



Real-time face alignment: evaluation methods, training strategies and implementation optimization

Constantino Álvarez Casado¹ · Miguel Bordallo López^{1,2}

Received: 11 February 2021 / Accepted: 13 April 2021 / Published online: 26 April 2021
© The Author(s) 2021

Abstract

Face alignment is a crucial component in most face analysis systems. It focuses on identifying the location of several key-points of the human faces in images or videos. Although several methods and models are available to developers in popular computer vision libraries, they still struggle with challenges such as insufficient illumination, extreme head poses, or occlusions, especially when they are constrained by the needs of real-time applications. Throughout this article, we propose a set of training strategies and implementations based on data augmentation, software optimization techniques that help in improving a large variety of models belonging to several real-time algorithms for face alignment. We propose an extended set of evaluation metrics that allow novel evaluations to mitigate the typical problems found in real-time tracking contexts. The experimental results show that the generated models using our proposed techniques are faster, smaller, more accurate, more robust in specific challenging conditions and smoother in tracking systems. In addition, the training strategy shows to be applicable across different types of devices and algorithms, making them versatile in both academic and industrial uses.

Keywords Face alignment · Real-time · Embedded devices · Cascaded regression · Optimization implementation · Training strategies

1 Introduction

Face alignment is a face analysis sub-problem that aims at estimating a sparse set of n specific points (landmarks) defining the shape of the face. Typically, this estimation includes the jaw, eyes, eyebrows, nose, and mouth. This identification of landmarks can be done either in images (a problem known as landmark detection) or on videos (a problem typically known as landmark tracking) [1, 2].

The set of points define the facial contour and face parts as shown in the example in the Fig. 1. That set of points located in a face is usually called a shape and can be considered as a vector describing the x , y coordinates (and z , in the

case of 3D shapes) of its n points in the context of the image that needs to be processed.

Most of the proposed face alignment methods in the literature focus on the inference of a face shape from static images, disregarding challenges under real scenarios in real-time applications such as fast face and head movements, extreme light conditions, artifacts from video streams such as noise or distortions, or smooth tracking of the landmark points. We see a lack of evaluation and analysis of the face alignment models beyond the accuracy of the inference for still images. Face alignment tracking systems involve other problems that have not been widely evaluated and mitigated. In this article, we propose a set of solutions to these challenges that are applicable especially in fast systems that require real-time operations.

This article focuses on improving the performance of state-of-the-art face alignment methods with emphasis on its operation in real time, both in embedded devices and desktop computers. In this context, the article leverages the original implementations and adapts it using multicore enabled C++ libraries. The techniques are applicable to several well-known fast-inference face alignment algorithms as used both in the industry and academic domains.

✉ Constantino Álvarez Casado
constantino.lvarez.casado@oulu.fi

Miguel Bordallo López
miguel.bordallo@vti.fi

¹ Center for Machine Vision and Signal Analysis, University of Oulu, Oulu, Finland

² VTT Technical Research Centre of Finland Ltd., Oulu, Finland

Fig. 1 The landmark locations defined by the Multi-PIE landmarks scheme [3]. (Left): exemplar face image with landmarks (right): schematic

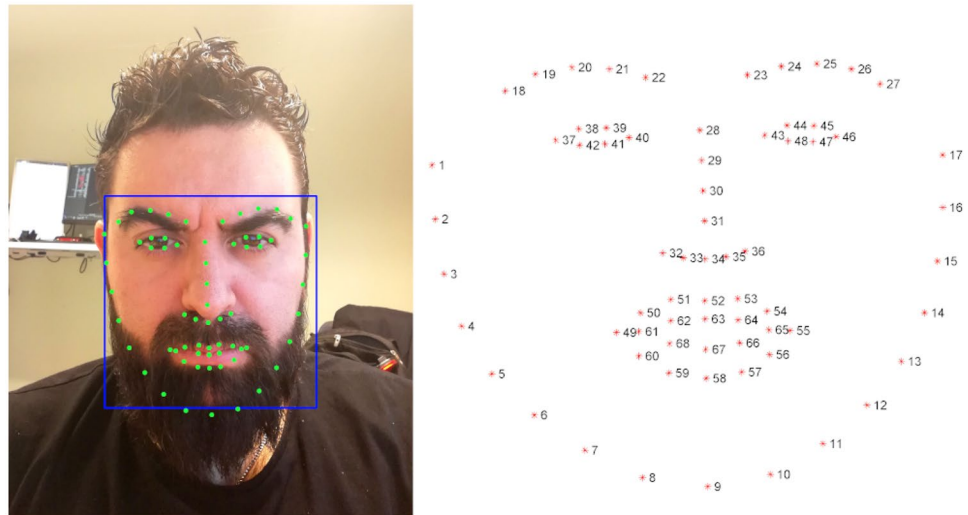


Fig. 2 Examples of face-filtering applications using face landmarks detectors as a core technology to implement them



We tackle the improvement of the accuracy of those models by proposing a set of training strategies, while optimizing the algorithms via parallelization strategies and the application of software engineering techniques, accelerating the inference of the models and making them applicable in systems that require both robust and fast inference of facial landmarks both in static images and video sequences.

1.1 Face alignment

Face is an important information source in many human contexts. It plays an important role for understanding a person from a simple interaction, identity recognition [4, 5], face recognition [6], personality [7], emotional state [8–10], non-verbal communication [11] or even health [12]. This understanding can be used in many fields, such as human-machine or human-computer interactions (HMI and HCI) [13], medical assistance applications, surveillance systems, robotics [14] or psychological and physiological analysis [15–17]. These applications support fields like education, commerce, health, therapy or entertainment, among others.

In recent years, face alignment has shown important progress. Nowadays, it is a key component in many computer vision systems such as facial recognition systems,

auto-focus systems on digital cameras in portrait and selfie modes, facial expression classification, augmented reality, and even face filters, widely used in many social media and social networking services [2]. Figure 2 shows a few of these applications. In general, we can say that all those systems integrate a face analysis module to produce semantic and quantitative information from faces.

Face analysis applications are usually comprised of a set of connected components, known as a pipeline. Face alignment is one of those core components that provides a precise map of facial points. Its results are in turn used further to compute intermediate tasks that feed the next steps. Geometrical transformations, face normalization, segmentation, region of interest selection, head pose, or movement information, all rely on facial alignment or landmark detection. For this reason, face alignment is a critical task in face analysis systems since the performance of most of the next steps in the pipeline rely completely on its performance.

1.2 Contribution

This article provides several contributions that are useful in both theoretical and practical contexts:

- First, we provide a set of original C++ implementations¹ of state-of-the-art face alignment methods based on cascaded regression, following the original publication [18] and its reference Matlab implementation. This custom implementation is used throughout the article, as a modular prototype that allows the adaptation and evaluation on low-end systems and the experimentation with the training of different models under different training strategies.
- We extend this implementation, initially thought for face landmark detection, and modify it to develop a facial landmark tracker, able to perform in video sequences and real-time video, by leveraging temporal information.
- Based on our observations, we derive in a set of guidelines depicting different performance problems and the related metrics for their minimization. We propose three new evaluation metrics to assess the model performance. The metrics measure small and noisy spatial variations around predicted face shapes in similar images, as well as the sensitivity, to other components integrated in the same system such as face detection.
- We derive a set of rules for optimal training strategies, and we name them General Training eXtensions (GTX). Derived from the proposed set of metrics, they include novel techniques such as initialization variations, or the addition of domain-specific data annotated using teacher-student architectures [19].
- We propose a set of software optimization techniques to accelerate the inference of these models, both in embedded systems and desktop computers.
- Finally extensively evaluate a set of models produced by both our implementations and training strategies by comparing them with three different state-of-the-art implementations and models available as open-source code (ERT-dlib, LBF-opencv and DAN).

2 Related work

As a general definition, we can define face alignment as the task of searching predefined facial key points or facial landmarks over a face image or video to obtain a facial shape. The number of key points is fixed and predefined by the annotation protocol and corresponds to specific physiological locations of the human face. The process of face alignment can be comprised of several steps depending on the selected approach and what information is used: facial appearance (pixels, textures), shape information (spatial information), or both.

The current state-of-the-art in face alignment is comprised of very slow methods based on deep learning that require computationally heavy inference and very fast methods based on cascades of regressors that lack the ability to cope with complicated cases or extreme poses [2, 20]. Both types show a wide range of approaches that can be synthesized by grouping them in high-level categories.

Most texts in the literature mainly adopt the classification of face alignment algorithms in two categories: generative methods and discriminative methods, depending on how they handle the probabilities and distributions based on the facial appearance and facial shape patterns [1, 2, 21].

2.1 Generative methods

Generative methods also known as holistic methods, try to model the appearance of the face as a parametric model of a deformable object (synthetic version of the target face), similar to how Eigenfaces approach works for face recognition [22]. Based on the global facial appearance information and the global facial shape patterns, it takes into account the deformation of the face and removes it to obtain the landmarks. The problem is typically formulated as an optimization problem that tries to maximise the probability of facial reconstruction from a deformable model to find the best fit to the target face [1, 2, 23]. Well-know generative methods in the face alignment literature are the Active Shape Models (ASM) [24] and its improved version, the Active Appearance Models (AAM) [25, 26]. Both introduced by Taylor and Cootes and still used nowadays, they play an important role in the field as reference benchmark to evaluate other methods [27]. The ASM method is considered the first one achieving to fit a Point Distribution Model (PDM) in the target face.

Other well-known methods, based on AAM improvements, focused on unconstrained training data or robust and fast fitting using strong statistical and mathematical optimizations [28]. Examples are the Simultaneous Inverse Compositional (SIC) algorithm [29] seems to be accurate but slow, and the Project-out Inverse Compositional (POIC) algorithm [30] which is faster but less robust. Other generative methods can be grouped in the so-called part-based generative methods. Following similar ideas as AAM, they are based on series of local patches. The best example is the Constrained Local Model (CLM) method, described by Cristinacce and Cootes [31], which uses the global facial shape patterns as well as the independent local appearance information around each landmark to compute the probability distribution of landmark coordinates inside the local patch by using local classifiers-based or regression-based local appearance models. It is more robust to illumination variations and occlusions [20].

¹ Available at: <https://gitlab.com/visualhealth/vhpapers/real-time-facealignment>.

2.2 Discriminative methods

Discriminative methods learn discriminative functions during the training stage that directly map the facial appearance to the facial landmarks coordinates by computing the facial points where the alignment is correct based on the error minimization of the discriminative feature vectors. This mapping functions are usually achieved by regression models, that output whether the parametric model is well aligned or not [1, 20, 23]. These methods are also known as Regression-based methods. The process typically consists in multiple and independent local regressors trying to estimate each facial landmark independently using a global Point Distribution Model (PDM) [32] to regularize and constrain the predictions as a deformable shape model with constraints. These methods have usually demonstrated a superior performance under uncontrolled conditions [20, 33]. We can classify the regression-based methods in three high-level categories: Direct Regression methods, Cascaded Regression methods and Deep-learning based regression methods.

Direct Regression methods, also called Ensemble regression-voting, are those able to map the face image appearance to the facial points without any kind of initialization [1]. There are two type of approaches: local and global approaches depending on using global face appearance or local patches. Typical methods inside of this category are the Regression Forests methods [34], simple and low computational complexity algorithms such as the Conditional Regression Forests algorithm proposed by Dantone et al. [35], the Random Forest Regression-Voting method described by Cootes et al. or the Structured-Output Regression forest method [36]. In general, these approaches are more robust than the previous ones due to the combination of votes from different face regions, but they have not demonstrated a good balance between accuracy and efficiency for face alignment in-the-wild [1].

Cascade Regression methods compute the facial landmarks by starting from an initial shape (e.g. canonical face shape). They normally geometrically transform the initial shapes by utilizing the face bounding box to fit it approximately to the target face, and gradually correct those landmark positions across different stages with their own regression function learnt during the training phase, until achieving the final shape in the last stage of the process [2, 23]. Cascaded regression methods have become as one of the most popular and state-of-the-art methods for this purpose. They show a very good balance between speed and high accuracy, making them a good choice for real-time face alignment detectors or trackers. The term was coined for first time by Cao et al. in the Explicit Shape Regression (ESR) approach [37], a boosted regression framework. Among the cascade regression methods, a well-known method is the Supervised Descent Method [38], similar to AAMs in many aspects,

but trained in a discriminative way using linear regression and using hand-craft SIFT features [39] as appearance features. The local patches are formulated as a non-linear least squares optimization problem, resulting in a simple but very fast and rather accurate approach. Other notable methods came as improvements up of the ESR method. Examples are the Regressing Local Binary Features (LBF) algorithm [18], re-implemented in this article and used currently for face alignment in OpenCV library [40] or the Ensemble of Regression Trees (ERT) [41] introduced by Kazemi et al. in 2014 and well-known for being utilized by the Dlib library [42]. A depiction of the stage-based inference of these kinds of methods can be seen in Fig. 3.

More recently, Deep Neural Networks and especially Convolutional Neural Networks approaches have gained attention and become popular tools in computer vision tasks [43], including face alignment. They show robust and accurate inferences of the facial landmarks [44–50]. CNNs can extract high-level image features, modeling complex non-linear relationships between the facial appearance and the face shape. They are able to carry out several tasks at the same time like the pose estimation or a 3D shape deformable model, computing the 2D face shape as a projection of the 3D model [51, 52]. Many of these approaches use heatmaps as a probability distribution map to compute accurately the coordinates of the facial landmark points in the input image [46, 50, 53, 54].

Although in the literature, the predominant research focuses in the estimation of face landmarks from still images, a few studies have faced the problem of face landmarks tracking in videos. Some of them have focused on the temporal coherence between frames [55], others in incremental learning [33, 56], using multiple initializations [57, 58], doing tracking-by-detection [59] or exploiting a synergistic approach between tracking and detection [60].

2.3 Datasets and protocols for face alignment

There are several well-known databases related to face alignment. A list of the datasets and its characteristics is depicted in Table 1. It can be seen that some are collected under "controlled" conditions, while others contain "in-the-wild" images. They are usually labelled with manual annotations. Among all these databases, there is not a common protocol for annotations, leading to different face alignment datasets that are difficult to combine due to bias and inconsistencies over them [1, 2]. When talking about annotation protocols related to face alignment, we refer to the features of the annotated facial points set used by the databases. The protocols can contain different number of facial landmarks, different positions in the face shape, and different ways of labeling those points [61] as shown in the Fig. 4.



Fig. 3 Stage-by-stage process in the LBF method. The searching area (local region) around each landmark and the radius for each stage decreases along the cascade [18]

Facial landmark annotations are mostly based on manual work, which could lead to inaccuracies due to factors such as human fatigue or variability in high-resolution images [68, 69], semi-automatic annotations [70] or even unsupervised and automatic annotations [69, 71]. The expected error in the manual annotations oscillates between 1 and 9%, depending on the landmark point, with a consensus average of an 8%. When a landmark prediction differs from the annotations below this threshold, we can consider that it could not be distinguished from the human annotation error.

In our experiments, we focus on the collection of datasets contained in the 300-W database [27] which is essentially a standardized compilation of several others (mainly XM2VTS, FRGCv2, LFPW, AFW, Helen) with a few images in the most challenging of the conditions.

3 Robust and fast face alignment system

The current face alignment state-of-the-art has placed the automatic face landmarks detection close to the domain of the human perception in terms of accuracy [46, 48–50], especially for still images in nonchallenging conditions.

Still, the face alignment methods show several shortcomings that would need to be overcome:

- *Tracking* even if the face alignment models work very well for still images, both in constrained and unconstrained environments, the smooth tracking of facial landmarks shows problems when presented with abrupt changes on the head pose in real-time situations. In addition, issues related to the sampling rate of the camera and the time processing of the face alignment algorithm add to the problem. Although there have been promising studies and results focused on this topic [85, 86], we are still far away from the results for still images, especially when accounting for the computational issues in low power devices.
- *Pose* The pose of the head affects directly on the face alignment model performance. Even the most advanced algorithms have problems when dealing with extreme poses, especially for pitch and yaw angles.
- *Illumination* Changes in the light also affect the performance of the models due to the texture dependence of the most of the approaches.
- *Occlusion* For images in-the-wild, occlusion is an essential artifact to take into account, since it appears

Table 1 Summary of publicly available face alignment databases and the most relevant features

Name	Year	Type of data	Number images/videos	Number of facial points	Type of annotations
XM2VTSDB [72]	1999	c, i	2360	68	2D, manual
BioID [73]	2001	g, i	1521	20	2D, manual
FRGC v2 [74]	2004	c, i	50,000	5	2D, manual
Multi-PIE [3]	2008	c, i	755,370	68	2D, manual
MUCT [62]	2010	c, i	3755	76	2D, manual
CK+ [75]	2010	g/c, v	593	68	2D, automatic
3DFAW-Video [76]	2019	c,v	66	51	2D/3D, automatic
AFLW [66]	2011	c, i	21,997	21	2D, manual
LFPW [77]	2011	c, i	1432	29	2D, manual
AFW [78]	2012	c, i	205	6	2D, automatic
Helen [63]	2012	c, i	2330	194	2D, manual
Ibug 300-W [27]	2013	c, i	3837	68	2D, manual
COFW [79]	2013	g/c, i	1852	29	2D, manual
MTFL [67]	2014	g/c, i	12,995	5	2D, manual
ACVF-2014 [80]	2014	c, v	201	9	2D, automatic
Ibug 300-VW [81]	2015	g/c, v	114	68	2D, semi-automatic
MAFL [82]	2016	c,i	20,000	5	2D, manual
LS3D-W [52]	2017	g/c, i	230,000	68	3D, automatic
Dlib 5-point [42]	2017	g/c, i	7364	5	2D, manual
WFLW [83]	2018	c, i	10,000	98	2D, manual
JD-landmark [84]	2019	c, i	15,393	106	2D, manual
LaPa [64] [65]	2020	c, i	22,000	106	2D, semi-automatic

The databases designed under control conditions are listed in blue in the first part of the table, while “in-the-wild” databases are listed in orange. We use the following notations to represent different features: c: color, g: grayscale, i: images, v: videos

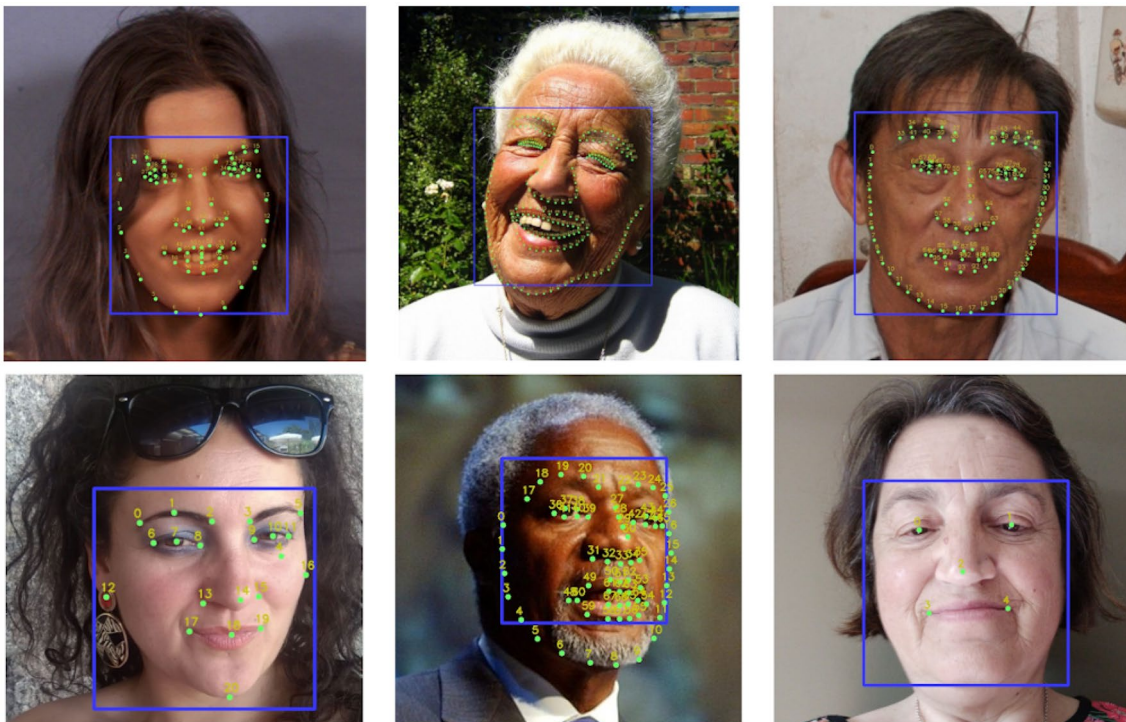


Fig. 4 Images using different annotation protocols. Some of those protocols are listed from left to right, top to bottom: MUCT (76 points) [62], Helen (194 points) [63], LaPa (106 points) [64, 65], AFLW (21 points) [66], Multi-PIE (68 points) [3] and MTFL (5 points) [67]

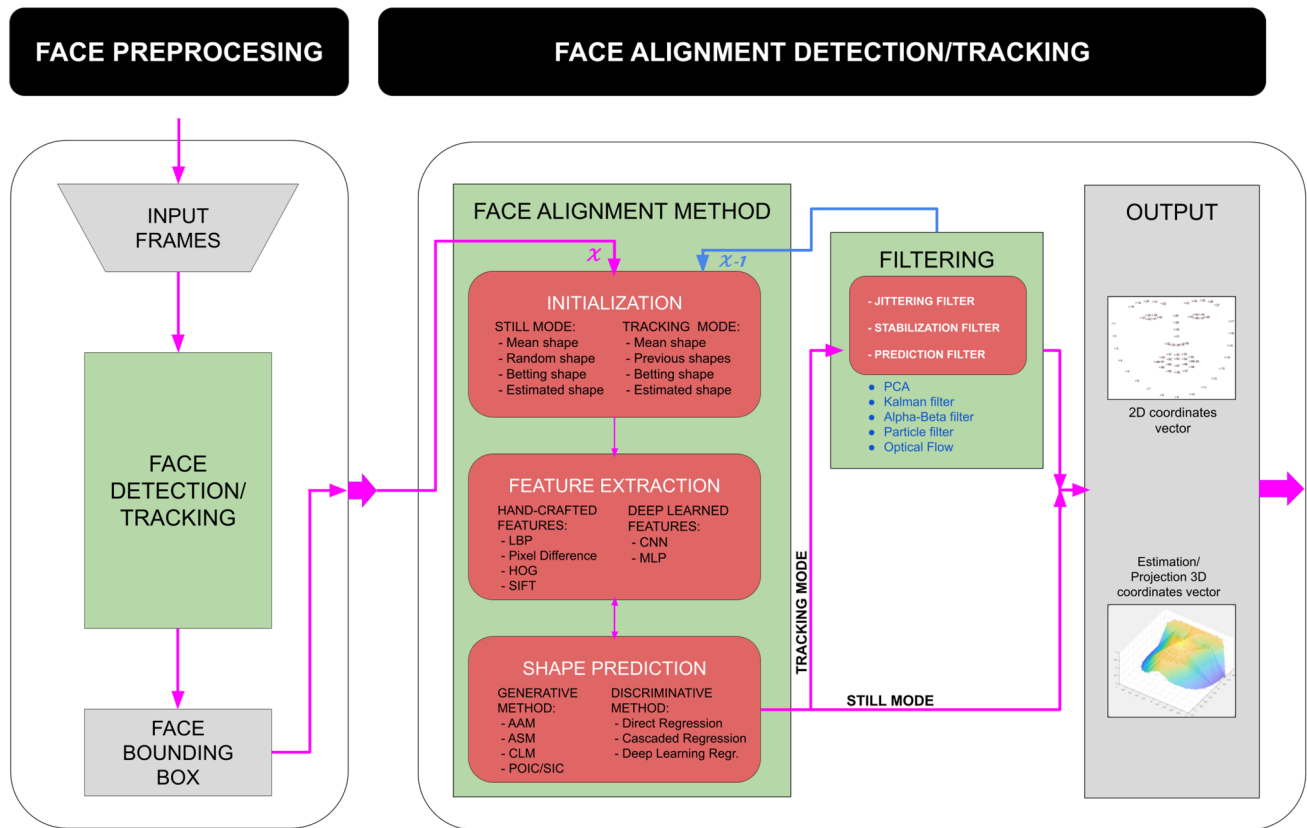


Fig. 5 System topology of a face alignment general approach, for both face landmarks detection and tracking. Green boxes represent the main modules and gray boxes represent dataflow steps

in multiple situations. Partial occlusions caused by hair, glasses, face masks, self-touching, or some external occlusion objects, hide several face regions complicating the landmark evaluation.

- **Expression Changes** in facial regions due to the influence of facial expression on the performance of the model. Several expression movements are very fast and hence difficult to track.

3.1 Face alignment pipeline

The system topology of our face alignment general approach is described as a modular system as shown in the Fig. 5, where the face analysis modules can be replaced depending on the context and target application. A modular pipeline means that individual component blocks can share semantic, spatial and temporal information between them to improve its performance, and they can be selected based on application needs.

3.2 Tracking of the facial landmarks

The proposed pipeline is designed for processing both still images and video streams. Several issues arise when using face alignment models trained on still images to perform tracking of landmarks in a sequence of frames. The main observed phenomenon is related to slight differences in the estimated landmarks between two consecutive frames practically identical. This generates an undesirable shaking effect known as jittering, that ends up showing as a pronounced shaking effect in the normalized facial shapes and textures to be used in the applications. We can tackle this problem by both improving the models during the training stage and leveraging temporal information during the inference using a filtering approach [87]. Another undesirable observed effect, especially for real-time applications, is the failure in the face landmark detection process when the head pose change in a very fast movement, due to the sudden change of the initialization conditions. We argue that many of these problems partially arise from poorly designed engineering

systems. This article tries to systematically assess some of these design challenges.

4 Evaluation metrics and benchmarks

Evaluation metrics refer to a set of measures that allow us to evaluate, quantify, and understand the performance of a model. We present several evaluation metrics, mainly focused on evaluating regression models, which we consider relevant for measuring the face alignment models performance, especially for real-time applications and models.

4.1 Standard face alignment metrics

The standard metrics for face alignment include typically those that are related to the quality of the predictions when compared to the ground truth annotations. This include mainly accuracy and error, and in some cases failure rate.

Accuracy is the fraction of predictions our model made right. Formally, the accuracy metric measures the ratio of correct predictions over all the processed data inputs. The accuracy metric is used as standard discriminative metric in the literature for classifying the performance of a model, but it suffers from presents shortcomings for understanding possible bias especially on unbalanced databases [88, 89].

Error can be measured by computing the difference between the inferred values and the ground truth values. The most commonly used error metrics are Mean Square Error (MSE), Root Mean Square Error (RMSE), Mean Absolute Error (MAE), and Mean Absolute Percentage Error (MAPE) [89, 90]. Standard evaluations in the face alignment literature are usually expressed as the point-to-point RMSE error between each point of the predicted shape and the ground truth annotations, normalized by the inter-pupil distance of the face [18]. A careful representation of the error distribution offers also a visual way of understanding the behaviour of a model.

Failure rate or *Hit rate* [91] is the percentage of face shapes with average error over or under an error threshold, respectively. They can be used as a way of knowing how many face shapes are estimated in an acceptable manner. It depends on an error threshold that can be arbitrarily set, that, in the context of face alignment, is usually set on values around 7–9%. [68, 92].

Computational efficiency refers to the time and effort required by a processor to generate a face alignment prediction. It is usually expressed with two complementary magnitudes; the time used for inferring a face shape (in milliseconds per face), and the number of the processed frames per second (fps). Both exclude all processes that are not directly related to the face alignment algorithm, such as

reading and loading images or pre-processing steps such as data augmentation.

4.2 Novel performance metrics

When applying the same face landmark detector independently to the frames of a video, sometimes an inter-frame noisy and small variation, known as jitter, is observed in the landmark points. Those keypoints do not follow well an anatomically defined point across the frames, generating an undesirable shaking effect. This effect could be produced by lack of training samples, imprecise annotations with high variance [69], unclean datasets, or non-deterministic inference methods that use some kind of (pseudo)random information during e.g. the initialization stage.

In addition, most of the face alignment algorithms rely on a face detection bounding box for shape initialization. It is well known that face detection is not always consistent in consecutive frames. The face rectangle may vary for the same image depending on the face detector used [93]. Most of the face alignment methods suffer from initialization changes leading to differences in the estimated face landmarks shape due to the aforementioned face detection jitter or different face rectangles. This is a phenomena known as Face Alignment Sensitivity [94], which describes the robustness of the model to changes in the initialization. This has been previously studied [94], by proposing the AUC_a as a metric to measure the sensitivity to the face detector rectangle. We extend the previous study by defining three complementary novel metrics, and use them to evaluate the model performance during tracking applications. We propose to compute that inter-frame error between the estimated shapes of two consecutive frames.

Landmarks Mean Squared Displacement metric (laMSD) is an adaptation of the image mean square displacement (iMSD) [95] adapted to video sequences. Measured in pixels and computed along a sequence of an arbitrary number of frames, it assumes the landmark points of the first frame as the reference, and computes the distance to the landmarks computed from the following frames.

Normalized Jitter Sensitivity Mean Square Error (NJS-MSE _{σ} ²) is a metric computed by inferring the face landmarks on a set of reference images using a number i of random variations of the ground truth face rectangle or the rectangle from the face detector used in our application for each image in the set. The random variations are based on small face rectangle center shifts in the horizontal and vertical axis. We use a high number of variations in pursuit of an error convergence, and the mean of the normalized error is then normalized by the variance, resulting in an evaluation metric to compare the jitter robustness of a face alignment model.

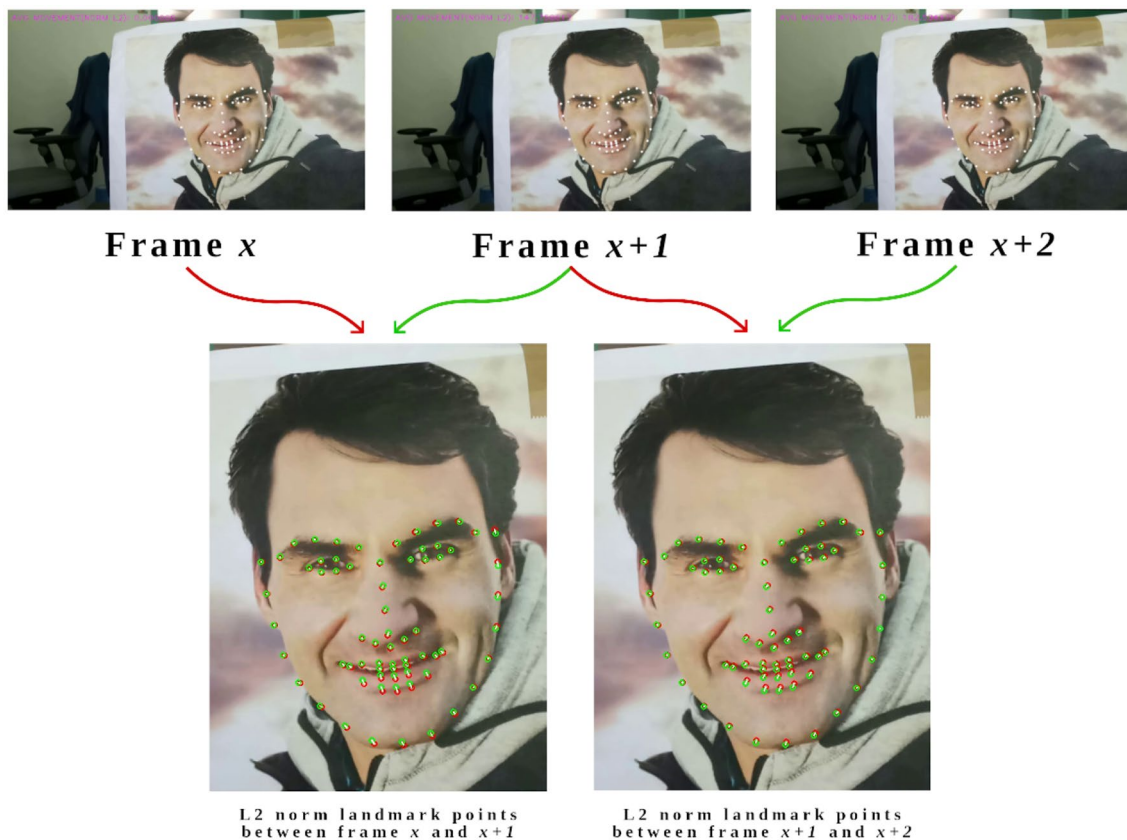


Fig. 6 Jittering benchmark flow. The L2 norm is computed between the current frame and the previous one to calculate the interframe movement distance of the facial landmarks

Normalized Face Detection Sensitivity Mean Square Error (NFDS-MSE_σ^2) is a metric computed by inferring the face landmarks on a set of reference images using a number i of random variations of the ground truth face rectangle or the rectangle from the face detector used in our application for each image in the set. For this metric, the random variations are based on face rectangle center shifts in the horizontal and vertical axis and random changes in the size (width and height) of the face bounding box. We use again a high number of variations in pursuit of an error convergence. The mean of the normalized error, is then normalized by the variance, resulting in an evaluation metric to compare the robustness of our face alignment model to different face detectors.

4.3 Benchmarks

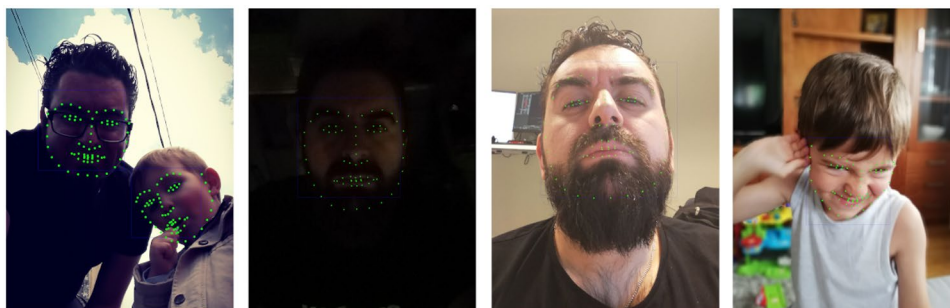
To evaluate our models, we test them on a mix of well established standard benchmarks previously proposed in the literature and a set of benchmarks carefully designed by us. Most of the related works are mainly concerned about two performance indicators, accuracy and efficiency, but we argue that they are not the only key indicators to evaluate

a face alignment model. We believe this combination of benchmarks are relevant for describing a more objective and realistic behavior in real-time applications running under unconstrained contexts. Those benchmarks are described as follows:

Common In-the-Wild benchmarks are a widely adopted set of benchmarks that focus on uncontrolled databases. In particular, they are useful to measure accuracy, error, and failure rates.

Jittering benchmarks are created ad hoc to test the accuracy of the models in pipelines implementing face tracking systems. The benchmark is designed as a set of 10 videos with 600 frames (1920×1080) where a static face (printed image) appears stuck on a stand without any face movement, as shown in the Fig. 6. The face rectangle is generated using a CNN face detector across all frames. The dimensions of the provided face bounding box in every frame is approximately 484×484 with little center shifts (between 1 and 3 pixels of difference). A small limitation of this setup is that this benchmark is not only measuring the jitter produced by the randomness in the initializations, but also due to the noise of the sensor of the camera. We think that there are not straightforward ways of avoiding this.

Fig. 7 Example images inside of the three domain-specific testsets. From left to right: strong back-light scenario, poor light scenario and extreme pitched head pose



Domain-specific benchmarks are a set of benchmarks focused on measuring the performance of the models in specifically challenging scenarios, as opposed as computing just a general mean error. In this context, we collected images to evaluate our face-alignment models in three new domain-specific testsets, as shown in Fig. 7. Each testset belongs to one of three scenarios: Strong back-light (images mostly containing frontal faces and a strong light in the back, from a natural or artificial light source and images with a low dynamic range), poor light (images with a low light condition scenario like photo-selfies taken in the night or close environments without so much illumination) and Extreme pose scenarios (containing faces looking up or down in forced poses).

Computation speed benchmarks evaluate the efficiency of face-alignment models, by computing the processing time needed for estimating the landmarks per face as main time performance metric. It is composed of a few different subsets. Assuming that a testset contains x images with only one face per image, we can compute the number of images (frames) processed per second. If the images on the testset contain more than one face per image, we can measure the time consumed per face shape estimation, and later compute the frames per second (fps). This is the mean frames per second of each testset and using still images, as used in the face-alignment literature. Apart from the measurement in still images, we can also measure the computation speed in tracking mode. In our experiments, we utilize the same video set as in the jittering benchmark.

5 Training strategies

This section presents a set of training strategies that can be generally used to train high-performance face alignment models. We detail them one by one, explaining its context, advantages, and limitations, but finally we group them in a set that will be referred throughout this article as General Training eXtensions (GTX), and this set refers to those techniques that improve the inference of our models in unconstrained environments or improve the results in the

test benchmarks. An overview of the training process can be seen in Fig. 8.

5.1 Data Augmentation based on image manipulations

Data Augmentation based on image manipulations offers a set of techniques to improve the size and quality of a training dataset. It is usually based on the generation of additional images or annotations based on transformations performed on the original dataset. Several methods for this exist:

Geometric transformations, also known as spatial transformations, are transformations of the image coordinate system. It refers to operations that transform an image using variations of the shape. Some of the most frequent geometric transformations are flipping, rotation, cropping, translation, or scaling.

Color transformation, also known as photometric transformations, are transformations over the pixels values in the matrices which compose an image, rather than the pixel positions. Some of the most frequent color transformations are changes in brightness, contrast, colorspace and normalizations.

5.2 Data Augmentation based on statistical manipulations

Statistical manipulations focus on numerical optimizations not interpreted from a visual perspective. Some algorithms are more stable working with a controlled range of numbers. Close numbers, in the sense of a continuous space with a linear scale, can produce smoother optimization paths and this can guarantee stable convergence of weights and biases [96]. Other statistical manipulations focus on finding discriminant properties through a statistical analysis to search for different classes in the database, such as discriminant power analysis (DPA) based on the DCT coefficients [97–99]. This opens the door for performing data augmentation in those classes with less representation. Statistical manipulations also refer to changes in some of the input conditions that affect the final estimation, such as noise, initializations or lighting. Some techniques are:

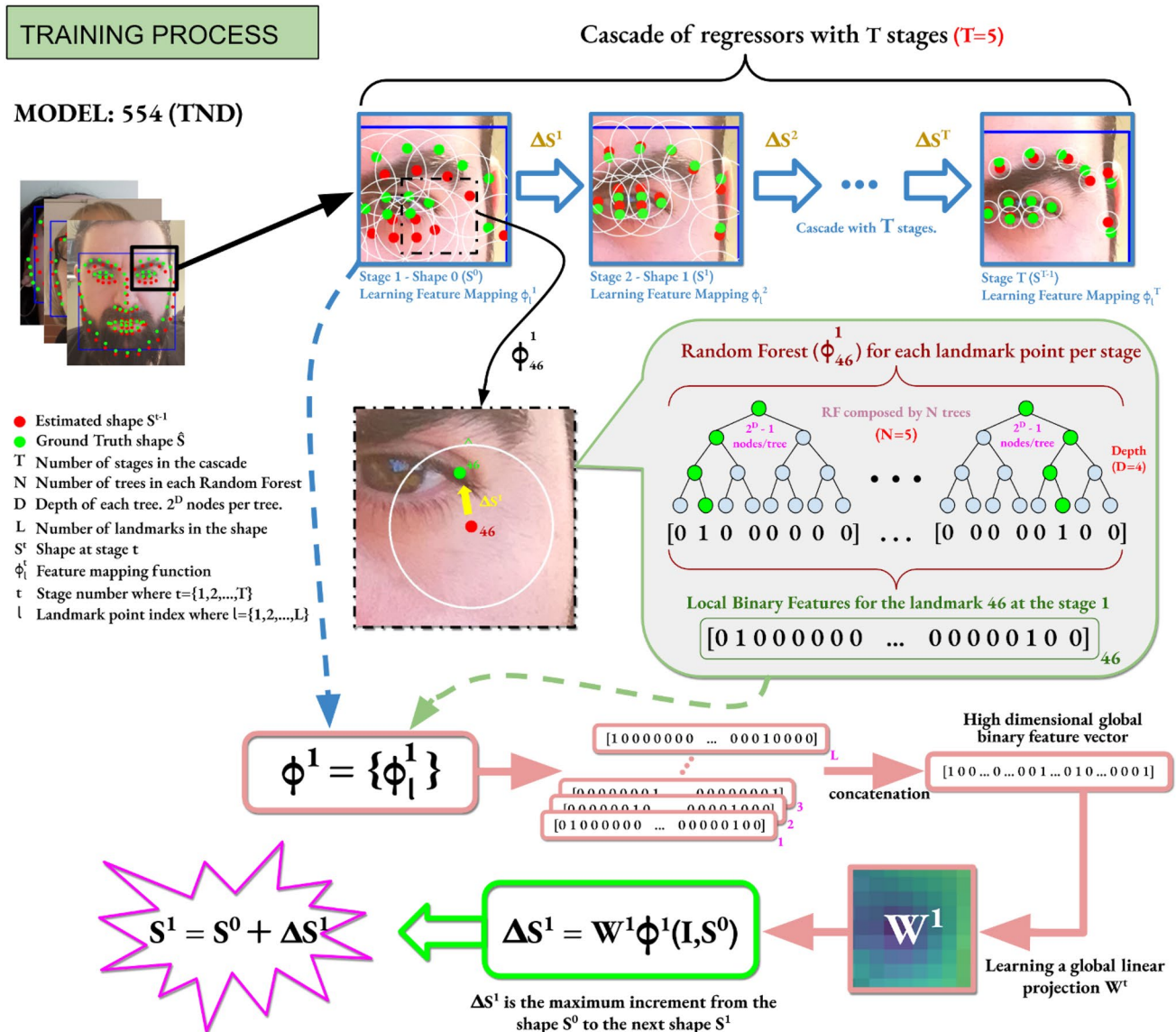


Fig. 8 Training process in the LBF method. The learning process follows a cascaded fashion topology. A linear regression of local features minimizes the distance between the current shape and the target ground truth shape

Image normalization changes the range of the pixels in an image by normalizing for a maximum and minimum value, by centering using the mean of all the channels or each individually, by scaling the pixel values to have a zero mean and unit variance or by histogram equalization to accomplish an uniform distribution of intensities. The main idea of the normalization is to enhance the discriminative information contained in the image, avoiding external components like ambient illumination changes [100].

Noise injection generates new images by adding certain amount of noise to the training dataset can lead to a better generalization error and fault tolerance by enhancing the learning capability [101]. Noise injection consists in creating a new matrix with random values following some

distribution, adding it to the original image. The effect of the noise injection ranges from a regularization effect [102, 103] to expand the samples in the dataset by adding variance and more samples, provoking that the models do not learn so much about specific training samples, making them able to learn more robust features. Adding noise to images is utilized also for simulating the noise patterns or noise jittering presented in camera sensors, or generating images in low light conditions [104].

Initialization variation consists on introducing different initializations of the landmark points from an initial shape fitted in the face bounding box given for the face detector. Initialization variations can be generated by slightly shifting the center of the face bounding boxes around and/or

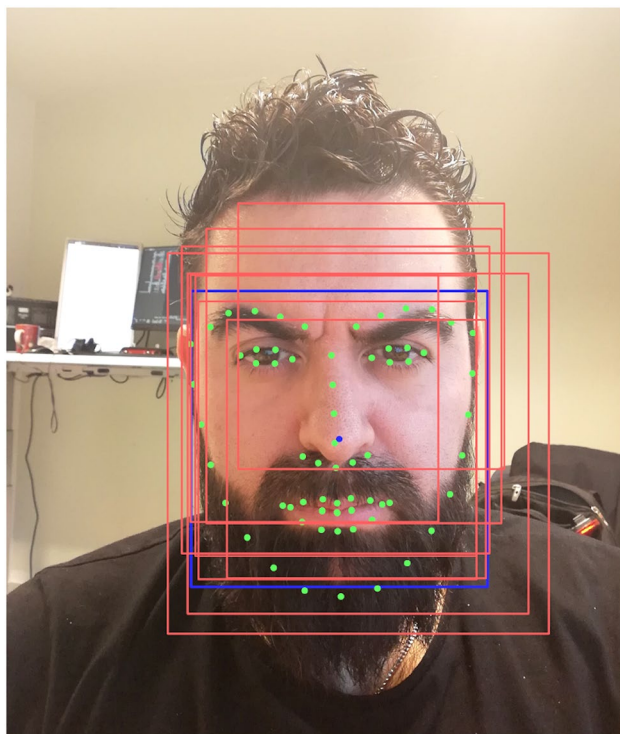


Fig. 9 Examples of image with multiple face bounding box initializations. The ground truth face rectangle is shown in blue. The random variations from the ground truth rectangle are shown in red colour

changing the size of the face rectangle. This technique has as an objective to reduce the variance caused when using different face detectors and the interframe variations caused by the noise. Initialization variations are shown in Fig. 9. To the best of our knowledge, this is the first time that experiments related to this technique are performed. Experimental results can be found in Sect. 7.

Outlier removal refers to removing the samples of the training dataset which are far away from the mean or other statistical indexes computed over the data, as a general explanation. Usually, it is a procedure carried out during the preprocessing step in the training pipeline, especially when raw data is used directly. Although we are not able to tell really which samples are outliers or not, we can set some constraints related to the ground truth data, such as, for example, limitations in the range of head angles. In our experiments, this was carried out after training the first models, and studying which samples had the biggest error, semi-automatically inspecting them and removing them from the training data.

5.3 Addition of domain specific data

To improve the model performance, it is possible to augment the training data by incorporating totally new data. In

particular, we study the collection of images that belong to specific conditions that have proven to fail in the base models. As an example, we tackle faces with extreme poses in terms of pitch (head looking up or down) and faces captured in very low illumination or against powerful backlights, presenting poor contrast and dynamic range. The images are annotated using a slow and accurate model and a teacher-student architecture. More details on the formulation of this technique can be seen in our previous publication [19]. Examples of the resulting models can be seen in Fig. 10.

6 Implementation and parallelization

This section shows a set of implementation techniques ranging from serial optimization to parallelization strategies. The solutions are analyzed later in terms of performance, and its possible accuracy trade-offs.

6.1 Reference algorithms

To evaluate our method, we focus on three face alignment methodologies that are considered to be state-of-the-art for different applications. The first model is trained using a Deep Alignment Network (DAN), one of the methods that has proven to have the best accuracy, at the expense of a relatively big computational time. The second model is based on a Cascade of Regressions of Local Binary Features (LBF). To provide a comparison and to show the validity of our method for different algorithms and models, we also train a complementary model based on an Ensemble of Regression Trees (ERT).

In this evaluation, we start from our own C++ single thread implementations of the algorithms which are loosely based on [46] (DAN), [18] (LBF) and [41] (ERT). The results obtained by our implementations closely match the ones published by the original authors of the methods.

During the experiments and evaluations, we will use the next nomenclature to refer to our models:

- $LBF(TND)_{base}$ LBF models with T stages, N number of trees per landmark per stage, and D bits of tree depth, trained using similar training strategies and data augmentation than in the original papers. The training and inference are our own C++ implementations.
- $LBF(TND)_{gtx}$ LBF models with T stages, N number of trees per landmark per stage, and D bits of tree depth, trained using the set of training strategies and data augmentation proposed in Sect. 5. The training and inference are our own C++ implementations.
- $LBF(TND)_{opencv}$ Default LBF models as implemented in the OpenCV library with T stages, N number of trees per landmark per stage and D bits of tree depth, trained

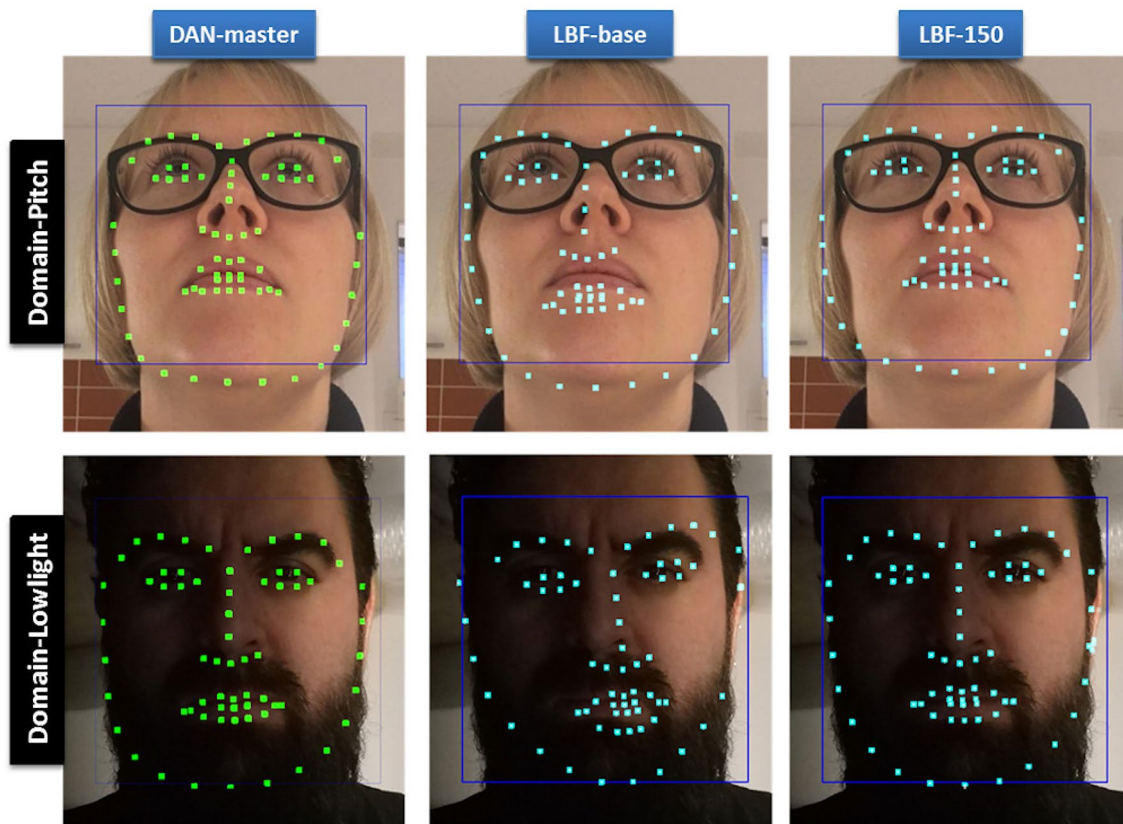


Fig. 10 Examples of images from the domain-specific testing data sets as aligned by the teacher and student models, including success and failure instances

using similar training strategies, and data augmentation than in the original papers. The inference code is the official C++ code from the open-source Face module of the Contrib module of the OpenCV library. We evaluate mainly the inference benchmarks of the default LBF565 model provided by the documentation and compare it with our own LBF565_{base} and LBF565_{gtx} models.

- ERT_{dlib} Official ERT model provided by the open-source C++ library Dlib. We use the model as provided. For the inference, we use the original Dlib C++ library. This is considered the base ERT model during all our evaluations and comparisons.
- ERT_{gtx} ERT model trained by us, by creating our own training project using the original Dlib code, but applying our set of training strategies to improve it. In the inference, we use the same code provided by the library.
- DAN model and implementation provided by the original paper in Python and C++.

6.2 Experimental setup

To carry out the implementation and evaluation experiments, we train our models using the 300W dataset and protocol

described in Sect. 1.1 and in the original publication [27]. Moreover, the computational complexity of the models and methods is measured by comparing the inference times using different hardware systems. We have divided all these systems in three groups as shown below:

For training only:

- *Server computer* A server computer with an Intel® Xeon processor with 20 cores (40 threads) at 2.2 Ghz, 250 Gigabytes of RAM, 1 terabyte Solid State Hard Drive for OS, 12 terabytes RAID Solid State Hard Drive, 4 Nvidia® Titan X and 4 Nvidia® RTX 2080, used exclusively for training and validation.

For both training and inference:

- *Laptop computer* MSI laptop with an Intel® Core i7 processor at 2.6 GHz, 16 Gigabytes of RAM, 1 terabyte Solid State Hard Drive and a Nvidia® GTX960M Graphics Processing Unit.
- *Desktop computer* personal desktop computer mainly used for testing and performing the evaluation benchmarks, but also for carrying out some basic training

attempts. It used an AMD® Ryzen 5 processor, with 16 Gigabytes of RAM, 1 terabyte SSD and a Nvidia® GTX1050.

For inference only:

- *Mobile device* Mediatek® MT6762 Helio P22 chipset with ARM Cortex-A53 Octa-core processor and 2 Gigabytes of RAM with 5 megapixels frontal camera. Used for testing the inference times of the final models.
- *Embedded system* NXP® Sabrelite board using Freescale i.MX6Q with ARMv7 Cortex A-9 mobile processor running at 792MHz and 2 Gigabytes of RAM. Used as main device for testing the inference times in low-end systems.

The development devices run Linux Ubuntu as the operating system except the smartphone, which runs Android OS. Algorithms and evaluated methods used in the experiments are implemented in C++ compiled using GCC (GNU Compiler Collection) version 7.5.0. For the implementations, we rely in a set of open-source libraries and machine learning model sources (OpenCV, DLib, LibLinear, and OpenMP). The reference slow method (DAN) also includes parts of the implementation in Python 3.6 (using Theano and Lasagne). Both training and inference implementations of the LBF algorithm were developed in C++ (using Qt Creator), mainly due to performance reasons [105, 106].

6.3 Implementation of LBF training

The training pipeline includes parameter setting, preprocessing, training, and model deployment. During the training stage, we set the parameters that define our models before the process. An example configuration can be seen in Code 1:

- *stages_n* the number of stages in the cascade, a rather critical parameter in the speed-accuracy trade-off.
- *tree_n* the number of trees per landmark per stage.
- *tree_depth* the depth of each tree. Always it is the same for every tree in the whole topology. The tree depth defines the number of nodes per tree, being $2^{tree_depth-1}$ nodes.
- *feats_m* the number of features sampled at each stage to train each split node where *feats_m* is an array setting a different number of features at each stage.
- *radius_m* searching radii around each landmark and at each stage to sample the above-mentioned feature.

```

Config::Config()
{
    dataset = "../data";
    test_input_filename = "/common_subset_test.txt";
    train_input_filename = "/base_trainset.txt";
    saved_file_name = "../model/LBF565_GTX.yaml";

    // MODEL 565
    stages_n = 5;
    tree_n = 6;
    tree_depth = 5;
    initShape_n = 10;
    bagging_overlap = 0.4;
    flip = true;
    rotation = false;
    noise = false;
    scaling = false;
    blur = false;

    int feats_m[] = { 400, 400, 400, 400, 400, 400, 30, 30, 30, 30 };
    double radius_m[] = { 0.32, 0.25, 0.18, 0.15, 0.10, 0.08, 0.08, 0.6, 1.0, 0.8 };

    number_random_faces = 10;
    landmark_n = 68;
    threshold = 0.05f;
    nbr_of_cores = 8;

    if (landmark_n == 68)
    {
        int pupils[] [6] = { { 36, 37, 38, 39, 40, 41 }, { 42, 43, 44, 45, 46, 47 } };
        for (int i = 0; i < 6; i++) {
            this->pupils[0].push_back(pupils[0][i]);
            this->pupils[1].push_back(pupils[1][i]);
        }
    }

    for (int i = 0; i < 10; i++) {
        this->feats_m.push_back(feats_m[i]);
        this->radius_m.push_back(radius_m[i]);
    }
}

```

Code 1: *Config* class in the project to set the LBF method parameters.

The rest of the parameters are related to data augmentation, preprocessing operations, error normalization, data bagging made for random forest training or for the definition of the landmarks that belong to the eye region, to perform the inter-pupil normalization.

The Pre-processing step includes several functions. The first function is a parser for reading the input data, both for training and testing models. It reads and process the annotations and ROIs of the facial images as .txt or .csv files. For performance reasons, we opt for loading all images at once in the memory, since the algorithm must access to different sections of the images for each tree.

Although the Data Augmentation step could be included as part of the pre-processing step, we describe it independently due its impact on the final performance of the models. This step implements the already defined General Training eXtensions (GTX), by using mostly OpenCV functions. After the process, we randomize the array of images before training to avoid too similar images in the same batch, since each original image could be augmented up to hundred-fold.

The Training step aims at providing the final model. The LBF method consists of a cascade of regression random forest and linear regression, where we learn “two models”, one feature mapping function to generate the local binary features, and another to regress a global linear projection using those concatenated local binary features. For the first one, we have implemented our own optimized random forest C++ class. For the global

linear regression at every stage, we have used the external library LibLinear.

After training our models, a file with extension .model is generated. It contains all information related to the model such as stages, number and depth of the trees, thresholds in the split nodes, regression weights, number of landmarks or mean shape among others. Those are the files used to test and evaluate our models before exporting them to include them in our face analysis library.

6.4 Acceleration and parallelization of the training process

To decrease the training times, and thanks to the independence of the local feature mapping functions in the training, we leverage the OpenMP library to boost the speed in the loops where the algorithm can perform each iteration independently. In particular, we parallelize the loop where the local binary features are sampled to learn the feature mapping function for each 68 landmark points, as shown in the Code 2, where we can observe the OpenMP directives.

```
void RandomForest::Train(std::vector<cv::Mat> &imgs, std::vector<cv::Mat> &gt_shapes, std
::vector<cv::Mat> &current_shapes, std::vector<BBox> &bboxes, std::vector<cv::Mat>
&delta_shapes, cv::Mat &mean_shape, int stage, int missingData)
{
    int N = static_cast<int>(imgs.size());
    double overlap_ratio = Config::GetInstance().bagging_overlap;
    int Q = int(N / ((1. - overlap_ratio) * trees_n));

    omp_set_num_threads(Config::GetInstance().nbr_of_cores);

    int i = 0;
    #pragma omp parallel shared(imgs, gt_shapes, current_shapes, bboxes, delta_shapes,
    mean_shape) private(i)
    {
        #pragma omp for
        for (i = 0; i < landmark_n; i++)
        {
            std::vector<int> root;
            for (int j = 0; j < trees_n; j++)
            {
                int start = max(0, int(floor(j*Q - j*Q*overlap_ratio)));
                int end = min(int(start + Q + 1), N);
                int L = end - start;
                root.resize(L);
                for (int k = 0; k < L; k++) root[k] = start + k;
                if (delta_shapes[0].cols > 1)
                {
                    random_trees[1][j].Train(imgs, current_shapes, bboxes, delta_shapes,
                    mean_shape, root, stage);
                }
            }
        }
    }
}
```

Code 2: Training function where the feature mapping function for each landmark point is learnt accelerated by using the *OpenMP* multiprocessing programming library. Typical *OpenMP pragmas* and definitions of the *shared* and *private* variables can be observed in the code.

In the code, several remarkable issues can be observed: one is the code where the input array of images is partitioned in batches (indexes ranges) with overlap to feed each tree at every random forest regression. Notice that we are using the same input arrays (images, bounding boxes, and shapes) for each thread, for that reason, those shared variables where

defined as shared, to avoid a private copy of each variable for each landmark training, and save memory in an already exhausted memory due to the big amount of input images.

By using OpenMP to accelerate the training stage, we have reduced the training time approximately by a factor of four ($\times 4$) in the laptop computer and a factor of ten ($\times 10$) in the server computer. In general, it is rather difficult to be precise when measuring times in multicore and multithread programming because it depends on many factors like the type of processor, scheduling algorithms, operative systems or shared data.

We have made measurements on two different models (LBF686 and LBF555 using same number of sampled features to train each split node) trained in a similar manner. We observe that the time reduction is proportional to the set number of threads while they are under the physical number of cores of the processor. When the number of threads is larger than the physical cores of the processor, the reduction decreases significantly. This is shown in the Tables 2 and 3 for two different systems.

One of the parameters with more impact on the training time of the LBF models is the number of features (feats_m) sampled to train each split node at every tree at every stage. Tables 4, shows the training times of training each landmark point. The results show four different versions of the same model LBF555_{base}, varying only the number of features among them.

Based on the result, an estimation of an LBF model training time can be done by multiplying these times by the number of training images and the number of landmarks used in the model. The same effects can be observed in both training computers.

These results are complemented by experimenting the effects of different depths and number of total training images. Table 5 shows the time needed to train four models with the same number of stages (5) and the same number of trees per random forest regression (5) but using different four depths (5, 6, 7 and 8 bits) and a growing number of input images. The rest of the parameters are kept constant. We can observe that when we increase the depth tree one bit, the training time for the same number of images increases around 35%. When we double the number of input images, the training time increases between 70 and 80%.

6.5 Implementation of LBF inference and testing

The inference and testing pipeline includes validation, visualization, exporting, and model loading of the results.

Inference and testing aim at evaluating the model. The methodology includes a validation stage after the training that measures all metrics mentioned in Sect. 4, implemented and visualized using OpenCV functions.

Table 2 Average training time in minutes per model, on an Intel® Core(TM) i7 processor at 2.6 GHz with 4 cores and 8 threads

Model	1-thread (min/model)	2-threads (min/model)	4-threads (min/model)	8-threads (min/ model)
LBF555 _{base}	87.6	47.9	26.8	21.4
LBF686 _{base}	144.1	79.5	45.1	40.4

Table 3 Average training time in minutes per model, on an Intel® Xeon processor with 20 cores (40 threads) at 2.2 Ghz

Model	1-thread (min/ model)	8-threads (min/model)	16-threads (min/model)	32-threads (min/ model)
LBF555 _{base}	96.6	14.1	2.3	1.4
LBF686 _{base}	157.6	22.7	3.5	2.1

Exporting, and model loading aim at integrating the model in the face analysis library. We have created a function to export it as a source code file to be included in the binary file of the library, instead of reading it from an external binary file of the model. There are some reasons for such decision: the first one is to make the generated files as small as possible when releasing the library. This was done by including the model data as header and source code, resulting in a reduced model size. The second reason is related to decreasing the model's memory loading times. A complementary reason is the protection of the models, since commercial releases might require to keep the model details hidden to the users.

Table 4 Average training time in milliseconds per landmarks per image, using different number of threads, in an Intel® Core(TM) i7 processor at 2.6 GHz with 4 cores and 8 threads

Sampled features per split node	1-thread (ms/land- mark per image)	2-threads (ms/land- mark per image)	4-threads (ms/land- mark per image)	8-threads (ms/ landmark per image)
100	0.99	0.55	0.32	0.29
200	1.19	0.66	0.37	0.34
400	1.57	0.86	0.48	0.41
800	2.29	1.24	0.69	0.58

Table 5 Average training time in minutes per model for four different models with different tree depths and a growing amount of input images (with and without data augmentation)

Model	300W trainset no data aug. (min/model)	300W trainset × 2 flip data aug. (min/model)	300W trainset × 6 flip + [− 45°, + 45°] rotation (min/ model)
LBF555	1.97	3.64	10.59
LBF556	2.62	4.86	13.53
LBF557	3.51	6.14	16.57
LBF558	4.79	8.12	20.95

6.6 Acceleration and parallelization of the inference process

After training our face alignment models, we obtain the mapping function to extract the features at each stage for every landmark, and a global linear projection matrix. For each tree, we obtain a set of indexes normalized between -1 and 1 , that express the appropriate geometrical transformations of the new shape. We have two options to parallelize the process; since they are independent, we can parallelize the computation for each landmark, or we can parallelize the computation for each tree, obtaining similar results for both.

In addition, we can also accelerate the multiplication of the high-dimensional features extracted in the above mentioned process and the global linear regression or transfer matrix W_l . This computation can be accelerated in two ways: by parallelization of the operations by computing the increment of each landmark in different threads, and/or by avoiding unnecessary multiplications, since those high dimensional features vectors are very sparse and hence have a majority of components equal to zero. We take advantage of this sparsity by trying several ways of avoiding multiplications, and show them in Code 3. In the Baseline, we can see how it is the normal multiplication to compute the global linear projection made in the LBF method inference. In the technique A, we avoid thousands of multiplications by checking if the feature is 0 or 1. In the technique B, we avoid even to use a statement if, by multiplying only the features which are 1. For that, we save the index where the feature vector is one when they are generated in the feature extraction step. Finally, in the technique C, we avoid the multiplication by adding the value of the transfer matrix where the feature vector is 1.

Hence, we avoid all the multiplications, and the process consists in additions to compute the increment for every landmark in the axis X and in the axis Y . Apart from this, we unroll the loop for, by reducing instructions and latencies, including the delay in reading data from memory [107].

```

short int (*transMatrix)[nbrOfFeatures] = model->stages[stage].transMatrix;
for (int l = 0; l < NBR_OF_LANDMARKS; l++)
{
    float inc_x = 0, inc_y = 0;
    short int * matrixProjX = &transMatrix[l][0];
    short int * matrixProjY = &transMatrix[NBR_OF_LANDMARKS + 1][0];

    /// BASELINE: ORIGINAL OPTION
    for(int j = 0; j < nbrOfFeatures; j++)
    {
        inc_x += model->stages[stage].features[j]*matrixProjX[j];
        inc_y += model->stages[stage].features[j]*matrixProjY[j];
    }

    /// TECHNIQUE A: AVOID MULTIPLICATIONS BY 0
    for(int j = 0; j < nbrOfFeatures; j++)
    {
        if(model->stages[stage].features[j] != 0)
        {
            inc_x += model->stages[stage].features[j]*matrixProjX[j];
            inc_y += model->stages[stage].features[j]*matrixProjY[j];
        }
    }

    /// TECHNIQUE B: MULTIPLICATION USING INDEX TO KNOW WHERE FEATURE VECTORS ARE 1
    for(int j = 0; j < NBR_TREES_PER_LEVEL; j++)
    {
        inc_x += model->stages[stage].features[model->index_vector[j]]*matrixProjX[
            model->index_vector[j]];
        inc_y += model->stages[stage].features[model->index_vector[j]]*matrixProjY[
            model->index_vector[j]];
    }

    /// TECHNIQUE C: MULTIPLY BY 1 IS NOT MULTIPLY + UNROLLING LOOP
    float dot1 = 0, dot2 = 0, dot3 = 0, dot4 = 0;
    for(int j = 0; j < NBR_TREES_PER_LEVEL; j += 2)
    {
        dot1+= matrixProjX[model->index_vector[j]];
        dot2+= matrixProjY[model->index_vector[j]];
        dot3+= matrixProjX[model->index_vector[j+1]];
        dot4+= matrixProjY[model->index_vector[j+1]];
    }
    inc_x = dot1 + dot3;
    inc_y = dot2 + dot4;
}

```

Code 3: Multiplication of the sparse linear projection W^t , computed at every stage in the face alignment process with 3 different techniques.

The difference between all these options is shown in the Table 6, where we can observe the significant improvement in terms of speed when we use option C, even with modern processors and compilers [108]. The time consumption decreases around 40% when we use the loop unwinding technique and we avoid all multiplications.

7 Comparative evaluation

This section empirically evaluates and analyses the impact of both training strategies and implementation optimization techniques in the final models, in terms of error, model size, computation time and performance in tracking mode and challenging scenarios. As a result of the GTX training strategies depicted in Sect. 5, we have obtained a set of boosted models, labelled as LBF_{gtx} models, that will be compared with the LBF_{base} models other complementary ones.

Table 6 The effect of the sparse multiplication techniques in the face alignment system

Model	Baseline (ms/face)	Technique A (ms/face)	Technique B (ms/face)	Technique C (ms/face)
$LBF555_{base}$	1.2–1.5	2.9–3.5	0.9–1.3	0.7–1.0
$LBF686_{base}$	4.4–5.1	10.5–11.8	2.6–3.2	2.4–2.9

Tests using a live webcam video stream ($640 \times 480@30fps$)

7.1 Benchmarks and real-time trade-offs

During the evaluation of our implementation and models, we have found several trade-offs that have a direct impact on the performance of our applications. They are related to the complexity of the models, the speed and/or precision of the computations, the number of accesses to the data allocated in the memory, the type of data used in the mathematical operations or the features of the hardware, among others.

The speed-accuracy trade-off is related to the accuracy of the system and its computation speed. Usually, a face alignment model is more accurate when it is more complex and includes a higher number of stages, trees per stage, depth of each tree or computed landmarks. When those parameters are larger, the complexity of the model increases, hence the inference time to detect the landmarks in a face increases too. Table 7 illustrates this trade-off.

The complexity of the model impacts on the computational time, and it is also directly linked with the size of the models. More complexity usually implies more information, and hence more memory space is needed to save and allocate it. Table 8, shows the size of every model configuration and the number of features extracted in each one to estimate the facial landmarks. Time consumption increases with the model size.

Model size plays a specially important role in low-end and embedded devices, where even the fastest of the models can compromise the real-time capabilities if implemented using nonoptimal data types.

Due to the limitations of these hardware devices (low power processors, low amount of cache memory, reduced memory bandwidth), floating-point numbers increase the time of computation, the memory access time [109] and the overhead in the computations [110]. A practical solution is the quantization of the models, to smaller data types [111].

We quantized the model applying Rounding Quantization [112] and fit the floating-point range in a short integer data type. We analyzed the impact of this quantization in a low-end device (LG k40 and NXP[®] Sabrelite board) and show it in Table 9. Quantization speeds up the inference in a noticeable manner, especially in low-end devices, without impacting heavily on the accuracy.

Table 7 Trade-off between average error, failure rate and computation times of different models in a laptop computer

Model	300W-common		300W-fullset		Speed test
	avg. er. (%)	fail.rate (%)	avg. er. (%)	fail.rate (%)	CPU (ms/face)
LBF554	6.70	18.59	10.62	31.93	1.1
LBF664	6.41	17.68	10.13	30.76	1.5
LBF645	6.31	16.06	10.11	28.73	1.6
LBF555 _{base}	6.54	16.96	10.35	29.61	1.6
LBF555 _{dnf}	6.52	17.49	10.36	30.33	1.6
LBF555 _{gtx}	6.24	14.62	8.93	27.86	1.6
LBF674	6.21	14.07	9.72	27.57	1.7
LBF565 _{gtx}	5.90	12.7	8.61	25.69	2.0
LBF547	6.73	18.05	10.85	31.20	2.7
LBF686 _{base}	6.02	11.55	9.57	24.67	5.1
LBF686 _{gtx}	5.57	8.75	7.72	21.77	5.1
LBF786	5.55	9.22	7.90	21.77	6.1
LBF698	5.99	12.27	7.81	23.80	7.2

In bold, best result

7.2 Jittering evaluation

We evaluated the inter-frame behavior of several LBF models, comparing base models with the ones trained using our GTX training strategies. The results are expressed using both the total amount of jitter in pixels and the laMSD metric, using the specific benchmarks previously described. We show the results in Table 10. We argue that the jittering effect is related to the generalization of the model, and it is interdependent with the number of images used to train them and the complexity of the model. We have also empirically observed that the parameters of the regularization function when training the global regression have impact in the amount of jittering in the trained model. The results show that our GTX models outperform base models in a consistent manner.

7.3 Impact of LBF parameters on model performance

To understand the optimal face alignment parameter and model selection, we extensively review next some results related to the topology and performance. The LBF algorithm, based on a cascade of regressions is defined by a number of stages (T) that affects to the complexity, and as a consequence, the speed and accuracy performance. We review the accuracy performance in terms of average error and failure rate stage by stage, to know how our models perform stage by stage and show them in Tables 11 and 12. We observe that the higher accuracy increment happens in the first two or three stages. Also we can observe that the time processing per stage is rather similar, because the feature mapping functions have the same size for every

stage, hence, the algorithm searches the same number of features at every stage.

When plotting the accuracy and the failure rate for several models, as shown in Fig. 11, we can see that the behaviour is common to all of them, and that both start decreasing until achieving almost asymptotically.

7.4 LBF error distributions

We believe that the average error does not sufficiently explain how the model is failing, specially since we can consider that individual errors under 8%, could be indistinguishable from human annotation errors. Figure 12, shows plots of the error distributions for two different LBF686 models and two different LBF555 models. We can observe the improvement between two different model configurations

Table 8 Sizes and dimensionality of the extracted features for several LBF models

Model	Model size (MB)	Features number	CPU (ms/face)
LBF-554	19.9	13,600	1.1
LBF-664	28.8	19,584	1.5
LBF-674	33.6	22,848	1.7
LBF-645	38.5	26,112	1.6
LBF-555 _{base}	40.1	27,200	1.6
LBF-555 _{dnf}	40.1	27,200	1.6
LBF-555 _{gtx}	40.1	27,200	1.6
LBF-547	128.5	87040	2.7
LBF-686 _{base}	154.2	104,448	5.1
LBF-686 _{gtx}	154.2	104,448	5.1
LBF-786	179.7	121,856	6.1
LBF-698	694.2	470,016	7.2

Table 9 Impact of quantization in terms of accuracy (300W-common dataset), speed and size for different hardware devices

Hardware	LBF555 _{float}		LBF555 _{quantized}	
	avg. error	ms/face	avg. error	ms/face
Laptop (Intel Core i7)	6.49%	1.3–1.9	6.54%	0.7–1.0
Desktop (AMD Ryzen 5)	6.49%	1.1–1.5	6.54%	0.6–0.9
LG K40 (ARM Cortex A53)	6.49%	14.7–21.4	6.56%	2.2–6.8
NXP (ARM Cortex A9)	6.51%	24.7–38.2	6.54%	2.9–7.8
Size (MB)	40.1		17.6	

In bold, best result

Table 10 Amount of total jitter and jitter per frame in our self-designed Jittering benchmark ordered by growing laMSD per frame

Model	Total jitter (pixels)	laMSD (pixels/frame)	Frame rate (fps)
LBF-686 _{gtx}	75,177.78	142.92	217.8
LBF-565 _{gtx}	100,903.73	191.83	605.7
LBF-555 _{gtx}	113,003.38	212.44	731.9
LBF-645	121,170.13	230.36	775.0
LBF-686 _{base}	124,206.93	236.13	218.6
LBF-674	136,166.44	258.87	612.8
LBF-565 _{opencv} [18, 40]	157,653.44	299.72	742.3
LBF-664	169,967.92	323.13	763.8
LBF-555 _{base}	171,945.42	326.89	731.9
LBF-554	203,154.56	386.23	1098.0
LBF-698	271,452.04	516.07	138.8

In bold, best result

(LBF686 vs LBF555—Left vs Right), and the improvement between two different versions of the same model configuration (LBF555_{base} vs LBF555_{gtx} and LBF686_{base} vs LBF686_{gtx}—Up vs Down). In general, similar size models benefit from training strategies and data augmentations in a consistent manner, by showing less errors in the higher part of the distribution.

Table 11 Average error, failure rate and average time consumed per stage by the LBF686_{base} model using the 300W-fullset and 300W-common subsets to evaluate it

Stage	300W-common		300W-fullset		Avg. Comp. Time CPU (ms/stage/face)
	avg. er. (%)	fail.rate (%)	avg. er. (%)	fail.rate (%)	
Initial	25.63	100.0	32.77	100.0	–
1	12.70	83.03	17.07	86.06	1.0
2	8.67	41.15	12.66	52.52	0.9
3	7.18	21.84	10.99	34.83	0.9
4	6.56	17.14	10.25	29.60	0.9
5	6.22	14.44	9.84	27.14	0.9
6	6.02	11.55	9.57	24.67	0.9

7.5 Sensitivity evaluation

To evaluate the sensitivity of the models to external components, we have computed the NJS-MSE_σ² and NFDS-MSE_σ² values according to our defined metrics in two different benchmarks, one using frontal faces, and another one using challenging faces with low light conditions and extreme head poses. Table 13 shows the results for both benchmarks and both metrics, where higher numbers mean better results.

For a better understanding of the metric, we provide the values of a simulated landmark detector with different amounts of jittering, created by adding variations to the ground truth shape. We have added different percentages over each ground truth landmark point.

Most of the real models show base jitter values equivalent to jitter between 0.5 and 2% in frontal faces, while the jitter due to the changes in the face detector range in values equivalent to 2% and 10% of jittering.

In general, for frontal faces, DAN has the best performance in both metrics as expected by a slow non-real-time model. Among the fast models, the LBF-686_{gtx} model has the best performance in both benchmarks for frontal and challenging images. In general, the GTX models work better for the metric related to the face detector sensitivity. This is expected since they are trained using random variation of the face detection initializations and learn to be robust against these changes.

7.6 Domain specific performance using Teacher-Student annotations

Our training strategies included a solution to annotate a small subset of facial images belonging to challenging domains, utilizing a very slow but accurate model (DAN) to annotate challenging data for us that could be included in the training.

The comparative analysis of the experiments with the base and improved student models can be seen in Table 14. It can be seen that the teacher model, based on DAN, has

Table 12 Average error, failure rate and average time consumed per stage by the LBF555_{base} model using the 300W-fullset and 300W-common subsets to evaluate it

Stage	300W-common		300W-fullset		Avg. Comp. Time CPU (ms/stage/face)
	avg. er. (%)	fail.rate (%)	avg. er. (%)	fail.rate (%)	
Initial	25.12	100.0	30.94	100.0	–
1	13.28	84.84	17.76	87.81	0.4
2	9.39	53.43	13.48	61.54	0.4
3	7.66	28.88	11.63	40.49	0.4
4	6.94	21.11	10.80	33.53	0.4
5	6.54	16.96	10.35	29.61	0.3

execution times that are not suitable for real-time computation when using a CPU. Both the base and improved models are from 100 to 300 times faster on desktop and mobile CPUs. As expected, the inclusion of the proposed training data does not have an impact on the computation time, since the models are not fundamentally changed and only the weights of the model vary with the new data.

When observing the error of the student models (LBF and ERT) on both Domain-Pitch and Domain-Lowlight testing sets, it can be seen that the average error is reduced from 15 to 30% adding as little as a 5% of data (150 images). An important characteristic of the training is that the proposed scheme can be utilized without any loss of generalization abilities for the more typical data, as it can be seen in Table 14. The results obtained are comparable for LBF and ERT models, showing applicability for multiple real-time. Additional experiments can be found in our previous work [19].

7.7 Qualitative results

In addition to quantitative experiments, we show examples of the qualitative performance of the models and implementations. We show examples from both common in-the-wild databases and our own benchmark datasets.

The performance of the most of the models and algorithms for frontal faces is most of the times excellent, as shown in Fig. 13, and we can assume that most of the models will perform reasonably good in optimal conditions. We can find slight differences among them, but mostly in little details of the eyes, lips or nose. Deep Alignment Network (DAN) models show the best results also when inspecting the quality of the landmarks, it estimates better all contours and definitions of the face parts. The differences between the ERT and LBF methods are minimal, and sometimes the estimation is marginally better in one than the other.

In challenging images, such as the ones included in the domain-specific benchmarks or the 300W challenging dataset, the differences are more noticeable. Figure 14 shows estimations of several challenging images. Again, the slow DAN method outperforms the others, being

highly accurate in all examples. However, we can observe how the data augmentation and training strategies improve substantially the accuracy of the real-time LBF and ERT models, as we can appreciate when observing regions such as the lips, eyes or face contours.

As the main observation, we can observe how the qualitative performance of the LBF686_{gtx} model is very close to the DAN method, but it is approximately $\times 200$ faster. This shows the applicability of our techniques for training and implementation to carry out fast but accurate fast alignment, directly usable in multiple applications and devices.

Despite the improvement show, some issues in challenging images remain. Figure 15, shows images with challenging issues that still are not solved in real-time face alignment. Occlusions, extreme head poses, make up or very dark scenarios where humans can predict well the shape, are still unsolved. Again, the DAN method can be considered to outperform the fast methods, although it also fails in the most challenging images.

For example, the first image shows that the best performance corresponds to fast LBF and ERT models trained using GTX, while the base ERT_{dlib} model and the LBF565_{opencv} included in the common libraries fail. This suggests that there is a clear benefit in using models trained with our proposed strategies, directly usable by the same libraries.

In general, the base models are the worst ones in almost all scenarios. The ERT_{dlib} and LBF565_{opencv} models and implementations have similar performance, only slightly better than the BASE ones with a small advantage for ERT_{dlib}. All of them show worse performance when compared to GTX models, in almost all metrics and scenarios.

8 Discussion and conclusion

The results of this article show the impact of a set of optimization and training strategies in the context of a face alignment system that is intended to be integrated in a solution that runs in real-time, both in desktop computers and mobile devices. Without any algorithmic development, we show how clever strategies for training, implementation

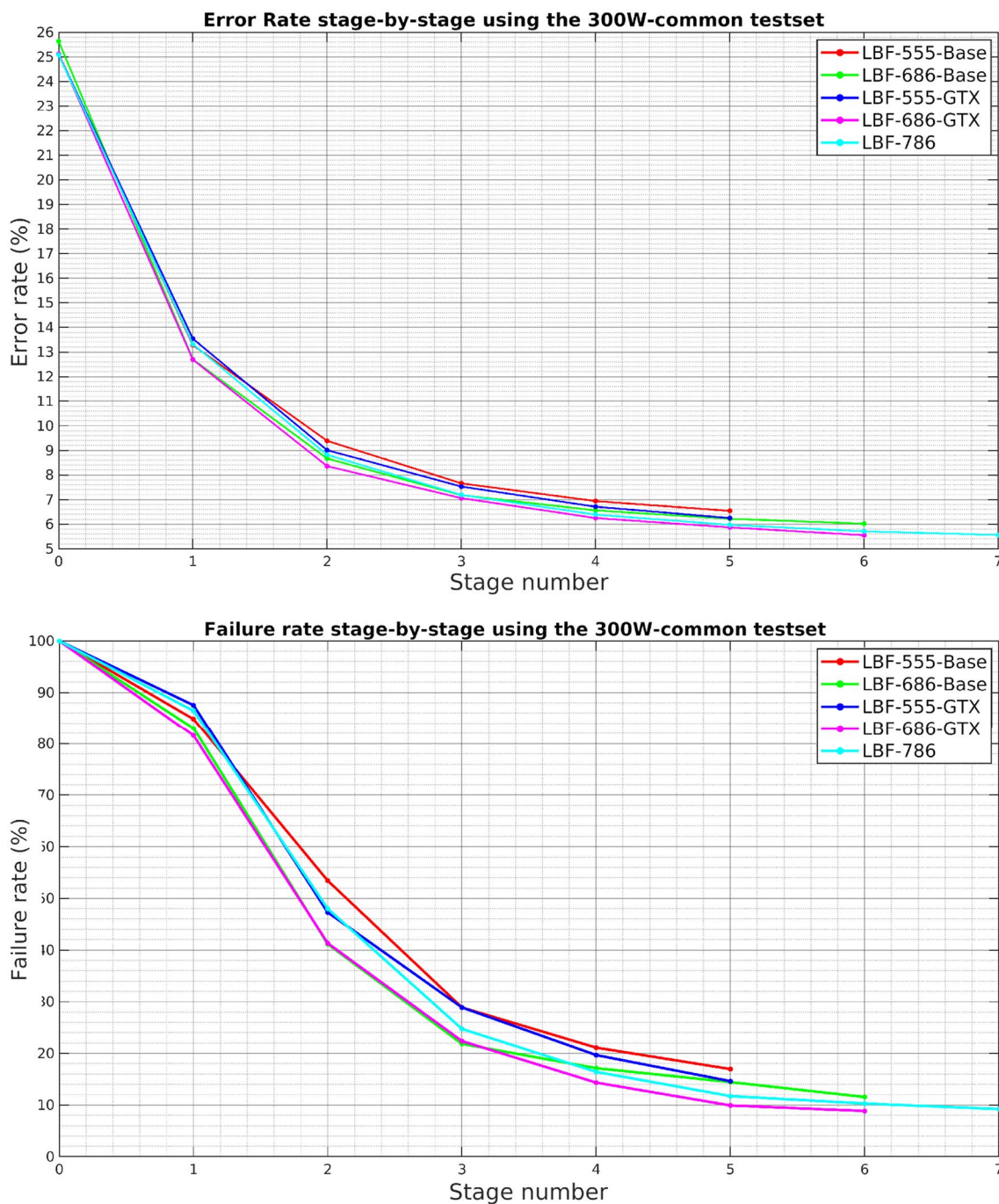


Fig. 11 Average error and failure rates of several LBF models stage-by-stage using the 300W-common testset

and system design can show tremendous impact on the performance of the models in terms of accuracy, speed, model size, or failure rate in challenging conditions.

We proposed a new set of specific metrics and benchmarks which focus on the jittering of landmark points that happens in consecutive frames and the performance in domain-specific scenarios. We proposed a set of different training strategies that directly impact the accuracy of the

face alignment models by incorporating data augmentation techniques that improve both the standard accuracy metrics and other important features of the models. We have shown several optimization techniques that can be directly applied to the inference stage, incorporating quantization of the models that impact especially the performance on embedded devices.

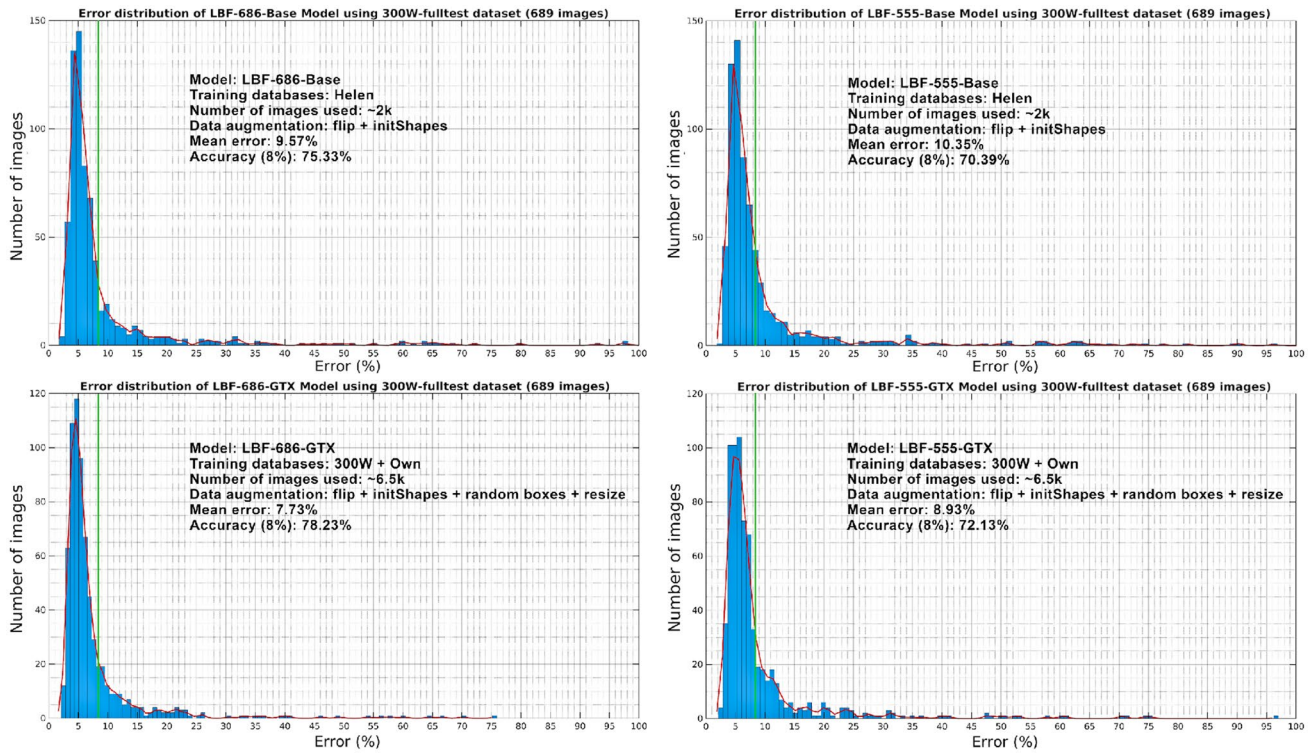


Fig. 12 Error distribution for the 300W-fulltest set using two different model configuration (LBF686 and LBF555), and two different versions of the two configuration (Base and GTX)

Table 13 Jitter and face detector sensitivity benchmarks

Model	Frontal faces test		Challenging faces test	
	NJS-MSE $^2_{\sigma}$	NFDS-MSE $^2_{\sigma}$	NJS-MSE $^2_{\sigma}$	NFDS-MSE $^2_{\sigma}$
SIM (0.5%)	0.2155	0.2158	2.1419	2.1387
SIM (1.0%)	0.1076	0.1079	1.0641	1.0672
SIM (1.5%)	0.0712	0.0716	0.7117	0.7166
SIM (2.0%)	0.0536	0.0536	0.5357	0.5355
SIM (2.5%)	0.0430	0.0433	0.4297	0.4308
SIM (5.0%)	0.0215	0.0214	0.2144	0.2154
SIM (10%)	0.0108	0.0107	0.1070	0.1067
DAN [46]	0.1668	0.0404	0.1781	0.0330
ERT _{gtx}	0.1663	0.0252	0.2430	0.0478
LBF-686 _{gtx}	0.1582	0.0556	0.3998	0.0849
ERT _{dlib} [41, 42]	0.1442	0.0096	0.3359	0.0568
LBF565 _{gtx}	0.1213	0.0321	0.2856	0.0859
LBF565 _{opencv} [18, 40]	0.1179	0.0113	0.2913	0.0496
LBF-686 _{base}	0.1055	0.0177	0.3679	0.0836
LBF-555 _{gtx}	0.1027	0.0390	0.3777	0.0919
LBF-555 _{base}	0.0930	0.0150	0.2897	0.0848

We provide a simulation results with different amount of jittering variations over the ground truth landmark points, to have a scale to quantify the evaluated models, In bold, best result

Table 14 Average error (%) and failed images (%) on the 300W-fullset and domain-specific testing subsets

Method	300-fullset		Domain Pitch		Domain Lowlight	
	avg. er. (%)	fail.rate (%)	avg. er. (%)	fail.rate (%)	avg. er. (%)	fail.rate (%)
DAN _{teac.} [46]	5.03	1.16	3.89	0	4.01	0
LBF _{base}	8.2	23.0	25.5	100	19.7	100
LBF-150	8.2	23.0	18.2	76	15.4	70
LBF-300	8.3	23.5	16.5	67	14.8	66
LBF-450	8.2	23.8	15.4	60	14.2	63
LBF-600	8.4	23.5	13.8	52	13.6	54
ERT _{base}	8.0	22.9	24.2	100	19.8	100
ERT-150	7.9	23.1	18.1	84	17.9	77
ERT-300	8.0	23.3	16.4	81	17.5	72
ERT-450	8.0	23.6	13.7	67	16.7	67
ERT-600	8.2	23.3	13.4	56	16.0	59

In bold, best result

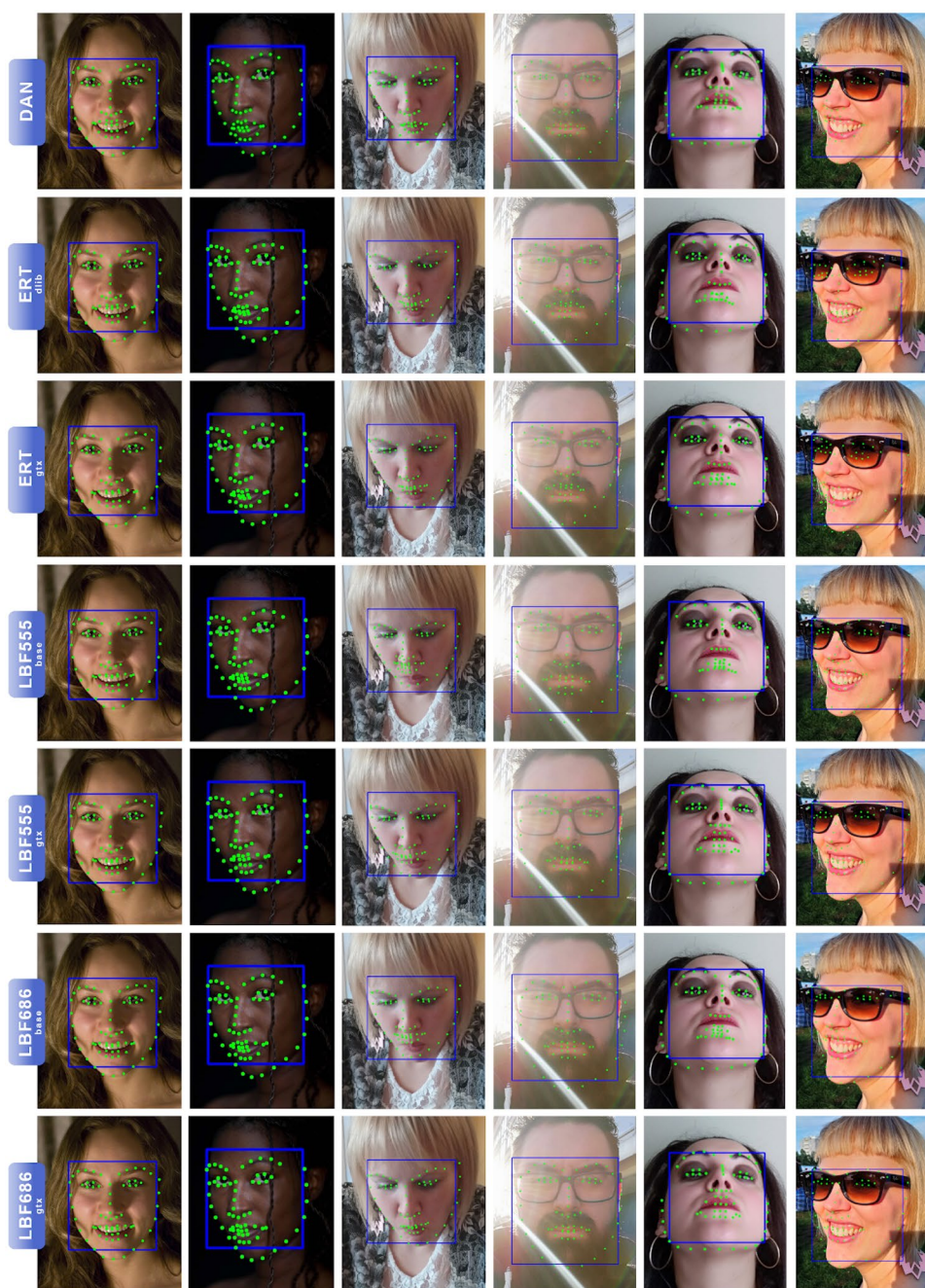
**Fig. 13** Inferences from DAN, ERT and LBF methods for images with frontal faces from our own testsets with and without GTX training

The resulting code and models are implemented and released using standard software platforms (Dlib, Opencv) that facilitate the adoption of these techniques and the direct utilization and retraining of similar models by other

researchers and industry practitioners making easier to understand the type of improvements that one could expect.

Although the proposed techniques here have direct application and show a framework to improve real-time face

Fig. 14 Inferences from DAN, ERT and LBF methods for some challenging images from 300W testsets and own datasets. We can observe images belonging to different specific-domain scenes. The effects of training strategies show results close to the slower models



alignment models based on cascaded regression methods, some open questions and limitations still remain.

There is a lack of face alignment algorithms using temporal information to track face landmarks in video streams, especially for real-time applications in low-end devices. The temporal information can help to improve the inference in consecutive frames, reducing artifacts. Widely used benchmarks are lacking of a well-defined sets aimed at measuring the jittering of face alignment tracker systems.

Exploiting the spatial relationships among landmark points, taking advantage of their relations due to, e.g., face symmetry, could lead to improving the speed, performance and parallelization in dedicated hardware, by reducing the size of the face alignment models or the amount of estimations, since these models rely on independent landmark regressions that have shown to be highly parallelizable.

We have some intuitions related to the learning capacity of the face alignment methods based on cascaded regressors. It is not clear yet how much the learning capacity depends

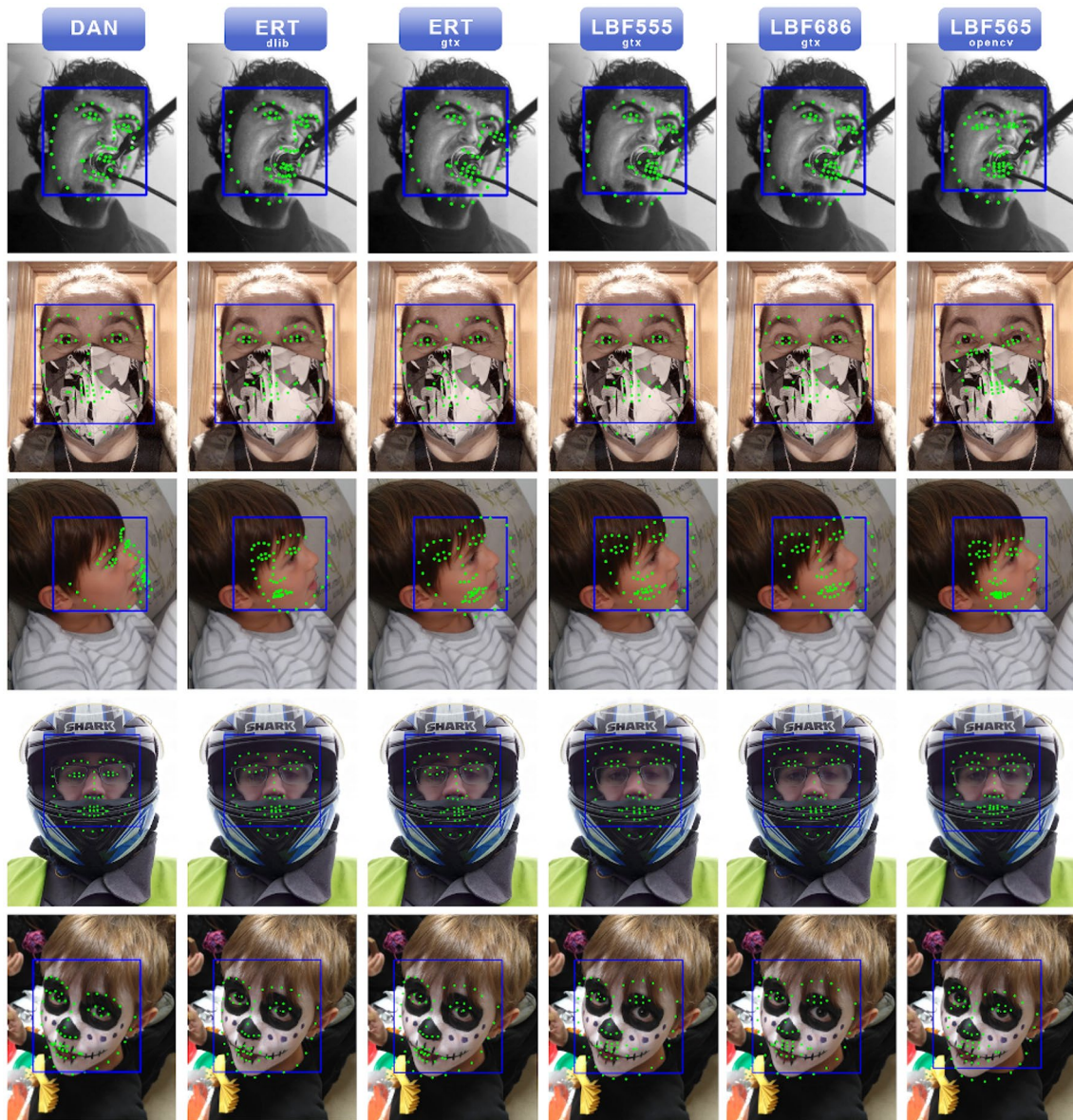


Fig. 15 Inferences from DAN, ERT and LBF methods for some images containing some of the main problems that the face alignment researching have to face: occlusions, lateral faces, extreme expressions, facial masks, etc

on the configuration of the model or on the topology of the algorithm. We think this is an interesting new horizon to explore and understand if we have a trade-off between complexity of the model and the amount of data in the training stage. In addition and related to the learning capacity, we have found a relationship between the amount of data, complexity, regression parameters, and jittering. We think that an exhaustive quantitative evaluation is still needed to better understand the learning process.

A critical issue that requires further evaluation, is the idea of classifying if an inference made by a face alignment

model is adequately accurate. We have explored some ideas to try to classify the quality of the inference by using a SVM classifier that leverages the same Local Binary Features used to estimate the face shape, or exploring the mirrorability property proposed by [93]. Unfortunately, we have not found an obvious and consistent technique to do it, or found any well-tested proposal in the literature. Some methods, especially the ones based on neural networks, offer confidence values, but its results are not always trustworthy.

In its current state of research, it is difficult to pinpoint a specific “best” model since numerous trade-offs

among different metrics have to be taken into account. Parameters such as the number of stages in the cascade, the number and depth of the trees, still need to be carefully selected based on the specific application needs. All these issues are still of importance, but lay beyond the scope of this article and should be studied in further future work.

Funding Open access funding provided by University of Oulu including Oulu University Hospital.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Jin, X., Tan, X.: Face alignment in-the-wild: a survey. *Comput. Vis. Image Understand.* **08** (2016)
- Wu, Y., Ji, Q.: Facial landmark detection: a literature survey. *CoRR* (2018). [arXiv:1805.05563](https://arxiv.org/abs/1805.05563)
- Gross, R., Matthews, I., Cohn, J., Kanade, T., Baker, S.: Multiple. *Image Vis. Comput.* **28**(5), 807–813 (2010)
- Haider, K., Malik, K., Khalid, S., Nawaz, T., Jabbar, S.: Deepgender: real-time gender classification using deep learning for smartphones. *J. Real Time Image Process.* **16**, 02 (2019)
- Tian, Q., Zhang, W., Mao, J.-X., Yin, H.: Real-time human cross-race aging-related face appearance detection with deep convolution architecture. *J. Real Time Image Process.* **17**, 02 (2020)
- Kortli, Y., Jridi, M., Atri, M.: Face recognition systems: a survey. *Sensors* **20**(342), 01 (2020)
- Júlio, C.S., Júnior, J., Güçlütürk, Y., Pérez, M., Güçlü, U., Andújar, C., Baró, X., Escalante, H.J., Guyon, I., van Gerven, M.A.J., van Lier, R., Escalera, S.: First impressions: a survey on computer vision-based apparent personality trait analysis. *CoRR* (2018). [arXiv:1804.08046](https://arxiv.org/abs/1804.08046)
- Ekman, P.: Darwin and Facial Expression: A Century of Research in Review. Ishk, California (2006)
- Ekman, P.: Darwin's contributions to our understanding of emotional expressions. *Philos. Trans. R. Soc. B Biol. Sci.* **364**(1535), 3449–3451 (2009)
- Robinson, P., Kaliouby, R.: Computation of emotions in man and machines. *Philos. Trans. R. Soc. B Biol. Sci.* **364**:3441–3447, 12 (2009)
- Li, H.Z.: Nonverbal Communication and Culture. American Cancer Society, Atlanta, pp. 1–7 (2015)
- Thevenot, J., López, M.B., Hadid, A.: A survey on computer vision for assistive medical diagnosis from faces. *IEEE J. Biomed. Health Inform.* **22**(5), 1497–1511 (2018)
- Jaimes, A., Sebe, N.: Multimodal human–computer interaction: a survey. *Comput. Vis. Image Understand.* **108**(1):116–134 (2007) (Special Issue on Vision for Human-Computer Interaction)
- Thomaz, A., Hoffman, G., Cakmak, M.: Computational human-robot interaction. *Found. Trends Robot.* **4**:104–223, 01 (2016)
- Suen, H.-Y., Hung, K.-E., Yu-Sheng, S.: Predicting behavioral competencies automatically from facial expressions in real-time video-recorded interviews. *J. Real Time Image Process.* **01** (2021)
- Kamenskaya, E., Kukharev, G.: Recognition of psychological characteristics from face. *Metody Informatyki Stosowanej*, nr **1**(Tom 13):59–73 (2008)
- Egger, M., Ley, M., Hanke, S.: Emotion recognition from physiological signal analysis: a review. *Electron. Notes Theor. Comput. Sci.* **343**:35–55 (2019) (The proceedings of AmI, the 2018 European Conference on Ambient Intelligence)
- Ren, S., Cao, X., Wei, Y., Sun, J.: Face alignment at 3000 fps via regressing local binary features. In: 2014 IEEE Conference on Computer Vision and Pattern Recognition, pp. 1685–1692 (2014)
- Casado, C.Á., López, M.B.: Face alignment: improving the accuracy of fast models using domain-specific unlabelled data and a teacher–student scheme. *Electron. Lett.* **55**(11):646–648 (2019)
- Johnston, B., Chazal, P.: A review of image-based automatic facial landmark identification techniques. *EURASIP J. Image Video Process.* **86**(09), 2018 (2018)
- Celiktutan, O., Ulukaya, S., Sankur, B.: A comparative study of face landmarking techniques. *EURASIP J. Image Video Process.* **1–27**, 2013 (2013)
- Turk, M.A., Pentland, A.P.: Face recognition using eigenfaces. In: Proceedings. 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 586–591, June (1991)
- Sánchez-Lozano, E., Tzimiropoulos, G., Martinez, B., De la Torre, F., Valstar, M.: A functional regression approach to facial landmark tracking. *IEEE Trans. Pattern Anal. Mach. Intell.* **40**(9), 2037–2050 (2018)
- Cootes, T.F., Taylor, C.J., Cooper, D.H., Graham, J.: Active shape models-their training and application. *Comput. Vis. Image Understand.* **61**(1), 38–59 (1995)
- Cootes, T.F., Edwards, G.J., Taylor, C.: Active appearance models. *Pattern Anal. Mach. Intell. IEEE Trans.* **23**:681–685, 07 (2001)
- Edwards, G.J., Taylor, C.J., Cootes, T.F.: Interpreting face images using active appearance models. In: Proceedings Third IEEE International Conference on Automatic Face and Gesture Recognition, pp. 300–305, April (1998)
- Sagonas, C., Tzimiropoulos, G., Zafeiriou, S., Pantic, M.: 300 faces in-the-wild challenge: the first facial landmark localization challenge. In: 2013 IEEE International Conference on Computer Vision Workshops, pp. 397–403, Dec (2013)
- Alabort-i-Medina, J., Zafeiriou, S.: A unified framework for compositional fitting of active appearance models. *CoRR* (2016). [arXiv:1601.00199](https://arxiv.org/abs/1601.00199)
- Baker, S., Gross, R., Matthews, I.: Lucas-kanade 20 years on: a unifying framework: part 3. *Int. J. Comput. Vis.* **56**, 12 (2003)
- Matthews, I., Baker, S.: Active appearance models revisited. *Int. J. Comput. Vis.* **60**, 03 (2004)
- Cristinacce, D., Cootes, T.: Feature detection and tracking with constrained local models. In: *Bmvc*, vol. 41, pp. 929–938. Citeseer, 01 (2006)
- Maria, M.V., Tavares, J.M.R.S.: Methods to automatically build point distribution models for objects like hand palms and faces represented in images. *CMES* **36**, 213–242 (2008)

33. Asthana, A., Zafeiriou, S., Cheng, S., Pantic, M.: Incremental face alignment in the wild. In: 2014 IEEE Conference on Computer Vision and Pattern Recognition, pp. 1859–1866, June (2014)
34. Liew, C.F., Yairi, T.: Robust face alignment with random forest: analysis of initialization, landmarks regression, and shape regularization methods. *IEICE Trans. Inf. Syst.* **99-D**:496–504 (2016)
35. Dantone, M., Gall, J., Fanelli, G., Gool, L.V.: Real-time facial feature detection using conditional regression forests. In: 2012 IEEE Conference on Computer Vision and Pattern Recognition, pp. 2578–2585, June (2012)
36. Yang, H., Patras, I.: Face parts localization using structured-output regression forests. In: Lee, K.M., Matsushita, Y., Rehg, J.M., Hu, Z. (Eds.) *Computer Vision—ACCV 2012*. Springer, Berlin, pp. 667–679 (2013)
37. Cao, X., Wei, Y., Wen, F., Sun, J.: Face alignment by explicit shape regression. In: 2012 IEEE Conference on Computer Vision and Pattern Recognition, pp. 2887–2894, June (2012)
38. Xiong, X., Torre, F. De la.: Supervised descent method and its applications to face alignment. In: 2013 IEEE Conference on Computer Vision and Pattern Recognition, pp. 532–539, June (2013)
39. Lowe, D.: Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vis.* **60**:91–111 (2004)
40. Bradski, G.: *The OpenCV Library*. Dr. Dobb's J. Softw. Tools (2000)
41. Kazemi, V., Sullivan, J.: One millisecond face alignment with an ensemble of regression trees. In: 2014 IEEE Conference on Computer Vision and Pattern Recognition, pp. 1867–1874, June (2014)
42. King, D.E.: Dlib-ml: A machine learning toolkit. *J. Mach. Learn. Res.* **10**, 1755–1758 (2009)
43. Voulodimos, A., Doulamis, N.D., Doulamis, A., Protopapadakis, E.: Deep learning for computer vision: a brief review. *Comput. Intell. Neurosci.* **2018** (2018)
44. Sun, Y., Wang, X., Tang, X.: Deep convolutional network cascade for facial point detection. In: 2013 IEEE Conference on Computer Vision and Pattern Recognition, pp. 3476–3483, June (2013)
45. Zhang, J., Shan, S., Kan, M., Chen, X.: Coarse-to-fine auto-encoder networks (cfan) for real-time face alignment. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (Eds.) *Computer Vision—ECCV 2014*. Springer, Cham, pp. 1–16 (2014)
46. Kowalski, M., Naruniec, J., Trzcinski, T.: Deep alignment network: a convolutional neural network for robust face alignment. *CoRR* (2017). [arXiv:1706.01789](https://arxiv.org/abs/1706.01789)
47. Feng, Z.H., Kittler, J., Awais, M., Huber, P., Wu, X.-J.: Wing loss for robust facial landmark localisation with convolutional neural networks. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2235–2245 (2018)
48. Park, B.-H., Se-Young, O., Kim, I.-J.: Face alignment using a deep neural network with local feature learning and recurrent regression. *Expert Syst. Appl.* **89**, 07 (2017)
49. Mahpod, S., Das, R., Maiorana, E., Keller, Y., Campisi, P.: Facial landmark point localization using coarse-to-fine deep recurrent neural network. *CoRR* (2018). [arXiv:1805.01760](https://arxiv.org/abs/1805.01760)
50. Dapogny, A., Bailly, K., Cord, M.: Decafa: Deep convolutional cascade for face alignment in the wild. *CoRR* (2019). [arXiv:1904.02549](https://arxiv.org/abs/1904.02549)
51. Zhu, X., Lei, Z., Liu, X., Shi, H., Li, S.Z.: Face alignment across large poses: a 3d solution. *CoRR* (2015). [arXiv:1511.07212](https://arxiv.org/abs/1511.07212)
52. Bulat, A., Tzimiropoulos, G.: How far are we from solving the 2d & 3d face alignment problem? (and a dataset of 230, 000 3d facial landmarks). *CoRR* (2017). [arXiv:1703.07332](https://arxiv.org/abs/1703.07332)
53. Chen, L., Su, H., Ji, Q.: Face alignment with kernel density deep neural network. In: 2019 IEEE/CVF International Conference on Computer Vision (ICCV), pp. 6991–7001, Oct (2019)
54. Wang, X., Bo, L., Li, F.: Adaptive wing loss for robust face alignment via heatmap regression. *CoRR* (2019). [arXiv:1904.07399](https://arxiv.org/abs/1904.07399)
55. Peng, X., Feris, R.S., Wang, X., Metaxas, D.N.: A recurrent encoder-decoder network for sequential face alignment. *CoRR* (2016). [arXiv:1608.05477](https://arxiv.org/abs/1608.05477)
56. Sánchez-Lozano, E., Martínez, B., Tzimiropoulos, G., Valstar, M.F.: Cascaded continuous regression for real-time incremental face tracking. *CoRR* (2016). [arXiv:1608.01137](https://arxiv.org/abs/1608.01137)
57. Yan, J., Lei, Z., Yi, D., Li, S.Z.: Learn to combine multiple hypotheses for accurate face alignment. In: 2013 IEEE International Conference on Computer Vision Workshops, pp. 392–396 (2013)
58. Peng, X., Zhang, S., Yang, Y., Metaxas, D.N.: Piefaf: Personalized incremental and ensemble face alignment. In: 2015 IEEE International Conference on Computer Vision (ICCV), pp. 3880–3888, Dec (2015)
59. Xiao, S., Yan, S., Kassim, A.A.: Facial landmark detection via progressive initialization. In: 2015 IEEE International Conference on Computer Vision Workshop (ICCVW), pp. 986–993, Dec (2015)
60. Khan, M.H., McDonagh, J., Tzimiropoulos, G.: Synergy between face alignment and tracking via discriminative global consensus optimization. In: 2017 IEEE International Conference on Computer Vision (ICCV), pp. 3811–3819, Oct (2017)
61. Zhu, S., Li, C., Loy, C.C., Tang, X.: Transferring landmark annotations for cross-dataset face alignment. *CoRR* (2014). [arXiv:1409.0602](https://arxiv.org/abs/1409.0602)
62. Milborrow, S., Morkel, J., Nicolls, F.: The MUCT landmarked face database. In: *Pattern Recognition Association of South Africa* (2010)
63. Le, V., Brandt, J., Lin, Z., Bourdev, L., Huang, T.S.: Interactive facial feature localization. In: Fitzgibbon, A., Lazebnik, S., Perona, P., Sato, Y., Schmid, C., (Eds.) *Computer Vision—ECCV 2012*. Springer, Berlin, pp. 679–692 (2012)
64. Liu, Y., Shi, H., Shen, H., Si, Y., Wang, X., Mei, T.: A new dataset and boundary-attention semantic segmentation for face parsing. In: *AAAI* (2020)
65. Liu, Y., Shi, H., Si, Y., Shen, H., Wang, X., Mei, T.: A high-efficiency framework for constructing large-scale face parsing benchmark. *CoRR* (2019). [arXiv:1905.04830](https://arxiv.org/abs/1905.04830)
66. Roth, P.M., Koestinger, M., Wohlhart, P., Bischof, H.: Annotated Facial Landmarks in the Wild: A Large-scale. Real-world Database for Facial Landmark Localization. In: *Proc, First IEEE International Workshop on Benchmarking Facial Image Analysis Technologies* (2011)
67. Zhang, Z., Luo, P., Loy, C.C., Tang, X.: Facial landmark detection by deep multi-task learning. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T., (Eds.) *Computer Vision—ECCV 2014*. Springer, Cham, pp. 94–108 (2014)
68. Fagertun, J., Harder, S., Rosengren, A., Moeller, C., Werge, T., Paulsen, R., Hansen, T.: 3d facial landmarks: Inter-operator variability of manual annotation. *BMC Med. Imaging* **14**, 35, 10 (2014)
69. Dong, X., Yu, S.-I., Weng, X., Wei, S.-E., Yang, Y., Sheikh, Y.: Supervision-by-registration: an unsupervised approach to improve the precision of facial landmark detectors. *CoRR* (2018). [arXiv:1807.00966](https://arxiv.org/abs/1807.00966)
70. Sagonas, C., Tzimiropoulos, G., Zafeiriou, S., Pantic, M.: A semi-automatic methodology for facial landmark annotation. In: 2013 IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp. 896–903, June (2013)

71. Guo, J., Mei, X., Tang, K.: Automatic landmark annotation and dense correspondence registration for 3d human facial images. *CoRR* (2012). [arXiv:1212.4920](https://arxiv.org/abs/1212.4920)
72. Messer, K., Matas, J., Kittler, J., Jonsson, K., Luettin, J., Maître, G.: Xm2vtsdb: the extended m2vts database. In: Proc. of Audio- and Video-Based Person Authentication, 04 (2000)
73. Jesorsky, O., Kirchberg, K.J., Frischholz, R.W.: Robust face detection using the hausdorff distance. In: Bigun, J., Smeraldi, F., (Eds.) Audio- and Video-Based Biometric Person Authentication. Springer, Berlin, pp. 90–95 (2001)
74. Phillips, P.J., Flynn, P.J., Scruggs, T., Bowyer, K.W., Chang, J., Hoffman, K., Marques, J., Min, J., Worek, W.: Overview of the face recognition grand challenge. In: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)—Volume 1—Volume 01, CVPR '05, USA. IEEE Computer Society, pp. 947–954 (2005)
75. Lucey, P., Cohn, J.F., Kanade, T., Saragih, J., Ambadar, Z., Matthews, I.: The extended cohn-kanade dataset (ck+): a complete dataset for action unit and emotion-specified expression. In: 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition—Workshops, pp. 94–101, June (2010)
76. Pillai, R.K., Jeni, L.A., Yang, H., Zhang, Z., Yin, L., Cohn, J.F.: The 2nd 3d face alignment in the wild challenge (3dfaw-video): dense reconstruction from video. In: 2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW), pp. 3082–3089, Oct (2019)
77. Belhumeur, P.N., Jacobs, D.W., Kriegman, D.J., Kumar, N.: Localizing parts of faces using a consensus of exemplars. *IEEE Trans. Pattern Anal. Mach. Intell.* **35**(12), 2930–2940 (2013)
78. Zhu, X., Ramanan, D.: Face detection, pose estimation, and landmark localization in the wild. In: 2012 IEEE Conference on Computer Vision and Pattern Recognition, pp. 2879–2886, June (2012)
79. Burgos-Artizzu, X.P., Perona, P., Dollár, P.: Robust face landmark estimation under occlusion. In: Proceedings of the IEEE international conference on computer vision, pp. 1513–1520 (2013)
80. Dhamecha, T., Verma, P., Shah, M., Singh, R., Vatsa, M.: Annotated crowd video face database. In: Proceedings of 2015 International Conference on Biometrics, ICB 2015, pp. 106–112, 06 (2015)
81. Shen, J., Zafeiriou, S., Chrysos, G.G., Kossaifi, J., Tzimiropoulos, G., Pantic, M.: The first facial landmark tracking in-the-wild challenge: benchmark and results. In: 2015 IEEE International Conference on Computer Vision Workshop (ICCVW), pp. 1003–1011, Dec (2015)
82. Zhang, Z., Luo, P., Loy, C.C., Tang, X.: Learning and transferring multi-task deep representation for face alignment. *CoRR* (2017). [arXiv:1408.3967](https://arxiv.org/abs/1408.3967)
83. Wu, W., Qian, C., Yang, S., Wang, Q., Cai, Y., Zhou, Q.: Look at boundary: a boundary-aware face alignment algorithm. *CoRR* (2018). [arXiv:1805.10483](https://arxiv.org/abs/1805.10483)
84. Liu, Y., Shen, H., Si, Y., Wang, X., Zhu, X., Shi, H., Hong, Z., Guo, H., Guo, Z., Chen, Y., Li, B., Xi, T., Yu, J., Xie, H., Xie, G., Li, M., Lu, Q., Wang, Z., Lai, S., Chai, Z., Wei, X.: Grand challenge of 106-point facial landmark localization. *CoRR* (2019). [arXiv:1905.03469](https://arxiv.org/abs/1905.03469)
85. Yang, J., Deng, J., Zhang, K., Liu, Q.: Facial shape tracking via spatio-temporal cascade shape regression. In: Proceedings of the 2015 IEEE International Conference on Computer Vision Workshop (ICCVW), ICCVW '15, USA. IEEE Computer Society, pp. 994–1002 (2015)
86. Sánchez-Lozano, E., Tzimiropoulos, G., Martínez, B., De la Torre, F., Valstar, M.: A functional regression approach to facial landmark tracking. *IEEE Trans. Pattern Anal. Mach. Intell.* **40**(9), 2037–2050 (2018)
87. Mörwald, T., Prankl, J., Zillich, M., Vincze, M.: Advances in real-time object tracking. *J. Real Time Image Process.* **10**, 01 (2013)
88. Hossin, M., Sulaiman, M.N.: A review on evaluation metrics for data classification evaluations. *Int. J. Data Min. Knowl. Manag. Process* **5**, 01–11, 03 (2015)
89. Ferri, C., Hernandez-Orallo, J., Modrou, R.: An experimental comparison of performance measures for classification. *Pattern Recogn. Lett.* **30**, 27–38, 01 (2009)
90. Botchkarev, A.: Performance metrics (error measures) in machine learning regression, forecasting and prognostics: properties and typology (2018). [arXiv:1809.03006](https://arxiv.org/abs/1809.03006)
91. Ahuja, S.: Introduction of a new metric hit rate and its variation with scaling on classification algorithms. *Int. J. Comput. Appl.* **125**, 13–16 (2015)
92. Sagonas, C., Antonakos, E., Tzimiropoulos, G., Zafeiriou, S., Pantic, M.: 300 faces in-the-wild challenge: database and results. *Image Vis. Comput.* **47**, 01 (2016)
93. Yang, H., Patras, I.: Mirror, mirror on the wall, tell me, is the error small? In: IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7–12, 2015, pp. 4685–4693 (2015)
94. Yang, H., Jia, X., Loy, C.C., Robinson, P.: An empirical study of recent face alignment methods. *CoRR* (2015). [arXiv:1511.05049](https://arxiv.org/abs/1511.05049)
95. Digiacomio, L., Digman, M., Gratton, E., Caracciolo, G.: Development of an image mean square displacement (imsd)-based method as a novel approach to study the intracellular trafficking of nanoparticles. *Acta Biomaterialia* **42**, 07 (2016)
96. Sutskever, I., Martens, J., Dahl, G., Hinton, G.: On the importance of initialization and momentum in deep learning. In: Dasgupta, S., McAllester, D. (Eds.) Proceedings of the 30th International Conference on Machine Learning, volume 28 of Proceedings of Machine Learning Research, Atlanta, Georgia, USA, 17–19 Jun (2013). PMLR, pp. 1139–1147
97. Leng, L., Zhang, J., Xu, J., Khan, K., Alghathbar, K.: Dynamic weighted discrimination power analysis: A novel approach for face and palmprint recognition in dct domain. In: International Journal of Physical Sciences, 2010, volume 5, no. 17, pp. 2543–2554, 12 (2010)
98. Leng, L., Li, M., Kim, C., Bi, X.: Dual-source discrimination power analysis for multi-instance contactless palmprint recognition. In: Multimedia Tools and Applications, 2017, volume 76, pp. 333–354, 01 (2017)
99. Dabbaghchian, S., Ghaemmaghami, M.P., Aghagolzadeh, A.: Feature extraction using discrete cosine transform and discrimination power analysis with a face recognition technology. *Pattern Recogn.* **43**(4), 1431–1440 (2010)
100. Štruc, V., Pavešić, N.: Image normalization techniques for robust face recognition. In: Proceedings of the International Conference on Signal Processing, Robotics and Automation (ISPRA'09), Cambridge, UK, pp. 155–160 (2009)
101. Moreno-Barea, F.J., Strazzera, F., Jerez, J.M., Urda, D., Franco, L.: Forward noise adjustment scheme for data augmentation. In: 2018 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 728–734, Nov (2018)
102. An, G.: The effects of adding noise during backpropagation training on a generalization performance. *Neural Comput.* **8**(3), 643–674 (1996)
103. Bishop, C.M.: Training with noise is equivalent to tikhonov regularization. *Neural Comput.* **7**(1), 108–116 (1995)
104. Carlson, A., Skinner, K.A., Johnson-Roberson, M.: Modeling camera effects to improve deep vision for real and synthetic data. *CoRR* (2018). [arXiv:1803.07721](https://arxiv.org/abs/1803.07721)

105. Hundt, R.: Loop recognition in c++/java/go/scala. In: Proceedings of Scala Days 2011 (2011)
106. Pereira, R., Couto, M., Ribeiro, F., Rua, R., Cunha, J., Fernandes, J.P., Saraiva, J.: Energy efficiency across programming languages: how do energy, time, and memory relate? In Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering, SLE 2017, New York, NY, USA, pp. 256–267 (2017) (Association for Computing Machinery)
107. Huang, J.C., Leng, T.: Generalized loop-unrolling: a method for program speedup. In: Proceedings 1999 IEEE Symposium on Application-Specific Systems and Software Engineering and Technology. ASSET'99 (Cat. No.PR00122), pp. 244–248, March (1999)
108. Kennedy, K., Allen, J.R.: Optimizing Compilers for Modern Architectures: A Dependence-Based Approach. Morgan Kaufmann Publishers Inc., San Francisco (2001)
109. Koskela, M., Viitanen, T., Jääskeläinen, P., Takala, J.H., Cameron, K.: Using half-precision floating-point numbers for storing bounding volume hierarchies. In: Proceedings of the 32nd Computer Graphics International Conference (2015)
110. Erick, L.O.: Fixed-point representation & fractional math. Oberstar Consulting, revision 1, 2 (2007)
111. Wu, H., Judd, P., Zhang, X., Isaev, M., Micikevicius, P.: Integer quantization for deep learning inference: principles and empirical evaluation (2020). [arXiv:2004.09602](https://arxiv.org/abs/2004.09602)
112. Widrow, B., Kollár, I.: Basics of Floating-Point Quantization. Cambridge University Press, Cambridge, pp. 257–306 (2008)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Constantino Álvarez Casado is a doctoral student at the Center for Machine Vision and Signal analysis at the University of Oulu. He received his M.Sc. degree in computer science from University of Oulu. He has several years of industrial experience in the development of real-time embedded computer vision algorithms, especially for face analysis. Contact him at constantino.lvarezCasado@oulu.fi

Miguel Bordallo López obtained his Ph.D. degree from the University of Oulu in 2014. During 15+ years he has worked at the Center for Machine Vision and Signal Analysis. He is currently Senior Scientist at VTT Technical Research Centre of Finland and Assistant Professor at the University of Oulu. His research interests include face analysis, embedded AI, image-based real-time sensing and energy-efficient embedded computer vision. Contact him at miguel.bordallo@vtt.fi