detectors. The first two rows indicate the starting configurations, see Table 4.2. The ID represents each configuration for easy reference, *Base conf. ID* indicates the starting configuration ID. *TransNet (ClipShots)* refers to the version of TransNet fine-tuned on the ClipShots dataset, while *TransNet (double thr.)* refers to the use of TransNet with a double thresholding strategy; both are described in Appendix C.

of faces sampled from each tracklet and the sampling strategy, some clustering hyperparameters that are influenced by the choice of features, i.e. by the choice of the specific feature extraction CNN, the face bounding box scale factor, and the minimum track length threshold t_{min}.

I changed the value of 3 samples per track to 1, 2 or 5. Results in Table 4.4 show that raising or lowering the number of samples extracted per track did not improve the accuracy of the pipeline. For this reason I fixed the number to 3 samples per track for the following tests.

Regarding the clustering hyperparameters, I changed the linkage method and the distance threshold σ_{cl}. I also replaced the average linkage with single and complete linkages, and I further lowered σ_{cl} from 0.4 to 0.3 (it was 0.5 initially).

Results are shown in Table 4.5. Single linkage worsened the performance by more than 0.6%, while complete linkage obtained a very similar score to average linkage. Lowering σ_{cl} to 0.3 had no impact on the average linkage

ID	Samples/track	MAP
11	3	93.71
15	1	93.59
16	2	93.69
17	5	93.67

Table 4.4: MAP (percentage) on the validation set computed by changing the number of faces sampled for each face track for feature extraction. The first row indicates the base configuration, see Table 4.3. All the other configurations were based on configuration 11.

method, while it slightly worsened the performance when using complete linkage. For this reason, I kept using the threshold $\sigma_{cl} = 0.4$ for most of the following tests (unless explicitly noted). Note that lowering the clustering distance threshold produces more clusters (since the agglomeration process is stopped earlier). This produces more feature vectors as a result, increasing the feature database size on disk and in memory, and the query time (since query vectors need to be compared with more vectors in the database). So, if the model accuracy is very similar, it is more convenient to set the threshold higher. To give an idea of the difference between a threshold of 0.4 and 0.3, configuration number 19 produced a feature database of 362 MB (containing 21,791 feature vectors), while configuration 21 produced a database of size 456 MB (containing 27,461 feature vectors), 25% more.

After that, I decided to further explore the face sampling strategy and tested a “quality-based” sampling strategy. Since the detection score output by a face detector has been shown to be a good predictor of face quality [328], instead of sampling the 3 middle faces of a track, I decided instead to choose the 3 faces with the highest detection score. Other factors related to face sampling are the bounding box enlargement factor (s_{det}), discussed previously in this section, and set to 1.2 for all the previous experiments, and the minimum track length t_{min}. Both changes also influence the feature extraction process, the first by directly changing the content of the input image,

ID	Linkage method	σ_{cl}	MAP
11	average	0.4	93.71
18	single	0.4	93.07
19	complete	0.4	93.74
20	average	0.3	93.71
21	complete	0.3	93.68

Table 4.5: MAP (percentage) on the validation set computed by changing HAC linkage method and clustering distance threshold σ_{cl}. The first row indicates the base configuration, see Table 4.3. All the other configurations were based on configuration 11.

and the second by removing more/less tracklets, thus potentially changing the amount of noise in the features. For this reason, it is important to choose the described hyperparameters appropriately before testing different feature extractors.

Table 4.6 shows the results. Both decreasing and increasing t_{min} led to worse performance. A very low t_{min} would keep many bad detections/distractor faces in the database, while a very high t_{min} would remove too much information by deleting tracks shorter than 6 or 9 seconds for a t_{min} of 7 or 10 respectively. The quality-aware sampling didn't have much effect, so it was discarded for the subsequent tests. The scaling factor instead had a significant effect, improving the starting MAP of 93.74% to 94.34%, a 0.6% increase. This means that including more context around the face is important for the VGGFace2 CNN, probably because the network was trained on faces with more context (or simply because the training faces were smaller and occupied a more "central" position in the input images, ignoring the borders). More face alignment considerations will be done next, with the evaluation of the other feature extraction networks.

Comparing different face feature extractors

I replaced the VGGFace2 CNN with the original VGGFace and with ArcFace.

ID	Sampling	s_{det}	t_{min}	MAP
19	Middle frames	1.2	4	93.74
22	Middle frames	1.2	1	92.47
23	Middle frames	1.2	2	93.20
24	Middle frames	1.2	7	91.20
25	Middle frames	1.2	10	83.12
26	Quality-aware	1.2	4	93.71
27	Middle frames	1.0	4	91.48
28	Middle frames	1.5	4	**94.34**
29	Middle frames	1.75	4	94.22
30	Middle frames	2.0	4	93.44

Table 4.6: MAP (percentage) on the validation set computed by changing the sampling strategy. The first row indicates the base configuration, see Table 4.5. All the other configurations were based on configuration 19. *Middle frames* indicates the standard sampling strategy used for all the previous tests, described in Section 4.2.1.4, while *Quality-aware* indicates the sampling strategy based on the face detection score. s_{det} is the face bounding box scaling factor (i.e. 2.0 means the detected bounding box width and height were doubled before feeding the face to the feature extraction network). The result in bold indicates the best performing configuration.

For VGGFace, since the entire VGG-16-based network was provided by the authors, including the prediction layer used to classify the 2622 identities of VGGFace, I tried extracting both the features from the penultimate fully-connected layer, like it was done in the paper, and the pooled convolutional features (before the fully-connected layers), like it is done in the other face recognition networks. Note that the fully-connected features are 4096-d vectors, while the pooled convolutional features are smaller 512-d vectors. I also tested both euclidean and cosine distance metrics.

For ArcFace, the original network configuration was used exactly like in the paper. Note that ArcFace requires face alignment before extracting the features, so the facial landmarks extracted using MTCNN were used to align the faces according to the original ArcFace implementation [329]. The clustering distance thresholds for both networks was kept at 0.4 for now, both for Euclidean and cosine distances[16]. Results are shown in Table 4.7.

VGGFace seems to perform consistently worse than VGGFace2. Using the euclidean distance helped a bit, but the results are still significantly worse than its newer version. Adding to that, VGGFace is based on the heavier and much slower VGG-16 CNN. The pipeline with the best performing VGGFace configuration (configuration 34) took 4336 minutes (3 days) to extract the features from all the videos using 4 parallel processes, while VGGFace2 only took 1825 minutes (1.27 days) to do the same. Moreover, since the FC features are 4096-dimensional, the feature database occupied 4.23 GB on disk, compared to 316 MB of the VGGFace2 version, and the query time rose accordingly: running all the 94,233 queries on 6 parallel processes took 590 minutes (2.7 queries/second on average), as opposed to just around 55 minutes (28.6 queries/second on average) for VGGFace2. In order to ensure that the clustering/feature aggregation process was not at fault for the bad accuracy of VGGFace, I performed some experiments without any aggregation step in

[16]Remember that for unit vectors, both the euclidean distance and the cosine distance (computed as one minus the cosine similarity) are in the range $[0, 2]$. All feature vectors in the pipeline are normalized to have unit norm after the extraction step.

ID	Base conf. ID	Feature extr. net	Distance metric	s_{det}	MAP
11	-	VGGFace2	Cosine	1.2	93.71
19	-	VGGFace2	Cosine	1.2	93.74
31	11	VGGFace1 (conv)	Cosine	1.2	55.71
32	11	VGGFace1 (conv)	Euclidean	1.2	67.32
33	11	VGGFace1 (FC)	Cosine	1.2	74.69
34	11	VGGFace1 (FC)	Euclidean	1.2	75.25
35	19	ArcFace (RN-50)	Euclidean	1.2	92.74
36	19	ArcFace (RN-50)	Euclidean	1.5	92.75
37	19	ArcFace (RN-101)	Euclidean	1.2	**94.59**
38	19	ArcFace (RN-101)	Euclidean	1.5	**94.58**
39	19	ArcFace (RN-18)	Euclidean	1.2	90.38
40	19	ArcFace (RN-18)	Euclidean	1.5	90.38

Table 4.7: MAP (percentage) on the validation set computed by changing the face feature extractor. The first two rows indicate the base configurations, see Tables 4.3 and 4.5. All the other configurations were based either on configuration 11 or 19, with the only difference being that configuration 11 uses average linkage, while 19 uses complete linkage (for each configuration, the *Base conf. ID* column indicates its starting configuration). *Feature extr. net* indicates the feature extraction network employed, where VGGFace1 is the original VGGFace, to distinguish it from VGGFace2, *(conv)* indicates that the pooled convolutional features were used, while *(FC)* indicates that the fully-connected features were used. RN-18, RN-50 and RN-101 indicate that a ResNet-18, ResNet-50 or ResNet-101 backbone was used, respectively. s_{det} is the face bounding box scaling factor (i.e. 2.0 means the detected bounding box width and height were doubled before feeding the face to the feature extraction network). The two best results are highlighted in bold.

the pipeline: I extracted a single face from each track, thus removing the need for feature aggregation, and saved the obtained feature vectors without performing inter-shot clustering. But this led to a terrible MAP of just 1.18%, close to the MAP for a random baseline retrieval system on our query set[17]. For all these reasons, I abandoned the use of VGGFace1 for all the subsequent tests.

Regarding ArcFace, the results were much better. The version with ResNet-101 performed the best, as expected since it is the largest and most complex model of the three ArcFace versions, but the ResNet-50 version still obtained 0.65% lower AP than the 93.40% AP of the VGGFace2 ResNet-50 version (configuration 6) and 1% worse AP than the VGGFace2 SENet-50 version, that requires only slightly more computational time (1759 minutes runtime for the ArcFace-based pipeline of configuration 35 vs. 1807 minutes of the SENet VGGFace2 pipeline of configuration 19). Using ArcFace in the pipeline seems to consistently result in a higher number of feature vectors in the feature database. Configuration 35 obtained 117,790 512-d feature vectors, which resulted in a database size of about 497 MB, as opposed to the 21,791 2048-d feature vectors of configuration 19, with a size of 364 MB.

In order to exclude the possibility of a clustering threshold problem, I ran some tests using a different clustering threshold, including a threshold of 0, which implies no inter-shot clustering was performed. Results are shown in Table 4.8. Differently from VGGFace2, where lowering the threshold further than 0.4 brought no improvement, using any $\sigma_{cl} \leq 0.25$ slightly increased the MAP by 0.1%. However, this increase was counterbalanced by a substantially higher number of feature vectors in the database, leading to 30% longer query times. It is interesting to note that even rising the clustering threshold up to 1 (out of a maximum of 2) did not degrade the performance significantly,

[17]I computed the expected MAP for the validation set using the exact formula presented in [330]. The expected random MAP is the average expected AP over all the queries of the considered dataset. Averaging the expected AP over the 94,233 queries in the validation set resulted in an expected MAP of 1.17% for a random retrieval system.

highlighting the advantage of the Additive Angular Margin loss employed in ArcFace training, which tends to impose strong angular margins between different classes. Another thing to note regarding ArcFace performance is the minimal impact of the bounding box scale factor s_{det}, which, differently from VGGFace2, didn't change the performance of the algorithm in a meaningful way.

ID	σ_{cl}	MAP
35	0.4	92.74
41	0.0	92.85
42	0.1	92.85
43	0.25	92.85
44	0.5	92.60
45	0.75	92.42
46	1.0	92.54

Table 4.8: MAP (percentage) on the validation set using ArcFace with different clustering distance thresholds σ_{cl}. The first row indicates the base configuration, see Table 4.7. All the other configurations were based on configuration 35.

Removing face alignment

Before testing different face detectors, since one of them, SSH, cannot predict face landmarks, and ArcFace needs them for face alignment, I tested the current configuration by removing face alignment during database creation, at query time, or both. I used the ResNet-18 version of ArcFace in order to speed up the feature extraction time, since the impact of face alignment on it would presumably be similar to its impact on the ResNet-50 or ResNet-101 ArcFace CNNs.

The results are shown in Table 4.9. Differently from VGGFace2, that does not need face alignment, ArcFace was trained with it, and the network is only able to function properly with it. Removing face alignment from the query step, or both steps, degraded the performance to random baseline, with 1-2% MAP. Reinstating face alignment only in the query step recovered part of the performance, which was however still significantly worse than the original configuration. This suggests that the face boxes extracted by MTCNN in the feature database creation stage, while not being explicitly aligned, were still closer to what ArcFace expected as input with respect to the face boxes from the VoxCeleb2 dataset, which were extracted using a SSD-based face detector, because the query face alignment was absolutely necessary to obtain a result that was significantly better than a random baseline.

ID	DB creation face align.	Query face align.	MAP
40	Yes	Yes	90.38
47	No	No	2.28
48	No	Yes	61.84
49	Yes	No	1.55

Table 4.9: MAP (percentage) on the validation set using ArcFace and removing face alignment from the database creation step, query step, or both steps. The first row indicates the base configuration, see Table 4.7. All the other configurations were based on configuration 40.

Replacing the face detector

I then moved on to test the pipeline with different face detectors: SSH and RetinaFace, the latter with the lighter and faster MobileNet-0.25 backbone.

I only used the MobileNet backbone and not the ResNet-152 backbone for two main reasons: first, ResNet-152 is way slower and much more memory-intensive than MobileNet-0.25, but also compared to MTCNN; using ResNet-152 would significantly slow down the tests, because the increased memory usage would force me to decrease the number of parallel processes used to create the feature database from 4 to 3 or even 2 in order to avoid GPU memory issues. The second reason is that the particular pyTorch implementation I used [331] only provided the trained weights for the MobileNet backbone. SSH is instead only available with the VGG-like backbone, and is thus slower, as we will see.

The results of the face detector changes are shown in Table 4.10. For VGGFace2, using SSH as a face detector slightly worsened the performance, while also being much slower: the database construction with 4 parallel processes took 5127 minutes for configuration 54, versus the 1807 minutes of the MTCNN version. RetinaFace had almost no impact on the performance of VGGFace2, but it was slightly faster, making it a more desirable choice over MTCNN: the feature database construction only took 1715 minutes. However, it is important to note that with 4 parallel processes the GPU speed is the main bottleneck of the pipeline, thus the biggest increase in performance is obtained by reducing the computation load of the GPU. Since MTCNN and RetinaFace use different amounts of CPU computation time, the speed comparison might not be representative of a serial (i.e. single-process) execution environment, where the GPU might not be the computational bottleneck at all times. Moreover, since the speed of MTCNN not only depends on the frame input size, like other face detectors, but also on the number of candidates for each frame[18], it is not so straightforward to claim which of the two networks is faster in an absolute way.

As expected from the earlier tests, since SSH does not compute facial landmarks and face alignment was not performed, ArcFace (configuration 56) performed poorly with SSH, with a 2% MAP. RetinaFace produced instead an increase in performance for ArcFace. The combination of ResNet-101 ArcFace with RetinaFace allowed the pipeline to reach 95.48% MAP, and the ResNet-50 ArcFace managed to beat the ResNet-50 VGGFace2, in contrast to what happened using MTCNN. The combination of ResNet-50 ArcFace and RetinaFace might be a good compromise between accuracy and performance, with the entire pipeline taking around 1600 minutes to extract the features from all the validation set videos. The pipeline with ResNet-101 ArcFace took

[18]For example, frames with no face candidates after the first step will avoid execution of the second and third steps in MTCNN.

a similar time, but again, this might be due to the randomness of parallel execution. Also note that ResNet-101 uses more GPU memory, which might be important when running multiple parallel query processes[19].

Interestingly, independently from the feature extraction network, the pipeline outputs less feature vectors using RetinaFace than using MTCNN, while it did not change too much with SSH. This is probably due to the lower number of faces detected using MobileNet. In order to test this, I ran the pipeline only up until the face detection step for each of the three tested face detectors, and counted the raw number of faces before any tracking, sampling or clustering was done. This showed that, using the same shot detector (TransNet) and the same frame sampling rate (1 frame per second), MTCNN extracted 2,434,216 faces, SSH extracted 2,332,164 faces, while RetinaFace extracted 1,940,093, confirming the fact that the MobileNet-based face detector tends to identify less faces in general. As already noted though, this did not reduce the retrieval accuracy, probably because the bigger, more easily recognizable faces are the foreground ones, which are also the ones the queries (and the TVBridge users) are looking for.

Experiments with feature quantization and inter-video clustering

In addition to all the experiments described above, I also tested two "experimental" features, already described in section 4.2.1: feature quantization and inter-video clustering.

Regarding feature quantization, I added it to the end of configuration 51, i.e. VGGFace2 + RetinaFace. I could only use VGGFace2 features because the AQBC algorithm needs non-negative vector values. The retrieval performance on the quantized features dropped from 94.44% to 93.71%, but at the same time the feature database size decreased from 297 MB to 41 MB. It is important to note that this was only a feasibility experiment, and the

[19]On the machine I used I was able to run 6 parallel processes with the ResNet-50, but I could only use 5 with the ResNet-101 without incurring in serious memory issues, and the system took 120 minutes to run all the queries instead of 104 minutes for ResNet-50.

ID	Base conf. ID	Feature extr. net	s_{det}	Face detector	MAP
19	-	VGGFace2	1.2	MTCNN	93.74
28	-	VGGFace2	1.5	MTCNN	94.34
35	-	ArcFace (RN-50)	1.2	MTCNN	92.74
37	-	ArcFace (RN-101)	1.2	MTCNN	94.59
50	19	VGGFace2	1.2	RetinaFace	93.75
51	28	VGGFace2	1.5	RetinaFace	94.44
52	35	ArcFace (RN-50)	1.2	RetinaFace	94.59
53	37	ArcFace (RN-101)	1.2	RetinaFace	**95.48**
54	19	VGGFace2	1.2	SSH	93.57
55	28	VGGFace2	1.5	SSH	94.16
56	37	ArcFace (RN-101)	1.2	SSH	2.07

Table 4.10: MAP (percentage) on the validation set by varying the face detector used. The first four rows show the base configurations, see Tables 4.5, 4.6 and 4.7. For each configuration, the *Base conf. ID* column indicates its starting configuration. The feature extraction networks and bounding box scale factor s_{det} are also reported for ease of consultation.

implementation was far from optimized. First of all, NumPy does not provide a native implementation of binary vectors, so I used 8-bit unsigned integer (uint8) vectors to store the quantized features. Using bit fields would have further reduced the feature size by a factor of 8, both on disk and in memory; NumPy allows to easily pack 8 binary values into a single uint8, but computing the cosine distance on these compacted vectors was extremely slow, since neither Python nor NumPy have a native, and thus efficient, implementation of popcount, a requirement to perform quick cosine distance computation, as described in [324]. The extreme loss in query time performance was not worth the space reduction gain: a single query on the validation set could take almost a minute on the compacted binary vectors, compared to the 0.2 seconds for the original features or the uint8 quantized features. Despite these problems, the experiment shows that with an efficient implementation of binary vectors, the use of quantized features could be a valid tradeoff choice between accuracy and disk/memory space, since it can in principle reduce the latter by about 64 times (since the original vectors were 64-bit floating points) while only losing less than 1% MAP and thus still providing a good accuracy.

For the inter-video clustering experiments, I used the same HAC algorithm on the final database, with the problems and limitations already described in Section 4.2.1.7. I tried to add inter-video clustering to configurations number 51 (VGGFace2 features), 52 (ResNet-50 ArcFace features) and 53 (ResNet-101 ArcFace features); however, while I was able to run the clustering step on the VGGFace2 features of all the 4260 videos in the validation set in around 13 seconds, I ran into problems with both ArcFace configurations. As explained previously, the pipeline produced a higher number of feature vectors when using ArcFace, and running the clustering algorithm on such a large number of samples incurred in RAM issues, so I was not able to cluster all the features in one go for those configurations.

In order to solve that I tried two approaches: first, I split the 100k feature vectors into 4 groups and only performed HAC inside those 4 groups (partial clustering); the second approach was raising the distance threshold σ_{cl}, performing the partial clustering as just described, aggregating the features and then performing a final round of clustering on the now smaller feature set (2-round clustering). The second approach was not feasible without raising σ_{cl}, since the partial clustering did not reduce the number of feature vectors enough for a full clustering to fit in memory.

Results are shown in Table 4.11. Inter-video clustering had a significant impact on the VGGFace2 features: the MAP raised from 94.44% to 95.60%, at the expense of additional running time in a real-world scenario, where the clustering needs to be updated every time a new video is added to the database (see Section 4.2.1.7). Neither the partial clustering nor the 2-round clustering on ArcFace improved the MAP. As a comparison, I also performed the partial clustering on the VGGFace2 features, and despite a smaller performance gain with respect to running a full clustering, it still obtained almost 0.8% increase in MAP. This might indicate that VGGFace2 features are slightly more unstable under different lighting conditions, so that averaging the features of a person across different videos leads to more stable features; another possibility is that the feature vectors produced by VGGFace2 are less well-separated, and aggregating distractor faces (such as background faces) reduces the number of false matches and improves the performance. In conclusion, inter-video clustering is a promising future direction of research. The use of a more efficient online clustering algorithm might lead to accuracy improvements and query speedup, since the query image features would need to be compared with a lower number of database feature vectors.

4.3.4.2 Running time considerations

While not the main focus of the thesis, some choices were still made with time efficiency in mind, as we have seen. Designing an efficient pipeline is in fact necessary for real-world applications, like TVBridge.

Running the pipeline in configuration 11 in single-process mode, it was able to extract the features from the 396.4 hours of footage of the validation set in 3189 minutes. This means that on average the pipeline was able to run

ID	Base conf. ID	Feature extr. net	Inter-video clustering	σ_{cl}	MAP
51	-	VGGFace2	None	0.4	94.44
52	-	ArcFace (RN-50)	None	0.4	94.59
53	-	ArcFace (RN-101)	None	0.4	95.48
57	51	VGGFace2	Full	0.4	95.60
58	52	ArcFace (RN-50)	Partial	0.4	94.55
59	53	ArcFace (RN-101)	Partial	0.4	95.44
60	51	VGGFace2	Partial	0.4	95.22
61	53	ArcFace (RN-101)	2-round clustering	0.75	95.48

Table 4.11: MAP (percentage) on the validation set after adding inter-video clustering on the final face features. The first three rows show the base configurations, see Table 4.10. For each configuration, the *Base conf. ID* column indicates its starting configuration. The feature extraction network is reported for ease of consultation. In the *Inter-video clustering* column, *None* means no inter-video clustering was performed, *Full* means a single-run clustering on the entire feature set was performed, *partial* means clustering was run separately on 4 splits of the feature set, and *2-round clustering* means the partial clustering was followed by a clustering over the aggregated features obtained from the partial clustering step. σ_{cl} is the clustering distance threshold.

shot detection, face detection, tracking, feature extraction, aggregation and clustering at 7.5 times the video real-time speed.

Tracking and feature aggregation take a negligible amount of time in the pipeline, feature extraction takes roughly 5% of the running time, shot detection takes about 30% of the runtime and face detection takes the remaining 65%, approximately. These figures were computed on configuration 11 and of course vary depending on the various pipeline hyperparameters, but it is a good rule of thumb to understand which steps of the pipeline are the most computationally expensive. It is important to note that the TransNet speed is close to the CPU video decoding speed: on a 1280x720 video, TransNet takes just about 1.4 times the FFmpeg frame decoding time. For this reason, there is limited room for improvement for the shot detector speed.

Regarding query speed, it grows linearly with the amount of feature vectors in the database. Even with more than 100,000 feature vectors, 6 parallel processes were able to perform about 15 queries per second. This means that the lower bound on single-process is about 2.5 queries per second, 0.4 seconds per query. The actual performance in single process can in fact be even better, since GPU access acts as a bottleneck for the 6 concurrent processes. When there is a large number of feature vectors in the database, the distance computation step dominates the query running time, surpassing the time needed to extract the single feature vector from the query image. Anyway, we can see that even with relatively large video datasets, the query time is reasonably small to allow for a smooth user experience in real-world systems like TVBridge, at least in a prototype stage.

4.3.4.3 Ablation studies

While shot detection, face detection and feature extraction are obviously mandatory for the proposed FBVR system, it would still be interesting to study if the tracking and feature aggregation processes really help with the performance, both in term of MAP and disk/memory footprint.

For this reason, I performed three ablation studies. First, removing the tracking step by feeding the single faces to the clustering algorithm; second, removing the clustering algorithm by saving the aggregated features from the tracking step without inter-shot clustering; third, keeping tracking but removing any feature aggregation by extracting a single face from each track and without any clustering.

Results are shown in Table 4.12. Removing tracking resulted in a loss of more than 1% MAP, indicating that pre-associating faces by their location in adjacent frames can lead to better feature vectors and helps to obtain a more stable clustering. Removing clustering didn't have any significant effect on the MAP, but produced a very high number of feature vectors, with a 2.5 GB feature database, as opposed to 362 MB for the original configuration. This resulted in an larger usage of RAM at query time, which led to heavy memory swapping; this, combined with the longer distance computation times, resulted in the queries taking 293 minutes to run, as opposed to the original 57 minutes. The advantage of inter-shot clustering is thus apparent, considering that its running time penalty is negligible. Finally, removing any feature aggregation resulted in a 0.74% worse MAP, with all the time/space complexity drawbacks just discussed for the configuration without clustering. This also highlights the importance of selecting multiple faces per track in order to smooth out pose and illumination differences, together with other sources of noise (e.g. motion blur or detection/tracking mistakes), that may arise along a track.

ID	Tracking	Samples/track	Clustering	MAP
19	Yes	3	Yes	93.74
62	No	3	Yes	92.58
63	Yes	3	No	93.76
64	Yes	1	No	93.00

Table 4.12: MAP (percentage) on the validation set for various ablation studies. The first row shows the base configuration, see Table 4.5. All the other configurations were based on configuration 19.

4.3.4.4 Test set experiments and analysis of the results

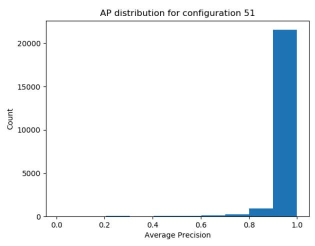
As already discussed, I collected a test set with no identity in common with the training sets of both VGGFace2 and ArcFace, in order to ensure a fair, unbiased evaluation of the models. On this dataset, I decided to run the best configuration for VGGFace2 (excluding the use of inter-video clustering), ArcFace with ResNet-50 and ArcFace with ResNet-101, since the choice of the feature extraction network is arguably more important than the choice of the shot or face detector, as seen in Section 4.3.4.1. I did not test a configuration with VGGFace1, given the poor performance on the validation set. Results are shown in Table 4.13.

ID	Feature extraction net	MAP (val.)	MAP (test)
51	VGGFace2 (SENet-50)	94.44	**97.25**
52	ArcFace (ResNet-50)	94.59	95.48
53	ArcFace (ResNet-101)	95.48	95.96

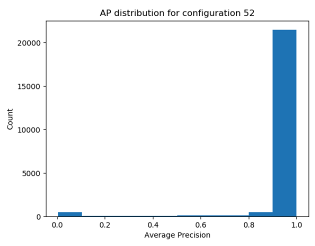
Table 4.13: MAP (percentage) on the test set for the best pipeline configurations involving VGGFace2 and ArcFace. These configurations were previously described in Table 4.10. Validation MAP is also shown for reference.

The pipeline using VGGFace2 reached the best performance on the test set, with 97.25% MAP. It obtained almost 1.3% MAP higher than the slower ArcFace with ResNet-101, despite obtaining a worse performance on the validation set. This suggests that VGGFace2 can more easily generalize on unseen identities. Remember that the fact the scores for all the networks are overall higher on the test set is at least partially due to the test set cleaning process, described previously. To give an idea of how much impact the dataset cleaning process had, configuration 51 obtained 88.15% MAP on the raw test set. This means that more than 9% MAP was lost due to annotation errors and not because of model mistakes. Another factor which might have lowered the validation performance is its bigger size: it is of course harder to find the correct videos among a higher number of possible candidates.

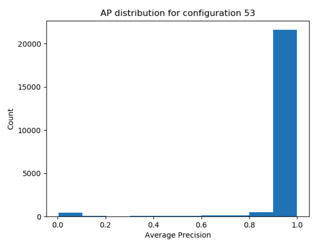
Figure 4.13 shows a comparison between the AP distribution over the 23,208 queries of the test set for the three considered configurations. As we can see, the vast majority of queries obtains near-perfect score: for example, for configuration 51, 21,555 queries (92.9%) obtain an AP higher than 90%, and 13,890 queries (almost 60%) obtain an AP higher than 99.99%, which is basically a perfect score, considering numerical errors in computing the AP. Remember that an AP score equal to 1 means that the k relevant videos for a query are returned exactly in the top k positions in the ranked video list. The ResNet-101 configuration achieves a similar number of queries (21,611) with AP higher than 90%, but also obtains a very low AP, less than 10%, for 451 queries (2% of the total), which is the main cause for the overall lower MAP. This is also evident in Figure 4.13c.



(a) Conf. 51 (VGGFace2)



(b) Conf. 52 (ArcFace – RN-50)

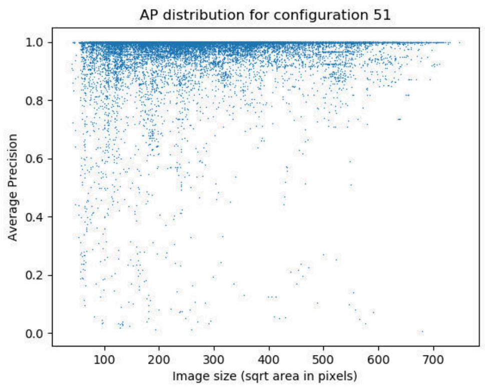


(c) Conf. 53 (ArcFace – RN-101)

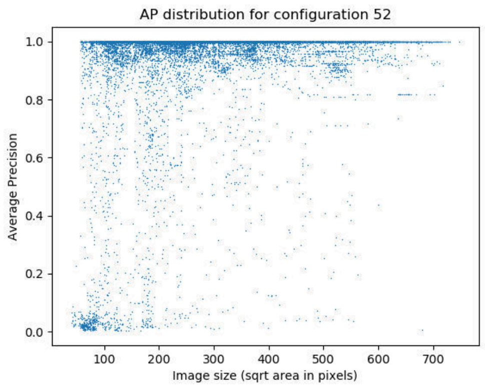
Figure 4.13: Distribution of the AP for the three tested configurations on the test set. The vast majority of queries obtained an AP $>$ 90%. The pipeline configurations with ArcFace still produce some queries with AP $<$ 10%.

In order to explore the performance difference between VGGFace2 and ArcFace on the test set, I decided to plot the AP of all the queries against the query image size, expressed as the square root of the image area (in pixels). The plots are shown in Figure 4.14. We can immediately see that differently from VGGFace2, ArcFace produces a substantial number of low-scores for low-resolution images (notice the clump of points in the bottom left corner of Figures 4.14b and 4.14c).

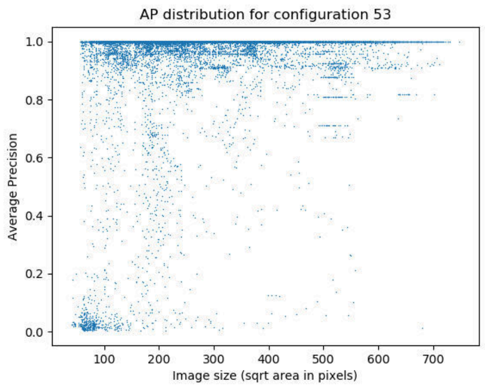
In order to understand which queries posed a problem for ArcFace, I



(a) Conf. 51 (VGGFace2)



(b) Conf. 52 (ArcFace – RN-50)



(c) Conf. 53 (ArcFace – RN-101)

Figure 4.14: AP vs. query image size for the three tested configurations on the test set. Note the group of low-score queries for low-resolution images in the bottom left corner of the ArcFace plots. This is not present in the VGGFace2 case.

sampled 16 random low-resolution face images among the ones that obtained AP lower than 10% with ArcFace (in configuration 53). I show those in Figure 4.15. While the resolution is low, the faces in these images still appear to be recognizable, not only by a human, but also by VGGFace2, which obtains an average AP of 75.5% on those same 16 images.

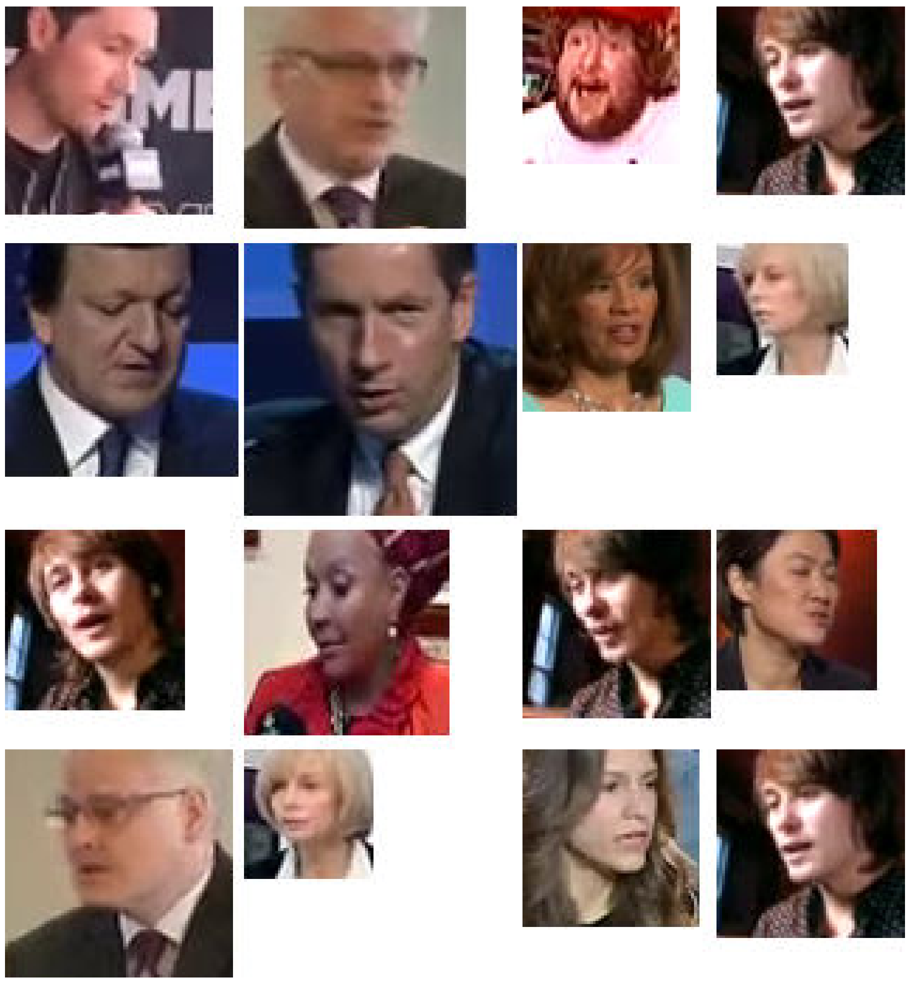


Figure 4.15: 16 randomly sampled low-resolution query image faces which gave AP $<$ 10% with ArcFace with ResNet-101. Images are shown at their original resolution and have not been resized.

As we have seen in Section 4.3.4.1, ArcFace only works well if the query image is properly aligned. For this reason I decided to run RetinaFace on those 16 images: the network was not able to detect any face in them, except for one. Without landmarks, these faces were not aligned in the query process and thus ArcFace returned unreliable feature vectors. The only image where

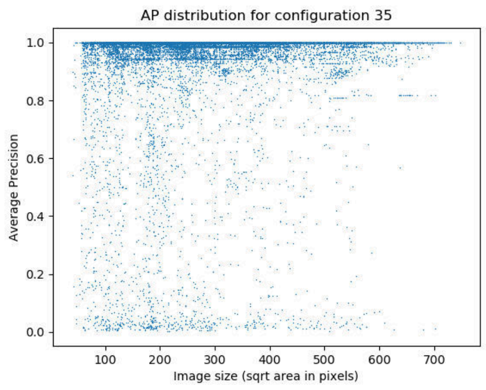
RetinaFace correctly detected a face (second row, second column in Figure 4.15) is actually another annotation error that bypassed the cleaning step. In fact, VGGFace2 also obtains a low score of 3.5% AP on it, as expected. Running RetinaFace on the 319 faces with low-resolution and low-score, I discovered that the CNN failed to detect a face in 309 of those. This strongly suggests that the bad performance of ArcFace on these images was caused by a face detection failure, and thus missing face alignment. This also highlights a weakness of ArcFace: VGGFace2 does not need face alignment and does not suffer from face detection errors like ArcFace. Besides, not requiring a face detection step at query phase also speeds up computation, making this a further reason to choose VGGFace2 over ArcFace in a FBVR system.

In order to check if changing the face detector could result in less errors and thus a better performance for ArcFace on the test set, I also ran the pipeline with MTCNN instead of RetinaFace (configurations 35 and 37 for ArcFace with ResNet-50 and 101 respectively). Results are shown in Table 4.14. Similarly to the results on the validation set, using MTCNN also has a negative effect on ArcFace, as opposed to using RetinaFace. But differently from RetinaFace, the mistakes in this case were not specific to low-resolution faces. Figure 4.16 shows the distribution of query AP with respect to image size. No clump of points in the lower left corner is present like the one in Figures 4.14b and 4.14c.

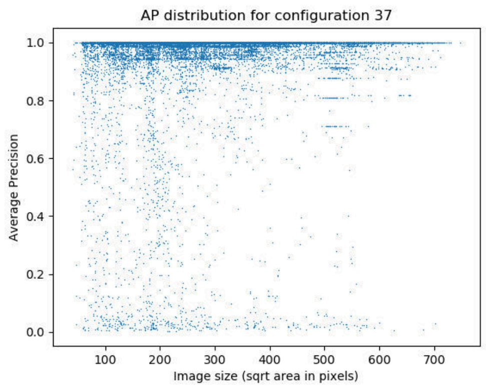
ID	Feature extraction net	MAP (val.)	MAP (test)
35	ArcFace (ResNet-50)	92.74	94.61
37	ArcFace (ResNet-101)	94.59	95.79

Table 4.14: MAP (percentage) on the test set when using ArcFace with the MTCNN face detector. These configurations were previously described in Table 4.7. Validation MAP is also shown for reference. Results were worse than using RetinaFace, like on the validation set.

Examining the mistakes, I found that also in this case many were due to face alignment errors. While, differently from RetinaFace, MTCNN was almost



(a) Conf. 35 (ArcFace – RN-50 with MTCNN)



(b) Conf. 37 (ArcFace – RN-101 with MTCNN)

Figure 4.16: AP vs. query image size for ArcFace with MTCNN on the test set. Differently from the distributions shown in Figures 4.14b and 4.14b, low-score queries were not concentrated in the low-resolution images.

always able to detect each face, the landmarks were instead not predicted correctly, especially in extreme poses, leading to bad face alignment. This probably caused ArcFace to compute unreliable feature vectors, resulting in bad results for the related queries. Figure 4.17 shows examples of landmark prediction failures among the query faces that obtained AP $<$ 10% for configuration 37 (ArcFace with ResNet-101).

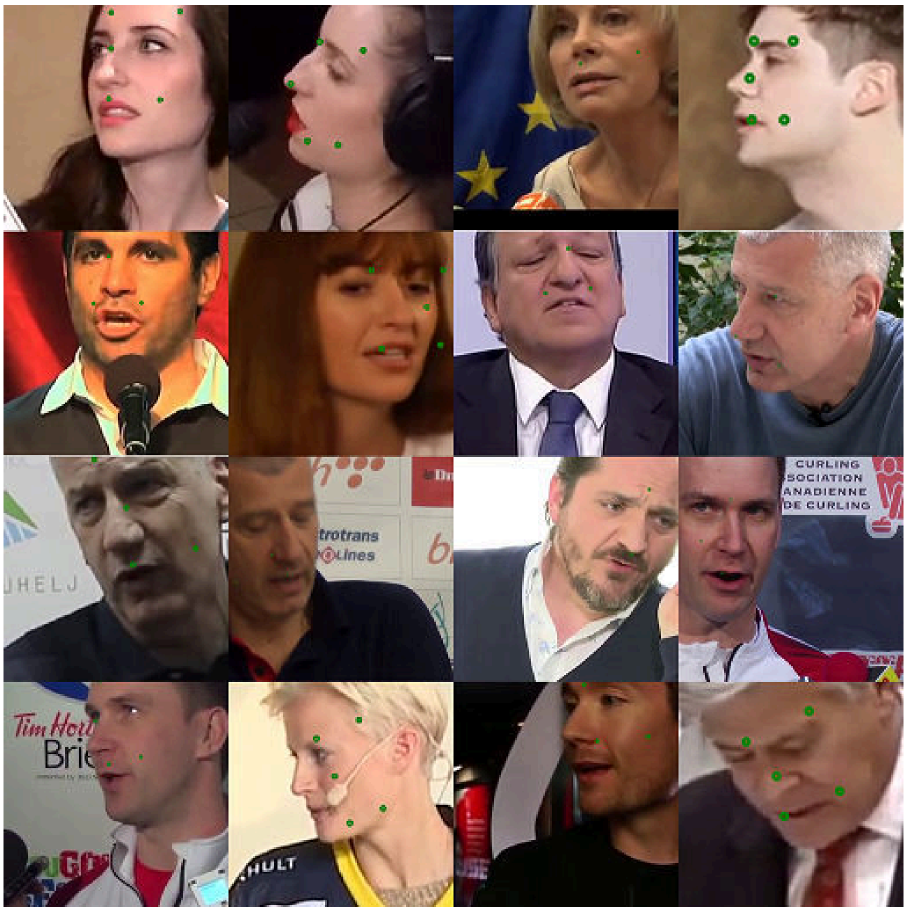


Figure 4.17: 16 examples of landmark prediction failures by MTCNN than resulted in low AP for ArcFace with ResNet-101 (configuration 37). Images have been resized to be 256 pixels wide. Zoom in to see the green circles representing the five facial landmarks (eyes, nose tip, mouth corners) predicted by MTCNN. Some faces have less than five landmarks because the predicted coordinates for some of them were outside of the image boundaries.

For completeness, I also decided to check the 16 worst-performing query images for VGGFace2, irrespective of their resolution (since for VGGFace2 there is no concentration of failures for low-resolution images like for ArcFace). They are shown in Figure 4.18. Almost every face shown is the result of an annotation error. The first five images are wrong boxes, where either the box does not surround the face correctly, or there is not even a face in the image. The 6th and 7th images (second row, columns 2 and 3) contain the wrong person, the interviewer instead of José Manuel Barroso. The next 3 images are more annotation errors and do not contain any face. The face in row 3, column 3 is the only correct face, despite containing some artifacts at the bottom which might have influenced the result. Moreover, the video in question has very low resolution, and the person (Piedad Córdoba) is not easily recognized. Then we have 2 more wrong boxes, and then the final 3 images also contain the wrong person: the interviewer instead of Élisabeth Guigou.

Another interesting question is whether there are some specific identities that are harder to recognize. To verify that, I computed the average AP for all the 74 identities in the test set, for configurations 51, 52 and 53. The results are shown in Figure 4.19. Most of the identities have a mean AP higher than 90%, and only one (for VGGFace2) or two (for ArcFace) have a MAP lower than 70%. The worst-recognized identity for VGGFace2 contains many low resolution query images and the identity sometimes wears sunglasses; moreover, there seems to be some notable age difference between the various videos of this person, which might contribute to his lower score. Example faces from this identity are shown in Figure 4.20a. This is also one of the two worst identities for ArcFace; the other one is shown instead in Figure 4.20b. We can see that he often wears sunglasses, which might again be a factor of its worse performance.

In conclusion, we can see that the pipeline performs reliably even on unseen identities, with MAP reaching over 97% for the best performing configuration. Together with the relative time efficiency, discussed in Section 4.3.4.2, the pipeline performed well enough to be integrated into a prototype version of TVBridge. The TVBridge integration will be briefly discussed in

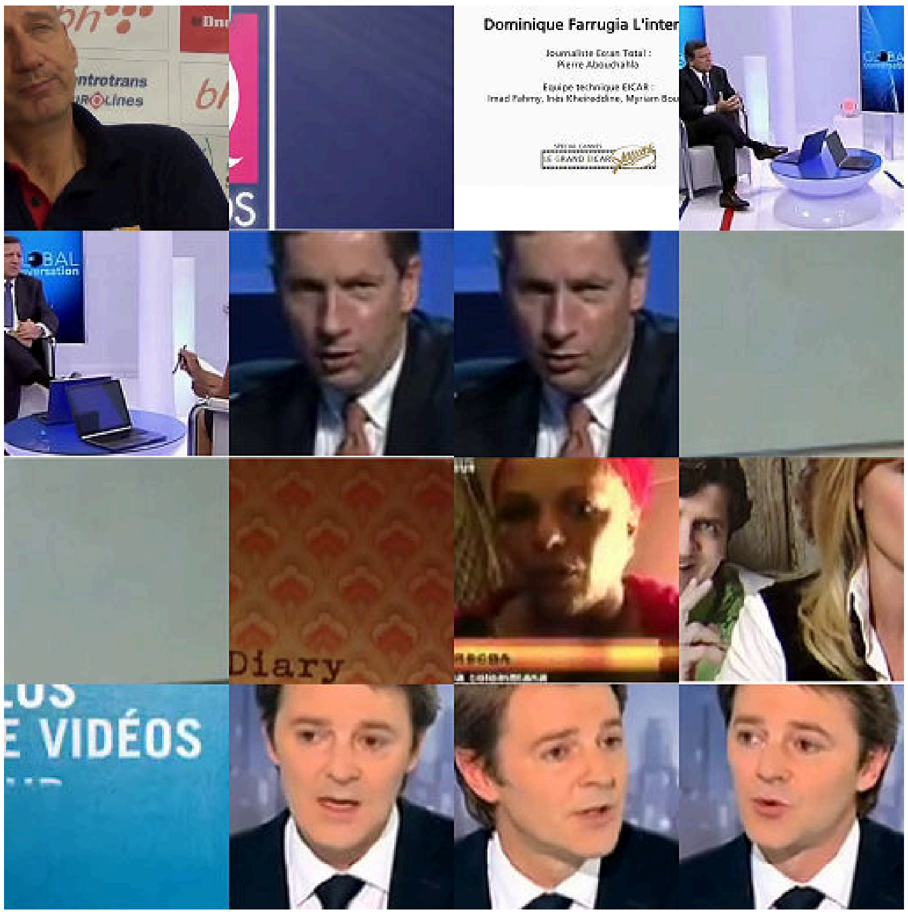
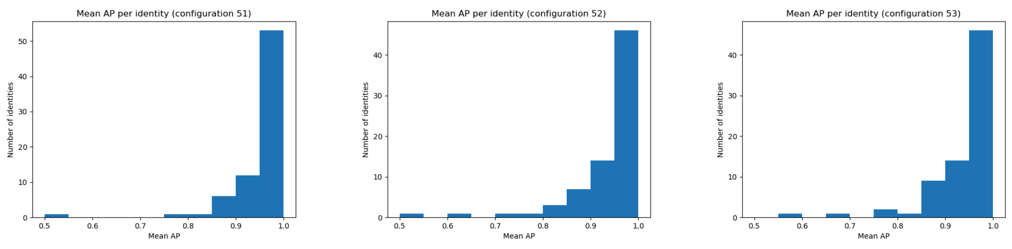
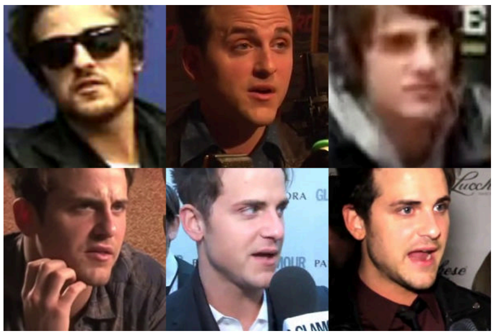


Figure 4.18: The 16 query image faces which gave the lowest AP for VGGFace2. Images have been resized to be 256 pixels wide. We can see that most of the images are annotation errors. More details are provided in the main text.



(a) Conf. 51 (VGGFace2) (b) Conf. 52 (ArcFace – RN-50) (c) Conf. 53 (ArcFace – RN-101)

Figure 4.19: Distribution of the average per-identity AP for the three configurations tested on the test set. Only one or two identities produce a MAP lower than 70%.


(a)

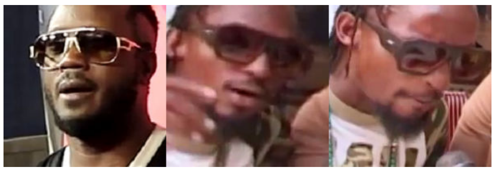

(b)

Figure 4.20: Sample images from the two worst performing identities in the test set. VGGFace2 obtains less than 70% MAP only on identity (a), while ArcFace (both ResNet-50 and ResNet-101 versions) on both identities (a) and (b).

the next section.

4.4 TVBridge implementation

As discussed at the start of the chapter, TVBridge is a platform that allows a broadcaster to enrich TV programs with second-screen content. Each unit of multimedia content is called a *bridget*. TVBridge contains various integrated components, one of which is the authoring tool (AT), which is the one in which I integrated the face-based video retrieval pipeline.

The pipeline was hosted on an Amazon AWS machine equipped with an NVIDIA T4 Tensor Core GPU with 16 GB of GPU memory. I implemented a Python RESTful web service which accepts requests to add and delete videos from the face feature database, and to perform queries on it. Since the TVBridge AT is user-based, a separate face feature database is constructed for each user, in order to avoid cross-searches. I used the dramatiq Python library [332] to manage a set of parallel workers to process the feature extraction and query requests, which can be done concurrently. A subset of the workers is exclusively dedicated to database querying, in order to avoid a situation in which all the workers are busy with long-running extraction processes and leave the user waiting for a long time for the query results.

Figure 4.21 exemplifies the system architecture when a user uploads a new video to the TVBridge AT and when the user performs a query. When the user uploads a new video (1) to the TVBridge AT, the AT asks the FBVR service to extract the features from it and sends it the new video (2). The request is put in a queue of tasks and when a worker assigned to the feature extraction process is ready, it processes the request and saves the features to the database related to the user that uploaded the video (3). When the user wants to search for videos containing a specific face, he uses the AT interface to select the face from a frame of the TV program he is working on (1), the AT asks the FBVR service to perform a query using that face, attached to the request (2). The request is put in a queue of tasks and when a worker assigned to the querying process is ready, it runs the query (3) and returns the result to the AT[20] (4), which presents it to the user (5). The results are thresholded (I set the matching score threshold to 0.6). In order to help the user to quickly locate the person of interest in the result videos, the list of shots is also returned to the AT, which presents the user with the option to quickly jump to each of those shots.

The prototype interface is shown in Figure 4.22. First, the user selects *Search by face* in the video selection box while editing a bridget (Figure 4.22a). The interface shows the currently selected frame and allows the user to manually draw a rectangle around the face of interest (Figure 4.22b). The user clicks *Search* and in a few tenths of a second the AT shows the list of videos that match the query (Figure 4.22c). The user can click the thumbnail to watch the video and he can click on one of the shots listed on screen to quickly jump to a shot containing the person of interest (Figure 4.22d). The user can then examine each video and easily choose the one to include in the bridget.

[20] In the prototype I have implemented, the AT polls the FBVR service in order to check the query status (which can be queued, in process, successful, or failed), and, once the query is successful, performs a separate HTTP request to fetch the results.

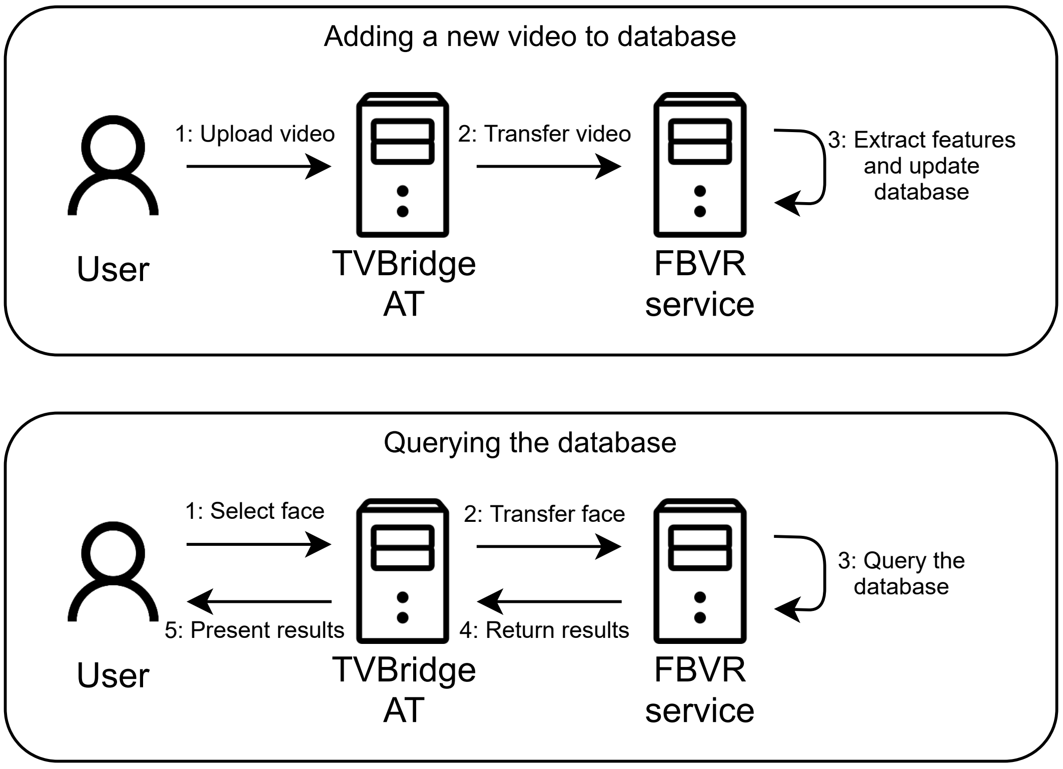
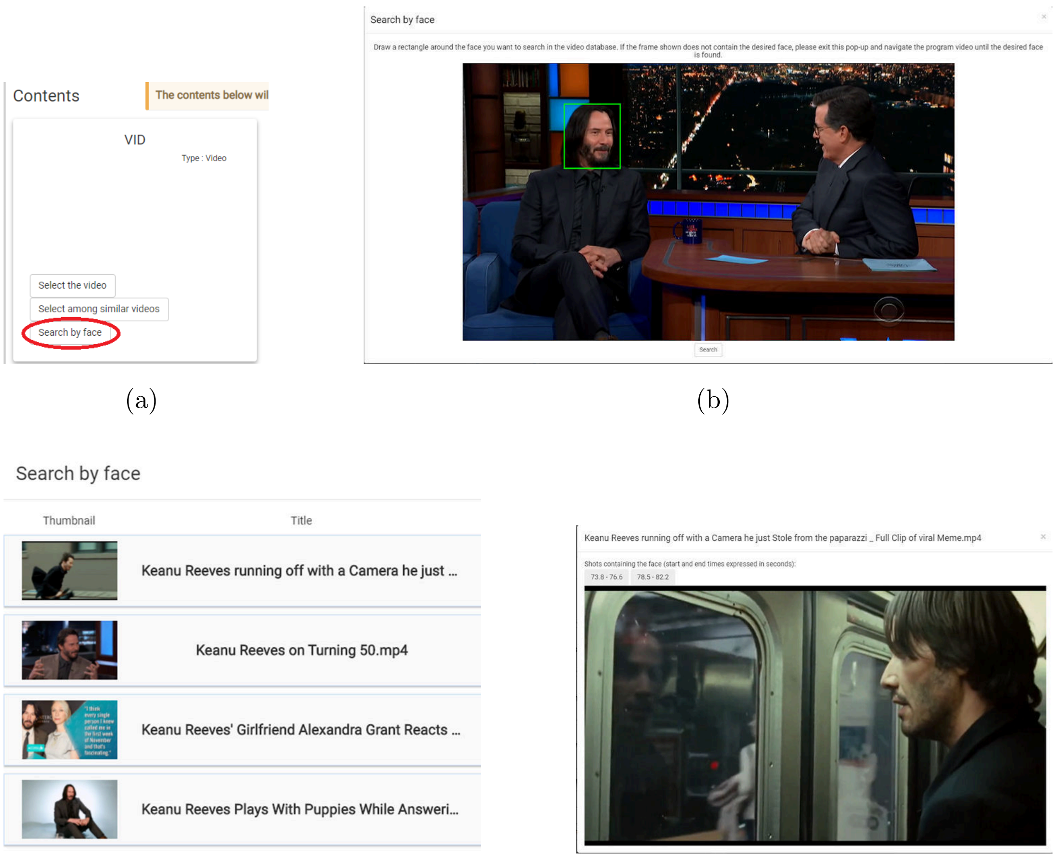


Figure 4.21: Integration of the FBVR pipeline into the TVBridge system. The upper section shows the video upload process, while the lower section shows the query process.



Draw a rectangle around the face you want to search in the video database. If the frame shown does not contain the desired face, please exit this pop-up and navigate the program video until the desired face is found.

Thumbnail	Title
	Keanu Reeves running off with a Camera he just ...
	Keanu Reeves on Turning 50.mp4
	Keanu Reeves' Girlfriend Alexandra Grant Reacts ...
	Keanu Reeves Plays With Puppies While Answeri...

Keanu Reeves running off with a Camera he just Stole from the paparazzi _ Full Clip of viral Meme.mp4

Shots containing the face (start and end times expressed in seconds):

73.8 - 76.6   78.5 - 82.2

(a) (b) (c) (d)

Figure 4.22: Interface to run a FBVR query in TVBridge. More details are provided in the main text.

4.5 Conclusion and future directions

I presented a novel pipeline for the task of face-based video retrieval on large-scale unconstrained multi-shot video datasets, with an application to a real-world commercial system, TVBridge. The pipeline includes shot detection, face detection, tracking and recognition, and feature aggregation, and presents a query protocol to perform video retrieval. I also built a large-scale video dataset for the proper evaluation of face-based video retrieval, derived from the VoxCeleb2 dataset, since existing datasets were not large and varied enough, or were not appropriate for an end-to-end evaluation of the proposed pipeline. I also briefly described how the algorithm was integrated into TVBridge.

The pipeline was able to efficiently analyze challenging videos and perform fast queries on databases containing features extracted from thousands of videos. It reached a Mean Average Precision of 97.25% over more than 23,000 queries on videos with unseen identities. I also compared a variety of approaches and network models for each step of the pipeline, and described the pros and cons of each model, highlighting the tradeoffs in terms of accuracy and time/memory efficiency, especially in the context of an integration in TVBridge.

However, the pipeline still has room for improvement. For example, one possible future line of research is the investigation of an efficient implementation of feature quantization, which can be useful for systems with a large number of users, for which space usage optimization is fundamental, and query execution time is even more important. Another unexplored route is the use of hashing strategies in order to reduce the query time complexity from linear (in the number of database feature vectors) to constant (on average); as we have seen, many hashing strategies exist in the literature, each of which may work differently depending of the type of features and the network used to extract them.

In addition to that, different tracking algorithms could be explored in order to help the feature aggregation process to produce more stable descriptors. Different aggregation strategies might also be investigated, by changing both the clustering algorithm and the feature merge process itself, with a smarter weighting of the feature vectors in the averaging process, in order to exclude outliers. A fast online clustering algorithm might also be investigated, with the aim to implement an efficient inter-video clustering process, that can help with accuracy in some cases, as we have seen.

Chapter 5

Conclusion

Deep learning has seen increasing popularity in the analysis of large amounts of complex data. Recently, it has been employed in many tasks related to video analysis. In particular, this thesis focused on the use of deep learning for multiple object tracking and face-based video retrieval.

Regarding MOT, I presented an extensive survey of the various trackers in the literature that employed deep learning models in their pipeline. I identified four steps that are shared by the majority of MOT algorithms: object detection, feature extraction and motion prediction, affinity/distance computation, and association. Each deep learning technique was then categorized based on these four steps, describing the different algorithms that employed it successfully. I also collected and presented the experimental results obtained by many of the described algorithms on the three main MOTChallenge datasets, discussing advantages and disadvantages of the most used techniques and identifying promising future lines of research.

For the task of face-based video retrieval I presented a novel pipeline for the retrieval on unconstrained, multi-shot videos in large-scale video datasets. The pipeline obtained 97.25% MAP on more than 1,000 unseen videos. Since, to the best of my knowledge, no existing dataset in the literature was appropriate for an end-to-end evaluation of the proposed pipeline in the context of television/media videos, I built an evaluation dataset, starting from the existing VoxCeleb2, that contained more than 5,000 videos with variable length, resolution, quality, number of shots and of people. I compared a variety of strategies and models for each of the main steps of the pipeline, such as shot detection, face detection, face feature extraction and aggregation. I also described how the pipeline was integrated into a prototype version of TVBridge, in order to aid the broadcaster editor in the process of searching for videos to include in second-screen content (i.e. the bridgets). Finally, I also presented some possible future directions of research to improve the pipeline performance, both in terms of accuracy and time/memory efficiency.

Appendices

Appendix A

Experiments on a customized MOT tracker

Following the findings of the survey presented in Chapter 3, I performed some preliminary experiments on MOT algorithms, in collaboration with the aforementioned team from the Universidad de Granada. The goal was to try to improve the performance of current MOT algorithms by accounting for various key observations described in Section 3.7. In particular, we wanted to solve the excessive reliance of MOT trackers on the quality of detection. Given the successes of SOT-based MOT trackers, we decided to apply a series of changes to one of them, specifically the DMAN tracker [137]:

- the use of an improved SOT tracker;
- the use of bidirectional tracking (the algorithm would not run in an online setting anymore, but in batch mode);
- the use of higher-quality detections.

As we will see, however, the experiments did not produce the expected results. For this reason, I did not include this section in the main text of the thesis, and just summarize the main experiments and results here.

A.1 DMAN with different SOT algorithms

We performed the first experiments on the MOT16 training set. The reason why we did not start by using the test set was that in order to evaluate an algorithm on the test set, its predictions have to be submitted on the MOTChallenge website. However, MOTChallenge limits the frequency of sumbissions in order to avoid fine-tuning models on the test set.

We chose to replace the ECO tracker [84] from the original DMAN algorithm with two possible SOT trackers: ATOM [333] and DIMP [334], the latter both in the ResNet-18 and ResNet-50 versions (which I will refer to as DIMP18 and DIMP50).

Testing the original DMAN with ECO on the private detections provided by [64] resulted in a MOTA of 54.3%. Replacing ECO with ATOM, DIMP18 and DIMP50 resulted in 46.8%, 46.1% and 47.9% MOTA, respectively.

We decided to add a localization correction strategy by adjusting the predicted location of a tracklet using the detections, whenever the IOU between the predicted location and a detection was higher than a threshold and the detection had a specific aspect ratio (to avoid considering false detections). If no matching detection was found, the track status was not changed. This resulted in an improvement over the original results, obtaining 53.1% MOTA with ATOM, 52.6% with DIMP18 and 53.1% with DIMP50.

A further improvement was made by re-initializing the tracker status every time a track was recovered after occlusion, in order to remove older appearance and motion information. This further improved the performance, resulting in 54.0% MOTA for ATOM, 53.7% for DIMP18 and 53.9% for DIMP50. However, these results were still worse than the 54.3% obtained by the original DMAN.

A.2 Bidirectional tracking experiments

We implemented a batch version of the algorithm that performed tracking in both directions. We hoped that integrating bi-directional information could help reduce the number of false negatives.

The algorithm was run first in the normal "forward" direction, and then backwards, i.e. by feeding it the video frames in reverse temporal order, starting from the end of the video. After the two runs, the results were merged to obtain the final tracks.

We implemented various association strategies, including a one-step "naive" association, performed using the Hungarian algorithm with IOU and box size ratio used as cost functions; a multi-step "naive" association, where multiple rounds of tracklet merging were performed; and an appearance-based association, where the DMAN was used to compute affinities between the tracks. We also used a hill climbing strategy in order to choose the best hyperparameters for the merging step, including IOU thresholds, track overlap thresholds, the use of a short track filter, the weight of IOU-based and appearance-based affinities in the cost matrix for the Hungarian algorithm, and so on.

In addition to testing the algorithm on the MOT16 training set, we also tested it on a custom subset of the training set of MOT15, which excluded videos that were used to train the DMAN. Moreover, we also performed tests using the lower-quality public detections included with both datasets. However, results were underwhelming. In general, false negatives were reduced, as expected, but this improvement was negated by a corresponding increase in false positives. The result was that the merged trajectories were either worse, or just marginally better than the unmerged ones. In particular, merged trajectories were only able to improve the MOTA score on the MOT16 training set. In order to test if the improvement was also reflected on the test set, we submitted results on the MOT16 test split to the MOTChallenge evaluation server. However, our modified algorithm obtained a MOTA score which was between 6 and 8% lower than the original DMAN algorithm. Tests on the subset of the MOT15 dataset also resulted in underwhelming results, with the merging strategies worsening the original result by no less than 3% MOTA.

Appendix B

Summary table of DL-based MOT algorithms

The following is a summary table for all the algorithms described in Section 3.5 for which the four described stages can be identified (i.e. excluding algorithms from Section 3.5.5). The papers are listed by roughly following the order in which they were presented in the main text (note that some papers appear in multiple sections). Besides describing the algorithms used for each of the four steps, a link to the source code or any other useful data is also provided, where available. This table was also published in [36].

	Detection	Feature extr. / mot. pred.	Affinity / cost computation	Association / Tracking	Mode	Source and data
[61]	Faster R-CNN	Kalman filter	IoU	Hungarian algorithm	O	Source
[64]	Modified Faster R-CNN	Modified GoogLeNet, Kalman filter	Cosine distance + IoU	Hungarian algorithm (online), modified H2T [111] (batch)	O+B	Detections and appearance features
[81]	Faster R-CNN	CNN (app.), AlphaPose CNN, pose joints velocities, interaction grid	Pose-based Triple Stream Network (LSTM-based)	Custom algorithm	O	
[82]	Faster R-CNN	CNN	Euclidean distance, cosine distance	Multifeature fusion re-tracking algorithm	B	
[83]	CNN	HOG + Colour Names	Variation of Discriminative Correlation Filter	Custom algorithm + Hungarian algorithm	O	
[90]	SSD	SSD, LSTM	Cosine similarity	Hungarian algorithm	O	
[87]	SSD	SSD	RNN	Hungarian algorithm, MLP (track scores)	O	
[88]	SSD	SSD + Correlation Filter	IoU + APCE	Hungarian algorithm	O	
[80]	Public / Mask R-CNN	Siamese Mask R-CNN	App. affinity, mot. consistency, spatial structural potential	Tensor-based high-order graph matching	O	Code will be released

	Detection	Feature extr. / mot. pred.	Affinity / cost computation	Association / Tracking	Mode	Source and data
[91]	YOLOv2	Tiny Yolo, Particle filter, Random Ferns, KLT	Pairwise overlap ratio, student Random Ferns, Euclidean distance	Greedy bipartite assignment	O	
[92]	RRC or SubCNN	Feature-based odometry, Pose Adjustment CNN, stacked-hourglass CNN	3D-2D cost + 3D-3D cost + appearance, shape and pose costs	Hungarian algorithm	O	Source
[95]	DPM or Tiny (CNN)	DPM or Tiny (CNN)	Implicit in Reverse Nearest Neighbour	Reverse Nearest Neighbour Matching	O	Code will be released
[97]	ViBe + SVM + CNN		IoU	Region Matching algorithm	O	
[101]	Multi-task Network Cascades (CNN)	Optical flow	Overlap of segmentation instances	Hungarian algorithm	O	
[106]	Dalal-Triggs detector	Autoencoders	SVM	Minimum spanning tree	O	
[108]	Public	CNN + PCA	Multi-Output Regularized Least Squares	Variation of Multiple Hypothesis Tracking	O	Source

	Detection	Feature extr. / mot. pred.	Affinity / cost computation	Association / Tracking	Mode	Source and data
[117]	Public	CNN, Kalman Filter	Multi-Output Regularized Least Squares + Kalman Filter + detection-scene score	Maximum Weighted Independent Set	B	
[118]	Public	R-CNN	Observation cost + transition cost + birth-death cost	Min-cost multi commodity flow problem, solved with Dantzig-Wolfe decomposition	O	
[119]	DoH [335]	CNN	CNN + Kalman filter	Custom algorithm, SVM	B	
[69]	From [64]	Kalman filter, Wide Residual Net	Mahalanobis dist. (mot.) + cosine distance (app.), IoU	Hungarian algorithm	O	Source
[70]	From [64]	CNN	appearance + motion + dynamic affinity	Hungarian algorithm	O	
[126]	Public	CNN	Bilinear LSTM	Variant of MHT-DAM [108]	B	
[120]	Public / SDP+RPN	CNN	Appearance + motion + shape affinities	Hungarian algorithm	O	Source

	Detection	Feature extr. / mot. pred.	Affinity / cost computation	Association / Tracking	Mode	Source and data
[122]	Public	GoogLeNet CNN	App. similarity	Bayesian inference using [336]	B	
[123]	Public / Faster R-CNN	GoogLeNet CNN	Recurrent Autoregressive Networks (GRU-based)	Bipartite graph matching	O	
[128]	Public	CNN	Hybrid Likelihood Function (Discriminative Correlation Filter + Gaussian Mixture Probability Hypothesis Density)	Hungarian algorithm	O	
[124]	Public	CNN	app. + HSV histogram + motion similarities	Pairwise update algorithm + SSVM	B	Will be available at this link
[125]	Public	GoogLeNet CNN, Optical flow	Distance between app. features, common superpixels, optical flow predictions	Multiple Hypotheses Tracking	B	
[127]	Public	CNN	LSTM (app.) + motion affinity	Batch Multi-Hypothesis	B	

	Detection	Feature extr. / mot. pred.	Affinity / cost computation	Association / Tracking	Mode	Source and data
[79]	Public / From [64]	DeepCut CNN [113], StackNetPose CNN	StackNetPose CNN	Lifted multicut problem, solved as in [163]	B	Source
[131]	Public	Siamese CNN	Euclidean distance (app. feat.) + IoU + box area ratio	Custom greedy algorithm	O	
[132]	DPM	Siamese CNN with temporal constraints	Mahalanobis distance (app. feat.) + motion affinity	Generalized Linear Assignment solved with Softassign [337], Dual-threshold strategy [338]	B	
[133]	HeadHunter [339]	CNN	Euclidean distance (app. feat.), temporal and kinematic affinities	Hungarian algorithm, Agglomerative clustering	B	Source
[134]	Public	Siamese CNN, contextual features	Gradient Boosting	Linear programming	B	
[136]	Public	CNN, sequence-specific statistics, optical flow, FC layers	FC layer combining app. and mot. distances	Minimax label propagation	B	
[142]	Public	CNN + various app. and non-app. feat.	embedding layer + bidirectional LSTM	Variation of Multiple Hypothesis Tracking	B	

	Detection	Feature extr. / mot. pred.	Affinity / cost computation	Association / Tracking	Mode	Source and data
[137]	Public	Linear motion model, Spatial Attention Network CNN	Temporal Attention Network (bidirectional LSTM)	Custom greedy algorithm, ECO (SOT tracker)	O	Source
[144]	Public	Siamese CNN, LSTM, WRN CNN, Siamese Bi-GRU + CNN	Euclidean dist. (app. feat.), spatial distance, GRU feature matching	Hungarian algorithm, bi-GRU RNN (track split), custom algorithm	B	
[138]	Public	DCCRF, visual-displacement CNN	Visual-similarity CNN, IoU	Hungarian algorithm	O	
[139]	Public	R-FCN + Kalman Filter, GoogleNet	Eucl. dist. (app. feat.), IoU	Hierarchical Data Association	O	Source
[140]	Public	Feature Pyramid Siamese Network, motion features	Feature Pyramid Siamese Network	Custom greedy algorithm	O	
[145]	Public	Kalman Filter, GoogLeNet	Distance between sparse coding of features using a learned dictionary	Hungarian algorithm	B	

	Detection	Feature extr. / mot. pred.	Affinity / cost computation	Association / Tracking	Mode	Source and data
[147]	Public	3 LSTMs (app., mot., interaction features) using CNN, bb velocity, occupancy map	LSTM	Hungarian algorithm, SOT tracker [155]	O	
[148]	Public	Linear motion model, CNN	CNN	Association to highest classification score	O	
[115]	Manually generated	Hidden Markov Models, CNN	Mutual information (app. feat.)	Dynamic programming algorithm from [336]	B	
[149]	Public	LK optical flow, Convolutional Correlation Filter CNN, Kalman filter	Optical flow aff., app. feat. aff., IoU, scale affinity, distance between detections	Custom algorithm (with Hungarian alg.)	O	Source
[152]	Public	Kalman filter + Deep RL agent	IoU	Hungarian algorithm + Deep RL agent	O	
[153]	N/A	LSTM (mot.)	Stitching score using IoU	Custom iterative tracklet-stitching algorithm	B	
[154]	Public	RNN (mot.)	LSTM	RNN	O	Source
[156]	Public	2 LSTMs, VGG16 CNN	SVM, Siamese LSTM	Greedy association	B	

	Detection	Feature extr. / mot. pred.	Affinity / cost computation	Association / Tracking	Mode	Source and data
[71]	From [64]	Kalman filter or LK optical flow, CNN + motion features	IoU, Siamese LSTM	Hungarian algorithm	B	
[157]	Public		FC layers + Bi-directional LSTM	Hungarian algorithm	O	
[165]	Public / from [166] (combines DPM, SDP and ACF)	Modified Faster R-CNN	Modified Faster R-CNN	Particle filter	O	
[167]	Public	DeepMatching, Siamese CNN	Edge potential as in [168], Siamese CNN	Lifted multicut	B	
[169]	Public	CNN (motion pred.), part of MDNet (CNN)	N/A	Deep RL agents	O	

Table B.1: Summary of the methods described in section 3.5. In each column, the approach for each paper in that step is shown. *app.* means appearance, *mot.* means motion, *feat.* means features, *pred.* means prediction; *O* and *B* in the Mode column indicate Online and Batch methods respectively. Text in the last column is clickable and contains links to the specified data.

Appendix C

TransNet experiments

In this appendix I will briefly describe two strategies I tested in order to improve the shot detector performance and to check if it might have a bigger impact on the overall pipeline MAP score.

First, I used the ClipShots dataset [316] in order to fine-tune TransNet on this newer, bigger-scale shot detection dataset. Second, I tried implementing the use of a double thresholding strategy to better differentiate between cut and gradual transitions and see if that could improve the performance. The next two sections will briefly describe the two experiments. A discussion of the results in the context of the main pipeline can be found in section 4.3.

C.1 Fine-tuning on ClipShots

ClipShots [316] is a new large-scale dataset specifically designed for training and testing shot detector algorithms. It contains 128,636 cut transitions and 38,120 gradual transitions from 4039 videos from YouTube and Weibo. 3539 videos are part of the training set, while the remaining 500 are used as test set. I used 75% of the training videos as actual training and the remaining 25% as validation videos, in order to choose the best model epoch and threshold σ, used to declare a frame as being part of a transition or not.

As evaluation metric, I used the F_1 score, following the literature. It is computed as

$$2 \cdot \frac{p \cdot r}{p + r},$$

where p and r are the precision and recall, respectively, computed in the usual way. Like previous works [316, 311], a true positive transition is one which overlaps with a ground truth transition, a false positive transition is one which does not overlap with any ground truth transition or overlaps only with a ground truth transition that was already matched to another predicted one; finally, a false negative transition is a ground truth transition with no overlapping predicted transition.

The original model, trained by the authors on TRECVID videos with some synthetic transitions, obtains 73.41% F_1 score on the ClipShots test set.

In the first training experiment I used the same exact hyperparameters and training strategy described in the original TransNet paper [317], and I trained the model for 20 epochs, considering that ClipShots is a much bigger dataset than the one used in the original TransNet paper, and that we are fine-tuning the network, not training from scratch; however, while the best combination (6th epoch weights, $\sigma = 0.1$) obtained a good 86.23% F_1 score on the validation set, this result did not translate well on the test set, with only 68.53% F_1 score, a worse result than the original model. Freezing the first layers of the pretrained weights led to even worse results, so I fine-tuned the entire network in the following experiments.

The authors of TransNet trained the original network by setting as positive only the middle frame of each transition. I experimented with changing that by setting as positive every frame belonging to a ground truth transition. This seemed to lead to better results: stopping fine-tuning at epoch 5 and using $\sigma = 0.6$ (computed on the validation set), the network obtained 74.61% F_1 score on the test set, an improvement of 1.2% over the original model. This is the *TransNet (ClipShots)* model included in Table 4.3 in Section 4.3.

C.2 Double thresholding experiments

Frames belonging to gradual transitions tend to have a smaller difference between them. For this reason, in order to correctly detect gradual transitions, the threshold σ must be kept relatively lower, so as to capture these smaller differences. However, this introduces some false positive cut transitions, when some isolated frames overcome the low threshold. To solve that, I decided to implement a double thresholding strategy, according to which cut transitions are predicted only if above a higher threshold σ_{high}, while gradual transition frames are detected only when there are at least l_{min} consecutive frames with a predicted transition score above σ_{low}.

I ran a grid search over σ_{low}, σ_{high}, l_{min} and the number of training epochs, where both σ_{low} and σ_{high} were chosen from the set $\{0.01, 0.05, 0.1, 0.15, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$, with the constraint that $\sigma_{low} < \sigma_{high}$, and l_{min} was chosen from the set $\{3, 5, 10, 20\}$.

The grid search on the fine-tuned weights returned a F_1 score of 74.13% on the ClipShots test set, using the best combination of hyperparameters obtained on the validation set: $\sigma_{low} = 0.2$, $\sigma_{high} = 0.5$, $l_{min} = 10$ and the weights obtained after the 5th epoch of fine-tuning. This is a worse result than using the standard single-value threshold σ, which gives 74.61% F_1, as seen in the previous section.

Running the grid search using the original weights led instead to an improvement. The best hyperparameters turned out to be $\sigma_{low} = 0.01$, $\sigma_{high} = 0.1$ and $l_{min} = 3$ (note that the original threshold used by the TransNet authors was $\sigma = 0.1$). The F_1 score on the test set was 73.71%, a slight improvement of 0.3% over the original version, which obtained 73.41%. This is the *TransNet (double thr.)* model mentioned in Table 4.3 in Section 4.3.

References

[1] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[2] Kaiming He et al. "Mask r-cnn. corr abs/1703.06870 (2017)". In: *arXiv preprint arXiv:1703.06870* (2017).

[3] Karen Simonyan and Andrew Zisserman. "Two-stream convolutional networks for action recognition in videos". In: *Advances in neural information processing systems* 27 (2014), pp. 568–576.

[4] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems* 27 (2014), pp. 2672–2680.

[5] Tom B Brown et al. "Language models are few-shot learners". In: *arXiv preprint arXiv:2005.14165* (2020).

[6] Yishuang Ning et al. "A review of deep learning based speech synthesis". In: *Applied Sciences* 9.19 (2019), p. 4050.

[7] David Silver et al. "Mastering chess and shogi by self-play with a general reinforcement learning algorithm". In: *arXiv preprint arXiv:1712.01815* (2017).

[8] Mihalj Bakator and Dragica Radosav. "Deep learning and medical diagnosis: A review of literature". In: *Multimodal Technologies and Interaction* 2.3 (2018), p. 47.

[9] Ke Zhang et al. "Automated IT system failure prediction: A deep learning approach". In: *2016 IEEE International Conference on Big Data (Big Data)*. IEEE. 2016, pp. 1291–1300.

[10] Richard Evans and Jim Gao. "Deepmind AI reduces Google data centre cooling bill by 40%". In: *DeepMind blog* (2016).

[11] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve restricted boltzmann machines". In: *ICML*. 2010.

[12] Bing Xu et al. "Empirical evaluation of rectified activations in convolutional network". In: *arXiv preprint arXiv:1505.00853* (2015).

[13] Anish Shah et al. "Deep residual networks with exponential linear unit". In: *Proceedings of the Third International Symposium on Computer Vision and the Internet*. 2016, pp. 59–65.

[14] Ning Qian. "On the momentum term in gradient descent learning algorithms". In: *Neural networks* 12.1 (1999), pp. 145–151.

[15] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. *Neural networks for machine learning lecture 6a overview of mini-batch gradient descent*. http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf. Accessed: 2021-01-22.

[16] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization." In: *Journal of machine learning research* 12.7 (2011).

[17] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[18] Yann Le Cun et al. "Handwritten digit recognition: Applications of neural network chips and automatic learning". In: *IEEE Communications Magazine* 27.11 (1989), pp. 41–46.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Communications of the ACM* 60.6 (2017), pp. 84–90.

[20] Olga Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[21] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).

[22] Christian Szegedy et al. “Going deeper with convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[23] Christian Szegedy et al. “Rethinking the inception architecture for computer vision”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826.

[24] Christian Szegedy et al. “Inception-v4, inception-resnet and the impact of residual connections on learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 31. 1. 2017.

[25] Saining Xie et al. “Aggregated residual transformations for deep neural networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1492–1500.

[26] Sergey Zagoruyko and Nikos Komodakis. “Wide residual networks”. In: *arXiv preprint arXiv:1605.07146* (2016).

[27] Shaoqing Ren et al. “Faster r-cnn: Towards real-time object detection with region proposal networks”. In: *Advances in neural information processing systems*. 2015, pp. 91–99.

[28] Ross Girshick et al. “Rich feature hierarchies for accurate object detection and semantic segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.

[29] R Girshick. “Fast r-cnn. arXiv 2015”. In: *arXiv preprint arXiv:1504.08083* (2015).

[30] Tsung-Yi Lin et al. “Microsoft coco: Common objects in context”. In: *European conference on computer vision*. Springer. 2014, pp. 740–755.

[31] Tsung-Yi Lin et al. “Feature pyramid networks for object detection”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 2117–2125.

[32] Wei Liu et al. “SSD: Single Shot MultiBox Detector”. In: *European conference on computer vision*. Springer. 2016, pp. 21–37.

[33] Joseph Redmon et al. “You only look once: Unified, real-time object detection”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[34] Joseph Redmon and Ali Farhadi. “YOLO9000: better, faster, stronger”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 7263–7271.

[35] Joseph Redmon and Ali Farhadi. “Yolov3: An incremental improvement”. In: *arXiv preprint arXiv:1804.02767* (2018).

[36] Gioele Ciaparrone et al. “Deep learning in video multi-object tracking: A survey”. In: *Neurocomputing* 381 (2020), pp. 61–88.

[37] Wenhan Luo et al. “Multiple object tracking: A literature review”. In: *arXiv preprint arXiv:1409.7618* (2014).

[38] Massimo Camplani et al. “Multiple human tracking in RGB-depth data: a survey”. In: *IET computer vision* 11.4 (2016), pp. 265–285.

[39] Patrick Emami et al. “Machine learning methods for solving assignment problems in multi-target tracking”. In: *arXiv preprint arXiv:1802.06897* (2018).

[40] Laura Leal-Taixé et al. “Tracking the trackers: An analysis of the state of the art in multiple object tracking”. In: *arXiv preprint arXiv:1704.02781* (2017).

[41] Laura Leal-Taixé et al. “Motchallenge 2015: Towards a benchmark for multi-target tracking”. In: *arXiv preprint arXiv:1504.01942* (2015).

[42] Anton Milan et al. "MOT16: A benchmark for multi-object tracking". In: *arXiv preprint arXiv:1603.00831* (2016).

[43] Jifeng Dai et al. "R-fcn: Object detection via region-based fully convolutional networks". In: *Advances in neural information processing systems*. 2016, pp. 379–387.

[44] Bo Wu and Ram Nevatia. "Tracking of multiple, partially occluded humans based on static body part detection". In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. Vol. 1. IEEE. 2006, pp. 951–958.

[45] Keni Bernardin and Rainer Stiefelhagen. "Evaluating multiple object tracking performance: the CLEAR MOT metrics". In: *Journal on Image and Video Processing* 2008 (2008), p. 1.

[46] Ergys Ristani et al. "Performance measures and a data set for multi-target, multi-camera tracking". In: *European Conference on Computer Vision*. Springer. 2016, pp. 17–35.

[47] Rainer Stiefelhagen and John Garofolo. *Multimodal Technologies for Perception of Humans: First International Evaluation Workshop on Classification of Events, Activities and Relationships, CLEAR 2006, Southampton, UK, April 6-7, 2006, Revised Selected Papers*. Vol. 4122. Springer, 2007.

[48] Rainer Stiefelhagen, Rachel Bowers, and Jonathan Fiscus. *Multimodal Technologies for Perception of Humans: International Evaluation Workshops CLEAR 2007 and RT 2007, Baltimore, MD, USA, May 8-11, 2007, Revised Selected Papers*. Vol. 4625. Springer, 2008.

[49] Piotr Dollár et al. "Fast feature pyramids for object detection". In: *IEEE transactions on pattern analysis and machine intelligence* 36.8 (2014), pp. 1532–1545.

[50] Pedro F. Felzenszwalb et al. “Object detection with discriminatively trained part-based models”. In: *IEEE transactions on pattern analysis and machine intelligence* 32.9 (2009), pp. 1627–1645.

[51] Ross B. Girshick, Pedro F. Felzenszwalb, and David McAllester. *Discriminatively trained deformable part models, release 5.* http://people.cs.uchicago.edu/~rbg/latent-release5/. 2012.

[52] Fan Yang, Wongun Choi, and Yuanqing Lin. “Exploit all the layers: Fast and accurate cnn object detector with scale dependent pooling and cascaded rejection classifiers”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2129–2137.

[53] Patrick Dendorfer et al. “Mot20: A benchmark for multi object tracking in crowded scenes”. In: *arXiv preprint arXiv:2003.09003* (2020).

[54] Patrick Dendorfer et al. *CVPR19 Tracking and Detection Challenge: How crowded can it get?* 2019. arXiv: 1906.04567 [cs.CV].

[55] Andreas Geiger, Philip Lenz, and Raquel Urtasun. “Are we ready for autonomous driving? the kitti vision benchmark suite”. In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2012, pp. 3354–3361.

[56] Andreas Geiger et al. “Vision meets robotics: The KITTI dataset”. In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1231–1237.

[57] Xiaoyu Wang et al. “Regionlets for generic object detection”. In: *Proceedings of the IEEE international conference on computer vision*. 2013, pp. 17–24.

[58] James Ferryman and Ali Shahrokni. “Pets2009: Dataset and challenge”. In: *2009 Twelfth IEEE International Workshop on Performance Evaluation of Tracking and Surveillance*. IEEE. 2009, pp. 1–6.

[59] Mykhaylo Andriluka, Stefan Roth, and Bernt Schiele. “Monocular 3d pose estimation and tracking by detection”. In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition.* IEEE. 2010, pp. 623–630.

[60] Longyin Wen et al. “UA-DETRAC: A new benchmark and protocol for multi-object detection and tracking”. In: *arXiv preprint arXiv:1511.04136* (2015).

[61] Alex Bewley et al. “Simple online and realtime tracking”. In: *2016 IEEE International Conference on Image Processing (ICIP).* IEEE. 2016, pp. 3464–3468.

[62] Rudolph Emil Kalman. “A new approach to linear filtering and prediction problems”. In: *Journal of basic Engineering* 82.1 (1960), pp. 35–45.

[63] Harold W Kuhn. “The Hungarian method for the assignment problem”. In: *Naval research logistics quarterly* 2.1-2 (1955), pp. 83–97.

[64] Fengwei Yu et al. “Poi: Multiple object tracking with high performance detection and appearance feature”. In: *European Conference on Computer Vision.* Springer. 2016, pp. 36–42.

[65] Sean Bell et al. “Inside-outside net: Detecting objects in context with skip pooling and recurrent neural networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2016, pp. 2874–2883.

[66] Spyros Gidaris and Nikos Komodakis. “Object detection via a multi-region and semantic segmentation-aware cnn model”. In: *Proceedings of the IEEE International Conference on Computer Vision.* 2015, pp. 1134–1142.

[67] Andreas Ess et al. “A mobile vision system for robust multi-person tracking”. In: *2008 IEEE Conference on Computer Vision and Pattern Recognition.* IEEE. 2008, pp. 1–8.

[68] Piotr Dollár et al. “Pedestrian detection: A benchmark”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2009, pp. 304–311.

[69] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. “Simple online and realtime tracking with a deep association metric”. In: *2017 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2017, pp. 3645–3649.

[70] Nima Mahmoudi, Seyed Mohammad Ahadi, and Mohammad Rahmati. “Multi-target tracking using cnn-based features: Cnnmtt”. In: *Multimedia Tools and Applications* 78.6 (2019), pp. 7077–7096.

[71] Xingyu Wan, Jinjun Wang, and Sanping Zhou. “An Online and Flexible Multi-Object Tracking Framework using Long Short-Term Memory”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2018, pp. 1230–1238.

[72] Takayuki Ujiie, Masayuki Hiromoto, and Takashi Sato. “Interpolation-based Object Detection Using Motion Vectors for Embedded Real-time Tracking Systems”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2018, pp. 616–624.

[73] Qizheng He et al. “SOT for MOT”. In: *arXiv preprint arXiv:1712.01059* (2017).

[74] Minghua Li et al. “Multi-person tracking by discriminative affinity model and hierarchical association”. In: *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*. IEEE. 2017, pp. 1741–1745.

[75] Wenbo Li, Ming-Ching Chang, and Siwei Lyu. “Who did What at Where and When: Simultaneous Multi-Person Tracking and Activity Recognition”. In: *arXiv preprint arXiv:1807.01253* (2018).

[76] Felipe Jorquera, Sergio Hernández, and Diego Vergara. "Probability hypothesis density filter using determinantal point processes for multi object tracking". In: *Computer Vision and Image Understanding* (2019).

[77] Zhao Zhong et al. "Decision Controller for Object Tracking with Deep Reinforcement Learning". In: *IEEE Access* (2019).

[78] Weigang Lu et al. "Multi-target tracking by non-linear motion patterns based on hierarchical network flows". In: *Multimedia Systems* (2019), pp. 1–12.

[79] Siyu Tang et al. "Multiple people tracking by lifted multicut and person re-identification". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2017, pp. 3539–3548.

[80] Zongwei Zhou et al. "Online Multi-Target Tracking with Tensor-Based High-Order Graph Matching". In: *2018 24th International Conference on Pattern Recognition (ICPR).* IEEE. 2018, pp. 1809–1814.

[81] Nan Ran et al. "A Robust Multi-Athlete Tracking Algorithm by Exploiting Discriminant Features and Long-Term Dependencies". In: *International Conference on Multimedia Modeling.* Springer. 2019, pp. 411–423.

[82] Haigen Hu et al. "An Automatic Tracking Method for Multiple Cells Based on Multi-Feature Fusion". In: *IEEE Access* 6 (2018), pp. 69782–69793.

[83] Lei Zhang et al. "Automatic Individual Pig Detection and Tracking in Pig Farms". In: *Sensors* 19.5 (2019), p. 1188.

[84] Martin Danelljan et al. "ECO: efficient convolution operators for tracking". In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2017, pp. 6638–6646.

[85] Navneet Dalal and Bill Triggs. “Histograms of oriented gradients for human detection”. In: *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR’05)*. Vol. 1. IEEE. 2005, pp. 886–893.

[86] Joost Van De Weijer et al. “Learning color names for real-world applications”. In: *IEEE Transactions on Image Processing* 18.7 (2009), pp. 1512–1523.

[87] Hilke Kieritz, Wolfgang Hubner, and Michael Arens. “Joint detection and online multi-object tracking”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2018, pp. 1459–1467.

[88] Dawei Zhao et al. “Multi-Object Tracking with Correlation Filter for Autonomous Vehicle”. In: *Sensors* 18.7 (2018), p. 2004.

[89] Mengmeng Wang, Yong Liu, and Zeyi Huang. “Large margin object tracking with circulant feature maps”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 4021–4029.

[90] Yongyi Lu, Cewu Lu, and Chi-Keung Tang. “Online video object detection using association LSTM”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 2344–2352.

[91] Sang Jun Kim, Jae-Yeal Nam, and Byoung Chul Ko. “Online tracker optimization for multi-pedestrian tracking using a moving vehicle camera”. In: *IEEE Access* 6 (2018), pp. 48675–48687.

[92] Sarthak Sharma et al. “Beyond pixels: Leveraging geometry and shape cues for online multi-object tracking”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 3508–3515.

[93] Jimmy Ren et al. "Accurate single stage detector using recurrent rolling convolution". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 5420–5428.

[94] Yu Xiang et al. "Subcategory-aware convolutional neural networks for object proposals and detection". In: *2017 IEEE winter conference on applications of computer vision (WACV)*. IEEE. 2017, pp. 924–933.

[95] Federico Pernici et al. "Memory based online learning of deep representations from video streams". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 2324–2334.

[96] Peiyun Hu and Deva Ramanan. "Finding tiny faces". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 951–959.

[97] Weidong Min et al. "A new approach to track multiple vehicles with the combination of robust detection and two classifiers". In: *IEEE Transactions on Intelligent Transportation Systems* 19.1 (2018), pp. 174–186.

[98] Olivier Barnich and Marc Van Droogenbroeck. "ViBe: A universal background subtraction algorithm for video sequences". In: *IEEE Transactions on Image processing* 20.6 (2011), pp. 1709–1724.

[99] Corinna Cortes and Vladimir Vapnik. "Support-vector networks". In: *Machine learning* 20.3 (1995), pp. 273–297.

[100] Shaoyong Yu et al. "A model for fine-grained vehicle classification based on deep learning". In: *Neurocomputing* 257 (2017), pp. 97–103.

[101] Sebastian Bullinger, Christoph Bodensteiner, and Michael Arens. "Instance flow based online multiple object tracking". In: *2017 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2017, pp. 785–789.

[102] Jifeng Dai, Kaiming He, and Jian Sun. “Instance-aware semantic segmentation via multi-task network cascades”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2016, pp. 3150–3158.

[103] Gunnar Farnebäck. “Two-frame motion estimation based on polynomial expansion”. In: *Scandinavian conference on Image analysis.* Springer. 2003, pp. 363–370.

[104] Jerome Revaud et al. “Deepmatching: Hierarchical deformable dense matching”. In: *International Journal of Computer Vision* 120.3 (2016), pp. 300–323.

[105] Yinlin Hu, Rui Song, and Yunsong Li. “Efficient coarse-to-fine patchmatch for large displacement optical flow”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2016, pp. 5704–5712.

[106] Li Wang et al. “Learning deep features for multiple object tracking by using a multi-task learning strategy”. In: *2014 IEEE International Conference on Image Processing (ICIP).* IEEE. 2014, pp. 838–842.

[107] Charles Cadieu and Bruno A Olshausen. “Learning transformational invariants from natural movies”. In: *Advances in neural information processing systems.* 2009, pp. 209–216.

[108] Chanho Kim et al. “Multiple hypothesis tracking revisited”. In: *Proceedings of the IEEE International Conference on Computer Vision.* 2015, pp. 4696–4704.

[109] Fuxin Li et al. “Video segmentation by tracking many figure-ground segments”. In: *Proceedings of the IEEE International Conference on Computer Vision.* 2013, pp. 2192–2199.

[110] Donald Reid. “An algorithm for tracking multiple targets”. In: *IEEE transactions on Automatic Control* 24.6 (1979), pp. 843–854.

[111] Longyin Wen et al. “Multiple target tracking based on undirected hierarchical relation hypergraph”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2014, pp. 1282–1289.

[112] Hao-Shu Fang et al. “Rmpe: Regional multi-person pose estimation”. In: *Proceedings of the IEEE International Conference on Computer Vision.* 2017, pp. 2334–2343.

[113] Leonid Pishchulin et al. “Deepcut: Joint subset partition and labeling for multi person pose estimation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2016, pp. 4929–4937.

[114] Mustafa Ozuysal et al. “Fast keypoint recognition using random ferns”. In: *IEEE transactions on pattern analysis and machine intelligence* 32.3 (2009), pp. 448–461.

[115] Mohib Ullah and Faouzi Alaya Cheikh. “A Directed Sparse Graphical Model for Multi-Target Tracking”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops.* 2018, pp. 1816–1823.

[116] Lawrence R Rabiner and Biing-Hwang Juang. “An introduction to hidden Markov models”. In: *ieee assp magazine* 3.1 (1986), pp. 4–16.

[117] Jiahui Chen et al. “Enhancing detection model for multiple hypothesis tracking”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops.* 2017, pp. 18–27.

[118] Min Yang, Yuwei Wu, and Yunde Jia. “A hybrid data association framework for robust online multi-object tracking”. In: *IEEE Transactions on Image Processing* 26.12 (2017), pp. 5667–5679.

[119] Shuo Hong Wang, Jing Wen Zhao, and Yan Qiu Chen. “Robust tracking of fish schools using CNN for head identification”. In: *Multimedia Tools and Applications* 76.22 (2017), pp. 23679–23697.

[120] Seung-Hwan Bae and Kuk-Jin Yoon. "Confidence-based data association and discriminative deep appearance learning for robust online multi-object tracking". In: *IEEE transactions on pattern analysis and machine intelligence* 40.3 (2017), pp. 595–610.

[121] Flip Korn and Suresh Muthukrishnan. "Influence sets based on reverse nearest neighbor queries". In: *ACM Sigmod Record* 29.2 (2000), pp. 201–212.

[122] Mohib Ullah and Faouzi Alaya Cheikh. "Deep Feature Based End-to-End Transportation Network for Multi-Target Tracking". In: *2018 25th IEEE International Conference on Image Processing (ICIP)*. IEEE. 2018, pp. 3738–3742.

[123] Kuan Fang et al. "Recurrent autoregressive networks for online multi-object tracking". In: *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2018, pp. 466–475.

[124] Longyin Wen et al. "Learning Non-Uniform Hypergraph for Multi-Object Tracking". In: *Thirty-Third AAAI Conference on Artificial Intelligence* (2019).

[125] Hao Sheng et al. "Heterogeneous Association Graph Fusion for Target Association in Multiple Object Tracking". In: *IEEE Transactions on Circuits and Systems for Video Technology* (2018).

[126] Chanho Kim, Fuxin Li, and James M Rehg. "Multi-object tracking with neural gating using bilinear LSTM". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 200–215.

[127] Longtao Chen, Xiaojiang Peng, and Mingwu Ren. "Recurrent Metric Networks and Batch Multiple Hypothesis for Multi-Object Tracking". In: *IEEE Access* 7 (2019), pp. 3093–3105.

[128] Zeyu Fu et al. "Gm-phd filter based online multiple human tracking using deep discriminative correlation matching". In: *2018 IEEE*

International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE. 2018, pp. 4299–4303.

[129] B-N Vo and W-K Ma. “The Gaussian mixture probability hypothesis density filter”. In: *IEEE Transactions on signal processing* 54.11 (2006), pp. 4091–4104.

[130] Jane Bromley et al. “Signature verification using a” siamese” time delay neural network”. In: *Advances in neural information processing systems*. 1994, pp. 737–744.

[131] Minyoung Kim, Stefano Alletto, and Luca Rigazio. “Similarity mapping with enhanced siamese network for multi-object tracking”. In: *Machine Learning for Intelligent Transportation Systems (MLITS), 2016 NIPS Workshop*. 2016.

[132] Bing Wang et al. “Joint learning of convolutional neural networks and temporally constrained metrics for tracklet association”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2016, pp. 1–8.

[133] Shun Zhang et al. “Tracking persons-of-interest via adaptive discriminative features”. In: *European conference on computer vision*. Springer. 2016, pp. 415–433.

[134] Laura Leal-Taixé, Cristian Canton-Ferrer, and Konrad Schindler. “Learning by tracking: Siamese CNN for robust target association”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2016, pp. 33–40.

[135] Laura Leal-Taixé, Gerard Pons-Moll, and Bodo Rosenhahn. “Everybody needs somebody: Modeling social and grouping behavior on a linear programming multiple people tracker”. In: *2011 IEEE international conference on computer vision workshops (ICCV workshops)*. IEEE. 2011, pp. 120–127.

[136] Jeany Son et al. “Multi-object tracking with quadruplet convolutional neural networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 5620–5629.

[137] Ji Zhu et al. “Online multi-object tracking with dual matching attention networks”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 366–382.

[138] Hui Zhou et al. “Deep continuous conditional random fields with asymmetric inter-object constraints for online multi-object tracking”. In: *IEEE Transactions on Circuits and Systems for Video Technology* (2018).

[139] Chen Long et al. “Real-time Multiple People Tracking with Deeply Learned Candidate Selection and Person Re-identification”. In: *ICME*. 2018.

[140] Sangyun Lee and Euntai Kim. “Multiple Object Tracking via Feature Pyramid Siamese Networks”. In: *IEEE Access* 7 (2019), pp. 8181–8194.

[141] Forrest N Iandola et al. “SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and¡ 0.5 MB model size”. In: *arXiv preprint arXiv:1602.07360* (2016).

[142] Andrii Maksai and Pascal Fua. “Eliminating Exposure Bias and Loss-Evaluation Mismatch in Multiple Object Tracking”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2019).

[143] Alexander Hermans, Lucas Beyer, and Bastian Leibe. “In defense of the triplet loss for person re-identification”. In: *arXiv preprint arXiv:1703.07737* (2017).

[144] Cong Ma et al. “Trajectory factory: Tracklet cleaving and re-connection by deep siamese bi-gru for multiple object tracking”. In: *2018 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE. 2018, pp. 1–6.

[145] Mohib Ullah et al. “A hierarchical feature model for multi-target tracking”. In: *2017 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2017, pp. 2612–2616.

[146] Stéphane G Mallat and Zhifeng Zhang. “Matching pursuits with time-frequency dictionaries”. In: *IEEE Transactions on signal processing* 41.12 (1993), pp. 3397–3415.

[147] Amir Sadeghian, Alexandre Alahi, and Silvio Savarese. “Tracking the untrackable: Learning to track multiple cues with long-term dependencies”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 300–311.

[148] Qi Chu et al. “Online multi-object tracking using CNN-based single object tracker with spatial-temporal attention mechanism”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 4836–4845.

[149] Lu Wang et al. “Online multiple object tracking via flow and convolutional features”. In: *2017 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2017, pp. 3630–3634.

[150] Chao Ma et al. “Hierarchical convolutional features for visual tracking”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 3074–3082.

[151] Bruce D. Lucas and Takeo Kanade. “An iterative image registration technique with an application to stereo vision”. In: *Proceedings of Imaging Understanding Workshop*. Vancouver, British Columbia, 1981, pp. 121–130.

[152] Pol Rosello and Mykel J Kochenderfer. “Multi-Agent Reinforcement Learning for Multi-Object Tracking”. In: *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems. 2018, pp. 1397–1404.

[153] Maryam Babaee, Zimu Li, and Gerhard Rigoll. "Occlusion Handling in Tracking Multiple People Using RNN". In: *2018 25th IEEE International Conference on Image Processing (ICIP)*. IEEE. 2018, pp. 2715–2719.

[154] Anton Milan et al. "Online multi-target tracking using recurrent neural networks". In: *Thirty-First AAAI Conference on Artificial Intelligence*. 2017.

[155] Yu Xiang, Alexandre Alahi, and Silvio Savarese. "Learning to track: Online multi-object tracking by decision making". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 4705–4713.

[156] Yiming Liang and Yue Zhou. "LSTM Multiple Object Tracker Combining Multiple Cues". In: *2018 25th IEEE International Conference on Image Processing (ICIP)*. IEEE. 2018, pp. 2351–2355.

[157] Kwangjin Yoon et al. "Data association for multi-object tracking via deep neural networks". In: *Sensors* 19.3 (2019), p. 559.

[158] Alexandre Robicquet et al. "Learning social etiquette: Human trajectory understanding in crowded scenes". In: *European conference on computer vision*. Springer. 2016, pp. 549–565.

[159] Mykhaylo Andriluka, Stefan Roth, and Bernt Schiele. "People-tracking-by-detection and people-detection-by-tracking". In: *2008 IEEE Conference on computer vision and pattern recognition*. IEEE. 2008, pp. 1–8.

[160] Kyunghyun Cho et al. "On the properties of neural machine translation: Encoder-decoder approaches". In: *arXiv preprint arXiv:1409.1259* (2014).

[161] Bjoern Andres, Andrea Fuksová, and Jan-Hendrik Lange. "Lifting of multicuts". In: *CoRR, abs/1503.03791* 3 (2015).

[162] Philippe Weinzaepfel et al. “DeepFlow: Large displacement optical flow with deep matching”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2013, pp. 1385–1392.

[163] Margret Keuper et al. “Efficient decomposition of image and mesh graphs by lifted multicuts”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 1751–1759.

[164] M Sanjeev Arulampalam et al. “A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking”. In: *IEEE Transactions on signal processing* 50.2 (2002), pp. 174–188.

[165] Long Chen et al. “Online multi-object tracking with convolutional neural networks”. In: *2017 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2017, pp. 645–649.

[166] Ricardo Sanchez-Matilla, Fabio Poiesi, and Andrea Cavallaro. “Online multi-target tracking with strong and weak detections”. In: *European Conference on Computer Vision*. Springer. 2016, pp. 84–99.

[167] Liqian Ma et al. “Customized Multi-Person Tracker”. In: *Computer Vision – ACCV 2018*. Springer International Publishing, Dec. 2018.

[168] Siyu Tang et al. “Multi-person tracking by multicut and deep matching”. In: *European Conference on Computer Vision*. Springer. 2016, pp. 100–111.

[169] Liangliang Ren et al. “Collaborative deep reinforcement learning for multi-object tracking”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 586–602.

[170] Hyeonseob Nam and Bohyung Han. “Learning multi-domain convolutional neural networks for visual tracking”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 4293–4302.

[171] Yifan Jiang, Hyunhak Shin, and Hanseok Ko. “Precise Regression for Bounding Box Correction for Improved Tracking Based on Deep Reinforcement Learning”. In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2018, pp. 1643–1647.

[172] Christopher John Cornish Hellaby Watkins. “Learning from delayed rewards”. PhD thesis. King’s College, Cambridge, 1989.

[173] Byungjae Lee et al. “Multi-class multi-object tracking using changing point detection”. In: *European Conference on Computer Vision*. Springer. 2016, pp. 68–83.

[174] Jun-ichi Takeuchi and Kenji Yamanishi. “A unifying framework for detecting outliers and change points from time series”. In: *IEEE transactions on Knowledge and Data Engineering* 18.4 (2006), pp. 482–492.

[175] Anthony Hoak, Henry Medeiros, and Richard Povinelli. “Image-based multi-target tracking through multi-bernoulli filtering with interactive likelihoods”. In: *Sensors* 17.3 (2017), p. 501.

[176] Reza Hoseinnezhad et al. “Visual tracking of numerous targets via multi-Bernoulli filtering of image data”. In: *Pattern Recognition* 45.10 (2012), pp. 3625–3635.

[177] Anton Milan et al. “Improving global multi-target tracking with local updates”. In: *European Conference on Computer Vision*. Springer. 2014, pp. 174–190.

[178] Roberto Henschel et al. “Fusion of head and full-body detectors for multi-object tracking”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2018, pp. 1428–1437.

[179] Russell Stewart, Mykhaylo Andriluka, and Andrew Y Ng. “End-to-end people detection in crowded scenes”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2325–2333.

[180] Weihao Gan et al. “Online CNN-based multiple object tracking with enhanced model updates and identity association”. In: *Signal Processing: Image Communication* 66 (2018), pp. 95–102.

[181] Jun Xiang, Guoshuai Zhang, and Jianhua Hou. “Online Multi-Object Tracking Based on Feature Representation and Bayesian Filtering within a Deep Learning Architecture”. In: *IEEE Access* (2019).

[182] Peng Chu et al. “Online Multi-Object Tracking with Instance-Aware Tracker and Dynamic Model Refreshment”. In: *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2019, pp. 161–170.

[183] Zhe Cao et al. “Realtime multi-person 2d pose estimation using part affinity fields”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 7291–7299.

[184] Liming Zhao et al. “Deeply-learned part-aligned representations for person re-identification”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 3219–3228.

[185] João F Henriques et al. “High-speed tracking with kernelized correlation filters”. In: *IEEE transactions on pattern analysis and machine intelligence* 37.3 (2014), pp. 583–596.

[186] *Sito web di CEDEO s.a.s.* https://cedeo.net/. Accessed: 2020-10-29.

[187] *TVBridge – Enrich your television program.* https://wimlabs.com/en/tvbridge.html. Accessed: 2020-10-29.

[188] *products - Cedeo.net.* https://cedeo.net/products/. Accessed: 2020-10-29.

[189] Gioele Ciaparrone and Roberto Tagliaferri. “A comparison of deep learning models for large scale face-based video retrieval”. In preparation.

[190] Costas Cotsaces, Nikos Nikolaidis, and Ioannis Pitas. “Face-based digital signatures for video retrieval”. In: *IEEE transactions on Circuits and Systems for Video Technology* 18.4 (2008), pp. 549–553.

[191] Murat Taskiran, Nihan Kahraman, and Cigdem Eroglu Erdem. “Face recognition: Past, present and future (a review)”. In: *Digital Signal Processing* (2020), p. 102809.

[192] Ognjen Arandjelović and Andrew Zisserman. “On film character retrieval in feature-length films”. In: *Interactive Video*. Springer, 2006, pp. 89–105.

[193] Christopher JC Burges. “A tutorial on support vector machines for pattern recognition”. In: *Data mining and knowledge discovery* 2.2 (1998), pp. 121–167.

[194] Josef Sivic, Mark Everingham, and Andrew Zisserman. “Person spotting: video shot retrieval for face sets”. In: *International conference on image and video retrieval*. Springer. 2005, pp. 226–236.

[195] David G Lowe. “Distinctive image features from scale-invariant keypoints”. In: *International journal of computer vision* 60.2 (2004), pp. 91–110.

[196] Christian Herrmann and Jürgen Beyerer. “Face retrieval on large-scale video data”. In: *2015 12th Conference on Computer and Robot Vision*. IEEE. 2015, pp. 192–199.

[197] Timo Ahonen, Abdenour Hadid, and Matti Pietikainen. “Face description with local binary patterns: Application to face recognition”. In: *IEEE transactions on pattern analysis and machine intelligence* 28.12 (2006), pp. 2037–2041.

[198] Taskeed Jabid, Md Hasanul Kabir, and Oksam Chae. "Local directional pattern (LDP) for face recognition". In: *2010 digest of technical papers international conference on consumer electronics (ICCE)*. IEEE. 2010, pp. 329–330.

[199] Josef Sivic and Andrew Zisserman. "Video Google: A text retrieval approach to object matching in videos". In: *null*. IEEE. 2003, p. 1470.

[200] Florent Perronnin, Jorge Sánchez, and Thomas Mensink. "Improving the fisher kernel for large-scale image classification". In: *European conference on computer vision*. Springer. 2010, pp. 143–156.

[201] Lior Wolf, Tal Hassner, and Itay Maoz. "Face recognition in unconstrained videos with matched background similarity". In: *CVPR 2011*. IEEE. 2011, pp. 529–534.

[202] Rodney Goh et al. "The CMU face in action (FIA) database". In: *International Workshop on Analysis and Modeling of Faces and Gestures*. Springer. 2005, pp. 255–263.

[203] Ognjen Arandjelovic. "Learnt quasi-transitive similarity for retrieval from large collections of faces". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 4883–4892.

[204] Jinjun Wang et al. "Locality-constrained linear coding for image classification". In: *2010 IEEE computer society conference on computer vision and pattern recognition*. IEEE. 2010, pp. 3360–3367.

[205] Yan Li et al. "Compact Video Code and Its Application to Robust Face Retrieval in TV-Series." In: *British Machine Vision Conference 2014*. 2014.

[206] Yan Li et al. "Hierarchical hybrid statistic based video binary code and its application to face retrieval in TV-series". In: *2015 11th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG)*. Vol. 1. IEEE. 2015, pp. 1–8.

[207] Yan Li et al. “Spatial pyramid covariance-based compact video code for robust face retrieval in TV-series”. In: *IEEE Transactions on Image Processing* 25.12 (2016), pp. 5905–5919.

[208] Ruiping Wang et al. “Covariance discriminative learning: A natural and efficient approach to image set classification”. In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2012, pp. 2496–2503.

[209] Yan Li et al. “Face video retrieval with image query via hashing across euclidean space and riemannian manifold”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 4758–4767.

[210] Zhen Dong et al. “Face video retrieval via deep learning of binary hash representations”. In: *Thirtieth AAAI Conference on Artificial Intelligence*. 2016.

[211] Zhen Dong et al. “Deep CNN based binary hash video representations for face retrieval”. In: *Pattern Recognition* 81 (2018), pp. 357–369.

[212] Markus Mühling et al. “Deep learning for content-based video retrieval in film and television production”. In: *Multimedia Tools and Applications* 76.21 (2017), pp. 22169–22194.

[213] Huaizu Jiang and Erik Learned-Miller. “Face detection with the faster R-CNN”. In: *2017 12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017)*. IEEE. 2017, pp. 650–657.

[214] Matthew D Zeiler and Rob Fergus. “Visualizing and understanding convolutional networks”. In: *European conference on computer vision*. Springer. 2014, pp. 818–833.

[215] Ralph Ewerth and Bernd Freisleben. “Video cut detection without thresholds”. In: *Proc. of 11th Workshop on Signals, Systems and Image Processing*. 2004, pp. 227–230.

[216] Ralph Ewerth and Bernd Freisleben. “Unsupervised detection of gradual video shot changes with motion-based false alarm removal”. In: *International conference on advanced concepts for intelligent vision systems*. Springer. 2009, pp. 253–264.

[217] Enrique G Ortiz, Alan Wright, and Mubarak Shah. “Face recognition in movie trailers via mean sequence sparse representation-based classification”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2013, pp. 3531–3538.

[218] Dong Yi et al. “Learning face representation from scratch”. In: *arXiv preprint arXiv:1411.7923* (2014).

[219] Neeraj Kumar et al. “Describable visual attributes for face verification and image search”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.10 (2011), pp. 1962–1977.

[220] Shishi Qiao et al. “Deep video code for efficient face video retrieval”. In: *Asian Conference on Computer Vision*. Springer. 2016, pp. 296–312.

[221] Chenchen Jing et al. “Fusing Appearance Features and Correlation Features for Face Video Retrieval”. In: *Pacific Rim Conference on Multimedia*. Springer. 2017, pp. 150–160.

[222] Xi Fang and Ying Zou. “Make the Best of Face Clues in iQIYI Celebrity VideoIdentification Challenge 2019”. In: *Proceedings of the 27th ACM International Conference on Multimedia*. 2019, pp. 2526–2530.

[223] *2019 iQIYI Celebrity Video Identification Challenge*. http://challenge.ai.iqiyi.com/detail?raceId=5c767dc41a6fa0ccf53922e6. Accessed: 2020-10-20.

[224] Shishi Qiao et al. “Deep heterogeneous hashing for face video retrieval”. In: *IEEE Transactions on Image Processing* 29 (2019), pp. 1299–1312.

[225] Minyoung Kim et al. “Face tracking and recognition with visual constraints in real-world videos”. In: *2008 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2008, pp. 1–8.

[226] Ankan Bansal et al. “Umdfaces: An annotated face dataset for training deep networks”. In: *2017 IEEE International Joint Conference on Biometrics (IJCB)*. IEEE. 2017, pp. 464–473.

[227] Ruikui Wang et al. “Hybrid Video and Image Hashing for Robust Face Retrieval”. In: *2020 15th IEEE International Conference on Automatic Face and Gesture Recognition (FG 2020)(FG)*, pp. 186–193.

[228] Guodong Guo and Na Zhang. “A survey on deep learning based face recognition”. In: *Computer Vision and Image Understanding* 189 (2019), p. 102805.

[229] Yaniv Taigman et al. “Deepface: Closing the gap to human-level performance in face verification”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 1701–1708.

[230] Gary B. Huang et al. “Labeled Faces in the Wild: A Database for studying Face Recognition in Unconstrained Environments”. In: *Workshop on Faces in 'Real-Life' Images: Detection, Alignment, and Recognition*. Erik Learned-Miller and Andras Ferencz and Frédéric Jurie. Marseille, France, Oct. 2008.

[231] Neeraj Kumar et al. “Attribute and simile classifiers for face verification”. In: *2009 IEEE 12th international conference on computer vision*. IEEE. 2009, pp. 365–372.

[232] Yi Sun, Xiaogang Wang, and Xiaoou Tang. “Deep learning face representation from predicting 10,000 classes”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 1891–1898.

[233] Dong Chen et al. “Bayesian face revisited: A joint formulation”. In: *European conference on computer vision*. Springer. 2012, pp. 566–579.

[234] Yi Sun et al. “Deep learning face representation by joint identification-verification”. In: *Advances in neural information processing systems*. 2014, pp. 1988–1996.

[235] Yi Sun et al. “Deepid3: Face recognition with very deep neural networks”. In: *arXiv preprint arXiv:1502.00873* (2015).

[236] Florian Schroff, Dmitry Kalenichenko, and James Philbin. “Facenet: A unified embedding for face recognition and clustering”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 815–823.

[237] Omkar M Parkhi, Andrea Vedaldi, and Andrew Zisserman. “Deep face recognition”. In: (2015).

[238] Qiong Cao et al. “Vggface2: A dataset for recognising faces across pose and age”. In: *2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018)*. IEEE. 2018, pp. 67–74.

[239] Yandong Guo et al. “Ms-celeb-1m: A dataset and benchmark for large-scale face recognition”. In: *European conference on computer vision*. Springer. 2016, pp. 87–102.

[240] Brendan F Klare et al. “Pushing the frontiers of unconstrained face detection and recognition: Iarpa janus benchmark a”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1931–1939.

[241] Yandong Wen et al. “A discriminative feature learning approach for deep face recognition”. In: *European conference on computer vision*. Springer. 2016, pp. 499–515.

[242] Ce Qi and Fei Su. “Contrastive-center loss for deep neural networks”. In: *2017 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2017, pp. 2851–2855.

[243] Weiyang Liu et al. “Large-Margin Softmax Loss for Convolutional Neural Networks”. In: *Proceedings of The 33rd International Conference on Machine Learning.* Vol. 48. Proceedings of Machine Learning Research. PMLR, June 2016, pp. 507–516.

[244] Yu Liu, Hongyang Li, and Xiaogang Wang. “Rethinking feature discrimination and polymerization for large-scale recognition”. In: *arXiv preprint arXiv:1710.00870* (2017).

[245] Weiyang Liu et al. “Sphereface: Deep hypersphere embedding for face recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2017, pp. 212–220.

[246] Xianbiao Qi and Lei Zhang. “Face recognition via centralized coordinate learning”. In: *arXiv preprint arXiv:1801.05678* (2018).

[247] Feng Wang et al. “Additive margin softmax for face verification”. In: *IEEE Signal Processing Letters* 25.7 (2018), pp. 926–930.

[248] Yutong Zheng, Dipan K Pal, and Marios Savvides. “Ring loss: Convex feature normalization for face recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2018, pp. 5089–5097.

[249] Hao Wang et al. “Cosface: Large margin cosine loss for deep face recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2018, pp. 5265–5274.

[250] Chen Huang et al. “Deep imbalanced learning for face recognition and attribute prediction”. In: *IEEE transactions on pattern analysis and machine intelligence* (2019).

[251] Jiankang Deng et al. “Arcface: Additive angular margin loss for deep face recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2019, pp. 4690–4699.

[252] Ira Kemelmacher-Shlizerman et al. “The megaface benchmark: 1 million faces for recognition at scale”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 4873–4882.

[253] Cameron Whitelam et al. “Iarpa janus benchmark-b face dataset”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2017, pp. 90–98.

[254] Brianna Maze et al. “Iarpa janus benchmark-c: Face dataset and protocol”. In: *2018 International Conference on Biometrics (ICB)*. IEEE. 2018, pp. 158–165.

[255] Yuanliu Liu et al. “iqiyi-vid: A large dataset for multi-modal person identification”. In: *arXiv preprint arXiv:1811.07548* (2018).

[256] Yongming Rao et al. “Learning discriminative aggregation network for video-based face recognition”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 3781–3790.

[257] J Ross Beveridge et al. “The challenge of face recognition from digital point-and-shoot cameras”. In: *2013 IEEE Sixth International Conference on Biometrics: Theory, Applications and Systems (BTAS)*. IEEE. 2013, pp. 1–8.

[258] Yongming Rao, Jiwen Lu, and Jie Zhou. “Attention-aware deep reinforcement learning for video face recognition”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 3931–3940.

[259] Changxing Ding and Dacheng Tao. “Trunk-branch ensemble convolutional neural networks for video-based face recognition”. In: *IEEE transactions on pattern analysis and machine intelligence* 40.4 (2017), pp. 1002–1014.

[260] Jingxiao Zheng et al. “An automatic system for unconstrained video-based face recognition”. In: *IEEE Transactions on Biometrics, Behavior, and Identity Science* 2.3 (2020), pp. 194–209.

[261] Jun-Cheng Chen et al. "A real-time multi-task single shot face detector". In: *2018 25th IEEE International Conference on Image Processing (ICIP)*. IEEE. 2018, pp. 176–180.

[262] Rajeev Ranjan et al. "An all-in-one convolutional neural network for face analysis". In: *2017 12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017)*. IEEE. 2017, pp. 17–24.

[263] Rajeev Ranjan et al. "Deep learning for understanding faces: Machines may be just as good, or better, than humans". In: *IEEE Signal Processing Magazine* 35.1 (2018), pp. 66–83.

[264] Nathan D Kalka et al. "IJB–S: IARPA Janus surveillance video benchmark". In: *2018 IEEE 9th International Conference on Biometrics Theory, Applications and Systems (BTAS)*. IEEE. 2018, pp. 1–9.

[265] *MegaFace*. http://megaface.cs.washington.edu/. Accessed: 2020-11-19.

[266] Paul Viola, Michael Jones, et al. "Robust real-time object detection". In: *International journal of computer vision* 4.34-47 (2001), p. 4.

[267] Ankan Bansal et al. "The do's and don'ts for cnn-based face verification". In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2017, pp. 2545–2554.

[268] *UMDFaces*. http://umdfaces.io/. Accessed: 2020-11-19.

[269] Brendan F Klare et al. "Pushing the frontiers of unconstrained face detection and recognition: Iarpa janus benchmark a". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1931–1939.

[270] *VoxCeleb*. http://www.robots.ox.ac.uk/~vgg/data/voxceleb/. Accessed: 2020-10-21.

[271] Arsha Nagrani, Joon Son Chung, and Andrew Zisserman. “Voxceleb: a large-scale speaker identification dataset”. In: *arXiv preprint arXiv:1706.08612* (2017).

[272] Joon Son Chung, Arsha Nagrani, and Andrew Zisserman. “Voxceleb2: Deep speaker recognition”. In: *arXiv preprint arXiv:1806.05622* (2018).

[273] Josef Sivic, Mark Everingham, and Andrew Zisserman. ““Who are you?”-Learning person specific classifiers from video”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2009, pp. 1145–1152.

[274] Martin Bauml, Makarand Tapaswi, and Rainer Stiefelhagen. “Semi-supervised learning with constraints for person identification in multimedia data”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2013, pp. 3602–3609.

[275] Arsha Nagrani and Andrew Zisserman. “From benedict cumberbatch to sherlock holmes: Character identification in tv series without a script”. In: *arXiv preprint arXiv:1801.10442* (2018).

[276] Qingqiu Huang, Wentao Liu, and Dahua Lin. “Person search in videos with one portrait through visual and temporal links”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 425–441.

[277] Zhengxia Zou et al. “Object detection in 20 years: A survey”. In: *arXiv preprint arXiv:1905.05055* (2019).

[278] Haoxiang Li et al. “A convolutional neural network cascade for face detection”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 5325–5334.

[279] Kaipeng Zhang et al. “Joint face detection and alignment using multitask cascaded convolutional networks”. In: *IEEE Signal Processing Letters* 23.10 (2016), pp. 1499–1503.

[280] Vidit Jain and Erik Learned-Miller. *Fddb: A benchmark for face detection in unconstrained settings*. Tech. rep. UMass Amherst technical report, 2010.

[281] Shuo Yang et al. "Wider face: A face detection benchmark". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 5525–5533.

[282] *kpzhang93/MTCNN_face_detection_alignment*. https://github.com/kpzhang93/MTCNN_face_detection_alignment. Accessed: 2020-12-22.

[283] Kaiming He et al. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[284] Rajeev Ranjan, Vishal M Patel, and Rama Chellappa. "Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 41.1 (2017), pp. 121–135.

[285] Koen EA Van de Sande et al. "Segmentation as selective search for object recognition". In: *2011 International Conference on Computer Vision*. IEEE. 2011, pp. 1879–1886.

[286] Martin Koestinger et al. "Annotated facial landmarks in the wild: A large-scale, real-world database for facial landmark localization". In: *2011 IEEE international conference on computer vision workshops (ICCV workshops)*. IEEE. 2011, pp. 2144–2151.

[287] Xiangxin Zhu and Deva Ramanan. "Face detection, pose estimation, and landmark localization in the wild". In: *2012 IEEE conference on computer vision and pattern recognition*. IEEE. 2012, pp. 2879–2886.

[288] Dong Chen et al. "Supervised transformer network for efficient face detection". In: *European Conference on Computer Vision*. Springer. 2016, pp. 122–138.

[289] Zekun Hao et al. “Scale-aware face detection”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2017, pp. 6186–6195.

[290] Shuo Yang et al. “Face detection through scale-friendly deep convolutional networks”. In: *arXiv preprint arXiv:1706.02863* (2017).

[291] Shifeng Zhang et al. “S3fd: Single shot scale-invariant face detector”. In: *Proceedings of the IEEE international conference on computer vision.* 2017, pp. 192–201.

[292] Mahyar Najibi et al. “Ssh: Single stage headless face detector”. In: *Proceedings of the IEEE international conference on computer vision.* 2017, pp. 4875–4884.

[293] Jianfeng Wang, Ye Yuan, and Gang Yu. “Face attention network: An effective face detector for the occluded faces”. In: *arXiv preprint arXiv:1711.07246* (2017).

[294] Tsung-Yi Lin et al. “Focal loss for dense object detection”. In: *Proceedings of the IEEE international conference on computer vision.* 2017, pp. 2980–2988.

[295] Xuepeng Shi et al. “Real-time rotation-invariant face detection with progressive calibration networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2018, pp. 2295–2303.

[296] Shuo Yang et al. “Faceness-net: Face detection through deep facial part responses”. In: *IEEE transactions on pattern analysis and machine intelligence* 40.8 (2017), pp. 1845–1859.

[297] Xu Tang et al. “Pyramidbox: A context-assisted single shot face detector”. In: *Proceedings of the European Conference on Computer Vision (ECCV).* 2018, pp. 797–813.

[298] Changzheng Zhang, Xiang Xu, and Dandan Tu. “Face detection using improved faster rcnn”. In: *arXiv preprint arXiv:1802.02142* (2018).

[299] Cheng Chi et al. “Selective refinement network for high performance face detection”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 2019, pp. 8231–8238.

[300] Jiankang Deng et al. “Retinaface: Single-stage dense face localisation in the wild”. In: *arXiv preprint arXiv:1905.00641* (2019).

[301] Andrew G Howard et al. “Mobilenets: Efficient convolutional neural networks for mobile vision applications”. In: *arXiv preprint arXiv:1704.04861* (2017).

[302] Jifeng Dai et al. “Deformable convolutional networks”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 764–773.

[303] Shaohua Teng, Wenwei Tan, and Wei Zhang. “Cooperative shot boundary detection for video”. In: *International Conference on Computer Supported Cooperative Work in Design*. Springer. 2007, pp. 99–110.

[304] HongJiang Zhang, Atreyi Kankanhalli, and Stephen W Smoliar. “Automatic partitioning of full-motion video”. In: *Multimedia systems* 1.1 (1993), pp. 10–28.

[305] Greg Pass, Ramin Zabih, and Justin Miller. “Comparing images using color coherence vectors”. In: *Proceedings of the fourth ACM international conference on Multimedia*. 1997, pp. 65–73.

[306] Jun Li et al. “A divide-and-rule scheme for shot boundary detection based on sift”. In: *International Journal of Digital Content Technology and its Applications* 4.3 (2010), pp. 202–214.

[307] Junaid Baber et al. “Shot boundary detection from videos using entropy and local descriptor”. In: *2011 17th International conference on digital signal processing (DSP)*. IEEE. 2011, pp. 1–6.

[308] Xue Ling et al. "A method for fast shot boundary detection based on SVM". In: *2008 Congress on Image and Signal Processing*. Vol. 2. IEEE. 2008, pp. 445–449.

[309] Hong Shao, Yang Qu, and Wencheng Cui. "Shot boundary detection algorithm based on HSV histogram and HOG feature". In: *2015 International Conference on Advanced Engineering Materials and Technology*. Atlantis Press. 2015.

[310] Evlampios Apostolidis and Vasileios Mezaris. "Fast shot segmentation combining global and local visual descriptors". In: *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2014, pp. 6583–6587.

[311] Lorenzo Baraldi, Costantino Grana, and Rita Cucchiara. "Shot and scene detection via hierarchical clustering for re-using broadcast video". In: *International Conference on Computer Analysis of Images and Patterns*. Springer. 2015, pp. 801–811.

[312] Jingwei Xu, Li Song, and Rong Xie. "Shot boundary detection using convolutional neural networks". In: *2016 Visual Communications and Image Processing (VCIP)*. IEEE. 2016, pp. 1–4.

[313] Michael Gygli. "Ridiculously fast shot boundary detection with fully convolutional neural networks". In: *arXiv preprint arXiv:1705.08214* (2017).

[314] Ahmed Hassanien et al. "Large-scale, fast and accurate shot boundary detection through spatio-temporal convolutional neural networks". In: *arXiv preprint arXiv:1705.03281* (2017).

[315] Du Tran et al. "Learning spatiotemporal features with 3d convolutional networks". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 4489–4497.

[316] Shitao Tang et al. “Fast video shot transition localization with deep structured models”. In: *Asian Conference on Computer Vision.* Springer. 2018, pp. 577–592.

[317] Tomáš Souček, Jaroslav Moravec, and Jakub Lokoč. “TransNet: A deep network for fast detection of common shot transitions”. In: *arXiv preprint arXiv:1906.03363* (2019).

[318] Fisher Yu and Vladlen Koltun. “Multi-scale context aggregation by dilated convolutions”. In: *arXiv preprint arXiv:1511.07122* (2015).

[319] George Awad et al. “Trecvid 2017: evaluating ad-hoc and instance video search, events detection, video captioning, and hyperlinking”. In: 2017.

[320] Tomáš Souček and Jakub Lokoč. “TransNet V2: An effective deep network architecture for fast shot transition detection”. In: *arXiv preprint arXiv:2008.04838* (2020).

[321] Lorenzo Baraldi, Costantino Grana, and Rita Cucchiara. “A deep siamese network for scene detection in broadcast videos”. In: *Proceedings of the 23rd ACM international conference on Multimedia.* 2015, pp. 1199–1202.

[322] Erik Bochinski, Volker Eiselein, and Thomas Sikora. “High-speed tracking-by-detection without using image information”. In: *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS).* IEEE. 2017, pp. 1–6.

[323] Jie Hu, Li Shen, and Gang Sun. “Squeeze-and-excitation networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2018, pp. 7132–7141.

[324] Yunchao Gong et al. “Angular quantization-based binary codes for fast similarity search”. In: *Advances in neural information processing systems* 25 (2012), pp. 1196–1204.

[325] Thuy-Diem Nguyen, Bertil Schmidt, and Chee-Keong Kwoh. "SparseHC: a memory-efficient online hierarchical clustering algorithm". In: (2014).

[326] Aditya Krishna Menon et al. "Online Hierarchical Clustering Approximations". In: *arXiv preprint arXiv:1909.09667* (2019).

[327] *sklearn.metrics.average_precision_score – scikit-learn 0.23.2 documentation*. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.average_precision_score.html. Accessed: 2020-12-09.

[328] Rajeev Ranjan et al. "Crystal loss and quality pooling for unconstrained face verification and recognition". In: *arXiv preprint arXiv:1804.01159* (2018).

[329] *deepinsight/insightface: Face Analysis Project on MXNet*. https://github.com/deepinsight/insightface. Accessed: 2020-12-15.

[330] Yves Bestgen. "Exact expected average precision of the random baseline for system evaluation". In: *The Prague Bulletin of Mathematical Linguistics* 103.1 (2015), pp. 131–138.

[331] *foamliu/InsightFace-v2: PyTorch implementation of Additive Angular Margin Loss for Deep Face Recognition*. https://github.com/foamliu/InsightFace-v2. Accessed: 2020-12-15.

[332] *Dramatiq: background tasks*. https://dramatiq.io/. Accessed: 2020-12-31.

[333] Martin Danelljan et al. "Atom: Accurate tracking by overlap maximization". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 4660–4669.

[334] Goutam Bhat et al. "Learning discriminative model prediction for tracking". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 6182–6191.

[335] Zhi-Ming Qian, Xi En Cheng, and Yan Qiu Chen. “Automatically detect and track multiple fish swimming in shallow water with frequent occlusion”. In: *PloS one* 9.9 (2014), e106506.

[336] Hamed Pirsiavash, Deva Ramanan, and Charless C Fowlkes. “Globally-optimal greedy algorithms for tracking a variable number of objects”. In: *CVPR 2011*. IEEE. 2011, pp. 1201–1208.

[337] Steven Gold, Anand Rangarajan, et al. “Softmax to softassign: Neural network algorithms for combinatorial optimization”. In: *Journal of Artificial Neural Networks* 2.4 (1996), pp. 381–399.

[338] Chang Huang, Bo Wu, and Ramakant Nevatia. “Robust object tracking by hierarchical association of detection responses”. In: *European Conference on Computer Vision*. Springer. 2008, pp. 788–801.

[339] Rodrigo Benenson et al. “Pedestrian detection at 100 frames per second”. In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2012, pp. 2903–2910.

List of Figures

List of Tables

La borsa di dottorato è stata cofinanziata con risorse del
Programma Operativo Nazionale Ricerca e Innovazione 2014-2020 (CCI 2014IT16M2OP005),
Fondo Sociale Europeo, Azione I.1 “Dottorati Innovativi con caratterizzazione Industriale”

UNIONE EUROPEA
Fondo Sociale Europeo

Ministero dell'Università e della Ricerca

PON
RICERCA
E INNOVAZIONE
2014 - 2020