



# A walk through randomness for face analysis in unconstrained environments

Arnaud Dapogny

## ► To cite this version:

Arnaud Dapogny. A walk through randomness for face analysis in unconstrained environments. Computer Vision and Pattern Recognition [cs.CV]. Université Pierre et Marie Curie - Paris VI, 2016. English. NNT : 2016PA066662 . tel-01588960

**HAL Id: tel-01588960**

**<https://theses.hal.science/tel-01588960>**

Submitted on 18 Sep 2017

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Université Pierre et Marie Curie

École Doctorale 391 :

Sciences mécaniques, acoustique, électronique et robotique de Paris

# **A Walk Through Randomness**

**for Face Analysis in Unconstrained Environments**

Thèse de doctorat

**Arnaud Dapogny**

**Date de soutenance : 01/12/2016**

**Composition du jury :**

Directrice de thèse :	Séverine Dubuisson	UPMC Sorbonne Universités, ISIR
Encadrant :	Kévin Bailly	UPMC Sorbonne Universités, ISIR
Rapporteurs :	Renaud Séguier	Centrale-Supélec
	Laurent Heutte	Université de Rouen
Examineurs :	Matthieu Cord	UPMC Sorbonne Universités, LIP6
	Lionel Prevost	ESIA, LRD
	Christophe Garcia	INSA Lyon



# Contents

<b>1</b>	<b>Introduction</b>	<b>9</b>
1.1	The JEMImE project . . . . .	9
1.2	Main contributions . . . . .	11
1.2.1	Publications . . . . .	13
1.2.2	Source code . . . . .	14
<b>2</b>	<b>Facial expression recognition: an overview</b>	<b>15</b>
2.1	Modelisation of affect . . . . .	15
2.1.1	Categorical representation . . . . .	16
2.1.2	Dimensional representation . . . . .	16
2.1.3	Facial action coding system . . . . .	17
2.2	Challenges in automatic facial expression recognition . . . . .	18
2.2.1	Intrinsic factors of variation . . . . .	20
2.2.2	Extrinsic factors of variation . . . . .	22
2.2.2.1	Lighting conditions changes . . . . .	22
2.2.2.2	Head pose variation . . . . .	23
2.2.2.3	Partial occlusions . . . . .	25
2.3	Available data . . . . .	27
2.3.1	Overview and taxonomy . . . . .	27
2.3.2	The 300-W databases . . . . .	27
2.3.3	The extended Cohn-Kanade database . . . . .	28
2.3.4	BU-4DFE database . . . . .	29
2.3.5	FG-NET FEED database . . . . .	29



2.3.6	BP4D database . . . . .	30
2.3.7	SFEW database . . . . .	30
2.3.8	DISFA database . . . . .	31
2.3.9	JEMImE database . . . . .	31
2.4	Discussion and conclusion . . . . .	32
<b>3</b>	<b>Pattern recognition tools</b>	<b>33</b>
3.1	Overview . . . . .	33
3.2	Image representations . . . . .	33
3.2.1	Geometric features . . . . .	34
3.2.2	Distances between pairs of feature points . . . . .	35
3.2.3	Angles between triplets of feature points . . . . .	35
3.2.4	Appearance features . . . . .	35
3.2.4.1	Hand-crafted representations . . . . .	36
3.2.4.2	Learned representations . . . . .	37
3.3	Machine learning for classification and regression . . . . .	37
3.3.1	Which is the best model? . . . . .	37
3.3.2	The overfitting issue . . . . .	39
3.3.3	Neural Networks . . . . .	40
3.3.3.1	Feed-forward neural networks . . . . .	40
3.3.3.1.1	Training with stochastic gradient descent and backpropagation. . . . .	42
3.3.3.1.2	A few tricks for training neural networks. . . . .	44
3.3.3.2	Autoencoders . . . . .	45
3.3.3.2.1	Unsupervised training via a reconstruction cri- terion. . . . .	46
3.3.3.2.2	Regularization schemes. . . . .	47
3.3.3.3	Convolutional neural networks . . . . .	48
3.3.4	Randomized Decision Trees . . . . .	50
3.3.4.1	A single decision tree . . . . .	51

3.3.4.2	Ensemble of randomized trees . . . . .	54
3.3.4.2.1	Enhancing prediction accuracy with perturb and combine approaches . . . . .	55
3.3.4.2.2	Popular variants. . . . .	57
3.3.4.2.3	Dealing with imbalanced data. . . . .	58
3.3.4.2.4	Out-of-bag error estimate. . . . .	59
3.3.5	Neural Decision Forests . . . . .	59
3.3.5.1	Soft trees with probabilistic routing . . . . .	59
3.3.5.2	Online learning with recursive backpropagation . . . . .	60
3.3.6	Evaluation protocols . . . . .	63
<b>4</b>	<b>Pairwise Conditional Random Forests</b>	<b>65</b>
4.1	Overview . . . . .	65
4.2	A static RF approach for FER . . . . .	67
4.2.1	Bootstrap generation . . . . .	67
4.2.2	Heterogeneous feature templates . . . . .	68
4.3	Pairwise Conditional Random Forests . . . . .	69
4.3.1	Pairwise conditional tree collections . . . . .	69
4.3.2	Heterogeneous derivative feature templates . . . . .	69
4.3.3	Model averaging over time . . . . .	72
4.4	Multi-view extension for Pose-Robust Facial Expression Recognition . . . . .	74
4.4.1	Averaging over time multi-view classifiers . . . . .	74
4.4.2	Multi-view dataset generation . . . . .	76
4.5	Experiments . . . . .	79
4.5.1	Evaluation framework . . . . .	80
4.5.2	Experiments on frontal data . . . . .	81
4.5.2.1	Experiments on prototypical data . . . . .	81
4.5.2.2	Generalization on spontaneous data . . . . .	84
4.5.3	Experiments on non-frontal data . . . . .	85
4.5.4	Complexity analysis . . . . .	87

4.5.5	Discussion	89
<b>5</b>	<b>Local Subspace Random Forests</b>	<b>91</b>
5.1	Overview	91
5.2	Training randomized decision trees on spatially defined local subspaces	93
5.2.1	Random mask generation	93
5.2.2	Local expression prediction features	95
5.2.3	Facial mesh refinement	96
5.3	Confidence weighting of local expression predictions for occlusion-robust expression prediction	97
5.3.1	Weighted Local Subspace Random Forests	97
5.3.2	Local manifold learning for confidence measurement	98
5.3.2.1	Hierarchical autoencoder network architecture	98
5.3.2.2	Training the network	99
5.3.2.3	Confidence measurement	100
5.4	Using local predictions for Action Unit detection	101
5.4.1	Using available categorical expression-related data for Action Unit detection	101
5.4.2	Confidence measurement in Action Unit activation	103
5.4.3	Multi-output prediction of Action Unit activation	104
5.5	Experiments	106
5.5.1	Experimental setup	106
5.5.1.1	Evaluation metrics	106
5.5.1.2	Hyperparameter setting	107
5.5.2	Experiments on non-occluded scenarios	107
5.5.3	Experiments on occluded scenarios	110
5.5.3.1	Targeted occlusions	110
5.5.3.2	Random occlusions	111
5.5.3.3	Realistic occlusions	113
5.5.4	Experiments on AU detection	113

5.5.4.1	Merging multiple datasets . . . . .	113
5.5.4.2	Multi-output strategies . . . . .	115
5.5.4.3	Impact of mesh refinement . . . . .	116
5.5.4.4	Relevance of AU confidence assessment . . . . .	117
5.5.4.5	Comparison with state-of-the-art approaches . . . . .	118
5.5.5	Runtime evaluation . . . . .	119
5.5.6	Discussion . . . . .	120
<b>6</b>	<b>Greedy Evaluation of Neural Decision Forests</b>	<b>121</b>
6.1	Greedy evaluation procedure . . . . .	121
6.2	An efficient Neural Decision Forest training algorithm . . . . .	122
6.3	Applications . . . . .	124
6.3.1	GNDF for FER . . . . .	125
6.3.1.1	Varying tree depth and number of trees . . . . .	125
6.3.1.2	Learning deep representations . . . . .	127
6.3.1.3	Runtime evaluation . . . . .	129
6.3.2	Feature point alignment with a cascaded semi-parametric deep greedy neural decision forest . . . . .	130
6.3.2.1	Face alignment on still images . . . . .	130
6.3.2.1.1	Parametric shape model. . . . .	131
6.3.2.1.2	Explicit shape model. . . . .	133
6.3.2.1.3	Regression with NDF predictors. . . . .	133
6.3.2.1.4	Feature extraction and dimensionality reduc- tion. . . . .	134
6.3.2.1.5	Avoiding overfitting. . . . .	135
6.3.2.1.6	Cascaded alignment. . . . .	136
6.3.2.2	Evaluation . . . . .	136
6.3.2.3	Feature point alignment on video . . . . .	140
6.3.2.3.1	Bounding box regeneration. . . . .	140
6.3.2.3.2	Error case detection. . . . .	143

6.3.2.4	Evaluation on the 300-W video challenge . . . . .	143
6.4	Discussion . . . . .	145
<b>7</b>	<b>Discussion and Conclusions</b>	<b>147</b>
7.1	Conclusion . . . . .	147
7.2	Future works . . . . .	148
7.2.1	Using all the labelled data . . . . .	148
7.2.2	Exploiting JEMImE database . . . . .	149
7.2.3	Tuning the algorithms for more efficiency . . . . .	149
7.2.4	Adapting the proposed methods to other problems . . . . .	150
	<b>Appendices</b>	<b>161</b>
<b>A</b>	<b>A lower bound for the depth of randomly initialized trees with constant prediction nodes</b>	<b>162</b>
A.1	Random uniform initialization for classification . . . . .	162
A.2	Gaussian initialization for regression . . . . .	164
<b>B</b>	<b>implementation details</b>	<b>167</b>
B.1	Dependencies . . . . .	167
B.2	Data structures . . . . .	168
B.3	Low-level operations . . . . .	169
B.4	Solution architecture . . . . .	170
B.4.1	The Decision Forests project . . . . .	170
B.4.1.1	Project overview . . . . .	170
B.4.1.2	Configuration options and class constructors . . . . .	172
B.4.1.3	Delving into the code: an example of RF induction . . . . .	173
B.4.1.4	Main methods for face analysis . . . . .	175
B.4.2	The NeuralNets project . . . . .	176
B.4.2.1	Project overview . . . . .	176
B.4.2.1.1	The NeuralNet class. . . . .	176
B.4.2.1.2	The NeuronLayer class. . . . .	178

B.4.2.2	How to train a NeuralNet . . . . .	178
B.4.3	The Real Time project . . . . .	180
B.4.3.1	Project overview . . . . .	180



# Chapter 1

## Introduction

By looking at somebody else’s face, one can infer a lot of information about that person, such as his or her age, gender or ethnicity, as well as information about that person’s mental state. This includes both low-level attributes such as, for instance, the precise localization of mouth or eye corners, as well as a person’s current emotional state (is he/she happy? angry? or does he/she look surprised?), or his/her level of depression or engagement in an interaction.

With the rise of machine learning-based prediction systems, one can wonder to what extent -or if at all- it is possible to design a system for retrieving these cues from a person’s face automatically, in an unobtrusive fashion. Such system would theoretically bring valuable information for a broad range of applications. For example, it would allow to map face deformations on an avatar [9] for markerless facial motion capture or gaming purposes. Also, it would enable richer interactions between a person and a robot, as well as the monitoring of the facial expressions associated with physical pain in the context of healthcare systems. Last but not least, face analysis also has applications in the context of serious games for educative purposes, as it will be discussed below.

### 1.1 The JEMImE project

This thesis takes place in the frame of the JEMImE project (supported by the French National Agency - ANR). JEMImE is a French acronym that stands for “Jeu Educatif



Multimodal d’Imitation Emotionnelle”. Specifically, it aims at designing a new automatic emotion prediction system to assess the quality of the emotions produced by children, and more specifically children suffering from Autism Spectrum Disorder (ASD). Those emotions are measured through integrating multiple modalities (facial expressions from RGB and depth streams as well as voice intonation).

It is well known that children suffering from ASD have trouble understanding socio-emotional signals. As a consequence, many of them cannot respond adequately to other people’s behavior. This limits their ability to socialize with other people. The JEMImE project was thus geared towards using recent advances in computer vision and pattern recognition to teach children with ASD how to better understand and respond to socio-emotional signals in order to behave more adequately in real-case scenarios. This shall be done in the context of a serious game, through multiple phases. First, children are asked to mimic an expression performed by an avatar. In the next step of the training, they have to produce a requested emotion without having access to a model. Finally, the last step is to produce the adequate expression given the social context represented in the video game (e.g. another character steals the child’s toy: he thus has to appear as either angry or sad), possibly with the help of a therapist.

The applicative context of the JEMImE project led us to develop a complete facial expression recognition system that must fulfil the following constraints:

- The proposed expression recognition module must be able to recognize the facial expressions in spite of intrinsic variations (e.g. morphology and expressiveness) as well as extrinsic ones (e.g. head pose variation, environmental lighting changes and self-occlusions).
- Specifically, in terms of designing predictive models, we have to propose solutions for both classification of the facial expression and regression of the expression quality value.
- It has to run in real-time on a standard computer and can be easily integrated into the serious game environment.

For those reasons, in this thesis we focused to a certain extent on the adaptation to face analysis of machine learning models based on ensemble of randomized trees as well as neural networks. Chapter 2 draws an overview of the facial expression recognition (FER) problem, with an emphasis on the different models for affect representation, as well as a review of the available datasets that will be used to assess the predictive capacity of the algorithms. We also provide a non-exhaustive survey of the most successful methods of the literature for face analysis in general, and FER in particular. Furthermore, in Chapter 3 we provide an overview of the computer vision representation and machine learning tools that we use to tackle the issue of automatic face analysis. Chapters 4, 5 and 6 thus depict the three main contributions of this PhD. Those contributions are introduced in the following section.

## 1.2 Main contributions

Throughout this thesis, we propose a number of innovative contributions that aim at addressing several challenges in face analysis and expression recognition in particular. Those contributions are summarized in Table 1.2, as well as a non-exhaustive list of applications of those methods for face analysis.

As it will be discussed in Section 3, most approaches for face analysis involve (a) feature point alignment, (b) feature extraction and (c) attribute prediction, such as expression recognition. The three main contributions of this thesis are geared towards addressing those three problems, and led to multiple publications in international conferences 1.2.1 as well as a source code framework for end-to-end facial expression recognition 1.2.2. In Chapter 4 we propose to train pairwise conditional random forests to perform FER from video sequences as well as head pose variation handling in FER. In Chapter 5 we propose to train spatially-constrained local trees to learn local representation related to facial expressions. These representations can be used for occlusion-robust FER as well as for AU activation detection. Last but not least, in Chapter 6 we propose improvements over the recent neural decision forests machine learning algorithm, as well as its adaptation for face analysis. Thus, we propose solutions to perform cascaded regression for facial

Method Application	Pairwise Condi- tional RF	Local Subspace RF	Greedy evaluation of NDF
Feature point alignment			
Learning representations for FER			
Dynamic FER			
Pose-robust FER			
Occlusion handling in FER			
AU activation prediction			
Online learning for FER			
Subject-specific calibra- tion			
Real-time FER			

Table 1.1: Summary of the proposed contributions

feature point alignment, learning deep representations for FER, as well as the possibility of online learning and subject-specific calibration *via* classifiers fine-tuning. Eventually, throughout this thesis, we focus on providing face analysis systems that operate upon live video stream with a standard webcam device in real-time with a decent framerate.

### 1.2.1 Publications

First, the present work led to some publications under the form of journal papers, papers in international conferences as well as presentations in the frame of French-speaking seminars, which are listed below.

#### Submitted papers

A. Dapogny, K. Bailly, and S. Dubuisson. Face Alignment with Cascaded Semi-Parametric Deep Greedy Neural Forests. *Submitted to IEEE International Conference on Computer Vision and Pattern Recognition, 2017.*

A. Dapogny, K. Bailly, and S. Dubuisson. Multi-Output Random Forests for Facial Action Unit Detection. *Submitted to IEEE International Conference on Automatic Face and Gesture Recognition, 2017.*

A. Dapogny, K. Bailly, and S. Dubuisson. Dynamic pose-robust facial expression recognition by multi-view pairwise conditional random forests. *arxiv preprint, 2016. Submitted to IEEE Transactions on Affective Computing.*

A. Dapogny, K. Bailly, and S. Dubuisson. Confidence-Weighted Local Expression Predictions for Occlusion Handling in Expression Recognition and Action Unit detection. *arxiv preprint, 2016. Submitted to Springer International Journal of Computer Vision.*

#### Conference papers

J. Aigrain, A. Dapogny, K. Bailly, M. Detyniecki, S. Dubuisson and M. Chetouani. On leveraging crowdsourced data for automatic perceived stress detection. In *International Conference on Multimodal Interaction*, p. 1-8, 2016.

A. Dapogny, K. Bailly, and S. Dubuisson. Pairwise Conditional Random Forests for Facial Expression Recognition. In *IEEE International Conference on Computer Vision*, p. 1-9, 2015.

A. Dapogny, K. Bailly, and S. Dubuisson. Dynamic facial expression recognition by

static and multi-time gap transition joint classification. In *IEEE International Conference on Automatic Face and Gesture Recognition*, p. 1-6, 2015.

### **Seminars and others**

A. Dapogny, K. Bailly, and S. Dubuisson. Pairwise Conditional Random Forests for Facial Expression Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, demo sessions, 2016.

A. Dapogny, K. Bailly, and S. Dubuisson. Random Forest pour la reconnaissance robuste des expressions faciales. *Atelier scientifique du GDR ISIS sur les interactions Homme/Machine*, 2015.

A. Dapogny, K. Bailly, and S. Dubuisson. Reconnaissance des expressions faciales par combinaison de classifieurs statique et dynamique. *Atelier scientifique du GDR ISIS sur les interactions Homme/Machine*, 2015.

## **1.2.2 Source code**

Another important aspect of this PhD is the development of a code framework that implements all the methods summarized in Table 1.2 above. In order to perform real-time face analysis from live video stream, the proposed code was developed in C++, although some functions are written in MATLAB, notably for generating the images used in Chapter 4 from the 3D models for expression recognition from arbitrary viewpoints. The source code is also further described in Appendix B, and directly within the *.h* and *.cpp* files, through the comments. Note that the proposed framework contains the data structures and methods to train and test all the algorithms introduced in Section 3 for general purposes, which are not limited to face analysis.

## Chapter 2

# Facial expression recognition: an overview

In this Chapter, we focus on introducing the issue of facial expression recognition. First, in Section 2.1 we discuss the main models used in the literature for affect representation. In Section 2.2 we introduce a classic pipeline for FER and highlight, from the view of both a human observer and an automated system, the main challenges for an accurate expression recognition. Finally, in Section 2.3, we describe the available data that we will use to train and evaluate our face analysis systems.

### 2.1 Modelisation of affect

In his book *The Expression of the Emotions in Man and Animals* (1872), Charles Darwin theorized that showing and recognizing emotions was an evolved trait universal to the whole mankind. However, an important current among anthropologists of the XX<sup>th</sup> century supported that the ability to produce and decipher facial expressions was determined through a behavioural learning process. This belief was later put into perspective by the works of Ekman on universally recognized expressions (Section 2.1.1), as well as more recent models (Sections 2.1.2 and 2.1.3).

### 2.1.1 Categorical representation

Through cross-cultural studies, Paul Ekman discovered that there existed a high agreement across people from diverse literate cultures across the world when it came to assigning a label to pictures representing expressive faces. In this work [27], Ekman came with a list of six universally recognized basic expressions (*happiness, anger, sadness, fear, disgust* and *surprise*). He also mentioned a *contempt* expression class, though the agreement appeared to be less clear. Furthermore, Ekman demonstrated that this finding extended to preliterate tribesmen in Papua, New Guinea, whose members could not have learned the meaning of facial expressions from exposure to media depictions of emotion. Along with a *neutral* state, this so-called categorical expression model has been used as an underlying model for most attempts at developing a prototypical expression recognition system [59], [103]. However, it faces limitations for dealing with spontaneous facial expressions [108], as many of our daily affective behaviors may not be translated in terms of prototypical emotions. Nevertheless, the annotation process is rather intuitive, thus there exists a large corpus of labelled data (see sections 2.3.3, 2.3.4, 2.3.5 and 2.3.7).

In a later work, Ekman proposed an expanded list of universally recognized basic emotions, which are not all encoded by facial muscles [26]. The newly included emotions are: *amusement, contempt, contentment, embarrassment, excitement, guilt, pride* in achievement, *relief, satisfaction, sensory pleasure*, and *shame*. However the data labelled with those expressions is currently very scarce, although some recent database begin to integrate these labels (see Section 2.3.6).

### 2.1.2 Dimensional representation

Another popular approach is the continuous affect representation [35] that consists in projecting expressions onto a restricted number of latent dimensions. Hence, a specific expression (e.g. a categorical expression such as *happiness*) can be described by its position in a low-dimensional space. A popular example of such model is the valence/arousal (relaxed vs. aroused)/power (feeling of control)/expectancy (anticipation) model. It is often simplified as a two-dimensional valence-arousal representation. Figure 2.1 shows

the projection of the six categorical expressions detailed above on a two-dimensional valence/arousal space. Following this diagram, *happy* is represented as *positive valence*, *high arousal*. On the contrary, *sad* is defined as *negative valence*, *low arousal*. One can see that using such a low-dimensional embedding of facial expressions can cause the loss of information. Indeed, some expressions can not be separated well (*fear* vs. *anger*), as both are defined as *negative valence*, *high arousal*. Last but not least, the annotation process is less intuitive than with the categorical representation. Thus, the labellers have to be trained prior to an annotation task, which in turn limits the data availability.

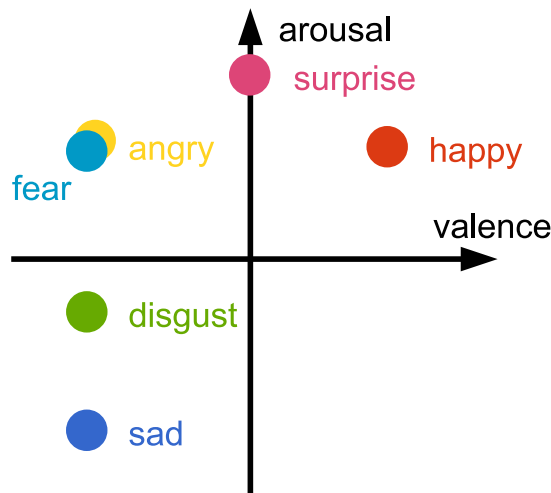


Figure 2.1: Projecting categorical facial expressions on a two-dimensional valence/arousal space.

### 2.1.3 Facial action coding system

Last but not least, an alternative facial expression model is the Facial Action Coding System (FACS) also proposed by Paul Ekman [28]. It consists in describing facial expressions as a combination of 44 facial muscle activations that are referred to as Action Units (AUs). Figure 2.2 illustrates the most common AUs from the upper and lower face parts. AUs is a face representation that may be less subject to interpretation. It can theoretically be used in accordance with the so-called Emotional FACS (EMFACS) rules in order to describe a broader range of spontaneous expressions. However, the main drawback of the



FACS-coding approach is that the annotation tends to be a time-consuming process. Furthermore, FACS coders have to be highly trained, hence limiting the quantity of available data. However, thanks to an effort from the research community, there exist a number of FACS-annotated databases depicting both prototypical (Section 2.3.3) and spontaneous (Section 2.3.8) expressive behaviors.

















<b>AU1</b>  Inner brow raiser	<b>AU2</b>  Outer brow raiser	<b>AU4</b>  Brow lowerer	<b>AU5</b>  Upper lid raiser
<b>AU6</b>  Cheek raiser	<b>AU7</b>  Lid tightener	<b>AU9</b>  Nose wrinkler	<b>AU10</b>  Upper lip raiser
<b>AU12</b>  Lip corner puller	<b>AU15</b>  Lip corner depressor	<b>AU17</b>  Chin raiser	<b>AU20</b>  Lip stretcher
<b>AU23</b>  Lip tightener	<b>AU24</b>  Lip pressor	<b>AU25</b>  Lips part	<b>AU26</b>  Jaw drop

Figure 2.2: Definition and illustration of some of the most commonly observed AUs. Images extracted from CK+ database (See Section 2.3.3).

## 2.2 Challenges in automatic facial expression recognition

Figure 2.3 illustrates a classic pipeline for automatic face analysis, and for expression recognition in particular. First, given a grayscale image (possibly a frame of a video clip), a subject's face rectangle is provided by a face detection algorithm. Then, a set of facial

feature points are aligned on the face, which correspond to specific locations of the face, e.g. lip corners, eye corners or nose tips. A set of representations are then extracted using the aligned feature points only (geometric features). Those features generally need to be normalized w.r.t. the location of one particular feature point (e.g. the nose tip) to ensure translation invariance, and w.r.t. the inter-ocular distance for scale invariance. Then, some machine learning algorithm is used to predict a classification of the facial expression, or to provide a regression of the value of the intensity of the activation of a set of AUs. Popular examples of machine learning methods are Support Vector Machines (SVM [18]), Random Forest (RF [11]) or Neural Networks (NN [39]).

In order to increase the prediction accuracy, many state-of-the-art approaches employ a combination of geometric (*i.e.* extracted from a set of facial feature points aligned on the face beforehand) and appearance features (*i.e.* sampling the texture of the face directly), that often need to be indexed w.r.t. the position of the facial feature points. For example, Senechal *et al.* [77] use a multi-kernel SVM to integrate those heterogeneous cues. Example of popular appearance features include Local Binary Pattern (LBP [78]), and Histogram of Oriented Gradients (HOG [46]).

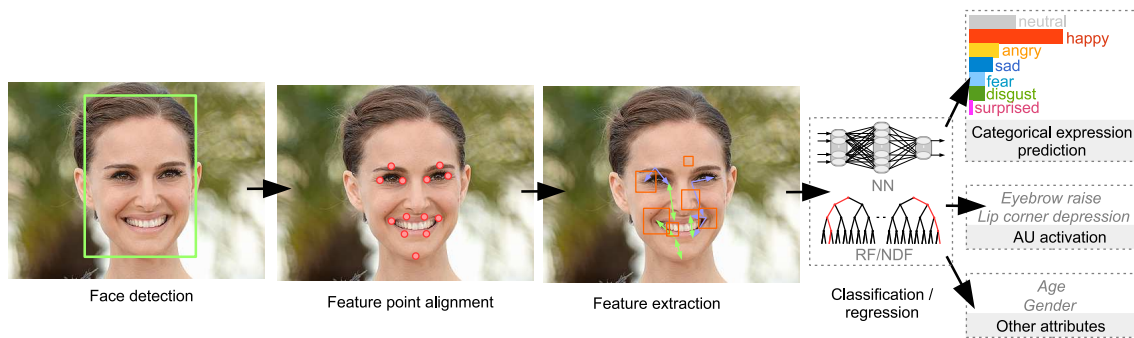


Figure 2.3: A classic pipeline for face analysis

Generally speaking, it is not easy for a human eye to distinguish between the expressions in the general case. Reasons for this are multiple: first, there are a number of factors of variations that are intrinsic to the person that produces the expressions, namely the morphology of this person’s face as well as, when talking about video records, the way his or her facial geometry changes over time to produce the expression (Section 2.2.1). There are also a lot of extrinsic factors of variation, such as changes in environmental

lighting conditions (2.2.2.1), head pose variations relatively to the camera device position (2.2.2.2), or the presence of partial occlusions of the face (2.2.2.3). In the following subsections, we illustrate those factors of variations, as well as a number of solutions that have been proposed in the literature for addressing these problems.

## 2.2.1 Intrinsic factors of variation

As it is illustrated on Figure 2.4, it can be difficult for a human eye to discriminate facial expressions when looking at a single picture. If we only look at the bottom row faces on Figure 2.4, we can hardly tell which expressions are being portrayed by the subjects. Reasons for this are multiple: we don't know how these persons look like when not displaying a specific expression (*i.e.* we don't have access to a *neutral* face), nor do we have any information about how they behave to display a specific expression. Hence, generally speaking, designing a static expression prediction system (*i.e.* one that works on still images) is a challenging task. In order to focus specifically on designing a FER system independent of a subject's morphology, most recent approaches covering static FER work in controlled conditions, on a frontal view and lab-recorded environments [59, 103]. Zhong *et al.* [112] proposed to learn active facial patches that are relevant for FER. Zhao *et al.* [110] designed a unified multitask framework for simultaneously performing facial alignment, head pose estimation and FER. Liu *et al.* [56] introduced a deep neural network that learns local features relevant for Action Unit prediction, and use it as an intermediate representation for categorical FER. The authors also studied the use of unlabelled data [55] to regularize the network training, further enhancing its predictive capacities for FER in the wild.

However, as it is illustrated on Figure 2.4, the morphology problem can be partially alleviated by looking at a neutral representation of the face (top-row images on Figure 2.4 - from left to right we have *anger*, *sadness*, *happiness*, *fear*, *disgust* and *surprise*). This way, it becomes easier to identify the expressions by looking more closely at which areas of the face are subject to appearance changes. From the perspective of an automatic recognition system, this means that the performances of such system shall be better if we have access to *relative* information using a *neutral* face representation. Examples of



Figure 2.4: Examples of corner-cases expressions from the BU-4DFE database. Without looking at a neutral face (top row), it may be difficult for a human observer to analyse the facial expressions adequately. Can you guess which ones are being portrayed?

this paradigm are the works of Khademi *et al.* [45] that respectively learn transition AU detectors, as well as the work of Chu *et al.* [15] which propose a new selective transfer machine framework that includes the possibility of subject-specific calibration.

Furthermore, as we generally do not have access to a neutral face representation, it is a good idea to use dynamic information, *i.e.* performing recognition on videos rather than on separate images. Dynamic information of facial expressions can be used in several ways: (a) at the feature-level, by using spatio-temporal image descriptors, and/or (b) at the semantic level, by modelling relationships between expressions or between successive phases (*onset*, *apex* and *offset*) of facial events. Generally speaking, effectively extracting suitable representations from spatio-temporal video patterns is a challenging problem as expressions may occur with various offsets and at different paces. There is no consensus either on how to combine those representations flexibly enough so as to generalize on unseen data and possibly unseen temporal variations. Common approaches employ spatio-temporal descriptors defined on fixed-size windows, optionally at multiple resolutions. Examples of such features include the so-called LBP-TOP [109, 37] and HOG3D [47] descriptors, which are spatio-temporal extensions of LBP and HOG features respec-

tively. Authors in [79] use histograms of local phase and orientations. However, those kinds of representations may lack the capacity to generalize to facial events that differ from training data on the temporal axis. More recently, Jung *et al.* [44] integrate both features in a deep temporal geometric/appearance neural network.

Approaches trying to address (b) aim at establishing relationships between high-level features and a sequence of latent states. Wang *et al.* [95] train hidden Markov models to perform early recognition of low-intensity facial expressions from videos. Wang *et al.* [96] integrate temporal interval algebra into a Bayesian network to capture complex relationships among facial muscles. Such approaches generally require explicit dimensionality reduction techniques such as PCA or  $k$ -means clustering for training. In addition, training at the sequence level reduces the quantity of available training and testing data as compared to frame-based approaches, as there is only one expression label per video.

## 2.2.2 Extrinsic factors of variation

The visibility of the face plays an important role in discriminating the facial expressions. We can divide the corresponding factors of variation in three groups: the lighting conditions (Section 2.2.2.1), the position of the camera w.r.t. the subject's head (Section 2.2.2.2), and whether or not the face is occluded (Section 2.2.2.3).

### 2.2.2.1 Lighting conditions changes

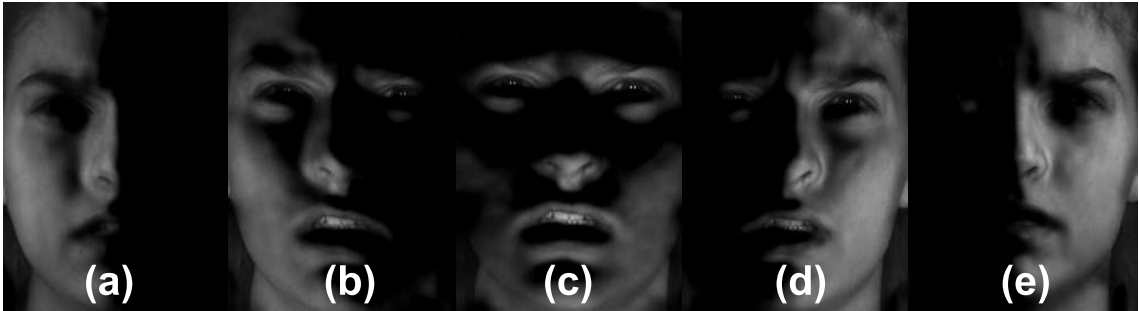


Figure 2.5: Examples of an expressive face under illumination variations. The images are generated using the high-resolution 3D face scans from the BU-4DFE database.

The way a one's face is illuminated is crucial in deciphering its facial expressions. For

example, how shall one label the faces portrayed on Figure 2.5, in terms of Ekman’s facial expressions? Such observer could, for instance, label (a) and (e) as *anger*, while (b) and (c) would be depicted as *disgust* and (d) as *sadness*. However, all five images were generated from a single video frame of the BU-4DFE database, with ground truth label *disgust*. This illustrates that the way the face is illuminated greatly influences the human perception of facial expressions. Furthermore, from the perspective of an automated system, drastic illumination conditions, such as those on Figure 2.5, can cause loss of information (for aligning the feature points or exploiting the texture information), as well as creating local gradients that may result in false detections. Indeed, most systems perform a global normalization or use gradient information to ensure invariance w.r.t. global luminosity variations. Many descriptors such as SIFT [58] or HOG [21] also perform some sort of local normalization for that purpose.

#### 2.2.2.2 Head pose variation

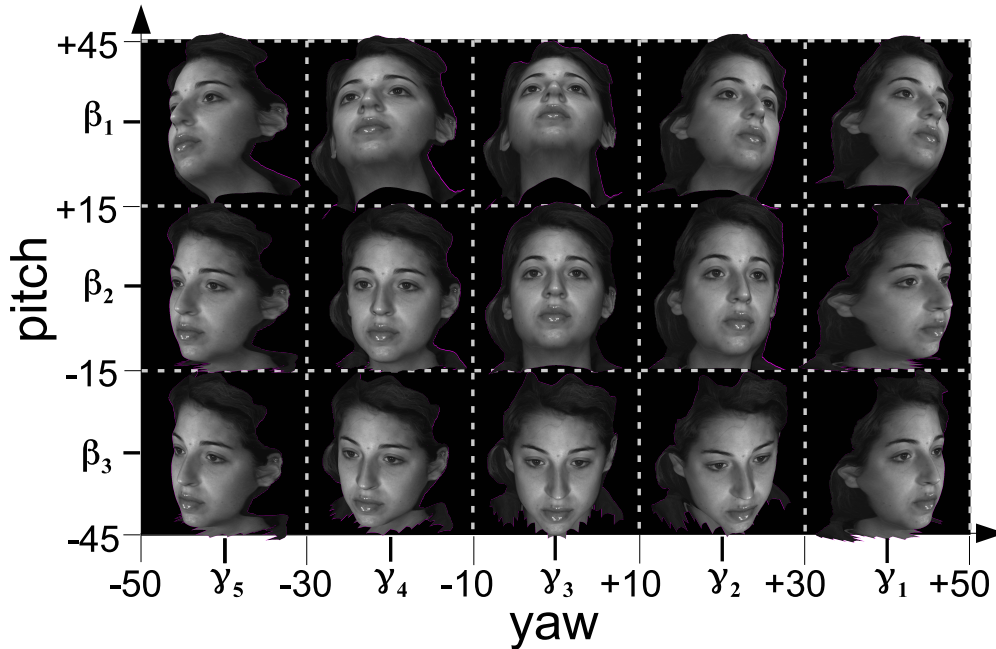


Figure 2.6: Examples of a neutral face under head pose variation. The images are generated using the high-resolution 3D face scans from the BU-4DFE database.



Head pose variations can also cause drastic changes in face appearance or geometry, as can be seen on Figure 2.6, that represents a subject’s face under yaw ( $\pm 45$  degrees) and pitch ( $\pm 45$  degrees) variations. For example, for positive pitches the eyes are less visible and the eyebrow look closer to the eyes (as it is the case when displaying prototypical anger with a frontal viewpoint), whereas for negative pitches the mouth becomes less visible. Furthermore, the feature points are more likely to be badly aligned for extreme poses in either pitch or yaw.

Many approaches for multi-view FER consist in training a single classifier to describe every viewpoint. Zheng *et al.* [111] introduce a regional covariance matrix representation of face images to infer static facial expressions on a corpus constructed from the BU-3DFE database [104] with 35 different head poses up to  $\pm 45$  yaw and  $\pm 30$  pitch. Tariq *et al.* [87] address the same problem by using a translation invariant sparse coding of dense SIFT features [58]. Eleftheriadis *et al.* [29] employ discriminative shared Gaussian processes to implicitly exploit the redundancy between multiple views of the same expressive images. However, such approach can struggle to capture the variability of the facial expressions when the number of training samples becomes important.

Alternatively, it is possible to learn a projection of a non-frontal views of a face image on a frontal one. Recently, Vieriu *et al.* [90] proposed to project 3D data of the face onto a head pose-invariant 2D representation. The visible fraction of the projected face is then used within a voting scheme to decipher the expression. FER can thus be performed using an off-the-shelf algorithm. In addition, the authors were able to perform FER under a broader range of poses, up to  $\pm 90$  yaw and  $\pm 60$  pitch. However, the proposed method requires high-resolution 3D face data that may not necessarily be available in multiple human-computer interaction scenarios, for instance when using images acquired with commercial lower-resolution depth sensors.

Eventually, some other works choose to learn one specific classifier per face view. During testing, the head pose is first estimated, then the best pose-specific expression classifier is applied. For instance, Moore *et al.* [64] learn multi-class SVMs upon LBP features for multiple viewpoints. Such approaches offer several advantages over the previous ones: first, learning classifiers upon separate and more homogeneous face view

data allows to considerably reduce the variability. As a consequence the classifiers can, in theory, more efficiently capture the subtle facial deformations between the expressions. Secondly, the runtime is the same as in the case of a single frontal view classifier, which may be a critical point for systems that try to project a given view on a frontal one. Finally, splitting the training data offers the advantage to reduce the memory usage, which can be important for learning on large databases. Those methods also face some impediments, such as the fact that (a) they require a reliable facial landmark alignment and head pose estimation, and (b) it implies dividing the data into several subsets. Nevertheless, (a) is barely a problem given that recent advances [98, 69, 99] for face alignment provide excellent results for head poses up to  $\pm 45$  yaw and  $\pm 30$  pitch, which is sufficient for most human-computer applications. Furthermore, (b) can be circumvented by the use of 3D face scans [103] from which we can generate a large corpus of videos for training multi-view dynamic classifiers.

### 2.2.2.3 Partial occlusions



Figure 2.7: Examples of expressive faces under partial occlusion. 3 images on the left: synthetic occlusions overlaid on images from the CK+ database. 3 images on the right: realistic occlusions from the SFEW database.

Another major challenge for the design of an automatic expression recognition system is the presence of partial occlusions of the face. The 3 images on the right of Figure 2.7 illustrate examples of occlusion that can happen in realistic scenarios. Partial occlusions include self-occlusions (e.g. with a body part such as a hand), or occlusions that are due to random accessories (e.g. a scarf or sunglasses). Countless configurations of partial occlusions cause a large variability of face appearance. This is all the more problematic



when designing an automatic FER system, as there is currently close to no labelled data for realistically occluded face, let alone expressive face images. For this reason, most of the literature focus on synthetic occlusions, that are usually generated by overlaying random or uniform patterns on some area of the face, as can be seen on Figure 2.7.

Kotsia *et al.* [49] studied the impact of human perception of facial expressions under partial occlusions, and the predictive capacities of automated systems thereof. In particular, they concluded that in the case of prototypical expressions, mouth occlusions seem to drastically hinder the recognition performance of both a human observer and an automated system. Cotter *et al.* [19] used sparse decomposition to perform FER on corrupted images. Ghiasi *et al.* [33] use a discriminative approach for facial feature point alignment under partial occlusions. Those approaches rely on explicitly incorporating synthetic occluded data in the training process, and thus struggle to deal with realistic, unpredicted occluding patterns. Zhang *et al.* [107] trained classifiers upon random Gabor-based templates. They evaluated their algorithms on synthetically occluded face images, showing that their approach leads to a better recognition rate when the same occluded examples are used for training and testing. Should this not be the case, unpredicted mouth/eye occlusions still lead to a significant loss of performance. Huang *et al.* [40] proposed to automatically detect the occluded regions using sparse decomposition residuals. However, the proposed approach may not be flexible enough, as the occlusion detection only outputs binary decisions, and as the face is explicitly divided into only three subparts (eyes, nose and mouth). This limits the capacities of the method to deal with unpredicted forms of occlusion. Finally, another approach consists in learning generative models of non-occluded faces, as it was done by Ranzato *et al.* [68]. When testing on a partially occluded face image, the occluded parts can be generated back and expression recognition can be performed. The pitfall of this approach is that training can be computationally expensive and does not allow the use of heterogeneous features (e.g. geometric/appearance descriptors).

Table 2.1: Summary of the available databases.

Database	300-W	CK+	BU-4DFE	FEED	BP4D	SFEW	DISFA	JEM-ImE
# subjects	3740	123	101	19	41	700	27	151
Landmarks								
Video								
Spontaneous								
Categorical								
AU								
Non-frontal poses								
Occlusions								

## 2.3 Available data

### 2.3.1 Overview and taxonomy

Table 2.1 presents an outline of the available databases for face analysis. For each of these datasets, we indicate the number of subjects that it contains, as well as the nature of the data (e.g. does it consist of separate images or video records? Are the expressive behaviors spontaneous or acted?). We also indicate whether or not the data contains non-frontal head poses and partial occlusions, in order to provide an idea of the difficulty of the different benchmarks. Finally, we also describe the nature of the provided ground truth annotations for each dataset (*i.e.* does the annotation consists in facial landmarks, categorical expressions or the activation of a set of AUs?).

### 2.3.2 The 300-W databases

The 300-W databases consist in multiple datasets for landmark analysis. They embrace a number of benchmarks with various degrees of difficulty, such as the LFPW or Labelled Faces Parts in the Wild database (834 images for training/224 images for test), HELEN (2000/334 images), and the more challenging ibug database (135 images). In

addition, there are also a number of images from the 2013 face alignment challenge (300 indoor/300 outdoor images) as well as 114 videos from the training partition of the 2015 video alignment challenge. Each image comes along with an annotated 68-points markup.

### 2.3.3 The extended Cohn-Kanade database



Figure 2.8: Examples of expressive face images from the CK+ database. From left to right: images labelled as *neutral*, *happy*, *angry*, *sad*, *fear*, *disgust*, *contempt* and *surprise*.

The CK+ or Extended Cohn-Kanade database [59] contains 123 subjects, each one associated with various numbers of expression records. Those records display a very gradual evolution from a *neutral* class towards one of the 6 universal facial expressions described by Ekman [27] (*anger*, *happiness*, *sadness*, *fear*, *digust* and *surprise*) plus the non-basic expression *contempt*. Expressions are acted and very prototypical. The sequences contains 20 grayscale images on average and the subjects exhibit practically no head pose variation as well as no self-occlusion. There are some illumination and face scaling changes between the records, but overall the recognition rates reported by state-of-the-art methods on this dataset are very high (around 90%).

As it is done in other approaches, we extract the first (*neutral*) and three apex frames for each of the 327 sequences for training classifiers to perform 8-class categorical FER. Figure 2.8 illustrates some expressive face images excerpted from the CK+ database. As some approaches discard the frames labelled as *contempt*, for the sake of comparison, we also report 7-class accuracy using only 309 sequences. Moreover, the CK+ database is also FACS-annotated, therefore we report results for the recognition of 14 of the most common AUs (AU1,2,4,5,6,7,9,10,12,15,17,20,25,26) on this dataset.



Figure 2.9: Examples of face images from the BU-4DFE database. From left to right: images labelled as *neutral*, *happy*, *angry*, *sad*, *fear*, *disgust* and *surprise*.

### 2.3.4 BU-4DFE database

The BU-4DFE of Binghamton University- 4D Facial Expressions database [103] contains 101 subjects, each one displaying the 6 acted facial expressions of Ekman with moderate head pose variations. Expressions are still prototypical but they are generally exhibited with much lower intensity and greater variability than in CK+, hence a lower baseline accuracy of about 70%. Sequence duration is about 100 frames. As the database does not contain frame-wise expression annotations, we manually selected neutral and apex of expression frames for all the subjects for training, making a total of 8218 images. Figure 2.9 shows some of these apex frames. Each frame is associated with high-resolution 3D model data recorded using a Di3D device, that we use in our experiments on non-frontal head poses to generate expression videos from multiple viewpoints (see Section 4.4.2).

### 2.3.5 FG-NET FEED database



Figure 2.10: Examples of face images from the FEED database. From left to right: images labelled as *neutral*, *happy*, *angry*, *sad*, *fear*, *disgust* and *surprise*.

The FG-NET FEED or Facial Expressions and Emotions database [93] contains 19 subjects, each one recorded three times while performing 7 spontaneous expressions (the six universal expressions, plus the *neutral* one). The data contain low-intensity emotions,

very short expression displays, as well as moderate head pose variations. We use the provided peak and neutral annotation metadatas for each sequence to generate a 716 images subset on which the predictive models can be trained and evaluated for frame-based classification.

### 2.3.6 BP4D database

The BP4D or Binghamton-Pittsburgh 4D database [108] contains 41 subjects. Each subject was asked to perform 8 tasks, each one supposed to give rise to 8 spontaneous expressions (*anger, happiness, sadness, fear, disgust, surprise, embarrassment* or *pain*). In [108] the authors extracted subsequences of about 20 seconds for manual FACS annotations, arguing that these subsets contain the most expressive behaviors. As done in the literature [108] we report results for recognition of 12 AUs (1,2,4,6,7,10,12,14,15,17,23,24). We randomly extract 10000 images for training and evaluate the AU classifiers on the whole dataset.

### 2.3.7 SFEW database



Figure 2.11: Examples of face images from the SFEW database. From left to right: images labelled as *neutral, happy, angry, sad, fear, disgust* and *surprise*.

The SFEW or Static Facial Expression in the Wild database [24] contains 700 images from 95 subjects displaying 7 facial expressions (including *neutral*) in a real-world environment. Data was gathered from video clips using a semi-automatic labelling process. The strictly person-independent evaluation (SPI) benchmark is composed of two folds of (roughly) same size. As done in other approaches, we report cross-validation results averaged over the two folds. As it can be witnessed on Figure 2.11 the data embraces a lot of variations, including uncommon morphological traits and lighting patterns, head pose

variations, self-occlusions as well as low-resolution images. Furthermore, the quantity of training data is very limited as each fold contains approximately 350 images.

### 2.3.8 DISFA database

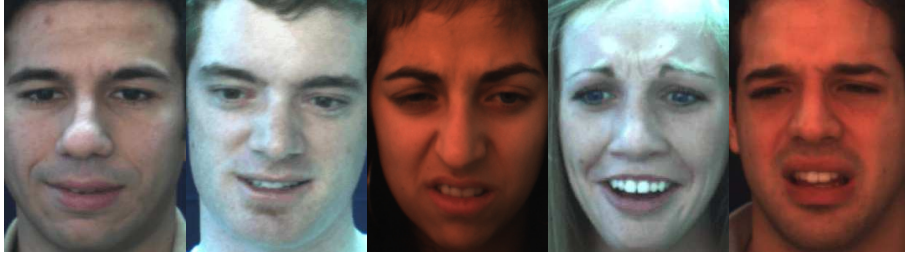


Figure 2.12: Examples of expressive face images from the DISFA database.

The DISFA or Denver Intensity of Spontaneous Facial Actions [62] contains videos of 27 subjects with different ethnicities and genders that were recorded watching a 4-minute emotive video stimulus. As shown on Figure 2.12, the face images are lab-recorded with mostly frontal head poses. However the induced behaviors are spontaneous, thus some AU activations are very subtle, hence a relatively low baseline accuracy. Data have been manually labelled frame by frame for 12 AUs (1,2,4,5,6,9,12,15,17,20, 25,26) on a 6-level scale by a human expert, and verified by a second FACS coder. For the purpose of predicting AU occurrence, we consider AUs which intensity is below 1 as non-activated. We randomly extract 6292 images for training and test on the 125832 images.

### 2.3.9 JEMImE database

The JEMImE database currently contains video records of 150 childs. Each one of these subjects was asked to watch and respond to videos displaying avatars performing 3 categorical facial expressions (*happiness*, *anger* and *sadness*), plus the *neutral* one. For each video, the child was asked to respond by imitating the proposed expression, through both *visual* and *audio-visual* modalities. In addition to that, childs were also asked to mimic the 4 expressions, without seeing the videos. Moreover, the order in which the tasks were presented to the subjects was randomized. Thus, the database contains a total of 2400

videos. For each of these videos, 2 raters were asked to answer, for each expression, if that expression was displayed on the video, in which case they also had to give a note to the quality of the displayed expression. In addition to that, the database will also contain data from children with ASD to study their capacity to produce facial expressions with and without a model, as well as to study the differences between them and typical children.

## 2.4 Discussion and conclusion

As we discussed, there exists a number of challenges in expression recognition, which can be divided in extrinsic (environmental, e.g. background and lighting) and intrinsic factors variations (subject morphology, head pose variation, self-occlusions). Furthermore, the available data is relatively scarce and do not quite cover all those factors of variations. Particularly in the case of pose or occlusion-robust FER as well as FER in the wild, we are limited with data quantities of the order of magnitude of 1000 images, which is very low, as compared to some other fields, e.g. face recognition or object recognition. As it was pointed out, this is mainly due to the fact that the annotation labels come from highly trained experts (e.g. for FACS coding) and are thus difficult to get.

For those reasons, in what follows, we mainly focus on adapting machine learning and pattern recognition tools presented in Chapter 3 for “making the most” of available data. Particularly, in Chapter 4, we highlight how we can train decision forests using dynamic information from video sequences with only a few labelled peak frames. Moreover, we explain how we can use 3D face scans from available data to allow pose-robust FER. In Chapter 5, we show how we can learn features from data labelled with categorical expressions to perform AU detection, as well as how we successfully perform occlusion-robust FER without any data related to partial occlusions. Last but not least, in Chapter 6 we discuss solutions for online learning in FER and feature point alignment, which can be applied for domain adaptation, e.g. in the frame of subject-specific calibration

# Chapter 3

## Pattern recognition tools

### 3.1 Overview

In this chapter, we describe the framework that we use to predict facial expressions, with an emphasis on the low-level image descriptors and machine learning methods to perform classification or regression. First, we employ the widely used Viola and Jones detector [92] to retrieve a rectangular face bounding box  $Face(\mathcal{I})$  from image  $\mathcal{I}$ . Then, a set of facial feature points  $f(\mathcal{I})$  are aligned on the face. In order to ensure reproducibility of the results, we use a state-of-the-art method (Intraface [98]). As it is somewhat classical in the FER literature, we use a combination of geometric (Section 3.2.1) and appearance features (Section 3.2.4).

### 3.2 Image representations

Generally speaking, it is not possible to use the raw gray level or RGB values from a given face image  $\mathcal{I}$  for a subsequent prediction task. Reasons for this are multiple: apart from the very high dimensionality of the input (1000000 for a  $1000 \times 1000$  image!), those pixel values are generally noisy. Furthermore, as we do not have access to unlimited amounts of training data for learning the prediction models, we generally have to design feature descriptors that exhibit some built-in invariance to the lower-level factors of variations, e.g. translation, planar rotation, scaling of the face or local luminosity intensity varia-



tions. To do so, we introduce a set of parametric feature templates  $\phi^{(i)}$  with  $i = 1, \dots, 3$  that include both geometric (Section 3.2.1) (*i.e.* computed from previously aligned facial feature points) and appearance features (Section 3.2.4). Figure 3.1 provides an example of feature point alignment and extracted geometric and appearance features. Those features are thus used as the input of a classification or regression machine learning system (Section 3.3) to either predict a categorical expression, a set of AU activations or intensity values or, to a larger extent, a number of attributes that can be inferred from the face (e.g. gender, age or ethnicity).

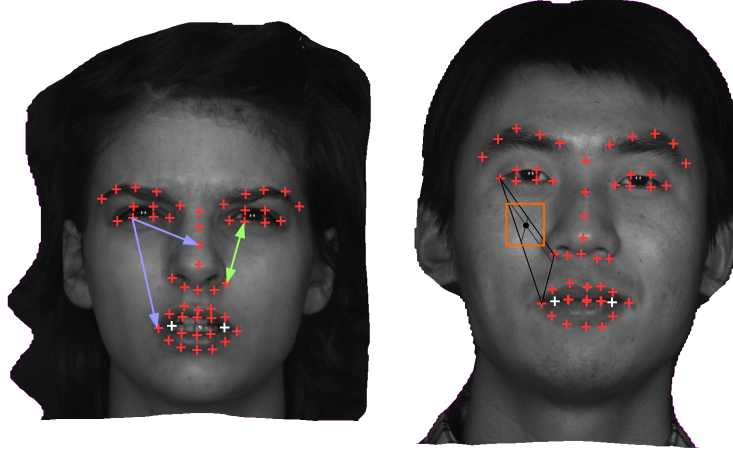


Figure 3.1: Feature points aligned on the face and features extracted for FER. Green: point distances  $\phi^{(1)}$ . Blue: point angles  $\phi^{(2)}$ . Orange: HOG  $\phi^{(3)}$ .

### 3.2.1 Geometric features

Given the set of facial feature points  $f(\mathcal{I})$  aligned on a face image  $\mathcal{I}$ , a simple and robust way of extracting information that is relevant for face analysis is to use (a function of) the positions of these feature points as a feature for the subsequent prediction step. Below are described two simple, yet robust features that efficiently sum up the face geometry.

### 3.2.2 Distances between pairs of feature points

The first geometric feature template  $\phi_{a,b}^{(1)}$  takes as its parameters the index of two facial feature points  $a$  and  $b$ .  $\phi_{a,b}^{(1)}$  is thus defined as the Euclidean distance between feature points  $f_a(\mathcal{I})$  and  $f_b(\mathcal{I})$ , that is normalized w.r.t. the inter-ocular distance  $iod(f(\mathcal{I}))$  (Equation 3.1). Note that  $iod(f(\mathcal{I}))$  can be computed once and for all for a specific image by knowing the indexes of 4 feature points (inner and outer corners for the left and right eyes).

$$\phi_{a,b}^{(1)}(\mathcal{I}) = \frac{\|f_a(\mathcal{I}) - f_b(\mathcal{I})\|_2}{iod(f(\mathcal{I}))} \quad (3.1)$$

It is easy to see that  $\phi^{(1)}$  is invariant w.r.t. translations and rotations of the face within the camera plane. Also, thanks to the normalization, it is invariant to the scaling of the face. However, this feature template only takes into account the distances between the feature points. Hence, we have to use a second geometric template that takes into account the orientation of those points.

### 3.2.3 Angles between triplets of feature points

The second geometric feature  $\phi_{a,b,c,\lambda}^{(2)}$  takes as parameters the indexes of three facial feature points  $f_a(\mathcal{I})$ ,  $f_b(\mathcal{I})$  and  $f_c(\mathcal{I})$ , as well as a boolean parameter  $\lambda$ , and is defined as follows:

$$\phi_{a,b,c,\lambda}^{(2)}(\mathcal{I}) = \lambda \cos(\widehat{f_a f_b f_c}) + (1 - \lambda) \sin(\widehat{f_a f_b f_c}) \quad (3.2)$$

Thus, depending on  $\lambda$ ,  $\phi_{a,b,c,\lambda}^{(2)}$  can be defined either as the sine or cosine of the angle between the three feature points. We use this formulation instead of the raw angle value so as to preserve the continuity of the function for angles around 0. Also note that  $\phi^{(2)}$  is also invariant to in-plane rotations, translations and face scaling. As a geometric feature, it is also invariant to intensity variations.

### 3.2.4 Appearance features

Geometric features are generally used along with appearance features, *i.e.* descriptors that represent the texture of the face image directly. Those two sets of features are compli-

mentary to each other and it is somehow classical to use a conjunction of both to increase the recognition accuracy [77, 44]. Appearance features can be divided into handcrafted, engineered descriptors and learned representations.

### 3.2.4.1 Hand-crafted representations

Perhaps the most popular engineered image descriptor for computer vision is Scale Invariant Feature Transform (SIFT [58]). The descriptor consists in dividing a window around a specific keypoint into (usually  $4 \times 4$ ) cells from which 8-bin quantizations of gradient orientations are computed. Those are thus weighted by the gradient magnitude, concatenated and  $\mathcal{L}_2$ -normalized to form a 128-dimensional local descriptor. Speeded-Up Robust Features (SURF [5]) was later introduced as an alternative descriptor, that essentially benefits from the integral image computational trick that allows faster processing. SIFT and SURF descriptors offer a good compromise between extraction speed, dimensionality and computation runtime for a variety of computer vision tasks such as image retrieval [4] or object segmentation [83]. However, for face analysis, the high-dimensionality of those features may slow down the extraction and (optional) dimensionality reduction step, which may be critical as the real-time constraint is prominent. Plus, local patterns are generally sufficient to sample the texture of the face (*i.e.* we do not need to use a grid composed of multiple cells).

For those reasons, we use Histogram of Oriented Gradients (HOG [21]) as our appearance features for their descriptive power and robustness to global illumination changes. In order to ensure fast feature extraction, we use integral feature channels as introduced in [25]. First, face images are extracted based on the position of the facial feature points. More specifically, a bounding box is determined by the min and max  $x-y$  coordinates and extended by 20% in all directions, then the corresponding region of interest is rescaled to a constant size of  $250 \times 250$  pixels. Then, horizontal and vertical gradients are computed on the image and used to generate 9 feature maps, the first one containing the gradient magnitude, and the 8 remaining correspond to a 8-bin quantization of the gradient orientation. Then, integral images are computed from these feature maps to output the 9 feature channels. Thus, we define the appearance feature template  $\phi_{\tau, ch, s, \alpha, \beta, \gamma}^{(3)}$  as an in-

tegral histogram computed over channel  $ch$  within a window of size  $s$  multiplied by the inter-ocular distance. Such histogram is evaluated at a point defined by its barycentric coordinates  $\alpha$ ,  $\beta$  and  $\gamma$  w.r.t. vertices of a triangle  $\tau$  defined over feature points  $f(\mathcal{I})$ . Also, we store the gradient magnitude in the first channel to normalize the histograms. Thus, HOG features can be computed with only 4 access to the integral channels (plus normalization).

### 3.2.4.2 Learned representations

An alternative approach to hand-crafted representations is to learn the image descriptors directly from the data. For that matter, one generally needs to use deep architectures that are composed of several differentiable layers. Neural networks (Section 3.3.3), neural decision forests (Section 3.3.5) and convolutional neural networks (Section 3.3.3.3) are examples of deep architectures that allows the extraction of hierarchically abstracted texture representations.

## 3.3 Machine learning for classification and regression

### 3.3.1 Which is the best model?

The No Free Lunch theorem [97] for machine learning states that there is no universal model that provides the best fit for every problem. Consider the analogy with the one-dimensional interpolation problem, where we want to find a “suitable” function  $f$  such that  $f(0) = f(0.25) = f(0.5) = f(0.75) = f(1) = 1$ . Of course, the concept of a “suitable” function is unclear and, without additional knowledge on the problem, there exists an infinity of very diverse functions that satisfy this criterion, such as every polynomial of the form  $f(x) = ax(x - 0.25)(x - 0.5)(x - 0.75)(x - 1) + 1$ , with  $a \in \mathbb{R}$  (Figure 3.2).

We now suppose we want  $f$  to be close from “test” points  $(0.125, 1.006)$ ,  $(0.375, 0.994)$ ,  $(0.625, 1.006)$ ,  $(0.875, 0.994)$ . However, those points remain unknown during the “training” step and cannot be used as control points for the interpolation. In such a case, to select a good polynomial function we need to introduce additional knowledge on the problem

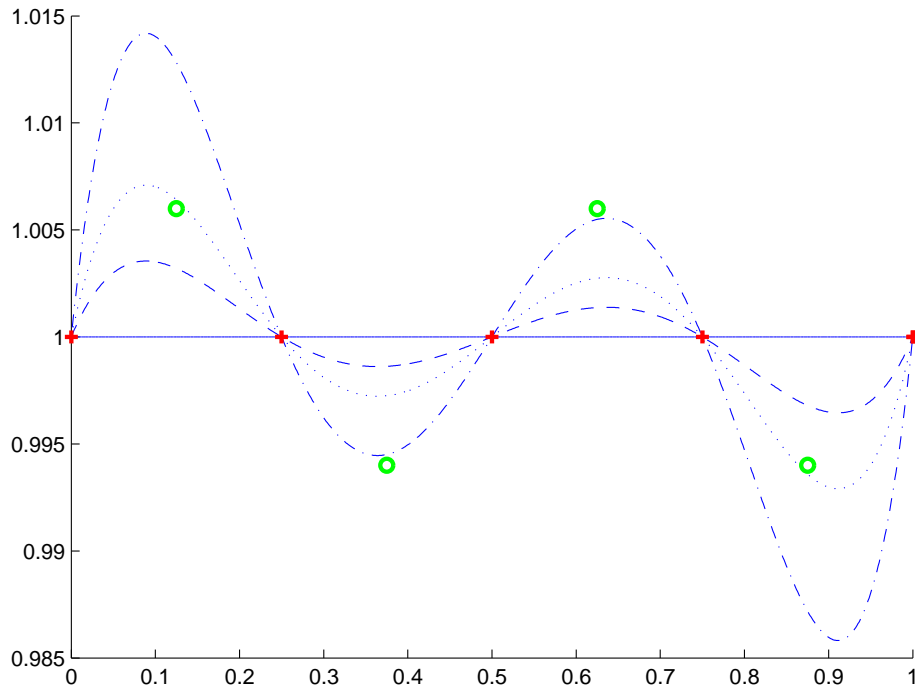


Figure 3.2: Analogy of learning a predictive model with one-dimensional polynomial interpolation. Red crosses indicate training samples, green circles indicate test samples. In this order, the dashed, dotted and dash-dotted lines indicate curves corresponding to polynomials with ascending value for the  $a$  parameter. Best viewed in color.

such as, for instance, the maximal/minimal value of  $f$  on the interval  $[0, 1]$ , or a boundary on the derivatives of  $f$ . The analogy with machine learning is pretty straightforward, except that (a) we have to deal with very high-dimensional inputs (10 up to uncountable infinite numbers of dimensions) and (b) constraints on the solutions generally cannot be analytically defined.

Consequently, there is no such thing as a silver bullet model in machine learning and it is therefore common -particularly when applied to computed vision tasks- to try multiple predicting models with various hyperparameter settings that seem reasonable given an applicative context, selecting the ones that offer an interesting trade-off between speed (training/testing), complexity and accuracy. Furthermore, one important aspect that we shall keep in mind when fitting a model is the overfitting issue, which is illustrated in the following paragraph.

### 3.3.2 The overfitting issue

Figure 3.3 illustrates the overfitting issue on a simple example that consists in a polynomial approximation of the cosine function on the interval  $[-2\pi, 2\pi]$ , given a restricted set of 13 control points. On the left, one can see that approximating the function with a polynomial of degree 4 does not provide satisfying results, as the error between the blue curve (the function to approximate) and the red one is still important. Consequently, the 4-degree polynomial model is not complex enough to approximate the function (it is also said to underfit the problem). The curves in the middle show the result of polynomial fitting using a 8-degree polynomial. In that case, the error on the control points is nearly non-existent and the approximation is qualitatively good. However, using a more complex model, e.g. a polynomial of degree 16 (on the right) produces a very different result. The error around the training data (*i.e.* the control points) is still very low, but the overall error function skyrockets completely. In machine learning, we refer to that phenomenon as overfitting, *i.e.* when a model is too complex to effectively capture the solution variations.

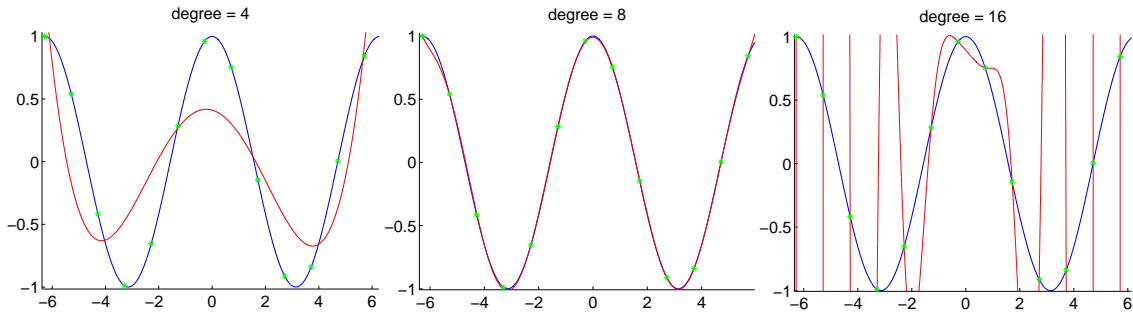


Figure 3.3: Approximation of the cosine using polynomials functions. Green stars: training samples. The blue line indicates the target (cosine) function. The red one indicates the polynomial approximation.

The balance between underfitting and overfitting can sometimes be tricky to find, as both issues may severely hinder the ability of a learning system to generalize on unseen data. In what follows, we will focus more specifically on two machine learning frameworks that are neural networks [39] and randomized decision trees [11] to predict facial expressions. Below we provide a description of those methods as well as an ensemble of properties that make them well suited for face analysis purposes regarding the issues that

are at stake.

### 3.3.3 Neural Networks

Neural networks (NNs) are among the oldest and most popular machine learning framework. Basic feed-forward neural networks allow non-linear embedding as well as efficient training using stochastic gradient descent and backpropagation. Also, recent developments involve dropout regularization [82] to prevent overfitting issues, unsupervised learning *via* a reconstruction criterion [8] as well as the possibility of a unified, top-down feature and prediction stages training using deep convolutional network architectures [31].

#### 3.3.3.1 Feed-forward neural networks

An individual neuron cell for a  $k$ -dimensional input  $\mathbf{x} = \{x_j\}_{j=1\dots k}$  has two kinds of parameters:  $\mathbf{w} = \{w_j\}_{j=1\dots k}$  the neuron weight vector and  $b$  the bias term. It realizes two operations: computing  $net(\mathbf{x})$  as a scalar product of the input by the neuron weights (plus bias), and a non-linear mapping of the output using an activation function  $\sigma$  (Figure 3.4).

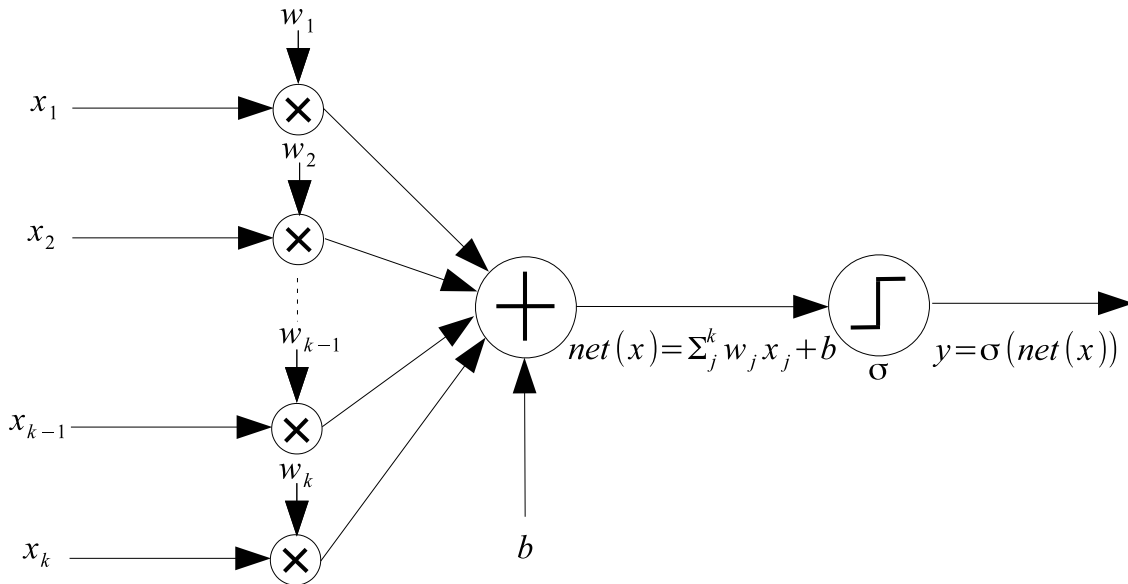


Figure 3.4: A single neuron

The activation function  $\sigma$  is usually defined either as an affine function:

$$\sigma(net(\mathbf{x})) = net(\mathbf{x}) \quad (3.3)$$

or, more commonly, as a non-linear activation such as the hyperbolic tangent function:

$$\sigma(net(\mathbf{x})) = \frac{1 - e^{-2net(\mathbf{x})}}{1 + e^{-2net(\mathbf{x})}} \quad (3.4)$$

the sigmoid function:

$$\sigma(net(\mathbf{x})) = \frac{1}{1 + e^{-net(\mathbf{x})}} \quad (3.5)$$

or the Rectified Linear Unit (ReLU) function:

$$\sigma(net(\mathbf{x})) = \max(0, net(\mathbf{x})) \quad (3.6)$$

In the case where multiple units indexed by  $i = 1, \dots, m$  are stacked to form a neuron layer, the output can be defined as a normalized sum over all the output net units to provide values in the  $[0, 1]$  interval, so that the sum of these values is equal to 1. This so-called softmax activation function is defined as:

$$\sigma(net_i(\mathbf{x})) = \frac{e^{net_i(\mathbf{x})}}{\sum_{i'=1}^m e^{net_{i'}(\mathbf{x})}} \quad (3.7)$$

Furthermore, multiple layers can be stacked, each one providing a non-linear encoding of the output of the previous layer. Figure 3.5 represents an example of a multi-layered neural net architecture.

In such a case, by abusing the notation, the output vector of each layer  $l \in \{1, 2, 3\}$  can be written in matrix form as  $\mathbf{y}^{(l)} = \sigma(\mathbf{w}^{(l)}\mathbf{x}^{(l)} + \mathbf{b}^{(l)})$ , with  $\mathbf{x}^{(l)}$  the input of layer  $l$  (with  $\mathbf{x}^{(0)} = \mathbf{x}$ ),  $\mathbf{w}^{(l)}$  and  $\mathbf{b}^{(l)}$  respectively the weight matrix and bias vector for layer  $l$  and  $\mathbf{y}^{(l)}$  the output of that layer. As shown on Figure 3.5, the output of each layer feeds the input of the following one, *i.e.*  $\mathbf{x}^{(l)} = \mathbf{y}^{(l-1)}$ . The network activation is thus computed in a feed-forward fashion, from the bottom levels (net input) to the top ones. This kind of architecture is often referred to as a Multi-Layer Perceptron (MLP) and its layers as fully-connected (FC) layers. Generally speaking, the input and output layer



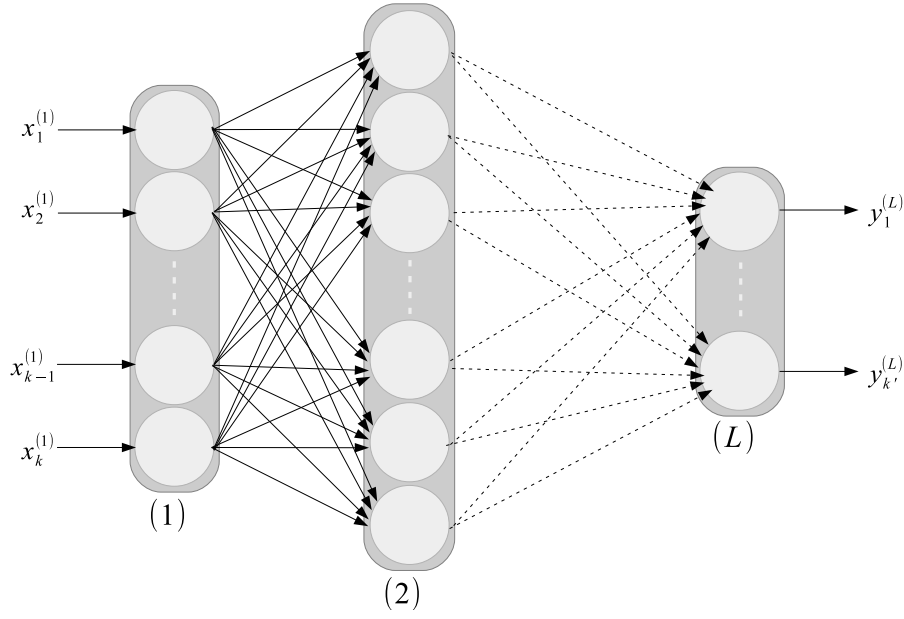


Figure 3.5: An example of multi-layered network architecture

sizes are determined by the problem (the (either raw or somehow reduced) number of input dimensions and output classes, respectively). However, there is no consensus on how to set the network hyperparameters that are the number of layers, the size of each layers, how the layer is initialized and the nature of the activation functions. Below are the most common approaches that are used to train such a network.

**3.3.3.1.1 Training with stochastic gradient descent and backpropagation.** Training a feed-forward neural network supposes finding a set of parameters  $\{\mathbf{w}^{(l)}, \mathbf{b}^{(l)}\}_{l=1,\dots,L}$  (with  $L$  the number of layers) that minimizes a cost function  $E(W)$  (where  $W = \{\mathbf{w}^{(1)}, \mathbf{b}^{(1)}, \dots, \mathbf{w}^{(L)}, \mathbf{b}^{(L)}\}$  is the set of parameters of the whole network) that corresponds to an error criterion  $E(W) = \epsilon(\hat{\mathbf{y}}, \mathbf{y}^*)$  between the ground truth label  $\mathbf{y}^*$  and the network output prediction  $\hat{\mathbf{y}} = \mathbf{y}^{(L)}$ , for every labelled example  $(\mathbf{x}, \mathbf{y}^*)$ . This error is usually defined as either the squared  $\mathcal{L}_2$  loss for classification and regression

$$\epsilon(\hat{\mathbf{y}}, \mathbf{y}^*) = \|\hat{\mathbf{y}} - \mathbf{y}^*\|_2^2 \quad (3.8)$$

or the cross-entropy loss

$$\epsilon(\hat{\mathbf{y}}, \mathbf{y}^*) = \sum_{j=1}^{k'} \hat{y}_j \ln(y_j^*) + (1 - \hat{y}_j) \ln(1 - y_j^*) \quad (3.9)$$

for classification, when the output activation function is the softmax function. However, minimizing this error criterion w.r.t. the network parameters is a non-convex optimization problem in the general case, and the high-dimensionality of the problem makes exhaustive search impossible. For these reasons, neural networks are generally trained by backpropagating [53] the error gradient from the top layers down to the bottom ones, by applying Stochastic Gradient Descent (SGD [71]). In practice, this consists in iteratively minimizing the loss function in Equation 3.8 by applying a sequence of updates to the model's parameters. Each of these updates only consider one or a mini-batch of examples that are randomly sampled from the training set. Generally SGD solvers need a certain amounts of epochs through the whole training set to converge. Formally, given a mini-batch of  $\mathcal{B}$  examples  $\{(\mathbf{x}_1, \mathbf{y}_1^*), \dots, (\mathbf{x}_{\mathcal{B}}, \mathbf{y}_{\mathcal{B}}^*)\}$ , the update for a parameter  $w_{ji}^{(l)}$ ,  $j \in [1, k']$ ,  $i \in [1, k]$  of a neuron layer  $l$  can be written:

$$w_{ji}^{(l)} \leftarrow w_{ji}^{(l)} - \frac{\alpha}{|\mathcal{B}|} \frac{\partial \epsilon(\hat{\mathbf{y}}, \mathbf{y}^*)}{\partial w_{ji}^{(l)}} \quad (3.10)$$

The same holds true for the bias terms, with  $\alpha$  the learning rate hyperparameter. If we consider a parameter of the top layer  $L$  we can write, using the chain rule:

$$\frac{\partial \epsilon(\mathbf{y}^{(L)}, \mathbf{y}^*)}{\partial w_{ji}^{(L)}} = \frac{\partial \epsilon(\mathbf{y}^{(L)}, \mathbf{y}^*)}{\partial y_j^{(L)}} \frac{\partial y_j^{(L)}}{\partial w_{ji}^{(L)}} = \frac{\partial \epsilon(\mathbf{y}^{(L)}, \mathbf{y}^*)}{\partial y_j^{(L)}} \frac{\partial y_j^{(L)}}{\partial net_j^{(L)}} \frac{\partial net_j^{(L)}}{\partial w_{ji}^{(L)}} \quad (3.11)$$

The first term is relative to the loss function ( $\mathcal{L}_2$  or cross-entropy). The second one depends on the activation function (e.g. softmax for classification or affine for regression tasks). Finally, for the last term we have:

$$\frac{\partial net_j^{(L)}}{\partial w_{ji}^{(L)}} = x_i^{(L)} = y_i^{(L-1)} \quad (3.12)$$

Notice that the equations are the same if we now consider a parameter of the second layer from the top  $L - 1$ :

$$\frac{\partial \epsilon(\mathbf{y}^{(L)}, \mathbf{y}^*)}{\partial w_{ji}^{(L-1)}} = \sum_{p=1}^{k'} \frac{\partial \epsilon(\mathbf{y}^{(L)}, \mathbf{y}^*)}{\partial y_p^{(L)}} \frac{\partial y_p^{(L)}}{\partial \text{net}_p^{(L)}} \frac{\partial \text{net}_p^{(L)}}{\partial w_{ji}^{(L-1)}} \quad (3.13)$$

Once again, using the chain rule (remember  $\mathbf{y}^{(L-1)} = \mathbf{x}^{(L)}$ ), we get:

$$\frac{\partial \epsilon(\mathbf{y}^{(L)}, \mathbf{y}^*)}{\partial w_{ji}^{(L-1)}} = \sum_{p=1}^{k'} \frac{\partial \epsilon(\mathbf{y}^{(L)}, \mathbf{y}^*)}{\partial y_p^{(L)}} \frac{\partial y_p^{(L)}}{\partial \text{net}_p^{(L)}} \frac{\partial \text{net}_p^{(L)}}{\partial x_j^{(L)}} \frac{\partial x_j^{(L)}}{\partial w_{ji}^{(L-1)}} = \delta_j^{(L-1)} \frac{\partial y_j^{(L-1)}}{\partial w_{ji}^{(L-1)}} \quad (3.14)$$

We define  $\delta^{(L-1)}$  as the error backpropagated to the  $(L-1)^{th}$  layer. Starting from the top level we have:

$$\delta_j^{(L)} = \frac{\partial \epsilon(\mathbf{y}^{(L)}, \mathbf{y}^*)}{\partial y_j^{(L)}} \quad (3.15)$$

and, for any layer  $l$  of size  $k''$  in the network,

$$\delta_j^{(l-1)} = \sum_{p=1}^{k''} \delta_p^{(l)} \frac{\partial y_p^{(l)}}{\partial \text{net}_p^{(l)}} w_{pj}^{(l)} \quad (3.16)$$

Thus, for each layer  $l$ , we can compute the derivative and the SGD or mini-batch update w.r.t. its parameters and compute the error  $\delta^{(l-1)}$  backpropagated to the layers below this one (if any).

**3.3.3.1.2 A few tricks for training neural networks.** Generally speaking, the weights and bias terms for each layer are randomly initialized (again, there is no consensus on what is an optimal value for the range of this random initialization, though uniform sampling in the interval  $[-0.01, 0.01]$  is a good rule of thumb). Then, examples are selected at random and a forward pass through the whole network provides the network output  $\hat{\mathbf{y}}$ . The error can be computed and backpropagated in a top-down fashion and the network parameters are updated using Equation (3.10). This process is usually repeated until a specified number of epochs ( $\approx 100$  is a good order of magnitude for our applications) through the whole training set are completed, with either a constant or a slightly decreasing learning rate hyperparameter.

Moreover, it is also generally a good idea to introduce a Gaussian prior with zero mean on the weights of the network. This assumption is equivalent to adding a weight decay

term to the cost function, that can be rewritten  $E(W) = \epsilon(\hat{\mathbf{y}}, \mathbf{y}^*) + 1/2\lambda W^2$ . Thus, for any parameter of the network the update equation becomes:

$$w_{ji}^{(l)} \leftarrow w_{ji}^{(l)} - \frac{\alpha}{|\mathcal{B}|} \frac{\partial \epsilon(\hat{\mathbf{y}}, \mathbf{y}^*)}{\partial w_{ji}^{(l)}} - \frac{\alpha \lambda}{|\mathcal{B}|} w_{ji}^{(l)} \quad (3.17)$$

In practice, this prevents the weights to reach extremal values, which can cause instability of the network. Usually  $\lambda$  is set to a constant, very small value. Other methods for preventing overfitting while training neural nets involve early stopping and dropout. On the one hand, the former simply consists in reducing the number of training epochs while the training error is still decreasing. While it is easy to qualitatively see why early stopping can prevent the networks from overfitting on the training data, there are a number of papers in the literature [75] that shows that it can be seen as some sort of Tikhonov regularization [102]. Dropout [82] or dropConnect [94], on the other hand, consists in randomly setting a fraction (usually one half) of the neurons' weights during the forward and backward pass for each SGD or mini-batch update. It has been shown by the authors in [82] to behave as an extreme form of Bagging [10], effectively preventing co-adaptations in the network, which in turn limits overfitting issues.

### 3.3.3.2 Autoencoders

Autoencoders are a special kind of neural network where the output vector is a reconstruction of the input, as illustrated on Figure 3.6. Thus, the output of the network has the same size as the input. The hidden layer can be smaller than the input, in which case the autoencoder essentially performs a non-linear dimensionality reduction. This is sometimes referred to as an under-complete representation of the input. In that case, as compared to PCA [43], autoencoders benefit from a more efficient training procedure that involves SGD and error backpropagation (Section 3.3.3.2.1). Manifold forests [20] are another method for non-linear encoding and manifold learning. However, the construction of a manifold forest assume that the data distribution of each node is unimodal with a Gaussian prior. It requires the computation of the determinant of the covariance matrix to estimate the volume of the hyper-ellipsoid. This causes low-rank deficiency problems when dealing with high-dimensional inputs. Last but not least, autoencoders can be used to learn

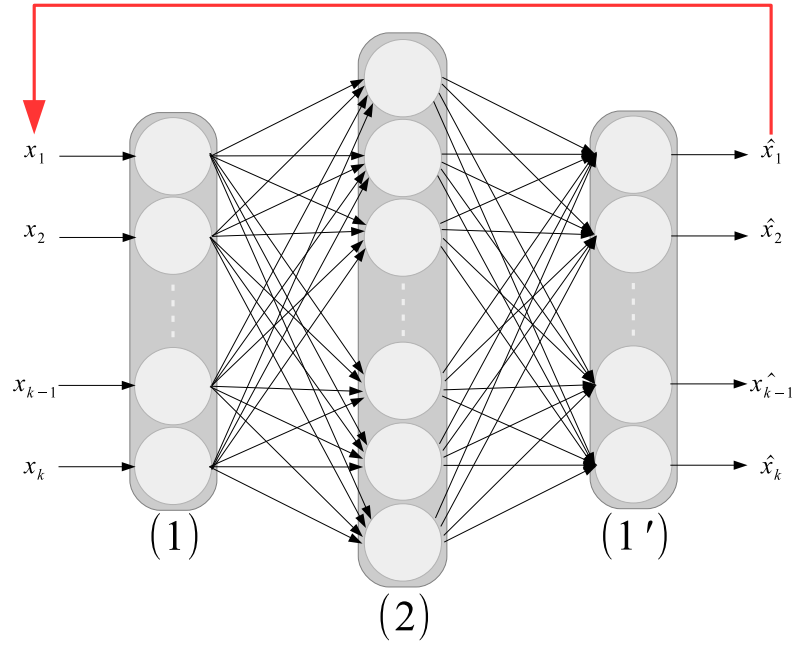


Figure 3.6: Architecture of an autoencoder

over-complete representation (*i.e.* if the hidden layer is larger than the input), that can be made more robust thanks to the introduction of a specific reconstruction criterion (Section 3.3.3.2.2).

**3.3.3.2.1 Unsupervised training via a reconstruction criterion.** As with standard NNs, autoencoders are trained by applying a sequence of SGD (or mini-batch) updates using backpropagation of an error criterion. However, as the output representation  $\hat{\mathbf{x}}$  of an example  $\mathbf{x}$  does not depend on a ground truth labelling, the error can be computed in an unsupervised way by using the  $\mathcal{L}_2$ -loss between the input and its reconstruction:

$$E(W) = \|\hat{\mathbf{x}} - \mathbf{x}\|_2^2 \quad (3.18)$$

In this equation, recall that  $\mathbf{x}$  is the *clean* input version and  $\hat{\mathbf{x}}$  is a reconstruction provided by the autoencoder from a *noisy* input. In order to provide a non-linear encoding of the input, we generally use a non-linear hidden layer parametrized by weight matrix  $\mathbf{w}$  and bias vector  $\mathbf{b}$ :

$$\mathbf{y} = \sigma(\mathbf{w} \cdot \mathbf{x} + \mathbf{b}) \quad (3.19)$$

Another trick to restrain the number of parameters of the autoencoder is to use *tied* weights, *i.e.* the decoding weight matrix is the transpose of the encoding matrix. With that regularization, an affine decoder is given by:

$$\hat{\mathbf{x}} = \mathbf{w}^T \cdot \mathbf{y} + \mathbf{c} \quad (3.20)$$

With  $\mathbf{c}$  being the decoder weight vector. The parameter updates can be computed easily using Equations 3.10, 3.11 and 3.16 (or 3.17). Furthermore, the cost function  $E(W)$  can be enhanced to impose a particular structure on the intermediate representation provided by the hidden layer.

**3.3.3.2.2 Regularization schemes.** There exists a wide number of regularization schemes for autoencoders. In this section we briefly review some of the most popular ones, with an emphasis on the denoising criterion [91] that we will use in Chapter 5 for face analysis. It consists in generating a randomly corrupted version  $\tilde{\mathbf{x}}$  of each training example  $\mathbf{x}$ , by adding either masking noise (*i.e.* a randomly chosen fraction of the input dimensions is set to 0), salt-and-pepper noise (*i.e.* a randomly chosen fraction of the input dimensions is set to 0 or 1) or gaussian additive noise with mean zero and spread  $\sigma$  (*i.e.*  $\tilde{\mathbf{x}} \sim \mathbf{x} + \mathcal{N}(0, \sigma^2)$ ). The cost function thus becomes:

$$E(W) = \|\hat{\mathbf{x}} - \mathbf{x}\|_2^2 \quad (3.21)$$

This means that the network is trained to reconstruct the *clean* version of the input  $\mathbf{x}$  knowing only a randomly corrupted version  $\tilde{\mathbf{x}}$ . From a manifold learning perspective, this allows to learn more robust intermediate representations by projecting back examples that lie further from the manifold. It can also be seen as a way to prevent co-adaptation of the units, similarly to dropout (see Section 3.3.3.1.2).

Others popular regularization schemes involves the  $k$ -sparse autoencoder [60], which consists in selecting the  $k$  larger values (at train and test time) among the hidden layer activations, zeroing out the others. This is generally used along with large numbers of hidden units to generate sparse, over-complete representations. An alternative approach is the contractive autoencoder [70], which consists in penalizing the Frobenius norm of

the Jacobian of the non-linear mapping. This encourages low-valued (*flat*) derivatives which implies robustness to small variations of the input, similarly to what is done with denoising autoencoders though in a more explicit way.

### 3.3.3.3 Convolutional neural networks

Convolutional Neural Networks (CNNs [52]) are NNs whose structure is adapted for signal processing in general, and image processing in particular. As illustrated on Figure 3.7, the major difference with regular NNs introduced in Section 3.3.3 is that the weights of CNN layer are shared across one or more dimensions of the input signal.

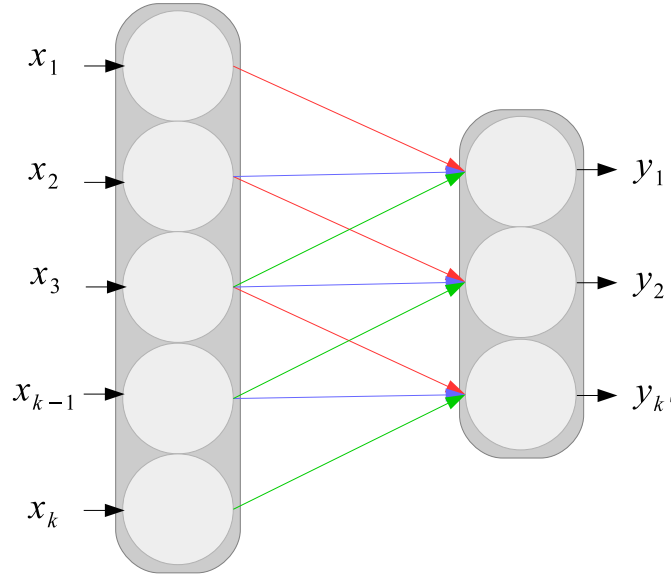


Figure 3.7: Shared weights across the input dimensions. Arrows with similar colors (blue, green and red) are associated with the same shared weights. Best viewed in color.

If the input  $\mathbf{x}^{(l)}$  of a CNN layer  $l$  is an image, its output  $\mathbf{y}^{(l)}$  can also be seen as a collection of images (often referred to as feature maps) which are obtained by “scanning” the pixels of  $\mathbf{x}^{(l)}$  with the units’ weights (symbolized by red, green and blue arrows on Figure 3.7). Hence, those weights act as filter kernels  $w_k^{(l)}$  we can write:

$$y_k^{(l)} = \sigma\left(\sum_j x_j^{(l)} * w_{kj}^{(l)} + b_k^{(l)}\right) \quad (3.22)$$

Everything is thus similar to the equations of generic feed-forward neural networks

introduced in Section 3.3.3.1, except that the net output of the layer is computed as a (valid) convolution product between kernels  $w_{kj}^{(l)}$  and input channels  $x_j^{(l)}$ , instead of a scalar product. Also, the bias term  $b_k^{(l)}$  is propagated to the whole feature map  $k$ . Figure 3.8 provides an example of a CNN architecture for image classification. Typically, a CNN architecture is composed of alternated convolutional and max-pooling layers that operate on small non-overlapping regions (generally  $2 \times 2$ ,  $3 \times 3$  or  $4 \times 4$  for very large images). Alternatively, one can use other forms of pooling such as mean-pooling,  $\mathcal{L}_2$  or stochastic pooling [105]. This pooling step is used to create position invariance over local regions as well as to reduce subsequent computation time by progressively downsample image size throughout the network layers. The last layers of the network are generally basic feed-forward prediction layers that are also called fully-connected (FC) layers - see Section 3.3.3.

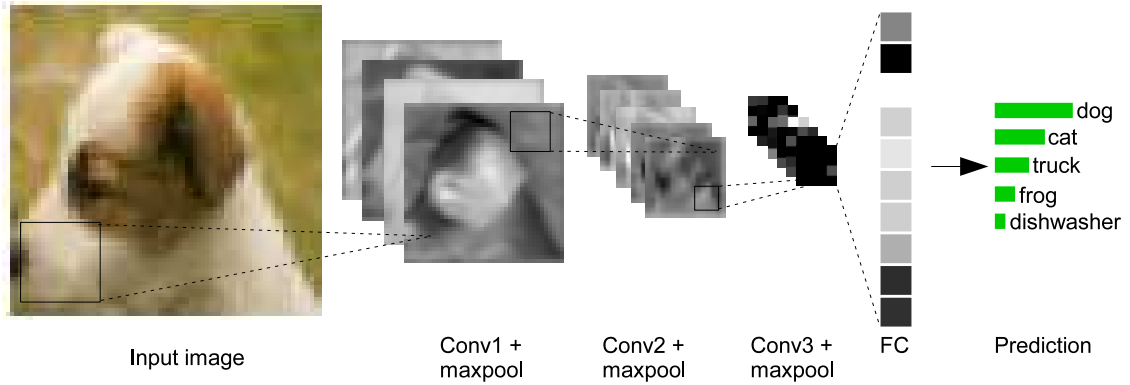


Figure 3.8: An example of CNN architecture for image classification. Credits to the Stanford “Convolutional Neural Networks for Visual Recognition” course for the pictures.

Similarly to regular feed-forward neural networks, the parameter update for kernel  $w_{ji}^{(l)}$  of layer  $l$  is given by:

$$\frac{\partial \epsilon(\mathbf{y}^{(l)}, \mathbf{y}^*)}{\partial w_{ji}^{(l)}} = \text{upsample}(\delta_j^{(l)}) * \frac{\partial y_j^{(l)}}{\partial w_{ji}^{(l)}} \quad (3.23)$$

With  $*$  the valid convolution operator and *upsample* the “inverse” of the pooling operator (in the case of max pooling, we need to keep track of the positions of the maxima). The error backpropagated from layer  $l$  to layer  $l - 1$  becomes:



$$\delta_i^{(l-1)} = \sum_{p=1}^{k''} \text{upsample}(\delta_p^{(l)} \frac{\partial y_p^{(l)}}{\partial \text{net}_p^{(l)}}) * \tilde{w}_{pi}^{(l)} \quad (3.24)$$

Where  $*$  is, this time, the full convolution operator (valid convolution with zero padding of the borders). The goal is thus to create a pyramid of gradually abstracted representations of the input image (as can be seen on Figure 3.8, the bottom layers usually learn Gabor wavelet-like edge detectors while the top CNN layers learn more sophisticated object detectors), by learning the prediction and representation stages in a unified, top-down fashion. Ideally, learning deep image representations using CNNs allows to avoid the use of *ad hoc*, hand-crafted representations such as SIFT or HOG. However, it also involves a lot of hyperparameter tuning, from the network architecture (e.g. the number of CNN/FC layers, the number and size of each layer kernel, the maxpool type and window size, or the weights' initialization) to the training hyperparameter (learning rate, weight decay, batch size). Furthermore, other machine learning algorithms can be used for the prediction stages, such as Support Vector Machines [86] or the recent Neural Decision Forests [48].

### 3.3.4 Randomized Decision Trees

Randomized Decision Trees, also referred to as Random Forests (RFs) are a popular machine learning framework introduced in the seminal work of Breiman [11]. In this section, we draw a non-exhaustive overview of the method, starting with the basic idea of growing a single decision tree to predict new data (Section 3.3.4.1), and highlighting the interest of learning collections of multiple randomized tree predictors, regarding some nice properties of RF w.r.t. overfitting on the training data as well as its generalization capacities on new data. We also show how the RF framework can be straightforwardly adapted for multiclass classification problems as well as regression tasks, with an emphasis on class balance issues.

### 3.3.4.1 A single decision tree

Without loss of genericity, a generic binary decision tree can be recursively described by a node  $n$ , which is either called a terminal node, or a split node, as shown on Figure 3.9. If node  $n$  is terminal (depicted with green arrows on Figure 3.9), then it is associated a terminal distribution, which consists in either an assignment probability  $p(c|\mathbf{x})$  for each class  $c \in \{1, \dots, \mathcal{C}\}$  in case of a classification task, or to a prediction value  $\hat{y}$  in case of a regression task. On the flip side, if node  $n$  is not terminal, it is called a split node (flat gray nodes on Figure 3.9) and contains a parametric split function  $\phi^n$  associated with a threshold  $\theta^n$ , and the address of left and right subtrees w.r.t. node  $n$ .

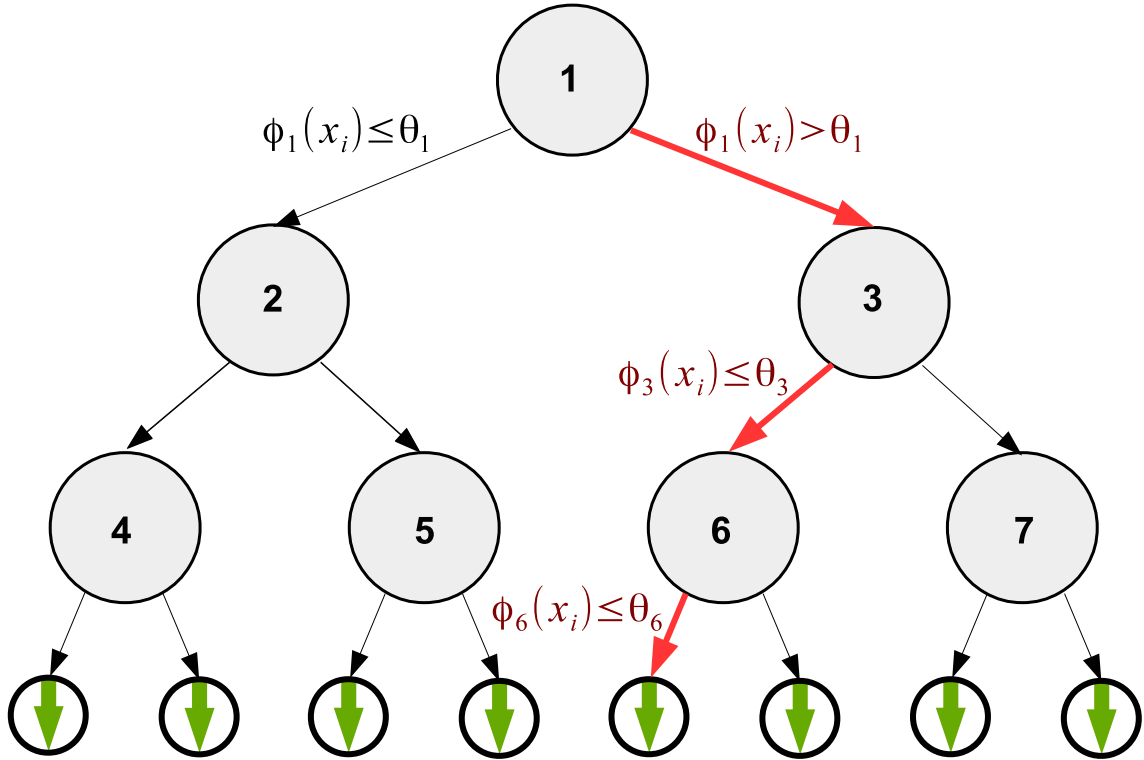


Figure 3.9: A single decision tree

Given an input  $\mathbf{x} = (x_1, \dots, x_k)^T$ , the split function associated to node  $n$  controls if  $\mathbf{x}$  will be routed to the left or to the right subtree w.r.t. node  $n$ . Formally, the split is defined as a binary function:

$$\delta^n(\mathbf{x}) = \begin{cases} 1 & \text{if } \phi^n(\mathbf{x}) > \theta^n \\ 0 & \text{otherwise.} \end{cases} \quad (3.25)$$

With the convention  $\mathbf{x}$  goes to the left if  $\delta^n(\mathbf{x}) = 0$ , and to the right if  $\delta^n(\mathbf{x}) = 1$ . Thus, generally speaking we can write the output probability that a decision tree outputs class  $c$  as:

$$p(c|\mathbf{x}) = \sum_l \mu^l(\mathbf{x}) p^l(c) \quad (3.26)$$

With  $p^l(c)$  the prediction outputted by leaf node  $l$  and  $\mu^l(\mathbf{x})$  the probability to reach leaf node  $l$ . For a regression task, we have:

$$\hat{y} = \sum_l \mu^l(\mathbf{x}) \hat{y}^l \quad (3.27)$$

In the case of a classical decision tree (Equation (3.28)), there is exactly only one non-zero  $\mu^l(\mathbf{x})$  coefficient, depending on a hard path that input  $\mathbf{x}$  takes in the tree. Formally, for each leaf  $l$  a path to the trees can be defined as two sets  $\mathcal{N}_l^{left}$  and  $\mathcal{N}_l^{right}$  of nodes  $n$  for which  $l$  respectively belongs to the left and right subtrees defined under node  $n$ . A hard path through the tree down to leaf  $l$  can thus be defined as a product of Kronecker deltas  $\delta^n(\mathbf{x})$  for each node  $n \in \mathcal{N}_l^{right}$ , and  $1 - \delta^n(\mathbf{x})$  for each node  $n \in \mathcal{N}_l^{left}$ .

$$\mu^l(\mathbf{x}) = \prod_{n \in \mathcal{N}_l^{right}} \delta^n(\mathbf{x}) \prod_{n \in \mathcal{N}_l^{left}} (1 - \delta^n(\mathbf{x})) \quad (3.28)$$

The split function  $\phi^n$  can be of two kinds: first, it can look at only one dimension of the input vector:

$$\phi^n(\mathbf{x}) = x_j \quad (3.29)$$

In that case, we refer to it as axis-aligned splits. Alternatively, it can consist in a linear combination of a number of input dimensions:

$$\phi^n(\mathbf{x}) = \sum_{j=1}^k \beta_j x_j \quad (3.30)$$

In such a case, the splits are called oblique splits. Moreover, in certain cases, the input dimensions cannot be explicitly represented in memory, as the potential number of combinations may be too important (consider, for example, the case where we want to take as features a number of triplets of pixels in an image). In those cases, as it will be shown in what follows, axis-aligned split candidates can be generated on-the-fly from the source data itself (e.g. the image in this case). Thus, growing a decision tree amounts to find successive “good” hyperplanes (*i.e.* split functions) that separate the training data from the different classes.

We now aim at growing a decision tree upon  $N$  examples  $\mathbf{X} = \{x_{i,j}\}_{i=1,\dots,N,j=1,\dots,k}$  and corresponding class label vector  $(c_1, \dots, c_N)^t$  with  $c_i \in \{1, \dots, \mathcal{C}\} \forall i \in \{1, \dots, N\}$ . There exists a number of variants for the induction of decision trees. Perhaps the most popular one is the ID3 algorithm proposed by Quinlan [67]. In the case of a classification task, for each node  $n$  starting from the root node (node 1 on Figure 3.9), it consists in evaluating, for each possible split candidate  $\{\phi^n, \theta^n\}$ , an impurity criterion  $H(\phi^n, \theta^n)$  relatively to the induced partition of the data. More specifically, for each split candidate, the input data at node  $n$  either goes to the left or to the right subtree and the class repartition  $\{r_1^l, \dots, r_C^l\}$  and  $\{r_1^r, \dots, r_C^r\}$  for those respective subtrees can be computed accordingly. For classification, the impurity criterion  $H$  is thus computed either as Shannon entropy:

$$H(\phi^n, \theta^n) = \sum_{c=1}^{\mathcal{C}} -r_c^l \log(r_c^l) + \sum_{c=1}^{\mathcal{C}} -r_c^r \log(r_c^r) \quad (3.31)$$

or as Gini impurity measurement:

$$H(\phi^n, \theta^n) = \sum_{c=1}^{\mathcal{C}} r_c^l (1 - r_c^l) + \sum_{c=1}^{\mathcal{C}} r_c^r (1 - r_c^r) \quad (3.32)$$

In the case of a regression task, we seek to minimize the variance of the subtrees in term of the ground truth predicted value  $y_i^*$  of elements that belong to either left or right subtrees:

$$H(\phi^n, \theta^n) = \frac{\sum_{i \in \mathcal{L}^n} (y_i^* - \bar{y}^l)^2}{2|\mathcal{L}^n|} + \frac{\sum_{i \in \mathcal{R}^n} (y_i^* - \bar{y}^r)^2}{2|\mathcal{R}^n|} \quad (3.33)$$

Where  $\mathcal{L}^n$  and  $\mathcal{R}^n$  are the set of examples that goes to the left and right subtrees

induced by split candidate  $\{\phi^n, \theta^n\}$ , respectively. Recall that those are defined as  $\mathcal{L}^n = \{i \in [1, N] / \delta^n(\mathbf{x}_i) = 0\}$  and  $\mathcal{R}^n = \{i \in [1, N] / \delta^n(\mathbf{x}_i) = 1\}$ . Finally,  $\bar{y}^l$  and  $\bar{y}^r$  are the mean predicted value from examples that belongs to those ensembles.

In order to avoid overfitting, in the case of a single decision tree, it is generally necessary to use some form of early stopping, for example by stopping the node splitting process either when the number of examples falling into a specific node becomes lower than a specific value, or when the information gain (the difference between the purity of a node and the average purity of its children) is below a threshold. In that case, a leaf node is set, which contains the class distribution of examples falling into this node. Similarly to neural networks (see Section 3.3.3), decision trees offer the advantage to potentially be able to model complex functions by setting non-linear subdivisions of the input space under the form of successive hyperplanes. Furthermore, the evaluation runtime is very low as a low number (*i.e.* logarithmic on the number of nodes) of nodes are actually evaluated for a given test example, especially when one considers axis-aligned splits. However, as is, they suffer from a number of drawbacks. First, contrary to neural nets, the training is essentially performed offline. Secondly, they tend to strongly overfit on the training data which is particularly relevant for computer vision tasks, as the data is generally high-dimensional with a lot of noisy, irrelevant features. This, in turn, limits the capabilities of decision trees to generalize on unseen data to a significant extent.

### 3.3.4.2 Ensemble of randomized trees

In order to highlight the limitations of using a single decision tree for classification, we propose a small benchmark on the *spiral* dataset. It consists in approximating two non-overlapping spirals in a two-dimensional space, by having only access to a restricted number of data points. Thus, we aim at generating classification models that encapsulate the spiral structure by providing only a restricted set of binary labelled coordinates. Those results are illustrated on Figures 3.10, 3.11, 3.12 and 3.13. On the one's hand, we can see on Figure 3.12 that with 10000 points the spiral is globally well “understood” by a single decision tree, even though some details are missing. Furthermore, it has a very coarse aspect with square borders, which is due to the limitations of using a single axis-aligned

split decision tree for classification. Moreover, looking at Figure 3.10, one can see that the spiral aspect is not satisfying when the tree is grown using less training data (e.g. 2000 points). This illustrates the limited generalization capacity of a single decision tree trained on restricted amount of data.

On the other hand, Figures 3.11 and 3.13 are obtained by using an ensemble of 50 randomized decision trees, upon 2000 and 10000-points datasets, respectively. The procedure for generating such tree collections will be detailed more in-depth within the following sections. Note that the spiral aspect is globally better understood by the models, especially when the number of points is low. This illustrates the fact that ensemble of randomized trees are in general more robust to label noise and less prone to overfitting than single decision trees.

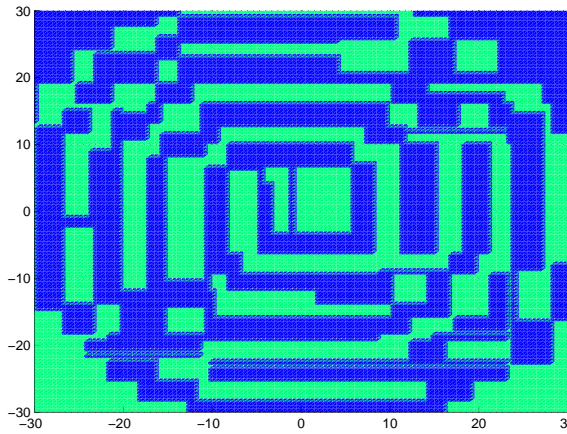


Figure 3.10: Spiral approximation using a single decision tree (2000 data points)

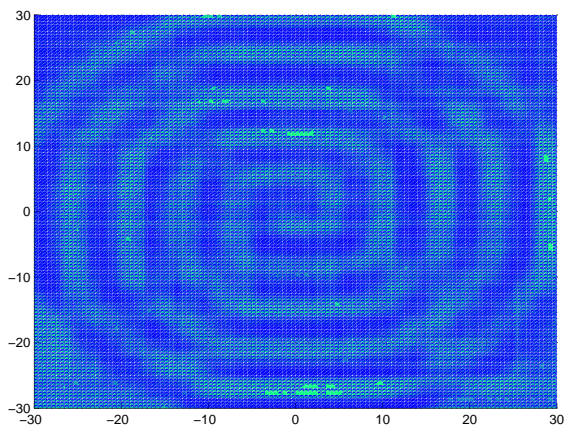


Figure 3.11: Spiral approximation using 50 randomized trees (2000 data points)

#### 3.3.4.2.1 Enhancing prediction accuracy with perturb and combine approaches

In order to obtain a more accurate model, it is common in the machine learning literature to gather a set of individually weak, but somehow complimentary, predictive models. Notorious examples of this paradigm are Boosting [30], Bagging [10] and Random Subspace [38]. RF associates two of the aforementioned methods to significantly increase the prediction accuracy of decision trees. Those methods are illustrated on Figure 3.14.

It has been proven by Breiman in [11] that an upper bound of the generalization error



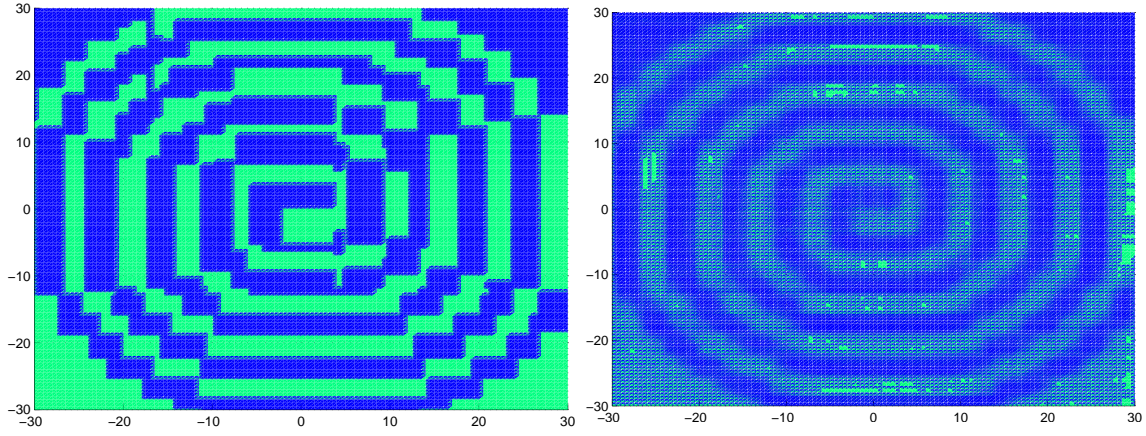


Figure 3.12: Spiral approximation using a single decision tree (10000 data points)      Figure 3.13: Spiral approximation using 50 randomized trees (10000 data points)

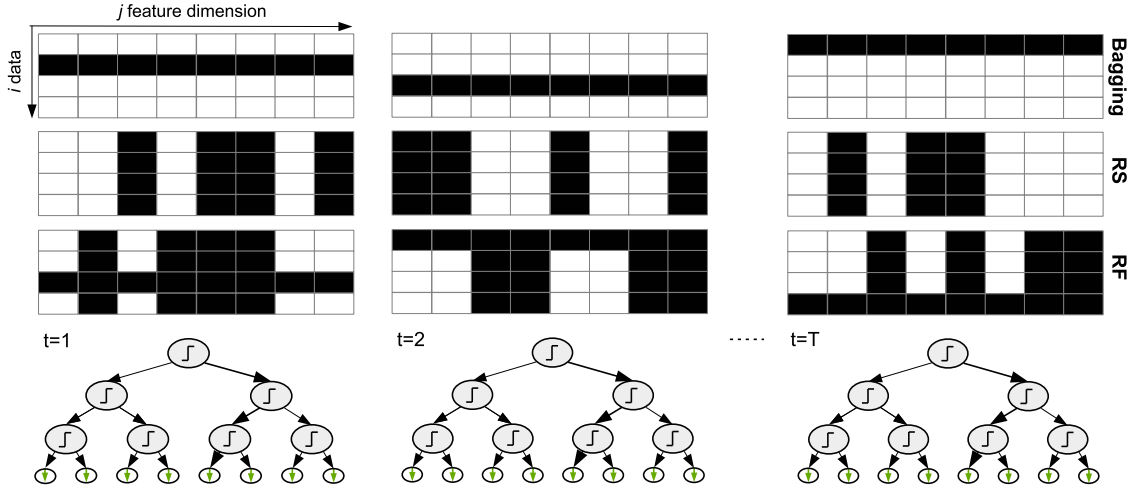


Figure 3.14: Ensemble of Randomized decision trees. Blacked out squares indicate unused data, as opposed to white ones. Top row: Bagging: each tree is grown upon a restricted bootstrap of the training data. Middle row: random subspace (RS): each tree is grown using only a subset of the input features. Bottom row: Random Forest (RF) each tree is grown using only a subset of the training data and input features.

for RFs is given by the ratio  $\rho/s^2$ , where  $s$  denotes the average strength of the individual trees, and  $\rho$  denotes the correlation between these. Thus, the smaller  $\rho$ , and the higher  $s$ , the higher the accuracy of the forest becomes. Generally speaking, adding randomness allows to decrease the correlation between the trees, which in turn increases the prediction accuracy. As explained on Figure 3.14, the most common frameworks to generate

ensemble of randomized trees involve Bagging (top row), which consists in growing the trees upon bootstraps that are randomly sampled with replacement from the whole training corpus. Another approach is Random Subspace (RS - middle row), in which only a subset of dimensions from the feature vector that corresponds to each example is used. Ultimately, Random Forests (RF) consists in using both bagging and random subspace methods, *i.e.* training each tree with a reduced subset of examples and dimensions. The prediction for the whole forest is thus given by:

$$p(c|\mathbf{x}) = \frac{1}{T} \sum_{t=1}^T p_t(c|\mathbf{x}) \quad (3.34)$$

in case of a classification forest, or

$$\hat{y} = \frac{1}{T} \sum_{t=1}^T \hat{y}_t \quad (3.35)$$

for a regression task. In these equations,  $p_t(c|\mathbf{x})$  and  $\hat{y}_t$  respectively denote the predicted probability for class  $c$  and regressed value outputted by a tree  $t$  of the forest. As a result, RFs are generally more robust to noisy and irrelevant features, as well as outliers within the training corpus, as compared to using a single decision tree trained using a deterministic method. They are also less prone to overfitting on the training data, as the recombination of multiple, independently weaker models allows to generate a smoother partition of the input space. It is stated in [7] that ensembles of randomized trees can be seen as one layer deeper as single decision trees, hence a more expressive and powerful prediction framework. It is illustrated on Figures 3.11 and 3.13, that were generated using RFs with 50 trees, each one grown on an uniformly-sampled 66% bootstrap of the input data, and using only 1 among the 2 coordinates for splitting.

**3.3.4.2.2 Popular variants.** There exists a number of RF variants that may differ from each other w.r.t. when to set a leaf or a split node, how the split candidates are selected or how the leaf predictions are computed. In Breiman's original RF [11], trees are grown upon bootstraps that each contains approximately 66% of the examples. It also uses axis-aligned splits and, although there is no value for the number of dimensions  $k'$  that can be examined at each node, usually  $k' = \sqrt{k}$  is a good rule of thumb. Thresholds  $\theta^n$



are optimally chosen for each dimension. In Extremely Randomized Trees [32] (ERT- $k'$  with  $k'$  being the number of dimensions that are looked up for setting each split node) the difference with RF is that the thresholds  $\theta^n$  are randomly selected. Thus for ERT-1 (which is sometimes referred to as *Totally Randomized Trees*) the splits are set independently of the labels. Moreover, in order to bring the decorrelation between the trees to another level, as in [80], one may choose to generate different random subspaces at each split node rather than for each separate tree.

Breiman [11] also suggests not to use early stopping. The rationale behind that is that even though individual trees may overfit on the training data (*i.e.* the corresponding bootstraps), this phenomenon is compensated by the post-hoc combination of predictions. Moreover, an interesting feature of RFs is that its accuracy does not rely too much on the hyperparameter setting, making it relatively easy (e.g. compared to neural networks) to find a suitable parametrization. Indeed, a setting that causes a decrease of the individual tree predictive strength  $s$  (e.g. decreasing the number of features  $k'$  for setting the split nodes, or decreasing the bootstrap sizes) may have the opposite effect on the correlation between the trees  $\rho$ , compensating the accuracy loss.

**3.3.4.2.3 Dealing with imbalanced data.** There are multiple ways to adapt the RF framework to train on imbalanced data. This section aims at providing a very coarse overview of what can be used for that purpose. For more thorough benchmarks and comparisons between the different approaches, the reader shall refer to [14]. The first possibility to overcome class imbalance is to assign to each class a weight that is used (a) to weight the contribution of each class during the computation of the impurity criterion and (b) to weight the probability distribution that is stored within the leaf nodes. However, there is no consensus on how those weights can be automatically set in the general case, thus this method may involve an additional hyperparameter setting. An alternative approach is to either apply upsampling of the minority classes, or downsampling of the majority classes, in order to enforce class balance within the bootstraps. As stated in [14], downsampling has a slight edge in the general case compared to upsampling and explicit class weighting when it comes to prediction accuracy. Furthermore, as the bootstraps are

essentially smaller, it is also significantly faster. Thus, in what follow, we will stick to downsampling for the purpose of ensuring class balance.

**3.3.4.2.4 Out-of-bag error estimate.** The Out-of-bag (OOB) error is an error estimate that is specific to ensemble methods. For RFs, it consists in evaluating each tree only on data that was not used to grow that tree (*i.e.* that does not belong to the corresponding bootstrap). As stated by Bylander *et al.* [13], the OOB estimate is generally more pessimistic than traditional cross-validation (e.g. 5 or 10 fold) error estimates. It is also, for instance, 10 times faster to evaluate than implementing a 10-fold evaluation. For FER, in order to evaluate the RFs in a subject-independent fashion, we associate an ID to each subject. During training, we then generate bootstraps at the ID level and test each tree on the subjects that were not included in the corresponding bootstrap, hence a valid subject-independent error estimate.

### 3.3.5 Neural Decision Forests

#### 3.3.5.1 Soft trees with probabilistic routing

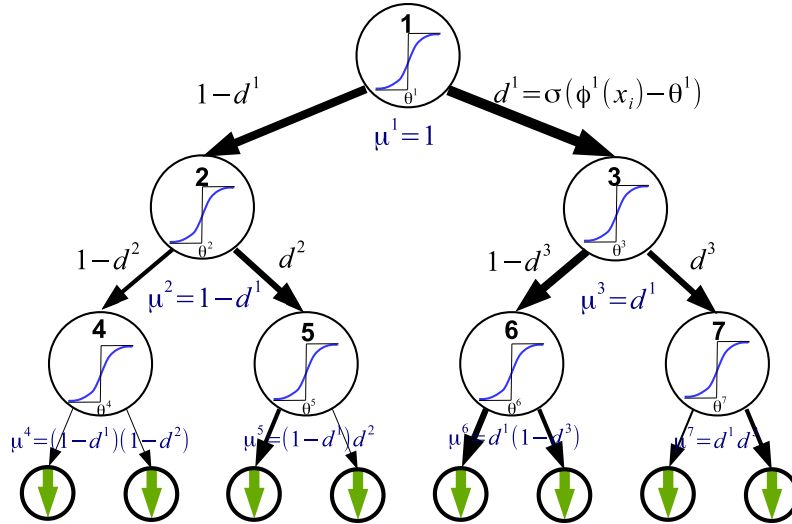


Figure 3.15: A single neural tree. Contrary to hard decision trees, an example  $x$  reaches each node of the tree with probability  $\mu^n \in [0, 1]$ . Those probabilities are computed as product of successive probabilistic splits  $d^n$  that correspond to a neuron activation.

As illustrated on Figure 3.15, Neural Decision Forests (NDFs) are a recent NN/RF hybrid introduced in [48]. In the case of a NDF, the probability  $\mu^n$  associated to each leaf node is defined as a product of continuous split probabilities associated to each probabilistic split node  $n$  (Equation (3.36)), that are parametrised by a Bernoulli random variable  $d^n \in [0, 1]$ . Taking the expected value for each node (which corresponds to an infinite number of samplings from tree  $t$ ), an example  $\mathbf{x}$  goes to the right subtree associated to node  $n$  with a probability given by the activation function  $d^n(\mathbf{x})$ , and to left subtree with probability  $1 - d^n(\mathbf{x})$ .

$$\mu^l(\mathbf{x}_i) = \prod_{n \in \mathcal{N}_l^{right}} d^n(\mathbf{x}) \prod_{n \in \mathcal{N}_l^{left}} (1 - d^n(\mathbf{x})) \quad (3.36)$$

The activation  $d^n(\mathbf{x})$  for node  $n$  is defined as a sigmoid function of the difference between a combination of the dimensions  $x_j$  parametrised by vector  $\beta^n$  and bias  $-\theta^n$ :

$$d^n(\mathbf{x}) = \sigma\left(\sum_{j=1}^k \beta_j^n x_j - \theta^n\right) \quad (3.37)$$

Thus, the calculus of  $d^n(\mathbf{x})$  can be seen as the activation of a neuron layer with weights  $\{\beta_j^n\}$  and bias  $-\theta^n$ . From a decision tree perspective, the successive activations  $d^n(\mathbf{x})$  define a soft routing through the trees, where each leaf node  $l$  is reached with probability  $\mu^l$ .

### 3.3.5.2 Online learning with recursive backpropagation

The prediction error  $\epsilon_t^l$  for a given class  $c$  and a leaf  $l \in \mathcal{L}$  of tree  $t$  can be computed as the Euclidean distance between the leaf prediction  $p_t^l(c)$  and ground truth label probability  $p^*(c|\mathbf{x}) = 1$  if  $c = c_i$ , 0 otherwise. The prediction error for the whole tree is thus equal to:

$$\epsilon_t(\mathbf{x}) = \sum_l \mu^l(\mathbf{x}) \epsilon_t^l \quad (3.38)$$

Hence, for any parameter  $\phi^n$  (*i.e.* a feature weight  $\beta_j^n$  or the threshold value  $\theta^n$ ), the (non-regularized) parameter update is given by Equation (3.39) with  $\alpha_t$  the learning rate hyperparameter for tree  $t$ .

$$\phi^n \leftarrow \phi^n - \alpha_t \frac{\partial \epsilon_t(\mathbf{x})}{\partial \phi^n} \quad (3.39)$$

As in a standard neural network, the learning rate can be set to a constant value  $\alpha_0$  (or a decreasing function of the number of training epochs). Alternatively, as proposed in [48], trees from the NDF can be randomly selected and updated, which can be seen as some way to add diversity, similarly to dropout regularization [82], but at the tree level. Formally,  $\alpha_t = \alpha_0$  if  $t = t_0$  (0 otherwise) with  $t_0 \sim \mathcal{U}_{[1, T]}$ . Moreover, the derivatives of  $\epsilon_t$  can be calculated recursively, as it is illustrated on Figure 3.16.

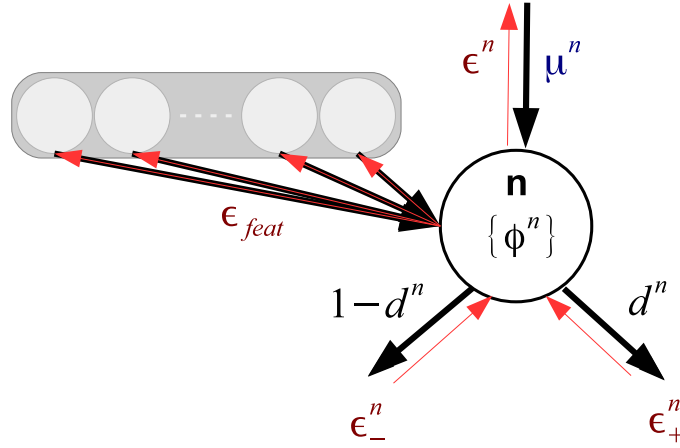


Figure 3.16: Error backpropagation on a single node of a NDF. The parameter update can be computed by applying a recursive call to find the values of the errors for the left and right subtrees, respectively denoted by  $\epsilon_-^n$  and  $\epsilon_+^n$ . Those quantities also give rise to the error for node  $n$   $\epsilon^n$ .

Specifically, for a split node  $n$  we can split the sum in Equation 3.38 in three, by grouping the leaves that belong to the left  $\mathcal{L}(n)$  and right subtrees  $\mathcal{R}(n)$ , and those who do not belong to those subtrees:

$$\epsilon_t(\mathbf{x}) = \sum_{l \in \mathcal{L}(n)} \mu^l(\mathbf{x}) \epsilon^l + \sum_{l \in \mathcal{R}(n)} \mu^l(\mathbf{x}) \epsilon^l + \sum_{l \notin \mathcal{L}(n), l \notin \mathcal{R}(n)} \mu^l(\mathbf{x}) \epsilon^l \quad (3.40)$$

While the first and second term respectively depend on  $1 - d^n(\mathbf{x})$  and  $d^n(\mathbf{x})$ , the last term does not depend at all on parameter  $\phi^n$ . We can thus write the summation as:

$$\epsilon_t(\mathbf{x}) = \mu^n(\mathbf{x})(1 - d^n(\mathbf{x}))\epsilon_-^n(\mathbf{x}) + \mu^n(\mathbf{x})d^n(\mathbf{x})\epsilon_+^n(\mathbf{x}) + \sum_{l \notin \mathcal{L}(n), l \notin \mathcal{R}(n)} \mu^l(\mathbf{x})\epsilon^l \quad (3.41)$$

With  $\epsilon_-^n(\mathbf{x})$  and  $\epsilon_+^n(\mathbf{x})$  the errors respectively for the left and right subtrees. Thus, we have:

$$\frac{\partial \epsilon_t(\mathbf{x})}{\partial \phi^n} = \mu^n(\mathbf{x}) \frac{\partial d^n(\mathbf{x})}{\partial \phi^n} (\epsilon_+^n(\mathbf{x}) - \epsilon_-^n(\mathbf{x})) \quad (3.42)$$

With:

$$\begin{cases} \frac{\partial d^n(\mathbf{x})}{\partial \theta^n} &= -d^n(\mathbf{x})(1 - d^n(\mathbf{x})) \\ \frac{\partial d^n(\mathbf{x})}{\partial \beta_j^n} &= x_j d^n(\mathbf{x})(1 - d^n(\mathbf{x})) \end{cases} \quad (3.43)$$

Moreover, the error up to node  $n$  is computed as:

$$\epsilon^n = d^n(\mathbf{x})\epsilon_+^n(\mathbf{x}) + (1 - d^n(\mathbf{x}))\epsilon_-^n(\mathbf{x}) \quad (3.44)$$

Once the trees are initialized, training samples are sequentially chosen from the data (SGD or mini-batch) and a forward pass through the trees provides the values of the probabilities  $\mu^n(\mathbf{x})$  and activations  $d^n(\mathbf{x})$  for each node  $n$ . Parameters can thus be updated using Equations 3.39, 3.42 and 3.43. Those steps are summarized in Algorithm 1. For each node  $n$ , `NDF_backprop` provides an update to the error that can optionally be backpropagated up to the feature level  $\epsilon_{\text{feat}}$  by recursively calling `NDF_backprop` on the left and right subtrees, which respectively return the errors  $\epsilon_-^n(\mathbf{x})$  and  $\epsilon_+^n(\mathbf{x})$ . The prediction error is thus recursively backpropagated from the leaves up to the root of the trees.

The authors of [48] suggest using a two-step iterative optimization scheme. First, a forward pass through the trees provides the activations  $d^n(\mathbf{x})$  and probabilities  $\mu^n(\mathbf{x})$  for each node for one (SGD) or a batch of examples. Then, after a specific number of epochs, the leaf probabilities are updated following a convex optimization scheme while the parameters for all the split nodes of the forests are fixed.

---

**Algorithm 1** NDF\_backprop

---

**Input:** example  $\mathbf{x}$  with label probability  $p^*$ , node  $n$  with probability  $\mu^n(\mathbf{x})$  and activation  $d^n(\mathbf{x})$ , learning rate  $\alpha$

**Output:** backpropagated error  $\epsilon^n(\mathbf{x})$ , error backpropagated to the feature level  $\epsilon_{\text{feat}}(\mathbf{x})$

**if**  $n$  is a leaf node **then**  $\epsilon^n(\mathbf{x}) \leftarrow \|p^n(\mathbf{x}) - p^*(\mathbf{x})\|^2$

**else**

$\epsilon_+^n(\mathbf{x}) \leftarrow \text{NDF\_backprop}(\mathbf{x}, \text{right}(n), \alpha)$   $\triangleright$  recursive call on right subtree

$\epsilon_-^n(\mathbf{x}) \leftarrow \text{NDF\_backprop}(\mathbf{x}, \text{left}(n), \alpha)$   $\triangleright$  recursive call on left subtree

$e^n(\mathbf{x}) = \mu^n(\mathbf{x})d^n(\mathbf{x})(1 - d^n(\mathbf{x})) \cdot (\epsilon_+^n(\mathbf{x}) - \epsilon_-^n(\mathbf{x}))$   $\triangleright$  split node parameter update

$\beta^n \leftarrow \beta^n - \alpha \mathbf{x} e^n(\mathbf{x})$

$\theta^n \leftarrow \theta^n + \alpha e^n(\mathbf{x})$

$\epsilon_{\text{feat}}(\mathbf{x}) \leftarrow \epsilon_{\text{feat}}(\mathbf{x}) - \alpha \beta^n e^n(\mathbf{x})$

$\epsilon^n \leftarrow d^n(\mathbf{x})\epsilon_+^n(\mathbf{x}) + (1 - d^n(\mathbf{x}))\epsilon_-^n(\mathbf{x})$

**end if**

---

### 3.3.6 Evaluation protocols

In machine learning in general, it is common that the training error estimate reaches very low values while the test error converges to a strictly positive value or, worse, increases (see Section 3.3.2). Thus, measuring the training error is not a good indicator of the performance of an automatic recognition system. From a face analysis point of view, imagine we train an algorithm for aligning feature points or recognizing the expressions using only training instances that come from one specific subject. The learned models will exhibit very low error on images that correspond to that subject, but will be completely useless when applied to other subjects.

A more significant test would be to evaluate our algorithms on examples that are not used to train the algorithms. However, from the point of view of designing an automated face analysis framework, the goal is to design a pattern recognition system that can generalize well on the facial morphology of new subjects. Hence, we have to ensure that the subjects that we evaluate our predictive models on are not used at training time. Traditional cross-validation estimates can be generated at the subject level by assigning

each subject a specific ID and generating the train/test partitions at the level of those IDs, as it was detailed in Section 3.3.4.2.4 in the case of the OOB error estimate for bagged classifiers. In what follows, we mainly evaluate RFs and NDFs using subject-independant OOB and 5/10-fold cross validation error estimates, and NNs 5-fold cross-validation. Last but not least, it is often interesting to evaluate the predictive models in a cross-database fashion (*i.e.* training on one database and testing on another one) in order to evaluate the capabilities of the algorithms to generalize on new environmental conditions (*i.e.* studying on how an expression recognition system trained on lab-recorded data with near-frontal head poses can perform in less controlled conditions).

# Chapter 4

## Pairwise Conditional Random Forests

### 4.1 Overview

As stated in Section 2.2.1, it is generally easier for a human observer to distinguish between the different expressions if one has access to the evolution of the face over time. However, effectively extracting suitable representations from spatio-temporal video patterns is a challenging problem as expressions may occur with various offsets and at different paces. There is no consensus either on how to combine those representations flexibly enough so as to generalize on unseen data and possibly unseen temporal variations. Towards this end, we introduce Pairwise Conditional Random Forest (PCRF) algorithm, which is a new formulation for training trees using low-level heterogeneous static (spatial) and dynamic (spatio-temporal derivative) features within the Random Forest (RF) framework. Conditional Random Forests have recently been used by Dantone *et al.* [22] as well as Sun *et al.* [84] in the field of facial alignment and human pose estimation, respectively. The authors of these papers generated collections of trees for specific, quantized values of a global variable (such as head pose [22] and body torso orientation [84]) and used prediction on this global variable to draw dedicated trees, resulting in more accurate predictions.

The basic idea developped in this chapter is to integrate spatio-temporal information under the form of transition classification in order to perform FER from video sequences. As shown on Figure 4.1, on the one hand, a static classifier (blue arrow) only use cues



relative to the current frame for which we want to predict the facial expression. On the other hand, the prediction outputted by our PCRF method is the combination of multiple transition classifications (green arrows). Those transitions are evaluated on pairs of images drawn from multiple time gaps in the sequence ( $n - n_3$  and  $n$ ,  $n - n_2$  and  $n$  as well as  $n - n_1$  and  $n$  on Figure 4.1). We will explain, in what follows, that, from a Random Forest perspective, it can be seen as extending the set of features that is used to grow the trees, effectively increasing individual tree strength as well as increasing decorrelation between the trees. The upside of the proposed approach is that it is fairly flexible, e.g. it is independent of the pace at which the expression is displayed (as compared to the use of descriptors designed on spatio-temporal volumes), and do not require temporal consistency of the sequences for training and testing (as compared to e.g. Hidden Markov Models).

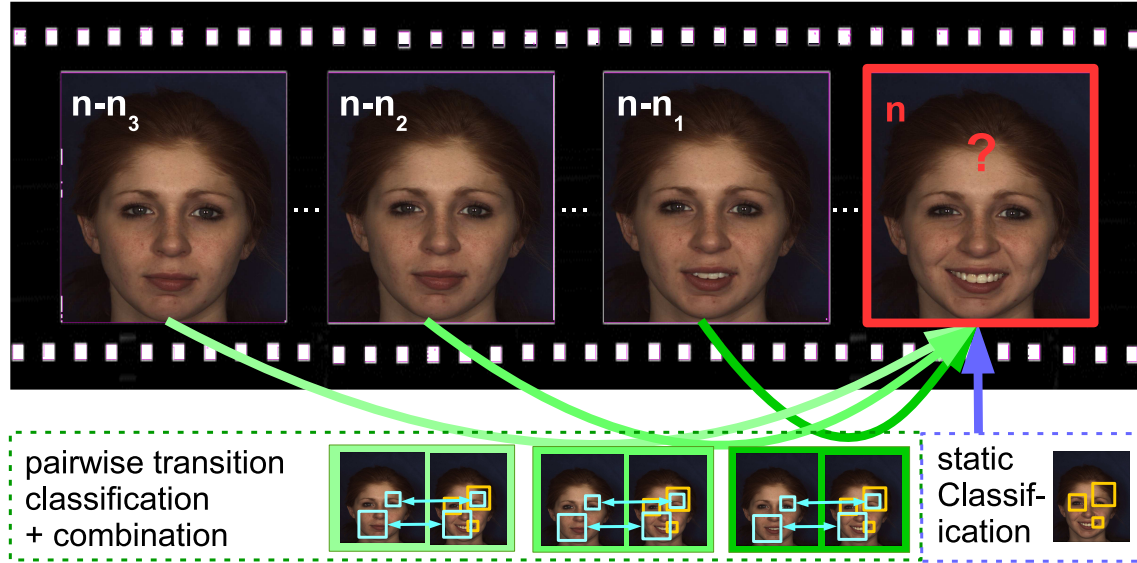


Figure 4.1: Transition (green) vs static (blue) classification.

Moreover, we propose to condition pairwise trees on specific expression labels to reduce the variability of ongoing expression transitions from the first frame of the pair to the other one. *In Extenso*, a head pose estimate can be used to draw trees from Multi-View PCRF (MVPCRF) collections to perform pose-robust FER.

This chapter is organized as follows: in Section 4.2 we describe our adaptation of the RF framework to learn expression patterns on still images from high-dimensional, hetero-

geneous (geometric/appearance) features. In Section 4.3 we present the PCRf framework for capturing spatio-temporal patterns that represent facial expressions. In particular, Section 4.3.1 illustrates how we can generate a pairwise dataset using available data. Section 4.3.2 highlights how we extend the pool of static features to capture (pairwise) dynamic information. In Section 4.3.3 we present the extension of traditional tree combination for averaging over time pairwise trees. In Section 4.4 we present an extension of PCRf for multi-view FER, with an emphasis on the preparation of a multi-view dynamic database for training and testing the models (Section 4.4.2) and on the evaluation of a sequence using MVPCRf models (Section 4.4.1). Finally, In Section 4.5 we show how our PCRf algorithm improves the accuracy on several FER datasets compared to a static approach as well as to state-of-the-art approaches. In Section 4.5.2 we report results from frontal view FER and in Section 4.5.3 we report accuracy for non frontal head poses, showing that our MVPCRf formulation substantially increases the robustness to pose variations. In Section 4.5.4 we report the ability of our framework to run in real-time. Finally, in Section 4.5.5 we discuss the interest of the proposed approach for real-time multi-view dynamic FER.

## 4.2 A static RF approach for FER

In order to perform static FER from still images, we adapt the framework presented in Section 3.3.4.2 in two main aspects: the bootstrap generation and the candidate feature selection process.

### 4.2.1 Bootstrap generation

We use a subject-wise bootstrap generation procedure for RF induction, as discussed in Section 3.3.4.2.4. More specifically, we assign each subject from a set of subjects  $\mathcal{S}$  a specific ID. Then, for each tree, we start with an empty bootstrap and sample with replacement on the vector of subject IDs until a specific fraction of elements (usually 66%) is added to the current bootstrap. From that point, we construct the index of OOB IDs simply by looking at the subjects' IDs that were not included in the bootstrap. Then,

we downsample the bootstrap iteratively, by removing at each step an element uniformly sampled from the majority class until all expression classes are represented by the same amount of elements (*i.e.* the lowest among all the classes). The rationale behind sampling at the ID level is that we want each subject morphology to influence the prediction on a equal foot regarding the number of annotated data for that subject (e.g. for certain datasets, one ID can correspond to only one frame whereas another one may give rise to ten times more samples). The downsampling step also allows to fight class imbalance at the tree level. Finally, it also allows to use OOB error estimate as a valid subject-independent error metric, which is more convenient for evaluating our algorithms than more traditional cross-validation or leave-one-subject-out estimates.

#### 4.2.2 Heterogeneous feature templates

As stated in Section 3.2, the information contained in geometric and appearance features is somewhat complementary. Moreover, as described in Section 3.3.4.2, collections of randomized trees benefit from using diverse information on the data, that in turns produces more decorrelated predictions from the individual trees. Following this idea, we grow trees using a combination of geometric and appearance feature templates  $\phi^{(1)}$  (Section 3.2.2),  $\phi^{(2)}$  (Section 3.2.3),  $\phi^{(3)}$  (Section 3.2.4.1).

During the RF induction step, for each node a number of split candidates are generated on-the-fly using one of the proposed feature templates with parameters sampled from a uniform distribution over their respective variation range. Namely, if we have a set of 49 feature points aligned on the face image (which corresponds to a mesh of 79 triangles), and we generated 9 feature channels (8 orientations plus gradient magnitude), we have for template  $\phi_{a,b}^{(1)}$   $a, b \sim \mathcal{U}[[1, 49]]$ . For template  $\phi_{a,b,c,\lambda}^{(2)}$ :  $a, b, c \sim \mathcal{U}[[1, 49]]$ ,  $\lambda \sim \mathcal{U}[[0, 1]]$ . Finally, for template  $\phi_{\tau,ch,s,\alpha,\beta,\gamma}^{(3)}$  we use  $\tau \sim \mathcal{U}[[1, 79]]$ ,  $ch \sim \mathcal{U}[[1, 9]]$ ,  $s \sim \mathcal{U}[0.1, 0.4]$  (recall  $s$  is multiplied by the inter-ocular distance),  $\alpha, \beta, \gamma \sim \mathcal{U}[0, 1]$ . Note that for the outer triangles (*i.e.* on the edge of the mesh) we instead sample the barycentric coordinates  $\alpha, \beta, \gamma \sim \mathcal{U}_{[-1,1]}$  in order to allow the sampling of the texture from outer regions (e.g. forehead, cheeks and chin). Each of these candidate features is associated to a set of thresholds  $\{\theta^i\}_{i=1,\dots,|\Theta|}$  to produce a binary split candidate. More precisely, the variation

range of each feature template is estimated beforehand and the candidate thresholds are uniformly sampled within that interval.

## 4.3 Pairwise Conditional Random Forests

### 4.3.1 Pairwise conditional tree collections

In this section we now consider pairs of images  $(\mathcal{I}', \mathcal{I})$  to train trees  $t$  that aim at outputting probabilities  $p_t(c|\mathcal{I}', c', l')$  of observing label  $c(\mathcal{I}) = c$  given image  $\mathcal{I}'$  and subject to  $c(\mathcal{I}') = c'$ , as shown in Figure 4.2. More specifically, for each tree  $t$  among the  $T$  trees of a RF dedicated to transitions starting from expression label  $c'$ , we randomly draw a fraction of subjects  $\tilde{\mathcal{S}} \subset \mathcal{S}$ . Then, for each subject  $s \in \tilde{\mathcal{S}}$  we randomly draw images  $\mathcal{I}'_s$  that specifically have label  $c'$ . We also draw images  $\mathcal{I}_s$  of every label  $c$  and create the pairs  $(\mathcal{I}'_s, \mathcal{I}_s)$  with label  $c$ . Note that the individual trees do not encode any sort of temporal evolution of the expressions, but rather differential information between pairs of images. In fact, two images of a pair need to belong to the same subject, but not necessarily to the same video. Indeed, we create pairs from images sampled across different sequences for each subject to cover all sorts of ongoing transitions. As we explained in Section 4.2.1, we then balance the pairwise bootstrap by downsampling the majority class w.r.t. the pairwise labels. Eventually, we grow tree  $t$  similarly to what we did in Section 4.2. The PCRf training algorithm is summarized in Algorithm 2.

### 4.3.2 Heterogeneous derivative feature templates

As shown on Figure 4.3, candidates for splitting the nodes are generated from an extended set of 6 feature templates  $\{\phi^{(i)}\}_{i=1,\dots,6}$ , three of which being the static features described in Section 4.2, that are applied to the second image  $\mathcal{I}$  of the pair  $(\mathcal{I}', \mathcal{I})$ , for which we want to predict facial expressions. The three remaining feature templates are dynamic features defined as the derivatives of static templates  $\phi^{(1)}, \phi^{(2)}, \phi^{(3)}$  with the exact same parameters. Namely, we have:

---

**Algorithm 2** Training a PCRF

---

**input:** images  $\mathcal{I}$  with labels  $c$ , number of candidate features  $\{k^{(i)}\}_{i=1,\dots,6}$  for templates  $\{\phi^{(i)}\}_{i=1,\dots,6}$

**for all**  $c' \in \mathcal{C}$  **do**  
    **for**  $t = 1$  to  $T$  **do**  
        randomly draw a fraction  $\tilde{\mathcal{S}} \subset \mathcal{S}$  of subjects  
         $pairs \leftarrow \{\}$   
        **for all**  $s \in \tilde{\mathcal{S}}$  **do**  
            draw samples  $\mathcal{I}'_s$  with label  $c'$   
            draw samples  $\mathcal{I}_s$  for each label  $c$   
            create pairwise data  $(\mathcal{I}'_s, \mathcal{I}_s)$  with label  $c$   
            add element  $(\mathcal{I}'_s, \mathcal{I}_s)$  to  $pairs$   
        **end for**  
        balance bootstrap  $pairs$  with downsampling  
        create new root node  $n$   
        call `treeGrowing`( $pairs, n, \{k^{(i)}\}_{i=1,\dots,6}$ )  
    **end for**  
**end for**

---

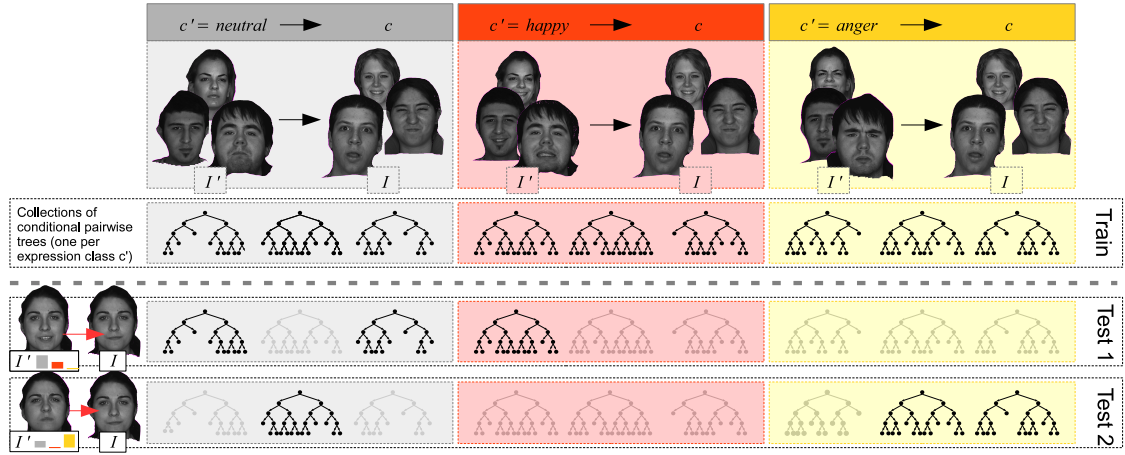


Figure 4.2: Example of pairwise tree collections for 3 basic expression classes. Expression probability predictions of previous images are used to sample trees from dedicated pairwise tree collections (one per expression class) that are trained using subsets of the (pairwise) training dataset, with only examples of ongoing transitions from a specific expression towards all classes. The resulting forest thus outputs an expression probability for a specific pair of images.

$$\left\{ \begin{array}{ll} \phi_{a,b}^{(1)}(\mathcal{I}', \mathcal{I}) & = \phi_{a,b}^{(1)}(\mathcal{I}) \\ \phi_{a,b,c,\lambda}^{(2)}(\mathcal{I}', \mathcal{I}) & = \phi_{a,b,c,\lambda}^{(2)}(\mathcal{I}) \\ \phi_{\tau, ch, s, \alpha, \beta, \gamma}^{(3)}(\mathcal{I}', \mathcal{I}) & = \phi_{\tau, ch, s, \alpha, \beta, \gamma}^{(3)}(\mathcal{I}) \\ \phi_{a,b}^{(4)}(\mathcal{I}', \mathcal{I}) & = \phi_{a,b}^{(1)}(\mathcal{I}) - \phi_{a,b}^{(1)}(\mathcal{I}') \\ \phi_{a,b,c,\lambda}^{(5)}(\mathcal{I}', \mathcal{I}) & = \phi_{a,b,c,\lambda}^{(2)}(\mathcal{I}) - \phi_{a,b,c,\lambda}^{(2)}(\mathcal{I}') \\ \phi_{\tau, ch, s, \alpha, \beta, \gamma}^{(6)}(\mathcal{I}', \mathcal{I}) & = \phi_{\tau, ch, s, \alpha, \beta, \gamma}^{(3)}(\mathcal{I}) - \phi_{\tau, ch, s, \alpha, \beta, \gamma}^{(3)}(\mathcal{I}') \end{array} \right. \quad (4.1)$$

As in Section 4.2, thresholds corresponding to the derivative features  $\phi^{(4)}$ ,  $\phi^{(5)}$ ,  $\phi^{(6)}$  are drawn from uniform distributions with new dynamic template-specific ranges estimated from the pairwise dataset beforehand. Also note that, as compared to a static RF, a PCRf model is extended with new derivative features that are estimated from a pair of images. When applied on a video, predictions for several pairs are averaged over time in order to produce robust estimates of the probability predictions.

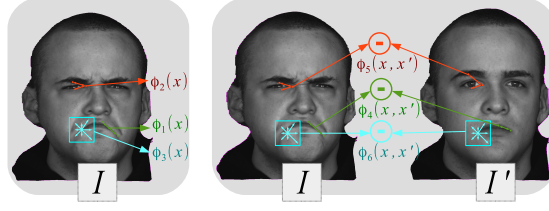


Figure 4.3: Static (left) and pairwise (right) feature templates.

### 4.3.3 Model averaging over time

We denote by  $p^n(c)$  the prediction probability of label  $c$  for a video frame  $\mathcal{I}^n$ . For a purely static RF classifier this probability is given by Equation (4.2):

$$p^n(c) = \frac{1}{T} \sum_{t=1}^T p_t(c|\mathcal{I}^n) \quad (4.2)$$

In order to use spatio-temporal information, we apply pairwise RF models to pairs of images  $(\mathcal{I}^m, \mathcal{I}^n)$  with  $\{\mathcal{I}^m\}_{m=n-N, \dots, n-1}$  the previous frames in the video. Those pairwise predictions are averaged over time to provide a new probability estimate  $p^n$  that takes into account past observations up to frame  $n$ . Thus, if we do not have prior information for those frames the probability  $p^n$  becomes:

$$p^n(c) = \frac{1}{NT} \sum_{m=n-N}^{n-1} \sum_{t=1}^T p_t(c|\mathcal{I}^m, \mathcal{I}^n) \quad (4.3)$$

In what follows, Equation (4.2) and Equation (4.3) will be referred to as the *static* and *full models*, respectively. Trees from the full model are likely to be stronger than those of the static one since they are grown upon an extended set of features. Likewise, the correlation between the individual trees is also lower thanks to the new features as well as the averaging over time. However, spatio-temporal information can theoretically not add much to the accuracy if the variability of the ongoing transitions is too large.

In order to decrease this variability, we assume that there exists a probability distribution  $p_0^m(c')$  to observe the expression label  $c'$  at frame  $m$ . Note that those probabilities can be set to purely static estimates (which is necessarily the case for the first video frames) or dynamic predictions estimated from previous frames. A comparison between those

approaches can be found in Section 4.5.2.1. In such a case, for frame  $m$ , pairwise trees are drawn from the tree collections (each one being conditioned to one expression label for the first frame of the pair) by sampling the distribution  $p_0^m$ , as shown in Figure 4.4. More specifically, for each previous frame  $m$  and expression label  $c'$ , we randomly select  $\mathcal{N}^m(c')$  trees over a PCRf model dedicated to transitions that start from expression label  $c'$ , trained with the procedure described in Section 4.3.1. We denote  $p_t(c|c')$  the probabilities outputted by a pairwise tree  $t$  conditioned on label  $c'$ . Equation (4.3) thus becomes:

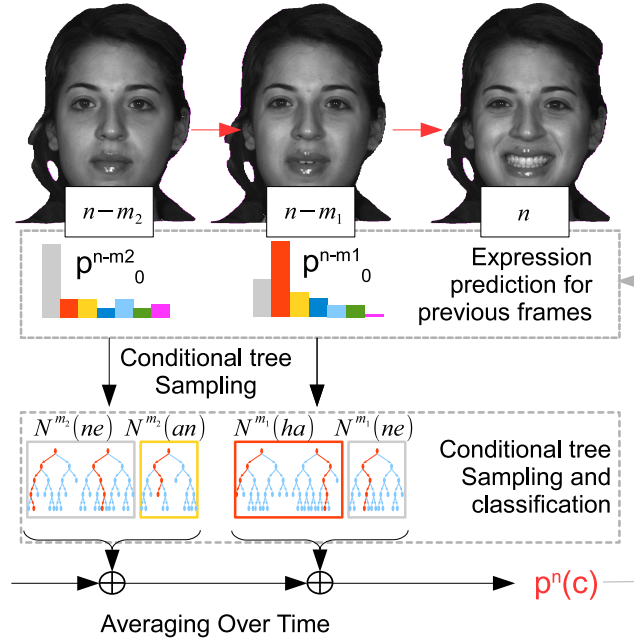


Figure 4.4: Averaging over time pairwise tree collections. For each previous frame in the sequence, trees are sampled from PCRf collections based on an expression probability prediction for those frames. the prediction probabilities outputted by those pairwise models are averaged over time to provide an estimate for current frame that takes into account the (pairwise) dynamics of the expression.

$$p^n(c) = \frac{1}{NT} \sum_{m=n-N}^{n-1} \sum_{c' \in \mathcal{C}} \sum_{t=1}^{\mathcal{N}^m(c')} p_t(c|\mathcal{I}^m, \mathcal{I}^n, c') \quad (4.4)$$



Where  $\mathcal{N}^m(c') \approx Tp_0^m(c')$  and  $T = \sum_{c' \in \mathcal{C}} \mathcal{N}^m(c')$  are the number of trees dedicated to the classification of each transition, which can be set in accordance with CPU availability. In our experiments, we will refer to Equation (4.4) as the *conditional model*. This conditional formulation helps to reduce the variability of the derivative features for each specialized pairwise RF. When predicting expression for a frame of a video, we can effectively use robust sequence-level expression estimates by averaging over time predictions conditioned on multiple, independent previous frames. Section 4.5 shows that using PCRF models for FER leads to significant improvements over both static and full models.

## 4.4 Multi-view extension for Pose-Robust Facial Expression Recognition

### 4.4.1 Averaging over time multi-view classifiers

In order to design a pose-robust recognition framework, we propose to condition the models described in Section 4.3.3 w.r.t a head pose estimate  $\omega(\mathcal{I}^n)$  for frame  $n$ . For that matter we quantize the pose space  $\Omega$  in  $k = \Gamma \times B$  pose bins  $\{\Omega_i = \Omega_{\gamma_i, \beta_i}\}_{i=1, \dots, k}$ , that are defined around yaw and pitch angles  $\gamma_i$  and  $\beta_i$ , respectively. We can thus rewrite Equation (4.2) as a static multi-view model (MVRF):

$$p^n(c) = \frac{1}{T} \sum_{\Omega_i \in \Omega} \sum_{t=1}^{\mathcal{N}(\Omega_i)} p_t(c|\mathcal{I}^n, \Omega_i) \quad (4.5)$$

At frame  $n$ , the head pose  $\omega(\mathcal{I}^n)$  is estimated first using an off-the-shelf posit algorithm [23]. Then, for each pose bin  $\Omega_i$ , a number  $\mathcal{N}(\Omega_i)$  of trees are selected based on a pose sampling probability distribution  $\mathcal{P}_{\Omega_i}(\omega^n)$  that we construct from the training data repartition, as it will be explained in Section 4.4.2. This is illustrated on Figure 4.5.

For that matter, we adapt Equation (4.4) by conditioning the expression-conditional model on pose estimation  $\omega(\mathcal{I}^n)$  (Equation (4.6)):

$$p^n(c) = \frac{1}{T} \sum_{m=n-N}^{n-1} \sum_{\Omega_i \in \Omega} \sum_{c' \in \mathcal{C}} \sum_{t=1}^{\mathcal{N}^m(c', \Omega_i)} p_t(c|\mathcal{I}^n, \mathcal{I}^m, \Omega_i, c') \quad (4.6)$$

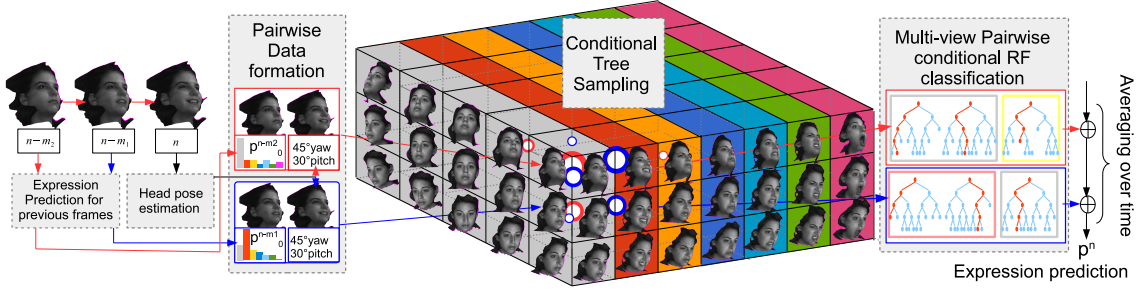


Figure 4.5: Flowchart of the MVPCRF method for FER. When evaluating a video frame indexed by  $n$ , pairs are created between this current frame and previous frames  $n-m_1, n-m_2, \dots$ . Randomized trees trained upon a pairwise dataset are then drawn conditionally to head pose estimation as well as expression probabilities for the previous frames. Finally, predictions outputted for each pair are averaged over time to give rise to an expression probability  $p^n$  for the current frame. This prediction is used as a tree sampling distribution for classifying the following frames. Best viewed in color.

In what follows, we refer to this model as the *multi-view PCRF* (MVPCRF) model. In this formulation, for computing the pairwise probability between frames  $n$  and  $m$ , we first estimate the head pose for frame  $n$ . Then, for each pose bin  $\Omega_i$  and expression label  $c'$ , we select a number of trees equal to  $\mathcal{N}^m(c', \Omega_i)$  (Equation (4.7)):

$$\mathcal{N}^m(l', \Omega_i) \approx T \mathcal{P}_{\Omega_i}(\omega(\mathcal{I}^n)) p_0^m(c') \quad (4.7)$$

Where  $p_0^m(c')$  is the probability of expression label  $c'$  for frame  $m$ . The number of trees allocated to classify each transition is thus:

$$T = \sum_{\Omega_i \in \Omega} \sum_{c' \in \mathcal{C}} \mathcal{N}^m(c', \Omega_i) \quad (4.8)$$

Note that the tree sampling distribution proposed in Equation (4.7) supposes that the head pose estimate do not vary that much between frames  $n-N$  and  $n$ . Should that be the case, MVPCRF can be trained from pairs of images from different viewpoints. It also assumes the independence of head pose and expression prior, which is not problematic for training on posed expression data. However, such assumption may not hold for

spontaneous datasets for which expressions as *surprise* or *fear* may involve specific head motion (e.g. recoil). In such case, prior conditionals may be estimated from the training corpus beforehand. Also, as stated in [22, 84] using conditional models usually involves one major pitfall, which lies in the reduction of the number of training examples used to train each separate classifier. This is barely a problem for the training of a PCRF model, as naturally many examples of each ongoing transition can be sampled from the datasets. Furthermore, for the MVRF and MVPCRF models we can generate a new database that contains a large number of training examples for each pose bin using the high-resolution 3D-models from the BU-4DFE database [103], as highlighted in Section 4.4.2.

#### 4.4.2 Multi-view dataset generation

Each texture frame of the BU-4DFE database is associated with a high-resolution 3D VRML model containing approximately 35000 vertices, that we use to train our MVPCRF classifier as well as to design a new dataset for multi-view video FER. Many approaches [87, 90] present results for static multi-view FER using the BU-3DFE database [104]. To do that, for each static image, the authors typically render 3D meshes from a viewpoint with fixed yaw and pitch rotation angles. However, for video FER, head pose does not necessarily remain constant throughout a video. Furthermore, from the perspective of a fully automatic multi-view FER system, we typically aim at covering a specific head pose range rather than a discrete, arbitrary set of viewpoints. Hence, we propose to generate rotated versions of the videos by assigning each sequence a yaw-pitch variation from the frontal video. More specifically, our goal is to cover the same “useful” range as in [87, 90] (i.e.  $\pm 45$  yaw,  $\pm 30$  pitch). We thus generate  $k = 5 \times 3$  bins  $\{\Omega_i = \Omega_{\gamma_i, \beta_i}\}_{i=1, \dots, k}$  with  $\{\gamma_i\} = \{0, \pm 17.5, \pm 35\}$  and  $\{\beta_i\} = \{0, \pm 25\}$  the mean rotation angles respectively in yaw and pitch. Each sequence  $s$  is thus associated with rotation angles:

$$\begin{cases} \gamma_i^s = \gamma_i + \gamma' \\ \beta_j^s = \beta_j + \beta' \end{cases} \quad (4.9)$$

Where  $\gamma'$  and  $\beta'$  are random variations uniformly drawn from the ranges  $[-\sigma_\gamma, \sigma_\gamma]$  and  $[-\sigma_\beta, \sigma_\beta]$ , respectively.  $\sigma_\gamma$  and  $\sigma_\beta$  respectively denote the expected yaw and pitch

width of the pose bins. In order to set those values, we measure the standard deviation of the head pose angles on the frontal view (3.6 and 5.9 in yaw and pitch respectively). We then set  $\sigma_\gamma = \sigma_\beta = 5$  to allow a small overlap, thus a smoother interpolation between adjacent pose bins. The data distribution among the generated pose bins can be seen in Figure 4.4.2. For each frame of each sequence  $s$ , we generate 15 frames by rotating the camera (position, direction and up vector). We also turn off the camera headlight and add an ambient light node to the VRML virtual environment.

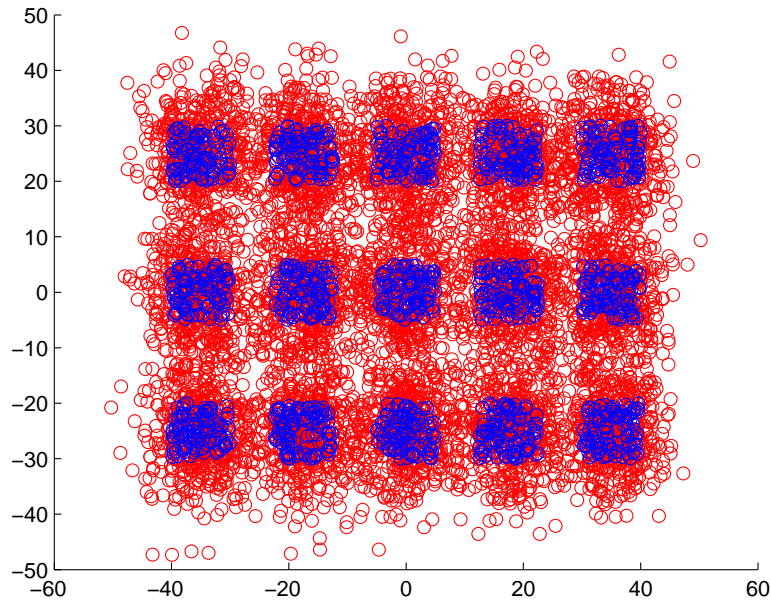


Figure 4.6: Data repartition across the 15 generated pose bins. Blue circles: angles associated with the sequences  $(\gamma_i^s, \beta_j^s)$ , red: individual frames

The next step is to align facial feature points on the rotated sequences. However, the standard pipeline of applying a frontal or full profile face detection before aligning the feature points from the output face rectangle is bound to fail when the yaw/pitch becomes important and only a few images can correctly be aligned. In order to circumvent those issues, we generate “boot” sequences using the first image of each video. Those sequences contain 20 frames and show a very progressive rotation of the first frame starting from a frontal view and ending on the expected viewpoint. We apply the OpenCV Viola-Jones face detector [92] on the first frame of the boot sequence (frontal view). Then we align

facial feature points with the SDM tracker [98] on the retrieved face rectangle. Feature points are then tracked throughout the boot sequence. Once the *boot* is completed, feature points are tracked on all the frames of the rotated expression videos (Figure 4.4.2). Finally, we crop the facial images to a constant size based on the feature point location and generate a total of 906030 images.



Figure 4.7: Boot process for multi-view data generation with aligned feature points

Lastly, we construct our multi-view training set by manually selecting the neutral and apical frames using the same subsets as in the frontal case. Also, in order to filter out the incorrectly aligned frames, we automatically discard the frames for which more than 5 feature points do not lie on the facial mesh. Our final training set thus consists of 122623 face images. Note however that we did not apply any manual check to remove the misaligned frames, or the ones for which the 3D models contain some distortions. The image generation process took about 5 days to complete on an *I7-4770* CPU on a `Matlab` environment. For each of the retrieved frames, we use the `posit` algorithm [23] to estimate head pose from the feature points. Such setting allows to use the same head pose estimation for training and testing, as compared to, e.g. constructing the pose sampling distribution from the ground truth generated positions. Then, we compute the pose sampling probability distribution for each pose bin  $\mathcal{P}_{\Omega_i}(\omega(\mathcal{I}^n))$  by applying a Gaussian smoothing on the training data repartition in the yaw/pitch space (Figure 4.4.2). Thanks to the booting procedure discussed above, the number of training samples between the different pose bins is roughly equivalent. However, this might not be the case for other datasets, where constructing a sampling probability from the data offers the advantage to implicitly downweight the sampling of pose-specific trees relatively to the amount of training data.

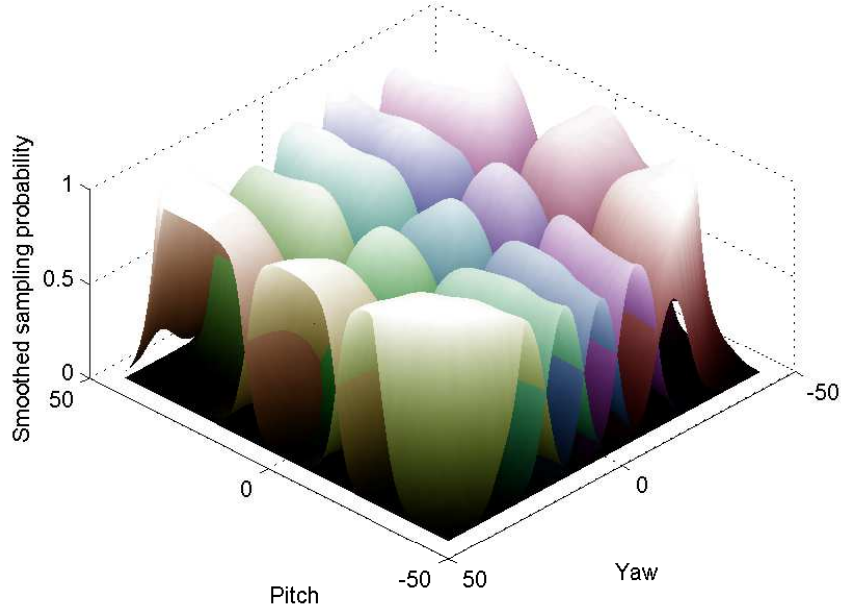


Figure 4.8: Pose sampling probability distributions  $\mathcal{P}_{\Omega_i}(\omega^n)$  constructed by smoothing the data repartition for each pose bin

## 4.5 Experiments

In this section, we report accuracies obtained on two different FER scenarios. In Section 4.5.2.1 we report comparisons between different classification models on two well-known frontal FER databases, CK+ (see Section 2.3.3) and BU-4DFE (Section 2.3.4) databases. Furthermore, in order to evaluate the capabilities of the learned models to generalize on spontaneous FER scenarios, in Section 4.5.2.2 we report classification results for cross-database evaluation on two spontaneous databases, namely the FG-NET FEED (Section 2.3.5) and BP4D (Section 2.3.6) databases. We highlight that our conditional formulation of dynamic integration substantially increases the recognition accuracy on such difficult tasks. Furthermore, in Section 4.5.3 we also evaluate our approach on multi-view video FER scenarios.

### 4.5.1 Evaluation framework

7-class RF (static) and PCRF (full and conditional) models are trained on the CK+ and BU-4DFE datasets using the set of hyperparameters described in Table 4.1. Note however that extensive testing showed that the values of these hyperparameters had a very subtle influence on the performances. This is due to the complexity of the RF framework, in which individually weak trees (e.g. that are grown by only examining a few features per node) are generally less correlated, still outputting decent predictions when combined altogether (as also stated in Section 3.3.4.2.2). Also, for a fair comparison between static and pairwise models, we use the same total number of feature evaluations for generating the split nodes.

Table 4.1: Hyperparameter settings

Hyperparameters	value(RF)	value(PCRF)
Nb. of $\phi^{(1)}$ features	40	20
Nb. of $\phi^{(2)}$ features	40	20
Nb. of $\phi^{(3)}$ features	160	80
Nb. of $\phi^{(4)}$ features	-	20
Nb. of $\phi^{(5)}$ features	-	20
Nb. of $\phi^{(6)}$ features	-	80
Data ratio per tree	2/3	2/3
Nb. of thresholds	25	25
Total nb. of features	6000	6000
Nb. of trees	500	500

During the evaluation, the prediction is initialized in a fully automatic way from the first frame using the static classifier. Then, for the full and conditional models, probabilities are estimated for each frame using transitions from previous frames only, bringing us closer to a real-time scenario. However, although it uses transitional features, our system is essentially a frame-based classifier that outputs an expression probability for each separate video frame. This is different from, for example, a HMM, that aims at predicting a probability related to all the video frames. Thus, in order to evaluate our classifier on



video FER tasks, we acknowledge correct classification if the maximum probability outputted for all frames corresponds to the ground truth label. This evaluates the capability of our system to retrieve the most important expression mode in a video, as well as the match between the retrieved mode and the ground truth label. Finally, both static and transition classifiers are evaluated using the Out-Of-Bag (OOB) error estimate [11].

## 4.5.2 Experiments on frontal data

### 4.5.2.1 Experiments on prototypical data

In order to validate our approach on frontal view videos, we compared our conditional model to a purely static model and a full model, for a variety of dynamic integration parameters (the number of frames in temporal window  $N$  and the step between those frames  $Step$ ) on the BU-4DFE database. We also evaluated the interest of using a *dynamic* probability prediction for previous frames (*i.e.* the output of the pairwise classifier for those frames) versus a *static* one. Average results are provided in Figure 4.9. For CK+ database, sequences are generally too short to show significant differences when varying the temporal window size or the step size. Thus we only report accuracy for full and conditional models with a window size of 30 and a step of 1. Per-expression accuracies and F1-scores for both Cohn-Kanade and BU-4DFE databases are shown in Figure 4.10.

Figure 4.10 reveals that facial expressions involving large deformations (e.g. *surprise* and *happiness*) are recognized with very high accuracies. *Disgust* is also recognized quite well for both databases and for all the models. However, more subtle expressions such as *anger* and *sadness* rank among the lowest. For those expressions, the addition of spatio-temporal information allows to increase the recognition accuracy as compared to a static RF model. As in many other works on facial expressions, accuracies for *fear* are lower than for the other expressions, as it can be quite subtle in some cases where the eyes are open a little bit wider. Moreover, this expression also displays a larger variability than the others on these databases. Overall, modelling transition patterns through PCRF allows to significantly increase the recognition accuracy as well as the balanced  $F1$ -score, for all expressions on both CK+ and BU-4DFE databases. We believe that this is due to the extra



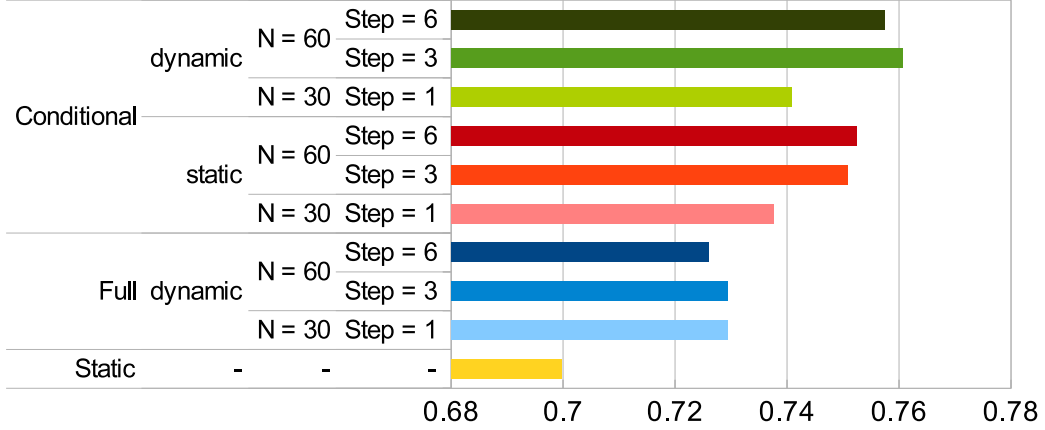


Figure 4.9: Average accuracy rates obtained for various temporal integration parameters on the BU-4DFE database

dynamic features that provide both robustness and decorrelation of the individual decision trees.

Figure 4.10 also shows that the conditional model outperforms the full model on both databases, which is probably due to the fact that using only a restricted set of ongoing expression transitions for training allows to better capture the variability of the spatio-temporal features for the dedicated pairwise forests. This is particularly true on the CK+ database, where the number of pairwise data points is not enough for the full model to capture the variability of all possible ongoing transitions, hence justifying the lower accuracy. Table 4.9 also shows that it is better to look backward for more frames in the sequence ( $N = 60$ ) with less correlation between the frames ( $Step = 3$  or  $6$ ). Again, such setting allows to take more decorrelated paths in the individual trees, giving a better recombination after averaging over time.

A compilation of comparisons to other state-of-the-art approaches for FER can be found in Tables 4.2 and 4.3. On the CK+ dataset, we compare our algorithms with recent works reporting results on the same subset of sequences (*i.e.* not including *contempt*). Such comparisons are to be put into perspective as the evaluation protocols differ between the methods. Nevertheless, PCRF provides slightly better results than those reported in [63] (+3.2%) as well as in [79] (+1.9%) and [36] (+2.3%). Furthermore,

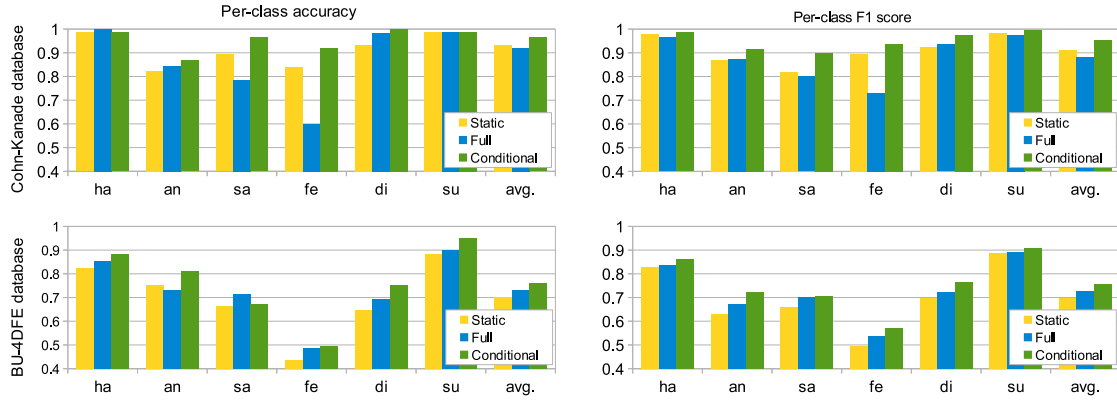


Figure 4.10: Per-class recognition accuracy rates and F1-scores on CK+ and BU-4DFE databases

those approaches explicitly perform normalization w.r.t. a neutral face and consider the last (apex) frame whereas our approach automatically retrieves the apex as the maximum probability throughout a sequence.

Moreover, to the best of our knowledge, our approach gives the best results on the BU-4DFE database for automatic FER from videos using 2D information only. It provides better results than the dynamic 2D approach [85] (+9.1%), as well as the LBP-TOP approach presented in [37] (+4.5%). Recently, Meguid *et al.* [1] obtained satisfying results using an original hybrid RF/SVM system. They trained on the static BU-3DFE database [104] and employed a post-classification temporal integration scheme. However our PCRF method achieved a significantly higher accuracy (+3%) which shows the benefits of using dynamic information at the feature level.

Table 4.2: CK+ database

method	Accuracy
Mohammadi <i>et al.</i> [63]	93.2
Happy <i>et al.</i> [36]	94.1
Shojaeilangari <i>et al.</i> [79]	94.5
This work, RF	93.2
This work, PCRF	<b>96.4</b>

Table 4.3: BU-4DFE database

Method	Accuracy
Sun <i>et al.</i> [85]	67.0
Hayat <i>et al.</i> [37]	71.6
Meguid <i>et al.</i> [1]	73.1
This work, RF	70.0
This work, PCRF	<b>76.1</b>

#### 4.5.2.2 Generalization on spontaneous data

Tables 4.4 and 4.5 respectively report results for cross-database evaluation (with training on the BU-4DFE database) on the FEED and BP4D databases. In order to provide a fair comparison between our approach and the one presented in [1], we used the same labelling protocol. One can see that the performances of their system are better than those of our static RF model, which can be attributed to the fact that they use a more complex classification and posterior temporal integration flowchart. Nevertheless, our PCRf model provides a substantially higher accuracy (+3.4%), which, again, is likely to be due to the use of spatio-temporal features as well as to an efficient conditional integration scheme. Furthermore, modelling spatio-temporal patterns for *every* possible transition (*i.e.* across the videos) allows to gather more training data than using spatio-temporal descriptors [109, 47] learnt on separate videos.

Regarding the experiments on the BP4D database, for the sake of a fair comparison, we used the same protocol as in [108], with training on the BU-4DFE database and using only a subset of the tasks (*i.e.* tasks 1 and 8 corresponding to expression labels *happy* and *disgust* respectively). However, we do not retrain a classifier with a subset of 3 expressions as it is done in [108], but instead use our 7-class static and PCRf models with a forced choice between *happiness* (probability of class *happiness*) and *disgust* (probability sum of classes *anger* and *disgust*). Such setting could theoretically increase the confusion in our conditional model, resulting in a lower accuracy. However, as can be seen in Table 4.5, using dynamic information within the PCRf framework allows to substantially increase the recognition rate as compared to a static RF framework (+8.2%). We also overcome the results reported in [108] by a significant margin (+5.8%), further showing the capability of our approach to deal with complex spontaneous FER tasks. Also note that in [108], the authors used the so-called *Nebulae 3D* polynomial volume features which are by far more computationally expensive than our geometric and integral HOG 2D features. All in all, we believe our results show that the PCRf approach provides significant improvements over a traditional static classification pipeline that translates very well to more complicated spontaneous FER scenarios, where a single video may contain samples of several expressions.

Table 4.4: FEED database	
method	Accuracy
Meguid <i>et al.</i> [1]	53.7
This work, RF	51.9
This work, PCRF	<b>57.1</b>

Table 4.5: BP4D database	
Method	Accuracy
Zhang <i>et al.</i> [108]	71.0
This work, RF	68.6
This work, PCRF	<b>76.8</b>

### 4.5.3 Experiments on non-frontal data

We also evaluate our approach on multi-view dynamic FER scenario on the database generated in Section 4.4.2. During evaluation, for each frame  $n$  of a sequence, head pose  $\omega(\mathcal{I}^n)$  is thus estimated from the set of aligned feature points and trees from the MVPCRF collections are sampled according to the values  $\mathcal{P}_{\Omega_i}(\omega(\mathcal{I}^n))$  for each pose bin  $\Omega_i$ . We compare the average accuracies outputted by RF, PCRF, MVRF and MVPCRF. RF and PCRF were trained on the central (frontal view) bin only. For PCRF and MVPCRF, we set the temporal integration parameters  $N = 60$  and  $Step = 6$  as it provided satisfying results in the frontal case (Figure 4.9). As in Section 4.5.2, a video is considered as correctly classified if the dominant expression mode (*i.e.* the maximum probability expression throughout the sequence) corresponds to the ground truth label for that video.

Table 4.6 displays per-expression accuracies averaged over the 15 pose bins for the three models. For all expressions, MVPCRF outperforms RF and PCRF by a significant margin. MVPCRF also outperforms the static multi-view MVRF on all expressions but *sadness* and *fear*. However, Table 4.7 reveals that the F1-score is a little higher for MVPCRF on those expressions, indicating that the static MVRF is more biased toward those expression classes. This seems particularly relevant in the positive pitch case, where using spatio-temporal information helps to disambiguate *anger* from *sadness*, which in some case differ only by a very subtle eyebrow frown or lip raiser. Also, *fear* appears as the most subtle expression as already reported in other works [1]. This is due to the fact that subjects often smile during the sequence, thus the videos may be misclassified as *happiness*. For this reason, many other approaches such as the one in [6] use a restricted number of subjects. However, we use the 101 subjects to ensure reproducibility of the results.

The overall classification accuracy is 72.2% against 76.1% for the benchmarks of Section 4.5.2.1 on frontal view video. This performance drop comes from a greater variability in face appearance as well as the feature point misalignment for non-frontal poses, as discussed in [41]. Classification rates are also a little lower than the static FER baseline ones [87, 90] on the BU-3DFE database. However, fully automatic FER from video is a much more difficult setup, as it involves the retrieval of the apex frames and expression classification on those frames. Furthermore, many approaches operate on high-resolution 3D data and require expensive projections on a frontal view, thus can not be applied easily to real-time FER from consumer camera.

Figure 4.11 shows the per-pose bin accuracy rates averaged over the six expressions. On the one’s hand, RF performances seems to drop dramatically when we move away from the central bin (from 70.4% to 44.7%). Interestingly, PCRf performs significantly better than RF on every pose bin, which proves that the captured dynamics generalize well on unseen data, as already shown on the cross-database settings. PCRf performance also drops significantly on off-center pose bins. On the other hand, MVPCRf performs significantly better on those bins: accuracy is nearly symmetrical for negative and positive yaws, as already reported by [87] for static multi-view FER. Furthermore, as stated in [87, 90] we observe lower classification rates on negative pitches (68.3% as compared to 74.1% on average for positive pitch). Our take is that the mouth area may be the most informative one for FER tasks: as such, the classifiers can struggle to disambiguate certain expressions (e.g. *anger* from *sadness*) when the mouth features become more subtle and difficult to capture.

Figure 4.12 shows per-expression, per-pose bin accuracies obtained for MVPCRf. Indeed, expressions such as *sadness* and *fear* are better recognized for positive pitches, as they specifically involve subtle mouth movements as well as eyebrow raising. Conversely, *anger* and *disgust* are characterized by eyebrow frowning that is better recognized on negative pitch views. Finally, *happiness* and *surprise* are expressions with the highest overall classification rates. They are typically better recognized on frontal views or for negative pitches, where the corresponding mouth motions are less frequently misclassified as *fear*.

Table 4.6: Per-expression accuracy rates averaged over all pose bins

Expression	RF (%)	PCRF (%)	MVRF (%)	MVPCRF (%)
Happy	57.8	73.4	83.3	<b>87.8</b>
Angry	59.2	73.3	71.9	<b>80.4</b>
Sad	56.0	52.2	<b>70.8</b>	64.4
Fear	29.6	25.7	<b>34.8</b>	33.0
Disgust	48.4	63.9	63.5	<b>74.3</b>
Surprise	81.6	88.3	85.3	<b>92.4</b>
Average	55.4	62.8	68.3	<b>72.1</b>

Table 4.7: Per-expression F1-scores averaged over all pose bins

Expression	RF (%)	PCRF (%)	MVRF (%)	MVPCRF (%)
Happy	62.6	74.4	80.7	<b>84.2</b>
Angry	48.6	61.5	62.9	<b>68.0</b>
Sad	46.2	48.8	65.4	<b>66.1</b>
Fear	34.7	34.7	43.8	<b>44.8</b>
Disgust	56.3	66.2	67.9	<b>73.1</b>
Surprise	71.4	77.6	83.6	<b>87.3</b>
Average	53.3	60.6	67.4	<b>70.6</b>

#### 4.5.4 Complexity analysis

An advantage of using conditional models is that with an equivalent parallelization they are faster to train than a full model learnt on the whole dataset. According to [57] the average complexity of training a RF classifier with  $M$  trees is  $\mathcal{O}(MKN \log_2 N)$ , with  $K$  being the number of features to examine for each node and  $N$  the size of (2/3 of) the dataset. Thus if the dataset is equally divided into  $P$  bins of size  $\tilde{N}$  upon which conditional forests are trained (and such that  $N = P\tilde{N}$ ), the average complexity of learning a conditional model now becomes  $\mathcal{O}(MKN \log_2 \tilde{N})$ .

Same considerations can be made concerning the evaluation, as trees from the full model are bound to be deeper than those from the conditional models. Table 4.8 shows

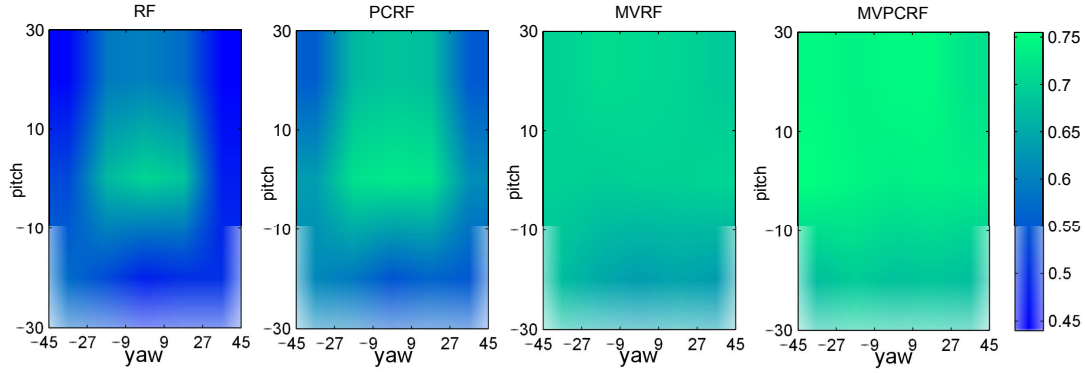


Figure 4.11: Per-pose bin accuracy rates averaged over all expressions

an example of profiling a MVPCRf on one video frame with an averaging over 60 frames and a step of 6 frames. We experiment with various total numbers of trees  $M$  to show that the proposed framework can perform real-time FER.

Table 4.8: Profiling of total processing time for one frame (in ms)

Step	Time (ms)
Facial alignment	10.0
Integral HOG channels computation	2.0
MVPCRf evaluation ( $M = 500$ )	2.6
MVPCRf evaluation ( $M = 1000$ )	4.8
MVPCRf evaluation ( $M = 2000$ )	7.8
MVPCRf evaluation ( $M = 6000$ )	19.0

This benchmark was conducted on a *I7-4770* CPU within a C++/OpenCV environment, without any code parallelization. As such, the algorithm already runs in real-time. Furthermore, evaluations of pairwise classification or tree subsets can be parallelized to fit real-time processing requirements on low-power engines such as mobile phones. In addition, the facial alignment step can be performed at more than 300 fps on a smartphone with similar performances using the algorithms from [69].

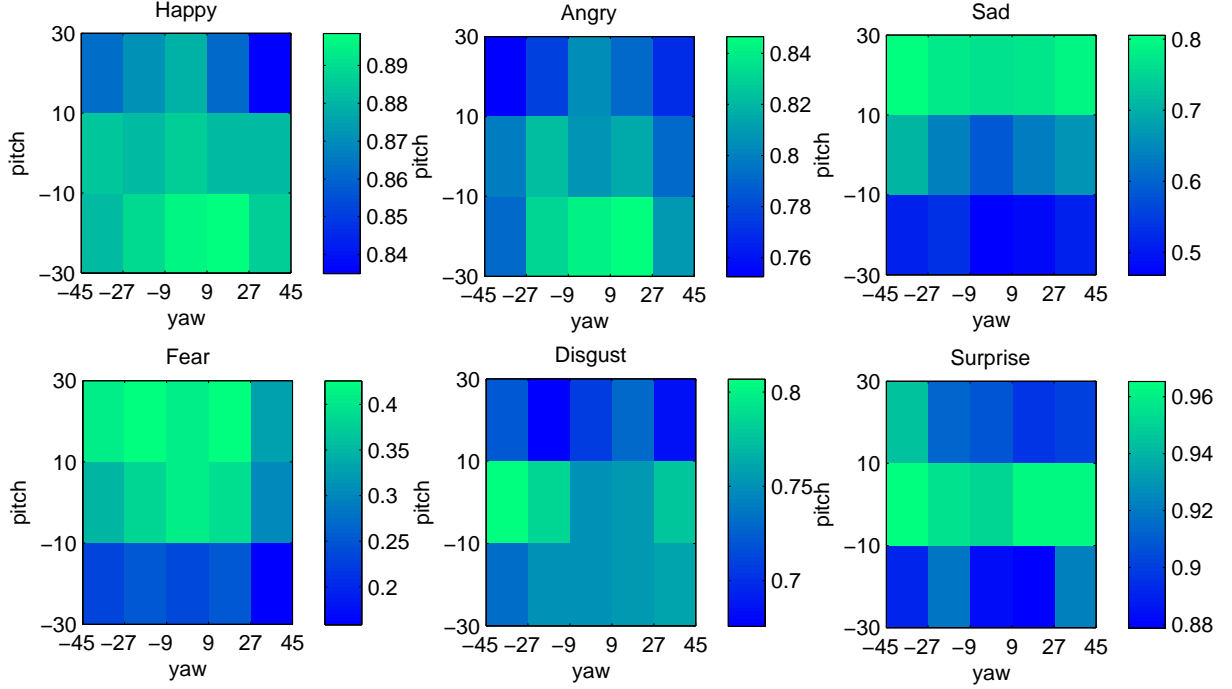


Figure 4.12: Per-expression, per-pose bin classification accuracy rates

#### 4.5.5 Discussion

In this chapter, we presented an adaptation of the RF framework for automatic dynamic pose-robust FER from videos. We also introduced a novel way of integrating the temporal information of expressions by considering pairwise RF classifiers. This formulation appears as a somewhat natural way to extend RFs for dealing with video sequences, and allows the efficient integration of high-dimensional, low-level spatio-temporal information through averaging over time pairwise trees. These trees are conditioned on predictions outputted for the previous frames to help reducing the variability of the ongoing transition patterns. In addition, we proposed an extension of the PCRF framework to efficiently handle head pose variation in an FER system. We showed that our models can be trained and evaluated efficiently given appropriate data, and lead to a significant increase of performances compared to a static RF. We also introduced a new multi-view video corpus generated using the BU-4DFE database to assess the pose-robustness of the proposed system. Finally, we showed that our method works on real-time without specific optimization schemes, and could be run on low-power architectures such as mobile phones by using an



appropriate parallelization scheme.

Nevertheless, the proposed algorithms are still not perfect and suffer from a number of limitations. First, in order to train a PCRF, one needs to explicitly consider frame-level annotations. In our work, we manually highlighted a set of peak frames for the database, which were used to train the classifiers. This is a recurrent drawback of frame-based classifiers as compared to sequence-level ones (e.g. HMMs, CRFs) and thus could not be solved easily. However, using only a subset of peak frames from the videos for training also allows to limit the memory usage, which is particularly relevant in our case, as we need to store the feature maps for each image. Moreover, we demonstrated that, during evaluation, our algorithm was successful at retrieving the correct expression modes and thus could be applied in a fully automatic fashion. Furthermore, an advantage of integrating the temporal information under the form of transition modelling is that it does not require continuity of the sequence. Hence, it has no problem handling failure from the detection or feature point alignment pipelines, as opposed to other spatio-temporal descriptors [109, 47]. Secondly, in order to build transition classifiers we need examples for each possible transition. This can be a hindrance when training on highly unbalanced datasets (as in CK+ with *contempt* expression class). Thirdly, multi-view classification requires loads of training data from multiple head poses. This, however, can be alleviated by the use of high-dimensional 3D models to generate training examples. Finally, we felt that, particularly for the experiments on multi-view data, we were at times limited by the robustness of the feature point alignment for non-frontal head pose, as well as from the quality of the pose estimation from the set of feature points. Such problem could in theory be alleviated by the use of recent robust algorithms such as the one proposed in [99]. The proposed system is also not robust to partial occlusions of the face, which are likely to happen in real-case scenarios, and will be discussed in the following chapter.

# Chapter 5

## Local Subspace Random Forests

### 5.1 Overview

In the traditional RF framework (Section 3.3.4.2), each tree in the forest is grown using a subset of training examples (bagging) and a subspace of the input dimension (random subspace). As stated in [11], the rationale behind training each tree on a random subspace of the input dimension is that the prediction accuracy of the whole forest depends on both the strength of individual trees and on the independence of the predictions. Thus, by growing individually weaker (e.g. as compared to C4.5) but more decorrelated trees, we can combine these into a more accurate tree collection. Following this idea, we propose an adaptation of the RF framework that uses spatially-defined Local Subspaces (LS) instead of the traditional Random Subspaces (RS), as described in Figure 5.1. Each tree is trained using a restricted subspace corresponding to a specific part of the face, that is generated under the form of a random facial mask (Figure 5.1-a) for each tree on a possibly refined facial mesh. Binary candidate features can be selected from those local subspaces (b). The aggregation of local models gives rise to representations that we call Local Expression Predictions (LEPs (c)).

When applied on a potentially occluded face image (e), the reconstruction error outputted by an autoencoder network provides a confidence measurement of how close a face region lies from the training data manifold, with high and low confidences depicted in green and red respectively. This local confidence measurement can be used to weight

LEPs (g) in order to provide an occlusion-robust expression prediction (WLS-RF). Finally, LEPs can be used to predict AU occurrence (h). Once again, the autoencoder network can be used to provide AU-specific confidence measurements (i).

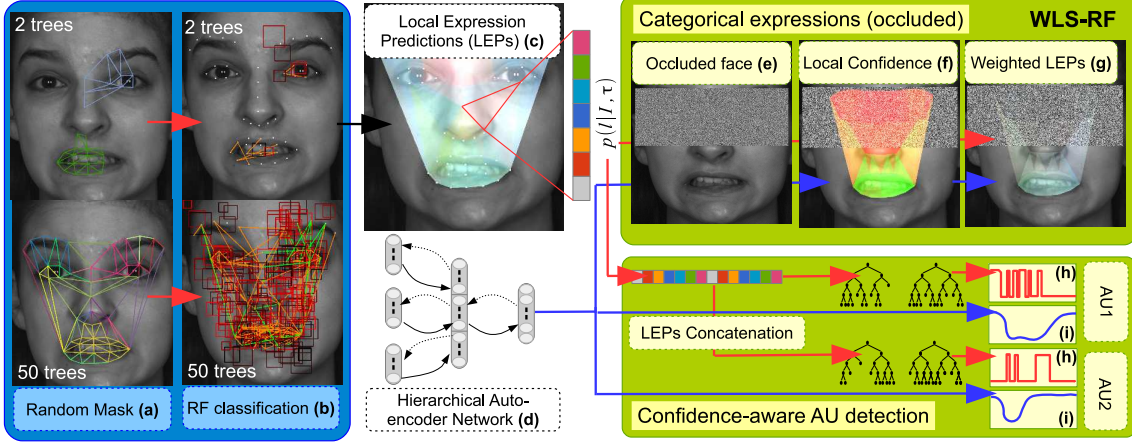


Figure 5.1: LEPs and applications to categorical expression recognition, occlusion handling in FER and AU detection. Randomized trees are trained upon local subspaces generated under the form of random facial masks (a), on which binary feature candidates are generated and selected (b). The local predictions outputted by the trees can be aggregated into categorical expression-driven high-level LEP representations (c). Given an occluded face image (e), an occlusion-robust categorical expression prediction can be outputted by weighting LEPs with confidence scores (f) given by a hierarchical autoencoder network (d). Furthermore, LEP features can be used to predict AU occurrence (h). For instance, AU 12 (lip corner raiser) can be described by a high value of LEPs associated to triangles around the lips with categorical expression *happiness*. Finally, for each AU, a confidence measurement can also be provided (i). Best viewed in color.

This chapter is organized as follows: Section 5.2 describes how we train Randomized trees upon spatially-defined local subspaces of the face, that are generated under the form of random facial masks (5.2.1). Then, Section 5.2.2 shows how we combine the output prediction of those trees into LEP features. Finally, Section 5.2.3 highlights how we can tessellate the facial mesh to arbitrary set the spatial resolution of LEP features.

Section 5.3 explains how we can weight the local predictions with a confidence measurement defined on triangles of the face mesh to produce occlusion-robust predictions. In

Section 5.3.2 we discuss the proposed autoencoder network architecture (Section 5.3.2.1) and how it is trained to capture the local manifold around facial feature points (Section 5.3.2.2). Eventually, in Section 5.3.2.3 we explain how we can use the autoencoder network to provide a confidence measurement to weight LEPs for occlusion-robust FER. In Section 5.4 we describe how LEPs can be used for AU detection, with a focus on how we can merge multiple expression datasets (Section 5.4.1), with AU-specific confidence assessment (Section 5.4.2). We also present a number of approaches for multi-output RFs classification upon LEP features (Section 5.4.3).

Finally, in Section 5.5 we show that our approach significantly improves the state-of-the-art for categorical FER on multiple datasets, both on the non-occluded (Section 5.5.2) and occluded cases (Section 5.5.3). We also demonstrate in Section 5.5.4 the interest of our LEP representation for AU activation prediction and the relevance of the AU-specific confidence measurement. Finally, Section 5.5.6 discusses a few perspectives raised by the proposed work.

## 5.2 Training randomized decision trees on spatially defined local subspaces

### 5.2.1 Random mask generation

Local trees are trained using Algorithm 3. For each tree  $t$  in the forest, we generate a face mask  $M_t$  defined over triangles  $\tau$  on facial feature points of a precomputed mean shape  $\bar{f}$ . The mask is initialized with a single triangle randomly selected from the mesh. Then, neighbouring triangles are added until the total surface covered by the selected triangles w.r.t.  $\bar{f}$  becomes superior to hyperparameter  $R$ , that represents the (approximate) surface that should be covered by the mask. Finally, tree  $t$  is grown on the subspace that corresponds to the facial mask  $M_t$ .

---

**Algorithm 3** Training Local Subspace Random Forest

---

**input:** images  $\mathcal{I}$  with labels  $l$  and feature points  $f(\mathcal{I})$

compute  $\bar{f}$ , the mean shape

pre-compute  $s(\tau(\bar{f}))$ , surfaces of triangles  $\tau$  on mean shape

**for**  $t = 1$  to  $T$  **do**

    randomly select a triangle  $\tau_i$

$r \leftarrow s(\tau_i)$

    initialize mask  $M_t \leftarrow \{\tau_i\}$

**while**  $r < R$  **do**

        draw a list of candidate neighbouring triangles

        randomly select a triangle  $\tau_j$  from that list

$r \leftarrow r + s(\tau_j)$

$M_t \leftarrow M_t \cup \{\tau_j\}$

**end while**

    randomly select a fraction  $\tilde{\mathcal{S}}_t \subset \mathcal{S}$  of subjects

    balance bootstrap  $\tilde{\mathcal{S}}_t$  with downsampling

    grow tree  $t$  on bootstrap  $\tilde{\mathcal{S}}_t$  and input subspace  $M_t$

**end for**

**output:** tree predictors  $p_t(c|\mathcal{I})$  with associated masks  $M_t$

---

### 5.2.2 Local expression prediction features

The output expression prediction of spatially-constrained trees can be used as features for face analysis. Note that this is not the first time that RF predictions are used as features for a subsequent task. For instance, Ren *et al.* [69] used local binary features to construct a cascaded feature point alignment method. However, contrary to [69], we construct our LEP representation by locally averaging predictions and not by directly using the output prediction of the trees. Furthermore, LEPs offer several advantages over using a set of trees defined on the whole face:

- LEPs can be aggregated to provide categorical FER. Those local models (LS-RF) can theoretically capture more diverse information compared to a global one by “forcing” the trees to use less informative features, that can still hold some predictive power.
- We can use the confidence outputted by the autoencoder network in Section 5.3.2.3 to weight the LEPs for which the pattern lies further from the training data manifold (WLS-RF). For example, in case of occlusion or drastic illumination changes, we can still use the information from the other face subparts to predict the expression.
- LEPs can be used as an intermediate representation for the task of describing Action Units (AUs). Noteworthy, AU classification could benefit from LEPs trained on larger corpus labelled with categorical expressions, as annotation is less time-consuming than FACS coding.

More specifically, when testing, a face image  $\mathcal{I}$  is successively rooted left or right for each tree  $t$  depending of the outputs of the binary tests stored in the tree nodes, until it reaches a leaf. The tree  $t$  thus outputs a probability vector  $p_t(c|\mathcal{I})$  for class  $c \in \mathcal{C}$ , whose components are either 1 for the represented class, or 0 otherwise. Prediction probabilities are then averaged among the  $T$  trees of the forest (Equation (5.1)).

$$p(c|\mathcal{I}) = \frac{1}{T} \sum_{t=1}^T p_t(c|\mathcal{I}) \quad (5.1)$$

Those prediction probabilities are computed similarly for the global RF and the LS-RF. However, for LS-RF the output probabilities of the trees have some degrees of locality and we can write the above formula as a sum over local probabilities defined for each triangle (Equation (5.2)).

$$p(c|\mathcal{I}) = \frac{1}{T} \sum_{\tau} Z_{\tau} p(c|\mathcal{I}, \tau) \quad (5.2)$$

Where  $p(c|\mathcal{I}, \tau)$  is the Local Expression Prediction (LEP) probability vector associated with triangle  $\tau$  on the facial mesh:

$$p(c|\mathcal{I}, \tau) = \frac{1}{Z_{\tau}} \sum_{t=1}^T \frac{\delta(\tau \in M_t) p_t(c|\mathcal{I})}{|M_t|} \quad (5.3)$$

With  $\delta(\tau \in M_t)$  being a function that returns 1 if triangle  $\tau$  belongs to mask  $M_t$ , and 0 otherwise.  $|M_t|$  is the number of times tree  $t$  is used in Equation (5.2), and  $Z_{\tau}$  is the sum of prediction values for all expression classes  $l$  and triangle  $\tau$ . Thus, a global expression probability is defined by a (normalized) sum of LEPs. Note that those LEP vectors  $p(c|\mathcal{I}, \tau)$  are not strictly limited to triangle  $\tau$  but defined within its neighbourhood, with a radius that depends on hyperparameter  $R$ . The setting of  $R$  thus controls the locality of the trees, as it will be discussed below.

### 5.2.3 Facial mesh refinement

A downside of the proposed local expression-driven features are constrained by the definition of a facial mesh, that we obtain by aligning a number of feature points using Intraface [98]. However, the coarseness of this facial mesh may be a hindrance to perform AU detection from said features, as the locality of the models is crucial to avoid using unrelated local information (e.g. a wide open mouth describing AU2 (Outer Brow Raiser)). In order to circumvent this issue, it is possible to arbitrary refine the facial mesh using an adaptive refinement strategy, as illustrated on Figure 5.2.

In order to do that, we first expand the mesh slightly above the eyebrows and below the mouth so that the appearance features can capture the forehead and jaw areas. Furthermore, we apply mesh tessellation on the resulting facial mesh. Note that, as shown

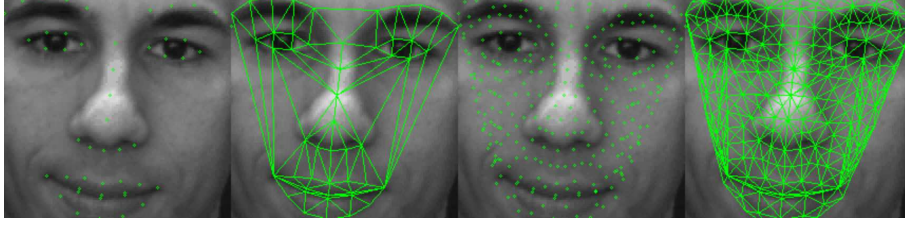


Figure 5.2: Facial mesh refinement. Left: original point-wise and triangle-wise mesh (49 points, 79 triangles). Right: refined mesh (500 points, 545 triangles)

on Figure 5.2, some triangles may have a long side without necessarily covering a wide portion of the face. This is something we want to avoid given our appearance feature extraction framework, which will be discussed later. Hence, contrary to [42], we do not use the  $\sqrt{3}$ -subdivision based on triangle surfaces for tessellating the mesh. Instead, we select the triangle with the longest side, which we split in two triangles on that side. We then iteratively apply this adaptive subdivision until a number of points  $N_p$  is reached (e.g.  $N_p = 500$  points,  $N_\tau = 545$  triangles on Figure 5.2). The result is that triangles from the mesh are more or less homogeneous in terms of its maximum side length, but not necessarily in terms of surface.

## 5.3 Confidence weighting of local expression predictions for occlusion-robust expression prediction

### 5.3.1 Weighted Local Subspace Random Forests

LEPs can also be weighted by local confidence measurements to give rise to the Weighted Local Subspace Random Forest model (WLS-RF):

$$p(c|\mathcal{I}) = \frac{\sum_{\tau} \alpha^{(\tau)} Z_{\tau} p(c|\mathcal{I}, \tau)}{\sum_{\tau} \alpha^{(\tau)} Z_{\tau}} \quad (5.4)$$

Where  $\alpha^{(\tau)}$  denotes a triangle-wise confidence measurement that is outputted by the autoencoder network described in Section 5.3.2.3, for triangle  $\tau$ . This weighting scheme allows to better handle partial occlusions, by downweighting the local RFs associated



with the most unreliable appearance patterns. In the following subsections, we highlight how we can obtain such a confidence measurement by modeling the local manifold of the texture around the feature points.

### 5.3.2 Local manifold learning for confidence measurement

We design a hierarchical autoencoder network for the purpose of modelling the local texture manifolds. In particular, Section 5.3.2.1 provides more details about the network architecture. Section 5.3.2.2 provides information on how we train the network by stage-wise reconstruction optimization and Section 5.3.2.3 explains how we can use the trained network to provide a confidence measurement that can be used to weight the LEP features (See Equation 5.4).

#### 5.3.2.1 Hierarchical autoencoder network architecture

As shown in Figure 5.3, we use a 2-layer architecture. First, we extract HOG descriptors within the neighbourhood of each feature point. The choice of learning a manifold of HOG patterns rather than gray levels comes from the fact that HOG are used for both facial alignment and the LEP generation pipeline (see Sections 3.2.4.1 and 4.2.2). Thus, the reconstruction error of these patterns provides a confidence measurement that is relevant for both tasks. The local descriptor  $\Psi^{(k)}$  for a specific feature point  $k$  consists in the concatenation of gradient magnitudes and quantized orientation values in  $5 \times 5$  cells around this feature point, with a total window size equal to a third of the inter-ocular distance. This descriptor of dimension 225 then feeds the  $N_p$  autoencoders (one per feature point) of the first layer ( $L_1$ ) which are trained to reconstruct non-occluded patterns. Because occlusion of local patterns extracted at the feature point level are not independent (*i.e.* a feature point close to an occluded area is more likely to be occluded itself), we employ a second layer ( $L_2$ ) of autoencoders, that are trained to reconstruct non-occluded patterns of groups of encoded feature point descriptors. Those groups represent five face subparts (left and right eyes, nose, left and right parts of the mouth) inside of which the local patterns are closely related. Specifically,  $L_1$  is composed of 125 units for each landmark.

$L_2$  layer for a feature point group contains  $65 \times N$  units ( $\frac{1}{2}$  compression), where  $N$  is 12,12,8,11 and 11 respectively for left/right eye, nose and left/right mouth areas.

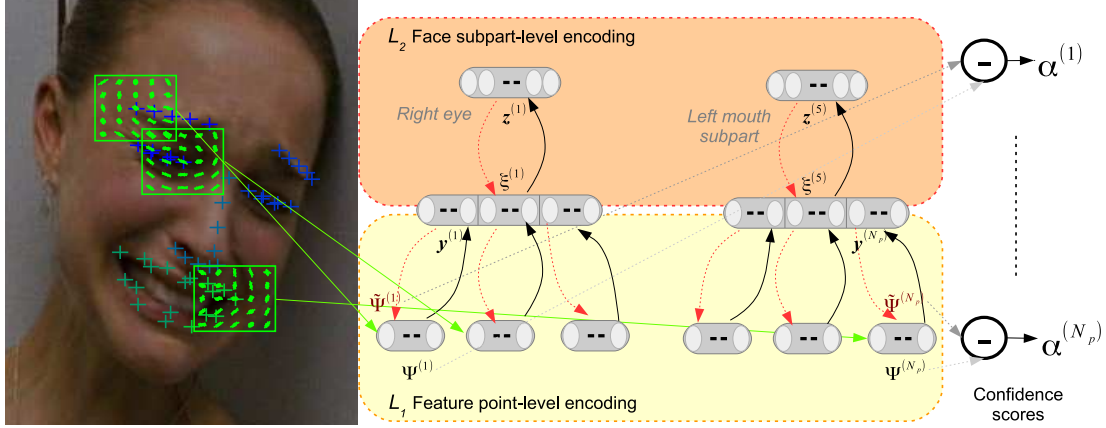


Figure 5.3: Architecture of our hierarchical autoencoder network. The network is composed of 2 layers: the first one ( $L_1$ ) captures the texture variations (HOG descriptors) around the separate aligned feature points. The second one ( $L_2$ ) is defined over 5 face subparts, each of which embraces multiple points whose appearance variations are closely related. The network outputs a confidence score  $\alpha^{(k)}$  for each of the  $N_p$  feature points.

### 5.3.2.2 Training the network

As highlighted in Section 3.3.3.2.1, the hierarchical network is trained in an unsupervised way, one layer at a time, by optimizing a reconstruction criterion. The input descriptor  $\Psi^{(k)}$  at feature point  $k$  is first encoded via the  $L_1$  encoding layer into an intermediate representation  $\mathbf{y}^{(k)} = h^1(\Psi^{(k)})$ :

$$\mathbf{y}^{(k)} = \sigma(\mathbf{w}^{(k)} \cdot \Psi^{(k)} + \mathbf{b}^{(k)}) \quad (5.5)$$

Where  $\sigma$  is a sigmoid function,  $\mathbf{w}^{(k)}$  and  $\mathbf{b}^{(k)}$  are respectively the neuron weight matrix and bias vector of the  $L_1$  neuron layer for feature point  $k$ . The output is then typically computed as the input reconstruction  $\tilde{\Psi}^{(k)} = g^1(\mathbf{y}^{(k)})$  using an affine decoder with tied input weights to reduce the number of parameters:

$$\tilde{\Psi}^{(k)} = \mathbf{w}^{(k)T} \cdot \mathbf{y}^{(k)} + \mathbf{c}^{(k)} \quad (5.6)$$

Where  $\mathbf{c}^{(k)}$  is the decoder bias vector. Then the set of  $K$  encoded descriptors  $\{\mathbf{y}^{(k)}\}_{k=1\dots K}$  associated to feature points  $k = 1\dots K$  that belong to the face subpart  $m$  are concatenated to form the input  $\xi^{(m)}$  of the layer  $L_2$  for that subpart. Once again, the input of the  $L_2$  layer is successively encoded into an intermediate representation  $\mathbf{z}^{(m)} = h^2(\xi^{(m)})$  and decoded in the same way into a reconstructed version  $\tilde{\xi}^{(m)} = g^2(\mathbf{z}^{(m)})$ :

$$\mathbf{z}^{(m)} = \sigma \left( \mathbf{w}'^{(m)} \cdot \xi^{(m)} + \mathbf{b}'^{(m)} \right) \quad (5.7)$$

$$\tilde{\xi}^{(m)} = \mathbf{w}'^{(m)T} \cdot \mathbf{z}^{(m)} + \mathbf{c}'^{(m)} \quad (5.8)$$

Thus, each layer is trained separately using stochastic gradient descent and backpropagation. More specifically, the input descriptors for each layer are presented sequentially. For example, a forward pass through the  $L_1$  layer provides a reconstructed version  $\tilde{\Psi}^{(k)}$  of  $\Psi^{(k)}$ . The squared  $\mathcal{L}_2$ -loss is then computed and weighted by a learning rate parameter to provide the parameter update  $(\delta \mathbf{w}^{(k)}, \delta \mathbf{b}^{(k)}, \delta \mathbf{c}^{(k)})$ . We tried various combinations of training parameters and the best reconstruction results were obtained by applying 15000 stochastic gradient updates with alternating sampling between the expression classes in the databases. Indeed, we want the network to be able to reconstruct local variations of all possible expressive patterns on an equal foot. We also use a constant learning rate of 0.01 as well as a weight decay of 0.001, which seem to provide good results in testing. Finally, we found that adding 25% randomly generated masking noise provided satisfying results (see Section 3.3.3.2.2 for an overview of regularization schemes for autoencoder training). From a manifold learning perspective, the goal of using such denoising criterion is to learn to project corrupted examples (e.g. partially occluded ones, which lie further from the manifold) back on the training data manifold. Such example will be reconstructed closer to the training data and its confidence shall be smaller.

### 5.3.2.3 Confidence measurement

Given a face image  $\mathcal{I}$ , we define the point-wise confidence  $\alpha^{(k)}(\mathcal{I})$  for point  $k$  as the  $\mathcal{L}_2$ -loss (*i.e.* the reconstruction error) between the HOG descriptor  $\Psi(\mathcal{I})$  extracted from

this feature point, and its reconstruction  $\tilde{\Psi}$  outputted by the network, after successively encoding by layers  $L_1$  then  $L_2$ , and decoding in the opposite order. By abuse of notation, we have:

$$\alpha^{(k)}(\mathcal{I}) = 1 - \frac{\|\Psi^{(k)} - g^1 \circ g^2 \circ h^2 \circ h^1(\Psi^{(k)})\|^2}{\|\Psi^{(k)}\|^2 \|g^1 \circ g^2 \circ h^2 \circ h^1(\Psi^{(k)})\|^2} \quad (5.9)$$

The choice of using an Euclidean distance as a confidence score seems natural as it is optimized during training. We also introduce a confidence measurement  $\alpha^{(\tau)}(\mathcal{I})$  defined over triangles  $\tau = \{k_1, k_2, k_3\}$  of the facial mesh as:

$$\alpha^{(\tau)}(\mathcal{I}) = \min(\alpha^{(k_1)}(\mathcal{I}), \alpha^{(k_2)}(\mathcal{I}), \alpha^{(k_3)}(\mathcal{I})) \quad (5.10)$$

As highlighted in the following experiments, this triangle-wise confidence measurement can thus be used to weight LEPs to enhance the robustness to partial occlusions to a significant extent.

## 5.4 Using local predictions for Action Unit detection

LEPs are local responses related to categorical facial expressions. Thus, it makes sense to assume that LEPs can somehow be related to AUs and constitute a good high-level representation for AU activation prediction.

### 5.4.1 Using available categorical expression-related data for Action Unit detection

More specifically, Figure 5.4 describes a basic AU recognition framework, in which LEP vectors corresponding to each triangle are extracted by a first layer of local trees, trained on a categorical expression dataset. The concatenation of all LEP vectors  $p(c|\mathcal{I}, \tau)$  for every expression  $c \in \mathcal{C}$  (6 universal expressions plus the neutral one) and triangle  $\tau$  of the facial mesh gives rise to a  $7 \times N_\tau$  feature vector used by a second layer of trees defined for each AU (with  $N_\tau$  the number of triangles of the facial mesh). Thus, the AU recognition layer is trained on a FACS-labelled dataset using only one feature template

$\phi_{c,\tau}^{(0)} = p(c|\mathcal{I}, \tau)$ , with associated thresholds  $\theta$  randomly generated from a uniform distribution in the  $[0; 1]$  interval.

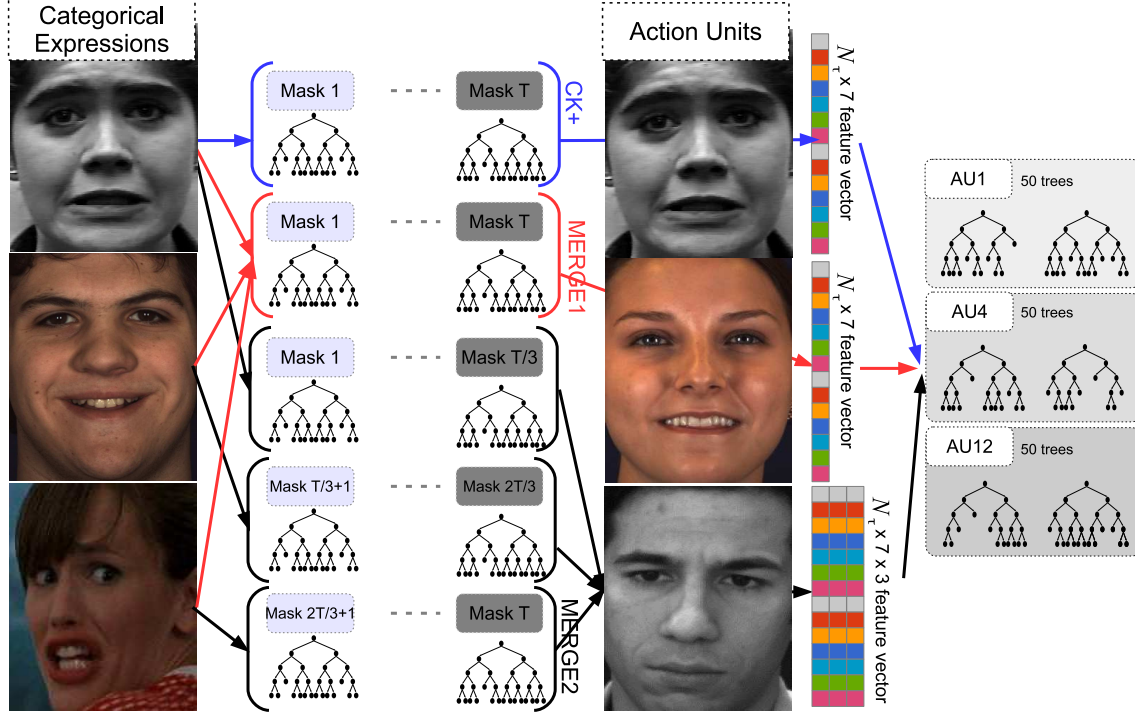


Figure 5.4: AU recognition using LEP features.

As illustrated on Figure 5.4, we also study the importance of using multiple available expression datasets for learning the first layer of trees (*i.e.* LEP representation). We can either train the models on a specific categorical expression database, or merge the datasets to learn LEP representation from all the available corpus (*M1*). Finally, we can also learn LEPs separately from the different categorical expression datasets and use a concatenation of the LEP feature vectors as an input for the second (AU prediction) tree layer (*M2*). Section 5.5.4 shows that those two approaches enhance the predictive power of the AU detection framework. Furthermore, those two strategies can complement each other well. Indeed, *M1* requires to simultaneously load multiple datasets at training time, *M2* involves computing multiple LEP features for evaluation. Thus, a combination of those two strategies can be used to fulfil the target memory/time requirements.

Also note that we voluntarily keep the AU recognition layer simple so as to showcase the usefulness of LEP representation for the AU prediction task, as compared to low-level

engineered descriptors and other state-of-the-art methods. However, as shown in other works on expression recognition [65], recent approaches such as multi-task formulations (e.g. training a single system for predicting multiple AUs) can significantly improve performances.

### 5.4.2 Confidence measurement in Action Unit activation

Because AUs are defined locally, chances are that AU activation relatively to an occluded area can not be predicted at all. Thus, we use the weights outputted by the autoencoder network to automatically derive a confidence score relatively to each AU indexed by  $m$ . To this end, we define as  $N_{l,\tau}^{(m)}$  the number of times that the LEP feature  $\phi_{l,\tau}^{(0)}$  was selected for splitting at the root of the trees, among all trees in the forest. This is highlighted on Figure 5.5. The reason for exclusively considering features at the root of the trees is that those features are selected from large numbers of training examples, as opposed to features from nodes deeper in the trees, that are essentially more noisy.

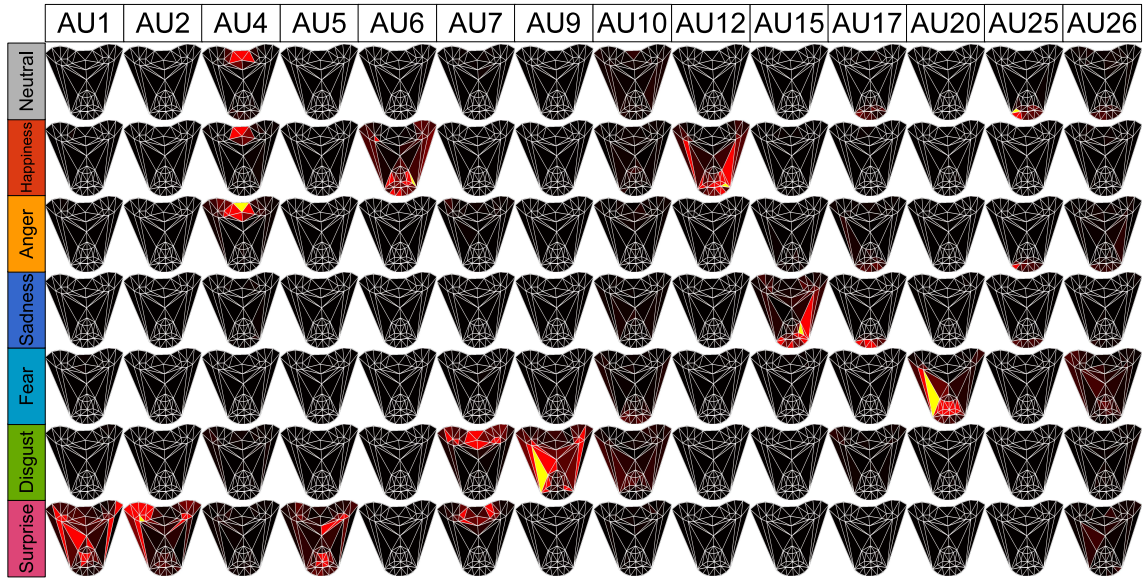


Figure 5.5: LEPs heat map for each of the 14 AUs (CK+ database). Only top-level LEP features are displayed for each AU. Best viewed in color.

Note that, while most approaches focus on describing expressions as a combination of AUs, we can decompose each AU as a set of local expression predictions. For exam-



ple, for AU1 (inner brow raiser) and AU2 (outer brow raiser), the most relevant LEPs are triangles corresponding to the inner and outer brows, associated with expression *surprise*, respectively. AU4 (brow lowerer) mainly uses triangles between the eyes associated with expression *anger*. AU9 (nose wrinkler) mainly uses triangles from the nose and cheek regions, associated with *disgust*. AU12 (lip corner puller) and AU20 (lip stretcher) respectively use triangles corresponding to lip corners with expressions *happiness* and *fear*.

We then define the AU-specific confidence measurement  $\alpha_m$  for AU  $m$  as the sum of confidences  $\alpha^{(\tau)}$  of triangles  $\tau$  of the facial mesh, weighted by the proportion of LEP features from that triangle, that are used to describe the activation of AU  $m$ :

$$\alpha_m = \frac{\sum_{\tau} \alpha^{(\tau)} N_{l,\tau}^{(m)}}{\sum_{\tau} N_{l,\tau}^{(m)}} \quad (5.11)$$

Thus, the AU-specific confidence measurement is proportional to the confidence of the face regions that are the most useful for describing the activation of a specific AU. We show in the following section that such simple setting allows to highlight the cases where the AU predictions are deemed unreliable.

### 5.4.3 Multi-output prediction of Action Unit activation

In Section 5.4.1 we trained independent detectors for each specific AU. However, by doing so, we essentially ignore the possible correlations between the AU activation prediction tasks. For that matter, we use the multi-output (MO) paradigm to train multiple trees that can each predict several AUs. From a RF induction perspective, each training strategy is defined by two aspects:

- Which AUs are defined to split the tree nodes during training.
- Which trees are used to predict which AUs.

In order to do that, we assign task weights  $w_i^{t,n}$  for a node  $n$  of a tree  $t$  and an AU detection task  $i \in \{1, \dots, N_{au}\}$  to compute the successive splitting objective functions. Thus, given a binary feature  $\{\phi, \theta\}$  the multi-output splitting criterion for node  $n$  shall become:

$$H_{\phi,\theta}^{1,\dots,N_{au}} = \sum_{i=1}^{N_{au}} w_i^{t,n} H_{\phi,\theta}^i \quad (5.12)$$

Where  $H^i$  is a single-output objective function defined, in our case, over a two-class problem regarding the activation of AU  $i$ . The tasks weights  $w_i^{t,n}$  can be generated at the tree level, in which case they can be used to weight the AUs at test time. They can also be generated independently for each node  $n$  of tree  $t$ . We investigate multiple strategies for setting the tasks weights, which are depicted on Figure 5.6.

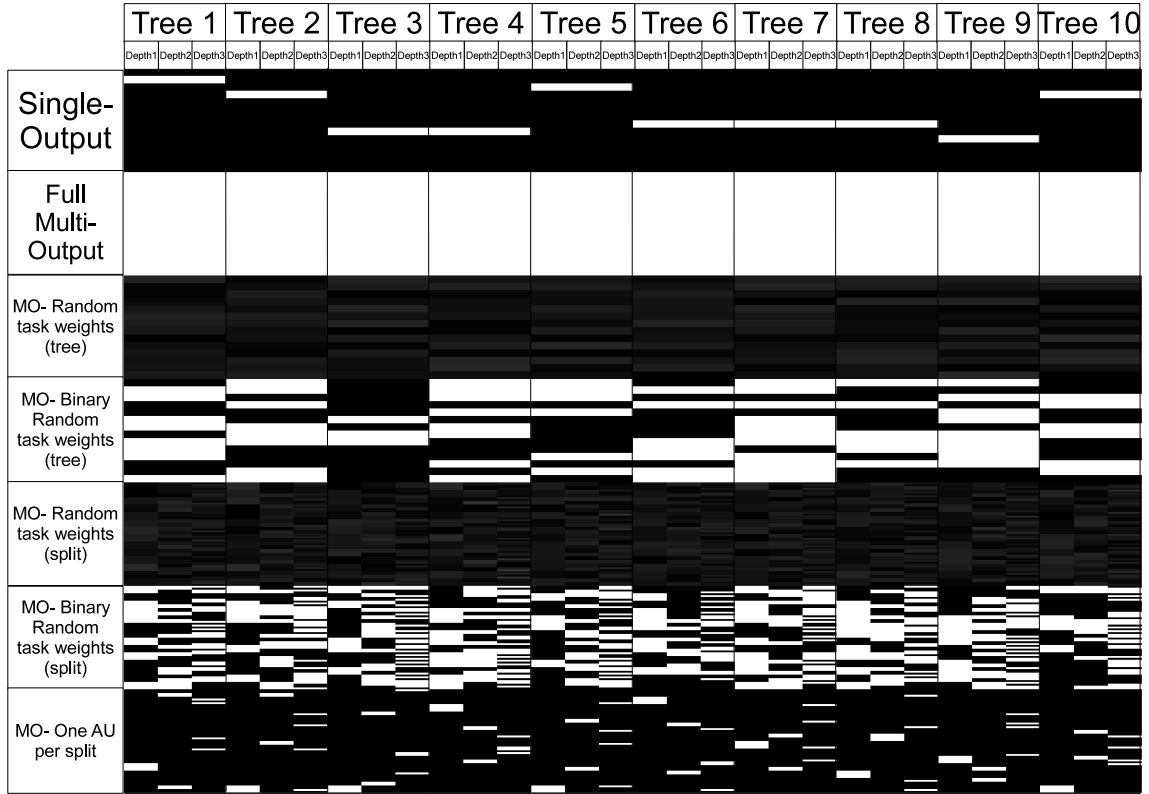


Figure 5.6: Illustration of different task weights assignment strategies. Grey level intensity bars represent task weight values in the  $[0, 1]$  interval.

- **Single-Output (SO):** we only predict one AU per tree by setting only one weight to 1 (only one white bar per tree on Figure 5.6) for each tree and the others to 0.
- **Full Multi-Output (MO):** each tree uses an unweighted combination of all AUs (*i.e.* all tasks weights  $w_i^{t,n}$  are set to 1) for splitting. Also, each tree predicts all the AUs simultaneously.



- **MO-random weights (tree):** we set the weights to uniformly sampled random values for each tree:  $w_i^{t,n} = w_i^t \sim \mathcal{U}[0, 1]$ . At test time the prediction of this tree for AU  $i$  is weighted by  $w_i^t$ .
- **MO-Binary random weights (tree):** we randomly set each weight to either 0 or 1 for each tree:  $w_i^t$  are sampled as a Bernoulli random variable with expected value  $1/2$ , *i.e.*  $w_i^t \sim \mathcal{B}(0.5)$ .
- **MO-random weights (split):** we set the weights to uniformly sampled random values for each split node:  $w_i^{t,n} \sim \mathcal{U}[0, 1]$ .
- **MO-Binary random weights (split):** we randomly set each weight to either 0 or 1 for each split node:  $w_i^{t,n} \sim \mathcal{B}(0.5)$ .
- **MO-One AU per split:** we randomly select only one AU for splitting each node (as suggested in [54]).

A comparison between those approaches can be found in Section 5.5.4.

## 5.5 Experiments

### 5.5.1 Experimental setup

#### 5.5.1.1 Evaluation metrics

For both occluded and non-occluded scenarios of categorical FER we use the overall accuracy as a performance metric. We also report confusion matrices to show the discrepancies between recognition of the expression classes. For AU detection we use the area under the ROC curve (AUC) as a performance metric, as it is widely used in the literature because it is independent of a decision threshold. For all the experiments, RF classifiers are evaluated with the Out-Of-Bag (OOB) error estimate. For the experiments involving a first layer of LEP features, these are generated for Out-Of-Bag examples for each tree. Moreover, AUs are evaluated with OOB error.

### 5.5.1.2 Hyperparameter setting

In order to decrease the variance of the error we train large collections of trees ( $T1 = 1000$  for LEP generation,  $T = 50$  for AU detection). For training the local models, we set the locality parameter  $R$  to 0.1 (which means that each local model uses 1/10 of the total face surface) which provides good robustness to occlusions. Finally, we use 40  $\phi^{(1)}$ , 40  $\phi^{(2)}$  and 160  $\phi^{(3)}$  features for learning LEPs, as well as 25 threshold evaluations per feature, as these seem to provide satisfying results (Section 4.5.1). For AU detection, we examine 100  $\phi^{(0)}$  features at each node, each associated with 25 threshold values. Note however that the values of these hyperparameters (except for  $R$ ) had very little influence on the performances. This is due to the complexity of the RF framework, in which individually weak trees (e.g. that are grown by only examining a few features per node) are generally less correlated, still outputting decent predictions when combined altogether.

For the occluded scenarios on CK+ and BU4DFE, the autoencoder networks are trained in a cross-database fashion (*i.e.* training on CK+ and testing on BU4DFE and vice versa). Lastly, on SFEW database, we use the autoencoder network trained on CK+, as SFEW embraces multiple examples of occluded faces.

## 5.5.2 Experiments on non-occluded scenarios

In Tables 5.1, 5.3, 5.5 we report the average accuracy obtained by our local subspace Random Forest (LS-RF) and the confidence-weighted version (WLS-RF). We also compare with standard RF (RS-RF).

Generally speaking, classification results of LS-RF are a little better than those of the RS-RF. Indeed, forcing the trees to be local allows to capture more diverse information. RS-RF relies quite heavily on the mouth region, but other areas (e.g. around the eyes, eyebrows and nose regions) may also convey information that can be captured by local models. Figure 5.7 displays the proportion of top-level features over all triangles of the face area.

While more than 90% of the features extracted by RS-RF are concentrated around the mouth, the repartition for LS-RF is more homogeneous. Hence, LS-RF is less prone

Table 5.1: CK+ database. <sup>†</sup>: CK database

CK+	Protocol	7em	8em
LBP [78]	10-fold	88.9 <sup>†</sup>	-
CSPL [112]	10-fold	89.9 <sup>†</sup>	-
iMORF [110]	10-fold	-	90.0
AUDN [56]	10-fold	93.7	92.0
RS-RF	OOB	92.6	91.5
LS-RF	OOB	94.1	<b>93.4</b>
WLS-RF	OOB	<b>94.3</b>	<b>93.4</b>

Table 5.2: Confusion matrix (CK+-8em)

	ne	ha	an	sa	fe	di	co	su
ne	92.4	0.3	0.9	0.6	1.2	0.6	3.97	0
ha	0	100	0	0	0	0	0	0
an	4.4	0	91.1	0	0	2.3	2.3	0
sa	22.6	0	0	77.4	0	0	0	0
fe	1.3	4	0	0	90.7	0	0	4
di	3.4	0	0.6	0	0	96.1	0	0
co	11.1	0	0	3.7	0	0	85.2	0
su	1.6	0	0	0	0.4	0	1.2	96.8

Table 5.3: BU4DFE database

BU4DFE	Protocol	% Acc
BoMW [100]	10-fold	63.8
Geometric [85]	10-fold	68.3
LBP-TOP [37]	10-fold	71.6
2D FFDs [74]	10-fold	73.4
RS-RF	OOB	73.1
LS-RF	OOB	74.3
WLS-RF	OOB	<b>75.0</b>

Table 5.4: Confusion matrix (BU4DFE)

	ne	ha	an	sa	fe	di	su
ne	89.5	0	1.8	4.4	0.9	0.9	2.6
ha	2	89.9	0	0	5	2	1
an	10.1	0	70.7	7.1	2	9.1	1
sa	11	0	15	71	3	0	0
fe	9.8	17.6	2.9	5.9	38.3	11.8	13.7
di	3	4	6.9	1	7.9	73.3	4
su	0	1	0	1	6.2	0	91.8

Table 5.5: SFEW database

SFEW	% Acc
PHOG-LPQ [24]	19.0
DS-GPLVM [29]	24.7
AUDN [56]	30.1
Semi-Supervised [55]	34.9
RS-RF	35.7
LS-RF	35.6
WLS-RF	<b>37.1</b>

Table 5.6: Confusion matrix (SFEW)

	ne	ha	an	sa	fe	di	su
ne	50.2	8.8	9.0	10.0	2.0	16.9	3.1
ha	10.6	67.5	6.2	6.9	2.6	3.5	2.6
an	25.4	16.1	31.3	10.1	3.7	0.9	12.5
sa	21.2	21.2	8.1	22.2	7.1	9.1	11.1
fe	14.2	16.2	13.0	5.0	23.1	7.1	21.3
di	31.3	23.7	10.4	7.1	3.7	15.6	8.2
su	15.4	11.0	12.1	3.3	7.7	6.6	44.0

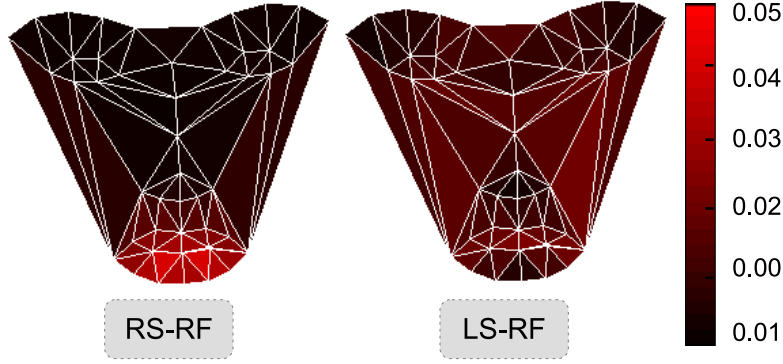


Figure 5.7: Proportion of top-level (tree root) features per triangle. Best viewed in color.

to a misalignment of the mouth feature points, or to occlusions of the mouth region. Furthermore, weighting the local predictions (WLS-RF) using the confidence score from the autoencoder network allows to enhance the results on BU4DFE and SFEW. The reason is that subjects from those datasets exhibit uncommon morphological traits, occlusion or lighting patterns. As such, more emphasis is put on reliable local patterns, resulting in a better overall accuracy. It also explains why the accuracy is equivalent for LS-RF and WLS-RF on CK+ database, where there is less variability. On the three databases, LS-RF and WLS-RF models provide better results compared to state-of-the-art approaches, even though some of these use complex FFD or spatio-temporal features (LBP-TOP), or use additional unlabelled data for regularization [55]. Note however that the evaluation protocols are different for some of these approaches. For example, authors in [29] use only the texture information and not the provided landmarks.

Tables 5.2, 5.4, 5.6 show the confusion matrices of WLS-RF on CK+, BU4DFE and SFEW respectively. Generally speaking, expressions *neutral*, *happy* and *surprise* are mostly correctly recognized, as they involve the most recognizable patterns (smile or eye-brow raise). *Anger* and *disgust* are also accurately recognized on CK+ and BU4DFE but not so much on SFEW. *Sadness* and *fear* seem to be the most subtle ones, particularly on BU4DFE and SFEW where those expressions can be misclassified as *surprise* or *happy*, respectively.

### 5.5.3 Experiments on occluded scenarios

#### 5.5.3.1 Targeted occlusions

In order to assess the robustness of our system to partial face occlusion, we first measured the average accuracy outputted by RS-RF, LS-RF and WLS-RF on CK+ (8 expressions) and BU4DFE (7 expressions) databases with synthetic occlusions of targeted areas of the face, namely the mouth and eyes regions. More precisely, for each image we use the feature points tracked on non-occluded images to highlight the eyes and mouth regions. We then overlay a noisy pattern (see Figure 5.8), which is a more challenging setup than black boxes used in [107, 40]. We add margins of 20 pixels to the bounding boxes to make sure we cover the whole eyes (with eyebrows, as they represent the most valuable source of information from the eye region) and mouth region. Finally, we align the feature points on the occluded sequences.

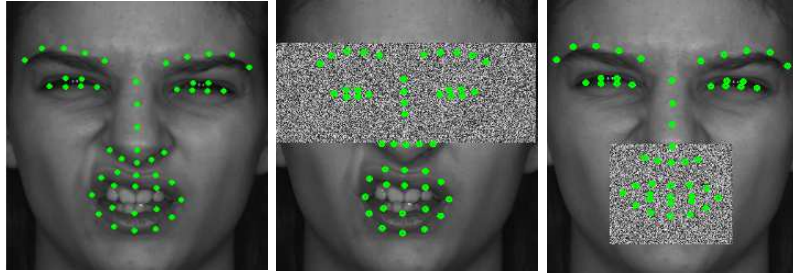


Figure 5.8: Examples of occluded faces from BU4DFE with aligned feature points. Left: non-occluded, middle: eyes occluded, right: mouth occluded. Also notice how the presence of an occlusion may have a critical effect on the quality of the feature point alignment.

Graphs of Figure 5.9 show the variation of average accuracy vs. hyperparameter  $R$  that controls the locality of the trees, respectively under eyes and mouth occlusion on CK+ database. Performances of RS-RF fall heavily when the mouth is occluded (from 91.5% to 25.4%), as observed in [107]. This further proves that the global model relies essentially on mouth features to decipher facial expressions. Forcing the trees to be more local (e.g. setting  $R$  to 0.1 or 0.2) allows to capture more diverse cues from multiple facial areas, ensuring more robustness to mouth occlusion. It also explains why LS-RF

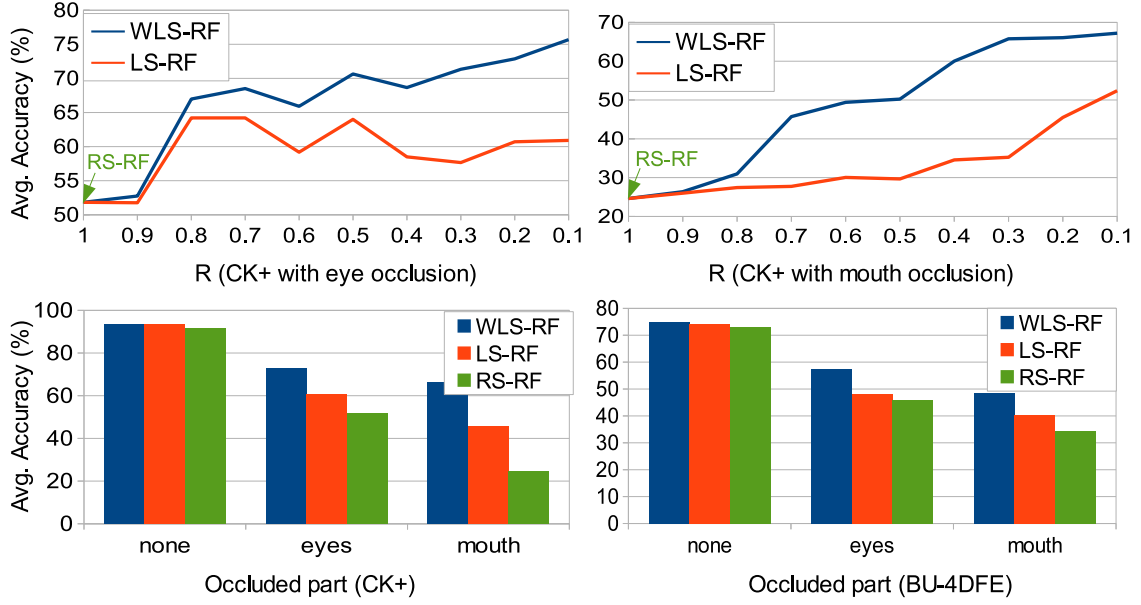


Figure 5.9: Accuracy outputted on occluded CK+ and BU4DFE databases

models with  $R = 0.8 - 0.5$  can already be quite robust to eye occlusions, as the majority of the information used on such models likely comes from mouth area. Nevertheless, on those two occlusion scenarios, WLS-RF achieves a substantially better accuracy than the unweighted local models. Figure 5.9 also shows the accuracy comparison for both eyes and mouth occlusion scenarios on CK+ and BU4D, with  $R = 0.1$ . On the two databases, LS-RF is more robust to partial occlusions than RS-RF. Furthermore, WLS-RF also provides better accuracy than both LS-RF and RS-RF.

### 5.5.3.2 Random occlusions

In order to quantify the capability of our method to deal with unpredicted occlusions as well as to compare our result to state-of-the-art methods for FER under partial occlusion, we evaluated our method on various occlusion scenarios using the same protocol as in [107]. Specifically, we evaluate the average prediction accuracy for 7-classes expression recognition on the CK+ database, where, for each image, a region of the face is being overlaid by white occluding patterns. These patterns are defined by the feature points locations for eye and mouth occlusions, and by white patches of size  $8 \times 8$  (R8),  $16 \times 16$  (R16) and  $24 \times 24$  (R24) for face crops of size  $48 \times 48$ . Similarly to what was done above,

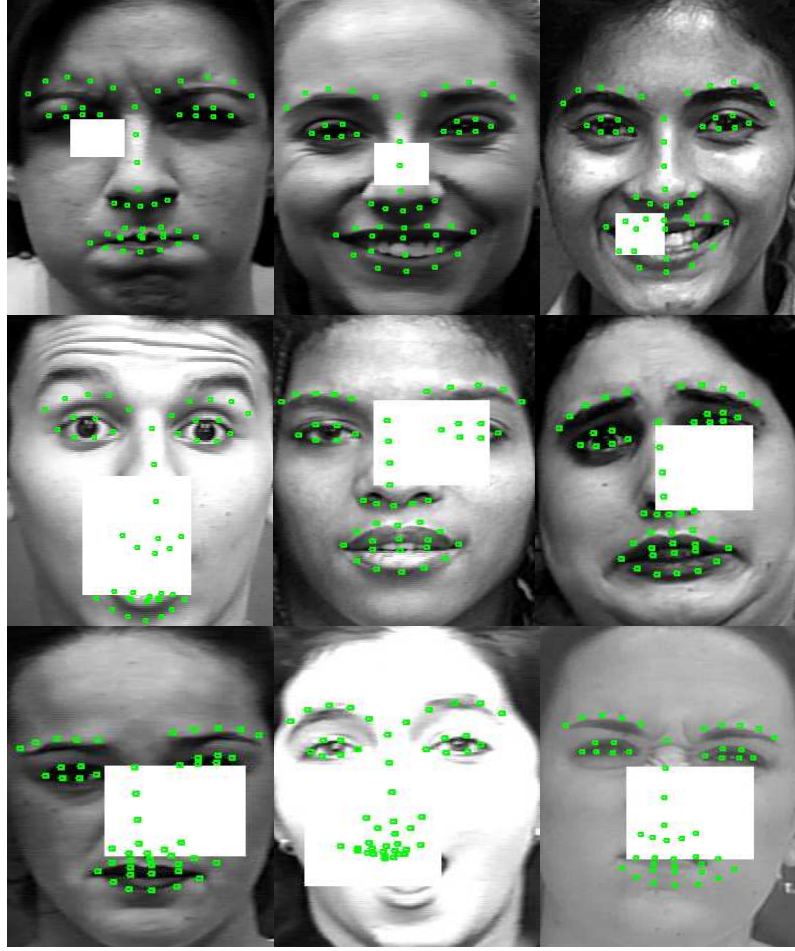


Figure 5.10: Examples of randomly occluded faces from CK+ with aligned feature points. Top row: R8 benchmark ( $8 \times 8$  overlaying patches). Middle row: R16 benchmark ( $16 \times 16$  overlaying patches). Bottom row: R24 benchmark ( $24 \times 24$  overlaying patches).

the facial feature points are aligned after the white patches are overlaid on the face, hence occlusions are likely to cause feature point misalignment in addition to texture corruption. Examples of randomly occluded faces can be observed on Figure 5.10. Table 5.7 shows the comparison of WLS-RF with [107] on several partial occlusion scenarios.

As one can see, WLS-RF provides similar accuracy as the randomly sampled Gabor templates introduced in [107] on the R8 and *eyes occluded* cases. However, on the most difficult benchmarks (R16, R24 and mouth occluded), WLS-RF provides significantly better results. This further shows that performing FER by using a confidence-weighted combination of spatially-constrained trees allows to flexibly handle partial occlusions

Table 5.7: Comparison with [107] on multiple scenarios

Protocol	WLS-RF	[107]
R8	<b>92.2</b>	<b>92</b>
R16	<b>86.4</b>	82
R24	<b>74.8</b>	62.5
Eyes occluded	<b>87.9</b>	<b>88</b>
Mouth occluded	<b>72.7</b>	30.3

without requiring any occluded data for training the classifiers.

### 5.5.3.3 Realistic occlusions

Our occlusion model is however quite “boring”, in the sense that the occluding noisy patterns are not realistic. For that matter, and because there is currently no FER database that includes annotated partial occlusion ground truth, we also present on Figure 5.11 qualitative results on more realistic occlusions. Notice how the autoencoder network (learnt on CK+) assign high confidences (green) to non-occluded feature points, whereas examples that lie further from the captured manifold (e.g. because of lighting conditions, self-occlusion with a hand or with an accessory) are given lower values (red). The corresponding triangles are thus downweighted for FER and appear transparent on the last row. Also note that different facial regions can vote for different expressions, as shown on the second column (*happy+angry/disgust*).

## 5.5.4 Experiments on AU detection

### 5.5.4.1 Merging multiple datasets

In this section we present results for AU detection using LEP features. Table 5.8 shows comparison of AUC for the prediction of AU activations on CK+ database obtained with LEPs trained on CK+, BU4DFE and SFEW databases, as well as models obtained *via* the *M1* and *M2* strategies.

For nearly every AU on CK+, the best AUC score is provided by the *M2* strategy.





Figure 5.11: Examples of local FER under realistic occlusions. Top rows: point-wise confidence scores (red: low confidence, green: high confidence). Middle rows: triangle-wise scores. Bottom rows: weighted local classification (transparent for low confidences, gray for neutral, red for *happy*, yellow for *angry*, blue for *sad*, cyan for *fear*, green for *disgust* and magenta for *surprise*). Best viewed in color.

However LEPs trained on CK+ only as well as the  $M1$  strategy also provide good prediction results. LEPs trained solely on BU4DFE and SFEW seem a bit lackluster, but using the additional categorical expression data in addition to data from CK+ can be beneficial for prediction accuracy. Interestingly, on BP4D, LEPs trained on CK+ only seem to have a slight edge over the two LEP models trained using all the available data. However, the  $M2$  strategy and, to a lesser extent,  $M1$  and training on BU4DFE only, provide close performances. Furthermore, on the DISFA dataset, the  $M1$  and the  $M2$  LEP models provide the highest AUC. Overall, the  $M2$  and  $M1$  models seem to perform better, followed by the models trained on CK+. This proves that AU detection can benefit from additional

Table 5.8: AUC scores on CK+, BP4D and DISFA databases. Results are presented for LEPS trained with multiple settings, *i.e.* Merge (M1 and M2) and training on CK+, BU4DFE and SFEW categorical expression databases.

	CK+					BP4D					DISFA				
AU	M1	M2	CK+	BU4DFE	SFEW	M1	M2	CK+	BU4DFE	SFEW	M1	M2	CK+	BU4DFE	SFEW
AU1	97.9	<b>98.4</b>	<b>98.4</b>	94.7	93.3	59.6	62.7	<b>63.6</b>	60.9	52.0	66.1	68.4	<b>71.3</b>	57.6	66.6
AU2	98	<b>98.2</b>	97.7	97.5	97.2	65.4	64.8	62.3	<b>66.0</b>	53.0	53.8	55.2	<b>67.3</b>	59.3	59.4
AU4	93.3	<b>95.4</b>	94.8	83.1	85.6	<b>68.7</b>	63.8	64.4	64.4	55.3	66.7	66.7	<b>67.3</b>	64.0	67.6
AU5	94	<b>97.5</b>	95.5	93.2	95	-	-	-	-	-	84.2	85.6	73.3	<b>88.6</b>	73.7
AU6	95.4	<b>95.7</b>	95.5	94.3	94.9	<b>83.1</b>	81.8	82.6	78.5	77.1	89.1	86.0	<b>89.2</b>	86.8	85.1
AU7	89.1	<b>90.2</b>	89.6	88.1	83	<b>76.8</b>	75.0	73.6	72.6	65.0	-	-	-	-	-
AU9	97.9	<b>99.3</b>	98.7	98.5	94.8	-	-	-	-	-	<b>79.0</b>	77.0	75.4	74.0	53.4
AU10	83.7	85.6	<b>86.5</b>	78.4	81.7	83.7	<b>83.8</b>	83.3	81.0	78.6	-	-	-	-	-
AU12	<b>97.6</b>	96	96.2	96	96.5	89.9	<b>90.0</b>	89.8	88.0	87.2	<b>95.5</b>	92.9	93.6	92.8	91.8
AU14	-	-	-	-	-	65.2	66.4	63.7	<b>66.5</b>	64.9	-	-	-	-	-
AU15	<b>91</b>	88.9	88.3	79	79.5	56.8	58.4	<b>58.5</b>	57.7	56.0	<b>69.5</b>	64.5	63.6	68.8	61.7
AU17	93.9	<b>95.1</b>	93.4	81.5	86.4	55.8	65.7	<b>68.9</b>	65.1	60.6	<b>67.8</b>	61.2	53.5	59.1	58.8
AU20	91.9	93.8	<b>94.5</b>	88.5	85.8	-	-	-	-	-	<b>65.0</b>	58.5	50.2	55.5	61.9
AU23	-	-	-	-	-	50.1	57.2	<b>60.2</b>	57.5	54.2	-	-	-	-	-
AU24	-	-	-	-	-	69.6	77.4	<b>78.2</b>	77.7	68.4	-	-	-	-	-
AU25	99	<b>99.1</b>	98.8	87.1	97.4	-	-	-	-	-	94.8	95.0	94.0	<b>95.6</b>	80.0
AU26	75.7	<b>81.2</b>	79.7	74.9	73.4	-	-	-	-	-	79.3	<b>81.4</b>	75.6	78.5	71.5
Avg	92.7	<b>93.7</b>	93.4	88.2	88.9	68.8	70.6	<b>70.8</b>	69.6	64.3	<b>75.9</b>	74.4	72.9	73.4	69.3

training data labelled with categorical expressions. Finally, LEPS trained on SFEW did not perform very well, probably due to the fact that the database embraces too much variability for too few training data. Thus, the categorical expressions can not be captured adequately, as can be seen from the low accuracies showed in Section 5.5.2.

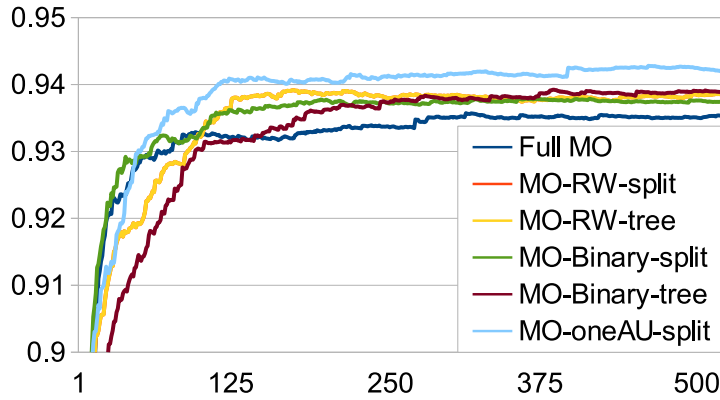
#### 5.5.4.2 Multi-output strategies

Table 5.9 shows the relative interest of the different multi-output learning strategies detailed in Section 5.4.3 on CK+. Overall, the SO strategy seems to output the lowest AUC score, as it essentially ignores the co-dependency between the multiple AU detection tasks. It is however closely followed by the full MO model. The MO-RW-tree and MO-RW-split provide slightly better results. Nevertheless, the latter are overshadowed by the MO-Binary-tree and MO-Binary-split which not only provides better overall accuracy, but also allows a significant decrease of the computational load during training, as

Table 5.9: AUC scores on CK+ database

training strategy	AU1	AU2	AU4	AU5	AU6	AU7	AU9	AU10	AU12	AU15	AU17	AU20	AU25	AU26	Avg.
SO	<b>98.4</b>	97.7	94.8	95.5	95.5	89.6	98.7	86.5	96.2	88.3	93.4	94.5	98.8	79.7	93.4
Full MO	97.8	98.1	94.6	96.4	96.1	<b>91.8</b>	99.2	84.9	97.7	89	94.9	93.1	99.2	76.7	93.5
MO-RW-tree	97.8	<b>98.2</b>	94.8	96.4	<b>96.2</b>	91.4	99.2	83.3	97.7	89.6	<b>95.1</b>	93.2	<b>99.3</b>	81.7	93.8
MO-Binary-tree	98.1	<b>98.2</b>	<b>94.9</b>	<b>96.8</b>	<b>96.2</b>	91.3	98.8	<b>87</b>	97.2	88.7	94.8	93.8	99.2	79.4	93.9
MO-RW-split	97.9	98.1	94.8	96.4	95.9	91.6	99.3	83.2	97.4	<b>89.8</b>	<b>95.1</b>	93.1	99.2	80.3	93.7
MO-Binary-split	97.9	98.1	94.8	96.5	96.1	91.6	99.3	83.3	97.7	89.7	94.9	93.1	99.2	80.3	93.7
MO-1task-split	98.1	<b>98.2</b>	<b>94.9</b>	96.4	96.1	91.6	<b>99.5</b>	86.8	<b>98</b>	88.6	<b>95.1</b>	<b>93.7</b>	99.2	<b>82.4</b>	<b>94.2</b>

only half the tasks are used to compute the splits. The best overall strategy is the recently proposed (see [54]) MO-1task-split, which is significantly faster and more accurate, as it further increases the decorrelation of the trees. Moreover, as illustrated on Figure 5.12, as all the AU prediction tasks are used for training each tree, the accuracy grows faster than the and the MO-RW-tree and MO-Binary-tree models, and quite as fast as the MO-RW-split and MO-Binary-split models. Thus, high accuracies can be obtained by using a restricted number of trees (e.g.  $T = 125$ ).

Figure 5.12: Influence of  $T$  (number of trees) on average AUC

### 5.5.4.3 Impact of mesh refinement

Table 5.10 shows the accuracy obtained on CK+ and DISFA for both the original 49-point mesh and a 500-point refined mesh, using Multi-output AU prediction performed upon LEP features trained using the M2 merge strategy. As one can see, the interest is limited on CK+ because a coarse mesh is sufficient to describe such prototypical facial behaviors.

On the DISFA database though, the refined facial mesh gives higher AUC for nearly every AU. Indeed, using a more fine-grained mesh allows to produce more diverse LEP features by subdividing large areas such as triangles corresponding to the cheeks.

Table 5.10: AUC scores on CK+ and DISFA databases

	CK+		DISFA	
AU	M2,MO(49)	M2,MO(500)	M2,MO(49)	M2,MO(500)
1	97.9	<b>98.3</b>	67.4	<b>72.8</b>
2	98.4	<b>98.8</b>	52.6	<b>61.9</b>
4	93.7	<b>94.1</b>	75.2	<b>77.8</b>
5	<b>96.8</b>	96.3	<b>88.3</b>	86.9
6	<b>96.2</b>	95.8	89.7	<b>90.5</b>
7	89.6	<b>91.6</b>	-	-
9	99.5	<b>99.6</b>	82.5	<b>84.9</b>
10	86.5	<b>86.8</b>	-	-
12	<b>98.1</b>	97.3	96.2	<b>96.2</b>
15	<b>88.2</b>	<b>88.2</b>	<b>74.5</b>	72.5
17	<b>94.5</b>	94.1	<b>64.2</b>	62
20	<b>95</b>	93.3	62.1	<b>63.1</b>
25	<b>99.5</b>	99.4	95.7	<b>95.8</b>
26	80.1	<b>85.1</b>	77	<b>79.3</b>
mean	94.0	<b>94.1</b>	77.1	<b>78.6</b>

#### 5.5.4.4 Relevance of AU confidence assessment

In order to assess the relevance of the AU-specific confidence measurement, we evaluated its average value on the occluded versions of the CK+ and BU4DFE databases generated in Section 5.5.3 for occlusion handling in categorical FER. From a general perspective, as can be seen on Figure 5.13, low confidence measurements can be observed for AUs from the upper face region on the two scenarios involving eye occlusion. The same holds for AUs from the lower face region and the “mouth occluded” scenario, whereas the confi-

dence scores are significantly higher in the non-occluded case. Interestingly, confidence scores for AU6 (cheek raiser) and, to a lesser extent, AU9 (nose wrinkle), are quite low even in the “mouth occluded” case. Indeed, as can be witnessed on Figure 5.5, the confidence measurement for these AUs also use LEP features from the nose and mouth area.

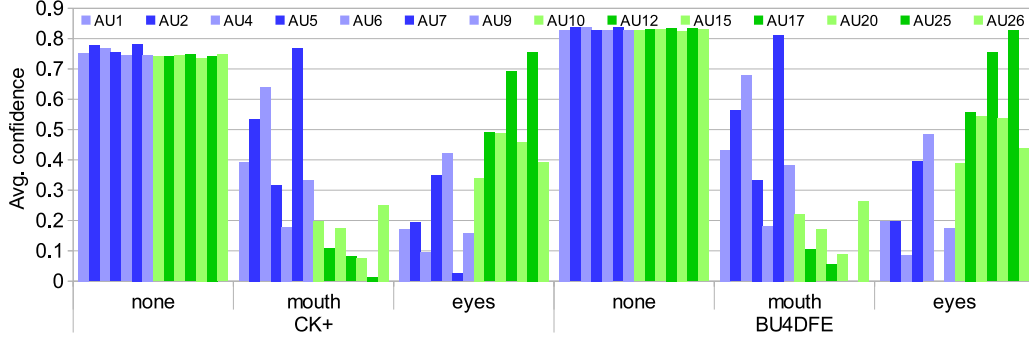


Figure 5.13: AU confidence scores outputted on occluded CK+ and BU4DFE database

#### 5.5.4.5 Comparison with state-of-the-art approaches

Table 5.11 reports the overall best AUC obtained on the three datasets. It also draws a comparison between the AUC scores obtained using our method and results reported in recent publications involving similar protocols (same databases and sets of AUs, same intensity threshold for AU occurrence on DISFA). Our approach provides better results than SHTL [72] on CK+, as well as accuracy similar to the multi-label CNN introduced in [34] on DISFA. Ruiz *et al.* [72] obtain an excellent AUC score of 81.5 on DISFA. However, they do not provide information to ensure that the same protocol is used for evaluation (e.g. the threshold that is applied on the AU intensity values). Last but not least, our method also provides better performance than baselines LBP-TOP features used in [108] on BP4D. This demonstrates that LEPs learned on large amounts of categorical expression data yield high discriminative power for AU detection tasks.

Table 5.11: Comparison with other works

Database	AUC(ours)	AUC(Other works)
CK+(14AU)	<b>94.2</b>	91.7 (SHTL [72])
BP4D(12AU)	<b>70.8</b>	68.9 (LBP-TOP [108])
DISFA(12AU)	<b>78.6</b>	75.7 (Multi-label CNN [34])

Table 5.12: Measured evaluation time per processing step (in milliseconds)

Processing step	time (ms)
Feature point alignment	10
Integral channels computation	2
Confidence weights computation	11
LEP computation (1000 trees)	7
12 AU detection - Single output(50 trees per AU)	1
12 AU detection - Multi output(50 trees total)	0.1
<b>Total</b>	<b>31</b>

### 5.5.5 Runtime evaluation

The proposed framework for occlusion-robust FER (WLS-RF) and AU detection operates in real-time on video streams, even with large tree collections. Table 5.12 displays the elapsed time for each step of the evaluation pipeline. The test was performed on an Intel Core I7-4770 CPU on a single-thread C++/OpenCV implementation.

It appears that the feature point alignment and confidence weight generation steps are the bottleneck of the system in term of computational load. However the runtime for the former can be reduced by the use of more efficient alignment algorithms such as the one proposed in Section 6.3.2. As for the confidence weights, the computation time can be significantly reduced by a proper multithreading (e.g. computing the confidence for each feature point in parallel). As it is, the framework already runs at more than 30 fps even with large collections of trees. As for training, learning LEPs with 1000 trees on a big database (BU4D containing more than 8000 face images) took approximately three hours without parallelization. Training the hierarchical autoencoder network took half a day

and learning the 12 AU detectors on DISFA database with 50 trees required one hour on the same I7-4770 CPU using a loose C++ implementation. Thus, our approach scales well both in terms of training and testing times, especially when compared to recent deep learning algorithms [34] for feature representation and learning.

### 5.5.6 Discussion

Throughout this chapter, we proposed a new high-level expression-driven LEP representation. LEPs are obtained from training RFs upon spatially-defined local subspaces of the face. Extensive experiments on multiple datasets highlight the fact that the proposed representation improves the state-of-the-art for categorical FER and yields useful descriptive power for AU occurrence prediction. Furthermore, we introduced a hierarchical autoencoder network to model the manifold around specific facial feature points. We showed that the provided reconstruction error could effectively be used as a confidence measurement to weight the prediction outputted by the local trees. The proposed WLS-RF framework significantly adds robustness to partial face occlusions. The ideas introduced in this chapter also open a lot of interesting directions for future works on face analysis. First, note that the confidence weights are representative of the spatially defined local manifold of the training data. Thus, these confidence values can be used to determine which parts of the face are the most reliable in a general way (e.g. to address unpredicted illumination patterns or head pose variations), and are not limited to occlusion handling. Furthermore, we could inject confidence weights into the feature point alignment framework in order to enhance the robustness of the feature point alignment w.r.t. occlusions. Compared to a discriminative approach using synthetic data [33], our manifold learning approach could in theory more efficiently deal with realistic occlusions. Moreover, the applications of LEPs for AU detection and intensity estimation are multiple. First, it would be interesting to learn LEPs using more expression data such as the datasets introduced in [76, 93], possibly with a more complex integration strategy.

## Chapter 6

# Greedy Evaluation of Neural Decision Forests

In this chapter, we provide adaptations of the very recent Neural Decision Forest (NDF [48]) machine learning algorithm to allow a more efficient, fully online training procedure (see Section 6.2) as well as an evaluation procedure (Section 6.1) that is an order of magnitude faster than the one proposed in [48]. We highlight on several applications that GNDF is particularly suitable for face analysis, for which it is crucial to work with a very high framerate upon high-dimensional feature vectors. Moreover, we show that the proposed GNDF framework can be adapted to tackle different machine learning problems, *i.e.* classification and regression. In order to do so, we benchmark our algorithms on facial expression classification 6.3.1 and feature point alignment within a cascaded regression framework 6.3.2.

### 6.1 Greedy evaluation procedure

As highlighted in Section 3.3.4.1, in the case of a classical decision tree, the probability  $\mu^l(\mathbf{x}_i)$  to reach each node given an example  $\mathbf{x}_i$  is given by a product of Kronecker deltas that successively indicate if  $\mathbf{x}_i$  is routed left or right:



$$\mu^l(\mathbf{x}_i) = \prod_{n \in \mathcal{N}_l^{right}} \delta^n(\mathbf{x}_i) \prod_{n \in \mathcal{N}_l^{left}} (1 - \delta^n(\mathbf{x}_i)) \quad (6.1)$$

In the case of a NDF (See Section 3.3.5.1), this probability is given by:

$$\mu^l(\mathbf{x}_i) = \prod_{n \in \mathcal{N}_l^{right}} d^n(\mathbf{x}_i) \prod_{n \in \mathcal{N}_l^{left}} (1 - d^n(\mathbf{x}_i)) \quad (6.2)$$

In the case where  $d^n(\mathbf{x}_i) \rightarrow \delta^n(\mathbf{x}_i)$ , a neural decision tree becomes a hard decision tree with oblique splits. Intuitively, from a NDF evaluation perspective, we successively choose the best path through the tree in a greedy fashion, node after node. Thus, we refer to this model as a Greedy Neural Decision Forest (GNDF).

On the one hand, in order to evaluate a NDF that is composed of  $T$  trees, we have to evaluate the probability to reach each leaf node of each tree. As a consequence, each split node has to be evaluated, thus the complexity of evaluating a given input of dimension  $k$  with NDF is  $T.k.(2^{\mathcal{D}+1} - 1)$ , *i.e.* exponential in the tree depth  $\mathcal{D}$ . By doing so, we essentially lose the runtime advantage of using ensemble of decision trees for prediction. In case of a GNDF, on the other hand, only a single, locally “best” path through the  $T$  trees has to be evaluated. Hence, its complexity is equal to  $T.k.\mathcal{D}$ , *i.e.* linear in  $\mathcal{D}$ .

## 6.2 An efficient Neural Decision Forest training algorithm

The authors in [48] suggest using a two-step iterative optimization scheme. Given a training example  $\mathbf{x}_i$ , a forward pass through the trees provides the activations  $d^n(\mathbf{x}_i)$  and probabilities  $\mu^n(\mathbf{x}_i)$ . The error is then backpropagated from the leaves up to the root of the trees. Then, after a number of epochs, the leaf probabilities are updated following a convex optimization scheme. This leaf prediction update may however be quite costly in terms of training time. It also involves additional hyperparameters and requires the use of all the data, thus it can not be performed online.

To circumvent those issues, we propose an alternative training procedure for NDF that involves fixed prediction nodes. First, we randomly initialize a number of trees (e.g. with a fixed depth  $\mathcal{D}$ , randomly generated feature weights  $\beta_j^n$  and thresholds  $\theta^n$  for each node).

In order to maximize the information gain w.r.t. class label assignment, leaf nodes of each individual tree should contain pure distributions. Following this idea, we assign random predictions to the tree leaves, that will be left unchanged during the whole training process. Thus, in case of a classification task, a class is randomly selected and assigned to each leaf node. For regression, the predictions are initialized with randomly sampled values in the range of the training ground truth values. Note that, depending on the distribution of these values, the prediction can be randomly sampled from specific distributions (e.g. uniform or gaussian).

Hence, only the split node parameters are updated at each SGD iteration. Note that such setting is not only intuitive, but it allows to perform the training faster and in a fully online fashion. It also offer the advantage to reduce the number of hyperparameters. However, in case of a classification task, the depth of the trees has to be set accordingly to the number of classes  $\mathcal{C}$ , so that each class is represented at least once among the leaf nodes. It is possible to show that setting  $\mathcal{D}$  according to Equation (6.3) provides a probability higher than 0.99 for this condition to be realized (See Appendix A). Furthermore, we also prove that this lower bound depth grows as the logarithm of the number of classes  $\mathcal{C}$ .

$$\mathcal{D} > \mathcal{D}_0 = \frac{1}{\ln(2)} \ln\left(\frac{\ln(1 - (1 - 0.99)^{1/\mathcal{C}})}{\ln(1 - 1/\mathcal{C})}\right) \quad (6.3)$$

SGD or mini-batch updates are then applied until a specific number of epochs through the whole training set are completed. SGD updates can be applied to each tree in parallel. After training is completed we convert the soft trees back to hard decision trees by applying  $d^n \rightarrow \delta^n$  for every node  $n$  to “convert” the NDF to a GNDF. As it will be shown in the experiments, this allows a faster evaluation at no expense in terms of prediction accuracy. Moreover, it also provides a deterministic prediction, as opposed to iteratively sampling the probabilistic split nodes a number of times. The whole training procedure is summarized in Algorithm 4.

---

**Algorithm 4** Efficient\_GNDF\_training

---

**Input:** data matrix  $\mathbf{X} = \{x_{i,j}\}_{i=1,\dots,N,j=1,\dots,k}$ , tree number  $T$ , tree depth  $\mathcal{D}$ , number of epochs  $E$ , learning rate  $\alpha$

**for**  $t = 1, \dots, T$  **do**

    initialize tree  $t$  with root node  $r_t$  and depth  $\mathcal{D}$  random split parameters  $\{\beta_i^n\}_{i=1,\dots,k}, \theta^n$  for each split node  $n$  and a random class prediction assigned to each leaf node

**end for**

**for**  $ep = 1 \dots E$  **do**

**for**  $ex = 1 \dots N$  **do**

        pick an example  $i$  at random from the training set

$\epsilon_{\text{feat}}(\mathbf{x}_i) = 0 \triangleright$  Recall that  $\epsilon_{\text{feat}}$  is the backpropagation delta to the feature level

**for**  $t = 1, \dots, T$  **do**

            Apply forward pass through tree  $t$  to compute probabilities  $\mu^n$  and activation  $d^n$

$\epsilon_{\text{feat}}(\mathbf{x}_i) = \epsilon_{\text{feat}}(\mathbf{x}_i) + \text{NDF\_backprop}(\mathbf{x}_i, r_t, \alpha)$

**end for**

        Optionally backpropagate error  $\epsilon_{\text{feat}}(\mathbf{x}_i)$  and update CNN weights

**end for**

**end for**

**for**  $t = 1, \dots, T$  **do**

**for all** node  $n$  in tree  $t$  **do**

$d^n \rightarrow \delta^n$

**end for**

**end for**

---

## 6.3 Applications

Below we present some applications of the proposed simplified NDF training procedure and greedy evaluation of NDF for real-time processing. More specifically, we present an application of GNDF for categorical FER (Section 6.3.1), with a focus on hyperparame-

ter setting (Section 6.3.1.1), learning (deep) texture representations with GNDF (Section 6.3.1.2) and real-time capabilities (Section 6.3.1.3). We also show that GNDF can be used in a cascaded regression framework to perform feature point alignment (Section 6.3.2).

### 6.3.1 GNDF for FER

In this section, we present an application of GNDF for categorical FER. For that matter, we use geometric features (point distances  $\phi^{(1)}$  - see Section 3.2.2) along with texture representations (Section 3.2.4.2) learned through a CNN architecture (Section 3.3.3.3). For the benchmarks of Section 6.3.1.1 we use geometric representations only, principally for evaluation speed reasons. For the deep learned models presented in Section 6.3.1.2, we use the deep architecture illustrated on Figure 6.1, which involves a 2-layers CNN. The first CNN layer (CNN-1) is applied on  $48 \times 48$  face crops and is composed of 40  $5 \times 5$  filters. The output feature maps are max-pooled and fed to the second layer that is composed of 40  $3 \times 3$  filters. The output feature maps (40  $10 \times 10$  feature maps for the second layer (CNN-2)) are concatenated and serve as input for the NDF classifier. Alternatively, both layers are used as the input for the prediction stage (Network connexions thus form a directed acyclic graph, hence we refer to that model as DAG-CNN). For each dataset, we augment the training data to learn CNN features by generating flipped and rotated versions (with angles randomly sampled within the interval  $[-15^\circ, 15^\circ]$ ) of the images and of the feature points.

#### 6.3.1.1 Varying tree depth and number of trees

First and foremost, we report accuracies obtained by training GNDFs with 10 trees of varying depth, with the same number of updates, slope and learning rate values as above. We run the experiment 10 times and report the accuracy (average and standard deviation) on Figure 6.2. As one can see, the depth  $\mathcal{D}$  of the trees is not a critical hyperparameter to set. However, classification accuracy is lower for very shallow trees (e.g.  $\mathcal{D} \leq \mathcal{D}_0 \approx 5$ ), as a number of expression classes may not be covered by some individual trees. For these benchmarks,  $\mathcal{D} = 6$  seems to be a good compromise.

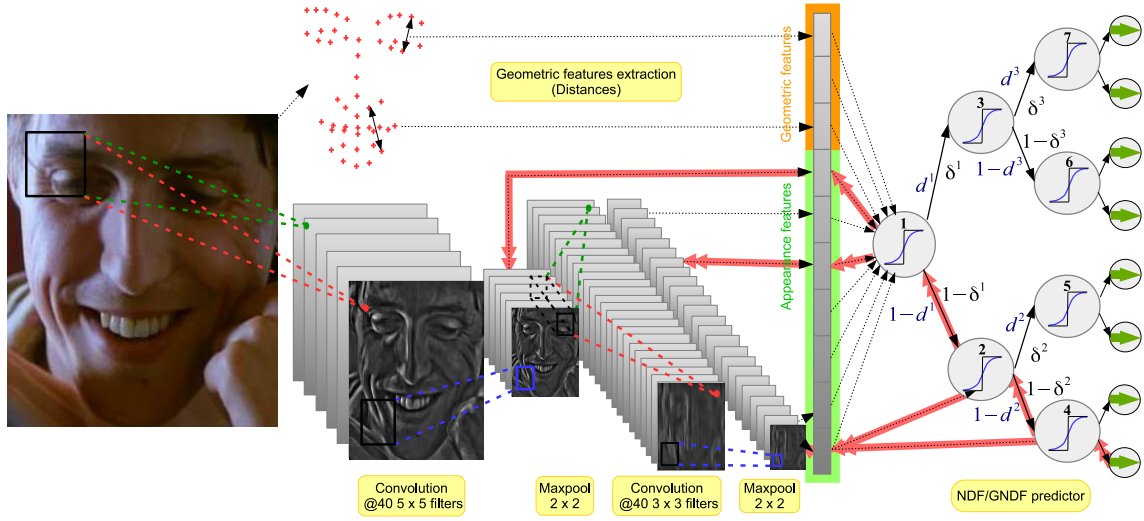


Figure 6.1: Deep architecture involving geometric and CNN features with GNDF/NDF predictor for FER. For each node  $n$  of the trees, NDF successive split nodes “dispatch” the data through the tree according to the activation  $d^n$  (right subtree) and  $1 - d^n$  (left subtree) of a neuron layer. Its corresponding GNDF takes hard decisions ( $\delta^n$  or  $1 - \delta^n$ ) upon oblique hyperplanes, which significantly decreases the evaluation runtime. The red arrows indicate (some of) the gradient backpropagation routes through the deep network, for multiple architectures: CNN-1 (the output of the first convolutional layer feeds the prediction pipeline), CNN-2 (same for the second layer) and DAG-CNN (both layers are used for prediction). Best viewed in color.

We then report accuracies obtained by training GNDFs with geometric features only and a fixed depth of  $\mathcal{D} = 6$ . We compare our results directly with the most common RF variations that are described in Section 3.3.4.2.2: RF and ERT- $k$  (with  $k = 50$ ). We also compare with NDF as well as with results obtained by sampling from the probabilistic trees (NDF-sample, mean and standard deviation over 5 samples). Figure 6.3 shows the accuracy evolution as a function of the number of SGD updates.

First, one can see that NDF, NDF-Sample and GNDF yield equivalent –if not better– results than the classical batch RF induction (RF and ERT-50), for  $T = 5, 20, 100$  trees. This is a promising result as other methods for online RF training such as the ones in [50, 73] are generally less efficient than the batch training procedure. Moreover, we also observe that the accuracies of GNDF and NDF-Sample lie a bit further than that of NDF

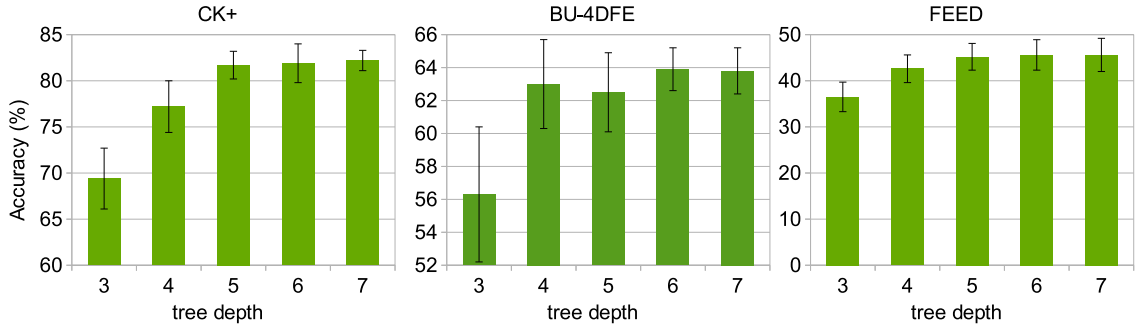


Figure 6.2: Accuracy as a function of tree depth

with small amounts of trees (e.g.  $T = 5, 20$ , not necessarily favouring NDF-see the graph with  $T = 5$  on FEED database). Overall, the accuracies of GNDF and NDF-Sample are equivalent or better than that of NDF (except on CK+ with  $T = 5$  trees) with a much lower runtime. This is due to the fact that the decorrelation induced by increasing the randomness of the evaluation may compensate for the loss caused by the greedy approximation/sampling procedure. Furthermore, the variance in accuracy that results from the random node sampling is quite important even after a number of SGD updates. Thus, GNDF appears as a more reliable predictor than NDF-sample, as the evaluation is a deterministic process. This proves the interest of GNDF as a standalone classifier, let alone its use for learning deep representations.

### 6.3.1.2 Learning deep representations

Table 6.1 displays the accuracy of NDF and GNDF on different datasets when used as shallow predictors upon geometric features, as well as their use inside a deep learning framework for learning texture representations. Note that for BU-4DFE and FEED databases, state-of-the-art methods usually report results for video classification on a restricted number of subjects or using an easier evaluation protocol [1], whereas we use all the subjects in our experiments. Thus, for comparison purposes, we provide a baseline for RF using geometric features alongside HOG (geo+HOG) in addition to a comparison with prior works that still give an insight of the relative difficulties of the classification task on those databases.

We observe that, as stated in [44], using a combination of geometric and deep learned

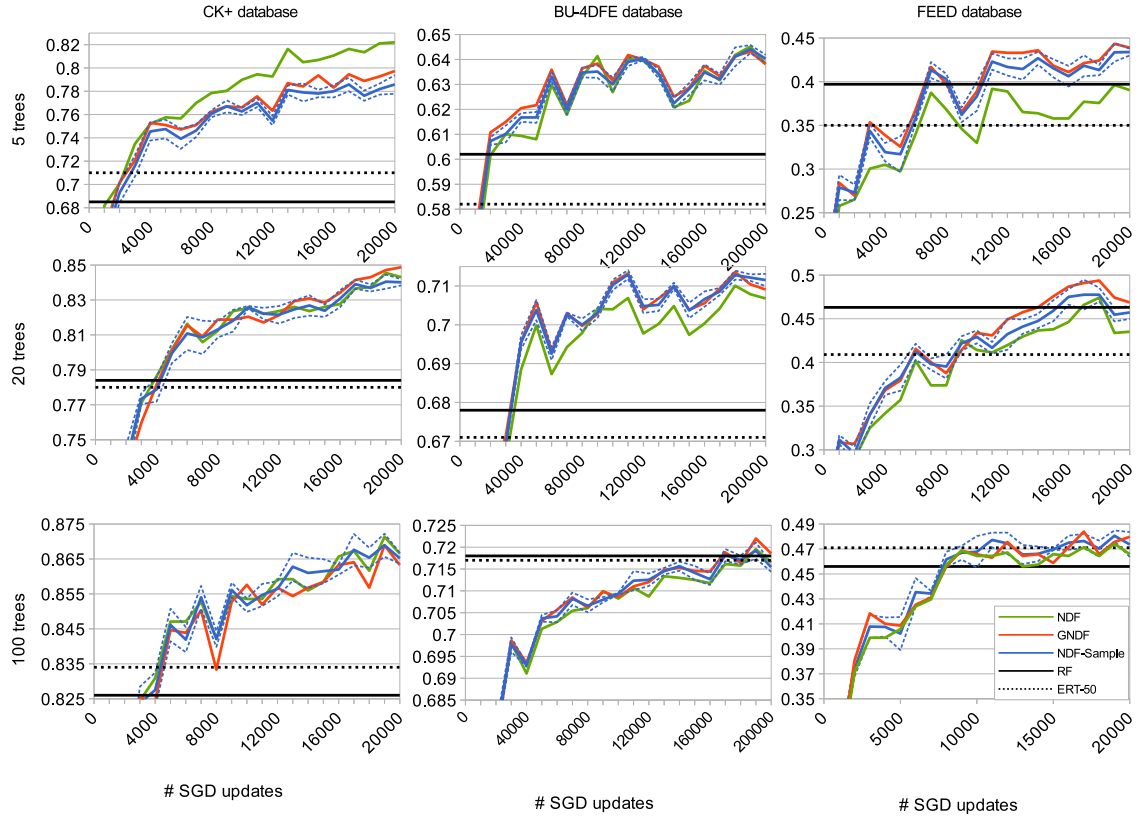


Figure 6.3: Accuracy comparison with standard RF variants for 5 (top row) 20 (middle row) and 100 trees (bottom row). The blue dashed lines indicate the mean  $\pm$  std of the NDF-Sample accuracy. Best viewed in color.

texture representations provides the best results by a significant margin. Results for geo+CNN-2 are better than the geometric features alone. However, the number of training examples may not be sufficient to learn the high number of parameters that a multi-layer CNN is composed of, even with data augmentation. Nevertheless, the use of DAG-CNN allows to slightly increase the accuracy by providing access to the outputs of the two CNN layers. Also, here again GNDF provides performances very close to those of NDF. Finally, when used inside a deep learning framework, results for NDF and GNDF are also above the RF baseline as well as state-of-the-art methods.

Table 6.1: Accuracies obtained with shallow/deep NDF/GNDF predictors. <sup>†</sup>: a correct classification is acknowledged if the ground truth label matches one of the two top proposals

	CK+ database		BU-4DFE database		FEED database	
features	NDF	GNDF	NDF	GNDF	NDF	GNDF
geo	87.3	87.3	71.6	71.8	46.6	47.9
geo+CNN-2	89.9	90.0	73.1	72.9	50.1	50.1
geo+DAG-CNN	<b>92.2</b>	<b>92.2</b>	<b>74.0</b>	<b>74.0</b>	51.8	52.0
LBP/SVM [78]	88.9		-		-	
RF/SVM [1]	-		73.1 <sup>†</sup>		<b>53.7<sup>†</sup></b>	
MRF/DBN [68]	90.1		-		-	
geo+HOG/RF	91.1		72.8		50.3	

### 6.3.1.3 Runtime evaluation

As stated in Section 6.1, the complexity of evaluating a GNDF is a linear function of the tree depth vs. exponential for NDF. Table 6.2 shows the runtime of NDF and GNDF for evaluating one expression frame with one tree of depth 6. GNDF is about 30 times faster than NDF and, as such, allows a real-time evaluation at more than 30 fps without parallelization with 100 trees, even when using high-dimensional DAG-CNN features. This, however, does not take into account the forward pass through the CNN that can be subject to a parallel implementation or dedicated GPU acceleration.

Table 6.2: Runtime evaluation (ms) for one example with one tree

features	#dimensions	NDF	GNDF
geo	1176	0.17	< 0.01
CNN-1	19360	3.05	0.11
CNN-2	4000	0.65	0.03
DAG-CNN	23360	3.76	0.13
geo+CNN-1	20536	3.97	0.12
geo+CNN-2	5176	0.90	0.03
geo+DAG-CNN	24536	3.99	0.16



As for training time, it took about 30 minutes to learn 100 trees of a shallow NDF on a Intel Core I7-4770 using a loosely-parallelized C++/OpenCV environment. Learning deep NDF with 100 trees and geometric features plus the proposed DAG-CNN architecture took about half a day to complete with 40 epochs on CK+.

### **6.3.2 Feature point alignment with a cascaded semi-parametric deep greedy neural decision forest**

As stated in Section 3.1, face alignment is a crucial step for face analysis in general, and expression recognition in particular. To tackle this issue, Section 6.3.2.1 focuses on learning and evaluating a face alignment system on still images. It consists in aligning a set of facial feature points (e.g. eyes or mouth corner, nose tip), which usually form either a 68-points markup (inner+cheeks points), or a 51/49-points markup with only the inner points. To this end, modern methods first rely on face detection to provide an initial bounding box, then apply a cascaded regression framework to infer the displacements of the points of a mean shape centered inside the face bounding box. The authors in [101] provide a comprehensive, yet compact survey of the most successful recent methods for face alignment. In this section, we propose to adapt the proposed GNDF framework to the problem of face alignment. *De facto*, GNDF is a well-suited predictor for that task, since the evaluation is as fast as a RF and since it benefits from differential training (hence allowing the learning of deep representations) with backpropagation and SGD (allowing efficient data augmentation schemes).

We also demonstrate in Section 6.3.2.2 that our method provides state-of-the-art accuracy on multiple benchmarks for face alignment on still images. Finally, in Section 6.3.2.3 we propose a few practical tricks to robustly perform feature point alignment on video sequences, as well as an evaluation of the proposed system.

#### **6.3.2.1 Face alignment on still images**

In this section, we present the proposed CSP-dGNDF method for face alignment, which is outlined on Figure 6.4. The proposed system mainly consists in two cascaded regressions

pipelines. The first one consists in a regression in the space of a parametric shape model 6.3.2.1.1. The subsequent one is an explicit cascaded regression 6.3.2.1.2. In particular, at each stage of those two cascades a displacement (either parametric or explicit) is regressed *via* a deep GNDF, which consists in a predictor (dGNDF 6.3.2.1.3) that is associated to a single neuron layer for dimensionality reduction 6.3.2.1.4. In particular, we discuss a few implementation choices for reducing the overfitting at each stage of the cascade (Section 6.3.2.1.5). Last but not least, we detail how we put all the pieces together inside the cascaded regression framework (Section 6.3.2.1.6).

**6.3.2.1.1 Parametric shape model.** Two constraints that may arise when training a NDF for the purpose of multi-output regression is (a) covering the output regression space by filling the leaf node predictions in a somewhat exhaustive manner and (b) limiting the number of nodes, which is a function of (the exponential of) the tree depth by the number of trees. Given those requirements, it is easy to see that trying to directly predict the shape displacement is a bad idea, as the output space is high-dimensional (dim.  $51 \times 2$ ) and the displacement value ranges can be important. For that matter, we employ an explicit shape parametrization in the first stages of the cascade, which is a classical setup for face alignment [17, 16, 3]. More specifically, shape  $\mathbf{s}$  is defined as:

$$\mathbf{s}(\mathbf{p}) = \alpha R(\gamma)(\mathbf{s}_0 + \phi \mathbf{g}) + t \quad (6.4)$$

Where  $\alpha = (\alpha_x, \alpha_y)$  is a scaling parameter,  $R$  is a  $2D$  rotation matrix parametrized by angle  $\gamma$ , and  $t = (t_x, t_y)$  is a translation parameter. Those are the rigid parameters of the transformation.  $\mathbf{s}_0$  is the mean shape and vector  $\mathbf{g}$  describes the non-rigid deformation of the shape in the space of the Point Distribution Model (PDM)  $\phi$ , as it was introduced in the seminal work of Cootes *et al* [16]. The vector of parameters is thus defined as  $\mathbf{p} = (\alpha_x, \alpha_y, \gamma, t_x, t_y, g_1, \dots, g_m) \in \mathbb{R}^{m+5}$ . In our experiments, we set  $m = 15$ , making a 20-dimensional parametrization of the shape.

Prior to constructing the PDM, we thus have to first detect the face, e.g. using OpenCV Viola and Jones algorithm [92]. The retrieved region of interest is thus resized to a  $200 \times 200$  window. For each shape, we then perform Procrustes analysis to remove the

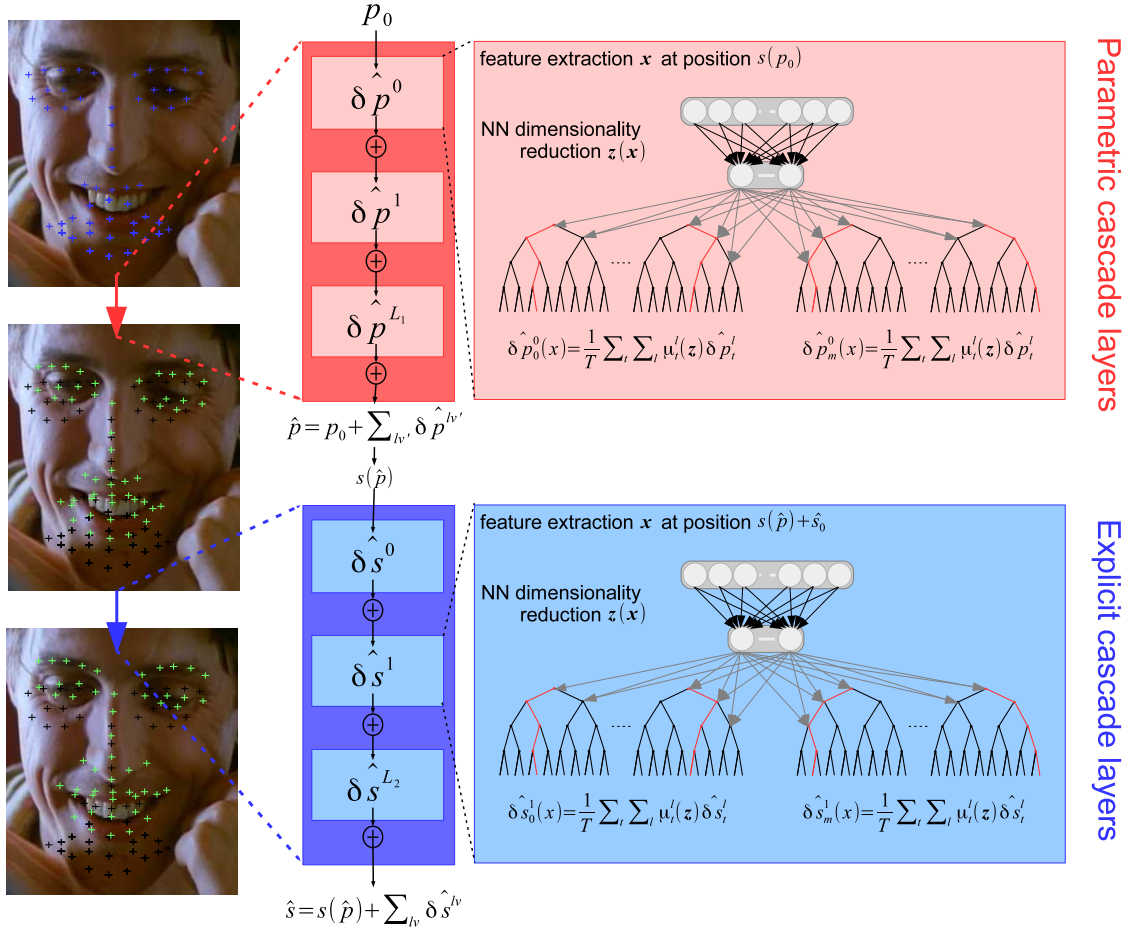


Figure 6.4: Flowchart of the Cascaded Semi-Parametric deep GNDF (CSP-dGNDF) alignment method. In the first stages of the cascade, a parametric shape is regressed using deep GNDF predictors. The alignment is performed in a fully online fashion, by presenting the augmented examples sequentially. The input vector for each stage of the cascade is a reduced concatenation of local SIFT descriptors computed around the current landmark estimates. In the later stages, more fine-grained explicit deformations of the shape are regressed.

rigid component. Thus, we generate the PDM matrix  $\phi$  using PCA on the rigidly-aligned shapes. After that, we apply 100 Gauss-Newton iterations to retrieve the parameter vector  $\mathbf{p}_i^*$  corresponding to each image  $i$  with ground truth shape  $\mathbf{s}_i^*$ . Each iteration is defined as  $\mathbf{p}_i \leftarrow \mathbf{p}_i + (J(\mathbf{p}_i)^t J(\mathbf{p}_i))^{-1} J(\mathbf{p}_i)^t (\mathbf{s}_i^* - \mathbf{s}(\mathbf{p}_i))$ , with  $J(\mathbf{p}_i)$  being the Jacobian of  $\mathbf{s}(\mathbf{p}_i)$ .

**6.3.2.1.2 Explicit shape model.** Contrary to parametric layers (see Paragraph 6.3.2.1.1), an explicit layer aims at directly predicting the displacements of the facial feature points. The output of such layer is thus defined as:

$$\delta \mathbf{s} = (\delta s_x^1, \dots, \delta s_x^{51}, \delta s_y^1, \dots, \delta s_y^{51}) \in \mathbb{R}^{51 \times 2} \quad (6.5)$$

However, as stated above, the output of an explicit layer is high-dimensional. Thus, in order to cover the range of those values with the leave nodes of a GNDF predictor that contains a moderate number of trees (to keep the runtime low), one shall restrict the ranges of the prediction values. Fortunately, the ranges of the deltas between a current predicted value and the ground truth feature point locations becomes smaller and smaller as cascade layers are stacked, allowing the use of explicit regression layers for the latter stages of the cascade.

**6.3.2.1.3 Regression with NDF predictors.** As illustrated on Figure 6.4, we propose a framework for multi-output regression upon an input feature vector  $\mathbf{z}_i$  of either the parameter update vector  $\delta \mathbf{p}$  or, in the case of an explicit layer, the displacement vector  $\delta \mathbf{s}$ .

In the case of a parametric layer, we first estimate the mean  $\bar{\delta}_k$  and standard variation  $\sigma_k$  of the delta between the initial position in parameter space  $\delta \mathbf{p}^0(k)$  (that corresponds to the mean shape in the shape space, for the first level of the cascade) and the ground truth objective  $\delta \mathbf{p}^*(k)$ , for each parameter  $k$ . We then generate  $T$  single-objective trees for each parameter  $k$  by assigning each leaf node  $l$  a single prediction  $\hat{\delta \mathbf{p}}^l \sim \mathcal{N}(\bar{\delta}_k, \sigma_k)$ . During training, all the  $T \times (m + 5)$  parameters are optimized jointly by updating each tree node with Equations 3.39, 3.42, 3.43 using a parameter-dependant learning rate  $\alpha_k = \frac{\alpha_0}{\sigma_k}$  in order to take into account the discrepancies in the dynamics of the different dimensions in the parameter space. The same holds true for an explicit layer by replacing the parameter

updates by the feature point displacements.

As in the classification case (Section 6.2) the tree depth has to be chosen carefully in order to ensure a minimal “resolution” in term of leaf predictions. For that matter, we provide in Appendix A.2 a proof that, in the regression case with constant leaf prediction initialized from a gaussian distribution, we have a sufficient condition to have each value  $y \in [\bar{\delta}_k - \sigma_k, \bar{\delta}_k + \sigma_k]$  close to at least one leaf node prediction  $y_l$  (in the sense that  $|y - y_l| < \epsilon$ , with a probability superior to  $1 - \epsilon'$ ). We essentially show that this condition is satisfied if  $\mathcal{D} > \mathcal{D}_0$ , with

$$\mathcal{D}_0 = \frac{1}{\ln(2)} \ln \frac{\ln(1 - (1 - \epsilon')^{\frac{1}{2\sigma_k}})}{\ln(1 - \frac{2\epsilon}{\sqrt{2\pi}\sigma_k} e^{-\frac{(\sigma_k + \epsilon)^2}{2\sigma_k^2}})} \quad (6.6)$$

In our case, using trees of depth 8 ensure that this condition is satisfied with  $\epsilon' = 0.99$  and  $\epsilon_k = \sigma_k/10$  for all the ranges  $\sigma_k$  of the model parameters (which experimentally vary from 10 to 0.1).

**6.3.2.1.4 Feature extraction and dimensionality reduction.** We use baseline SIFT features (Section 3.2.4.1) for their robustness and fast extraction speed. The 9-dimensional orientation bins and magnitude maps are generated and SIFTs are extracted from those feature maps with  $4 \times 4$  non-overlapping cells in a fixed size window ( $40 \times 40$ ,  $36 \times 36$ ,  $32 \times 32$  and  $28 \times 28$  for the first, second, third and fourth cascade layers, respectively) and concatenated to form 6528-dimensional descriptors  $\mathbf{x}_i$  in the case of a 51-landmark shape (8704 for a 68-landmark one).

Learning NDFs with such high-dimensional descriptors would be quite slow in terms of memory and training time, let alone overfitting issues. For those reasons, as in [98] we perform dimensionality reduction. However, as stated above, as NDF are differential classifiers, we can use a single neuron layer to perform dimensionality reduction and learn the weights of that layers in a single, top-down, supervised training pass (as opposed to, e.g., applying PCA beforehand [98]). We thus plug the descriptors  $\mathbf{x}_i$  into a single neuron layer with 500 output units with an hyperbolic tangent activation function. Those units’ weights are initialized in the range of  $[-0.01, 0.01]$  and, during training, their weights are

updated by applying Equation 6.7 for all nodes of all trees. The backpropagated error corresponding to the  $j^{th}$  component corresponding to an example  $i$   $\epsilon_{i,j}^{feat}$  is thus:

$$\epsilon_{i,j}^{feat} = \frac{1}{T \times (m+5)} \sum_{t=1}^{T \times (m+5)} \sum_{n \in \mathcal{N}(t)} \mu^n(\mathbf{z}_i) \beta_j d^n(\mathbf{z}_i) (1 - d^n(\mathbf{z}_i)) (\epsilon_+^n(\mathbf{z}_i) - \epsilon_-^n(\mathbf{z}_i)) \quad (6.7)$$

Additionally, during training the weights of the neurons are regularized using  $\mathcal{L}_1$ -penalty. We use the truncated gradient algorithm introduced in [51] to enforce sparsity among the neurons' weights. In practice, this reduces the number of non-zero coefficients down to 10%, thus allowing fast dimensionality reduction while fully preserving the prediction accuracy.

**6.3.2.1.5 Avoiding overfitting.** Within the frame of a cascaded landmark alignment [98], it is crucial that each stage of the cascade (*i.e.*, in our case, each NDF predictor) does not overfit on the training data so that the residual deltas  $\delta \mathbf{p}_i^* - \hat{\delta \mathbf{p}}_i$  (for a parametric layer) do not shrink too much after one or two stages. Even though the NDF predictors embraces a whole lot of parameters ( $20 \times 25 \times 255 \times 500 = 63750000$  parameters for a 20 parameters model and  $T = 25$  trees of depth 8!), three mechanisms limit overfitting in practice:

- We use dimensionality reduction to limit the number of parameters (see Section 6.3.2.1.4).
- We use early stopping by training each NDF predictor with a restricted number of SGD updates. Moreover, as the proposed NDF training framework is fully online, for each example, we generate random perturbations that are randomly sampled within the variation range for that parameter (for scaling and translation parameters only).
- The switching from a NDF model to a GNDF may possibly introduce some noise in the predictions, which will be compensated in the further stages of the cascade, while significantly reducing the evaluation runtime (exponential to linear function of the tree depth).

**6.3.2.1.6 Cascaded alignment.** As it is somewhat classical in the landmark alignment literature, we propose a cascaded alignment procedure. However, in our work, we use a semi-parametric shape model, in which a shape prediction is provided as the sum of multiple displacements in parameter space, starting from an initial guess (usually defined as the mean shape parameterization). Then, in the latter stages, the displacement is fine-tuned using explicit layers. The final prediction can thus be written as:

$$\begin{cases} \hat{\mathbf{s}} = s(\hat{\mathbf{p}}) + \sum_{lv'} \hat{\delta \mathbf{s}}^{(lv')} \\ \hat{\mathbf{p}} = \mathbf{p}_0 + \sum_{lv} \hat{\delta \mathbf{p}}^{(lv)} \end{cases} \quad (6.8)$$

This allows (a) a constrained shape regression that is, theoretically speaking, more stable than a fully explicit method, and (b) a flexible alignment procedure that captures the fine-grained feature point displacements. After each step, the shape is updated using Equation (6.8) and the SIFT descriptors can be computed using the current feature point location. Those descriptors are then used to feed the next level of the cascade. As stated before, each cascade layer  $lv$  consists in (a) a feature extraction step, (b) a separate NN for dimensionality reduction and (c) a NDF predictor that is evaluated in a greedy fashion for faster processing. The steps for training a CSP-dGNDF cascade are summarized in Algorithm 5.

Note that, as the parameters for all the training examples converge towards the ground truth values, the parameter ranges standard deviation  $\sigma_k^{lv}$  decreases accordingly. Recall that, in the case of a parametric layer, the leaf predictions are generated as  $\hat{\delta \mathbf{p}}^l \sim \mathcal{N}(\bar{\delta}_k, \sigma_k)$ . Thus, as the number of stages increases, the trees are automatically constrained to cast more precise predictions in a much smaller variation range, hence a coarse-to-fine alignment.

### 6.3.2.2 Evaluation

For evaluating the proposed CSP-dGNDF cascade, we train our models on a concatenation on the training partition of the LFPW and HELEN databases, as well as images from the AFW database. The total training corpus contains 3148 training images. For each of these training images, we use the provided ground truth bounding boxes, as it is com-

---

**Algorithm 5** Learning a CSP-dGNDF

---

**Input:** Images  $\mathcal{I}_i$  with ground truth shapes  $\mathbf{s}_i^*$ , tree number  $T$  for each parameter, tree depth  $\mathcal{D}$ , number of updates  $N_u$ , base learning rate  $\alpha$ , number of PDM dimensions  $m$ , number of parametric cascade stages  $L_1$  and explicit cascade stages  $L_2$

Perform Procrustes analysis on  $\mathbf{s}_i^*$  and GN iterations to find ground truth  $\mathbf{p}_i^*$

Initialize parameters to mean values  $\hat{\mathbf{p}}_i \leftarrow \bar{\mathbf{p}}$

**for**  $lv = 1, \dots, L_1$  **do**

    Compute parameter ranges  $(\bar{\delta}_k^{lv}, \sigma_k^{lv})$ , initialize  $NN^{lv}$  and  $NDF^{lv}$

**for**  $up = 1, \dots, N_u$  **do**

        Draw an example  $i$  and augment scaling and translation parameters

**for**  $lv'' = 1, \dots, lv - 1$  **do**

$\hat{\mathbf{p}}_i \leftarrow \hat{\mathbf{p}}_i + GNDF^{lv''}(\mathbf{z}_i)$

            Update shape  $\mathbf{s}(\hat{\mathbf{p}}_i)$ , descriptors  $\mathbf{x}_i$  and reduced vector  $\mathbf{z}_i == NN^{lv''}(\mathbf{x}_i)$

**end for**

        Forward pass through  $NDF^{lv}(\mathbf{z}_i)$  to compute node probabilities

        Backpropagate error through  $NDF^{lv}$  and  $NN^{lv}$

**end for**

**end for**

**for**  $lv' = 1, \dots, L_2$  **do**

    Compute displacement ranges  $(\bar{\delta}_k^{lv'}, \sigma_k^{lv'})$ , initialize  $NN^{lv'}$  and  $NDF^{lv'}$

**for**  $up = 1, \dots, N_u$  **do**

        Draw an example  $i$  and augment scaling and translation parameters

**for**  $lv'' = 1, \dots, L_1$  **do**

$\hat{\mathbf{p}}_i \leftarrow \hat{\mathbf{p}}_i + GNDF^{lv''}(\mathbf{z}_i)$

            Update shape  $\mathbf{s}(\hat{\mathbf{p}}_i)$ , descriptor  $\mathbf{x}_i$  and reduced vector  $\mathbf{z}_i == NN^{lv}(\mathbf{x}_i)$

**end for**

**for**  $lv'' = 1, \dots, lv - 1'$  **do**

$\hat{\mathbf{s}}_i \leftarrow \hat{\mathbf{s}}_i + GNDF^{lv''}(\mathbf{z}_i)$

            Update descriptor  $\mathbf{x}_i$  and reduced vector  $\mathbf{z}_i == NN^{lv''}(\mathbf{x}_i)$

**end for**

        Forward pass through  $NDF^{lv}(\mathbf{z}_i)$  to compute node probabilities

        Backpropagate error through  $NDF^{lv'}$  and  $NN^{lv'}$

**end for**

**end for**

---



monly done in the literature. However, in practice the bounding boxes can be generated using OpenCV Viola and Jones face detector [92].

We crop each face image according to the corresponding bounding box, add horizontal and vertical margins equal to a third of the bounding box width and height, respectively. Then we resize the crops to a  $200 \times 200$  scale, and proceed to initialize the mean shape from the mean model parameters. Then we perturb the mean shape position  $(t_x, t_y \sim \mathcal{N}(0, 10))$  and scale  $(s_x, s_y \sim \mathcal{N}(0, 0.1))$ . We then train a 4-level cascade, where each layer contains a single neuron layer with 6528-dimensional input (8704 for a 68-landmark shape) and 500-dimensional output, and a CSP-dGNDF with 25 trees per parameter, hence a total of 500 trees of depth 8.

We report results on the test partition of LFPW (224 images), HELEN (330 images) as well as the challenging IBUG database (135 images). To do that, we align the feature points from the mean shape initialized using the exact same setting, and measure, for each test example, the average point-to-point Euclidean distance. As it is done in the literature, this distance is normalized by the inter-pupil distance. Figure 6.5 shows the cumulative error distribution curves, for 1, 2 and 3 parametric stages cascade, as well as a semi-parametric one, with the last layer being an explicit one (with  $n_2$  normalization) on the three databases and with both 51 and 68-landmarks markups. As one can see, on both LFPW and HELEN databases, the error is below 5% of the inter-pupil distance, which is very close to the human performance on that task, as stated in [101]. Also, the error is higher for the 68-landmarks markup, as the 17 landmarks located on the cheeks are subject to a greater appearance variability. The alignment is also subject to higher errors on IBUG, as the database contains extreme poses variations as well as a number of self-occlusions. Finally, one can see that each cascade stage substantially increases the alignment accuracy. In particular, for every benchmark, the precision “deltas” between the gains provided by the third and forth layers are roughly equivalents. Thus, using explicit layers for the latter stages of the cascade allows to reduce the diminishing returns effect of stacking regression layers for alignment.

Table 6.3 shows a comparison of our results with results reported in the literature. As one can see, the accuracies reported for our 4-levels CSP-dGNDF are among the best

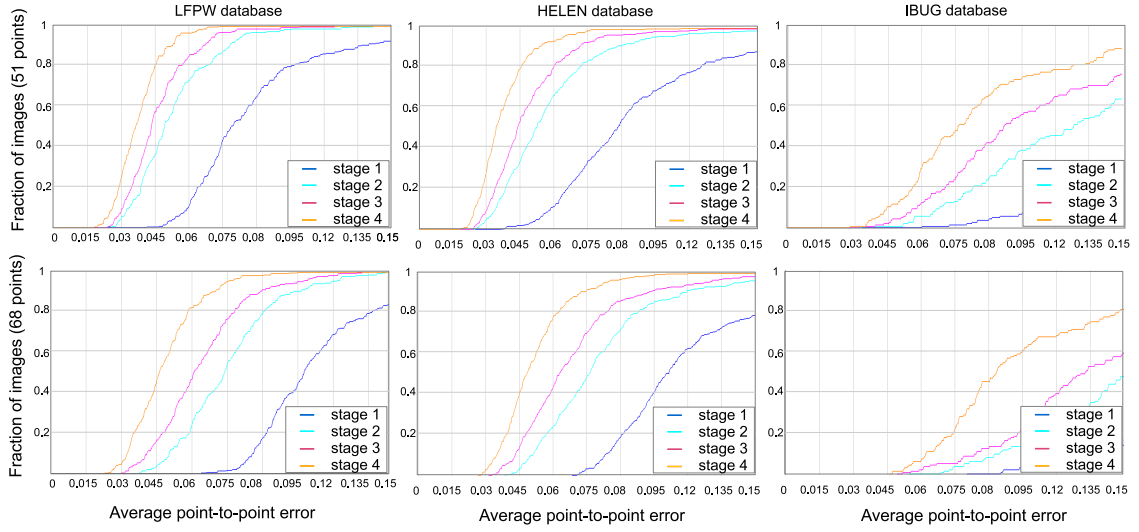


Figure 6.5: Cumulative error distribution curves on the LFPW and HELEN test partitions.

results in the literature (including recent approaches such as L21-Cascade (2016, [61]), PO-CR (2015, [88]) and GN-DPM (2014, [89])) on the three databases, for both 51 and 68-landmark markups. In addition to that, thanks to the greedy evaluation procedure that was introduced in Section 6.1, our method largely runs in real-time on an single Intel *I5* core.

Table 6.3: Comparison of normalized mean error

	LFPW		HELEN		IBUG	
method	51 pts	68 pts	51 pts	68 pts	51 pts	68 pts
SDM [98]	4.47	5.67	4.25	5.50	-	15.40
RCPR [12]	5.48	6.56	4.64	5.93	-	17.26
IFA [3]	6.12	-	5.86	-	-	-
DRMF [2]	4.40	5.80	4.60	5.80	-	19.79
CFAN [106]	-	5.44	-	5.53	-	-
L21-Cascade [61]	3.80	-	4.1	-	16.3	-
GN-DPM [89]	4.43	5.92	4.06	5.69	-	-
PO-CR [88]	4.08	-	3.90	-	-	-
CSP-dGNDF	<b>3.76</b>	<b>4.84</b>	<b>3.87</b>	<b>5.16</b>	<b>10.45</b>	<b>12.74</b>

Figure 6.6 shows qualitative alignment results respectively obtained by applying 1, 2, 3-levels parametric cascade, as well as 4-levels semi-parametric cascade (with 3 parametric layers and 1 explicit layer), on a subset of images for the LFPW test partition. The column on the right shows the ground truth landmark position for those images. The alignment quality is noticeably better for the semi-parametric cascade, which allows to more correctly fit fine-grained details such as the positions of the outer lip and eye corners. Moreover, the eyebrow landmarks seem to be closer to the ground truth labelling, illustrating how using a semi-parametric cascade allows to overcome one limitation of parametric cascades, *i.e.* the rigidity of the model which sometimes prevent from correctly fitting the landmarks in the case of specific facial expressions.

### 6.3.2.3 Feature point alignment on video

In order to perform feature point alignment on video, we propose a simple, yet efficient framework that is illustrated on Figure 6.7. It uses the proposed CSP-dGNDF framework, in conjunction with two “tricks” to respectively update the face bounding box and control the alignment quality throughout the sequence.

**6.3.2.3.1 Bounding box regeneration.** The problem of feature point alignment on video is analogue to that of aligning on still images, except that it is inefficient to perform re-detection of the face at each separate frame. Instead of that, what we do is that we use the predicted parameter and shape updates to predict a new bounding box that will be used as an initialization for the next frame. Formally, we define the bounding box for a video frame  $\mathcal{I}^n$  as a rectangle  $rect(\mathcal{I}^n) = (x_0^n, y_0^n, w^n, h^n)^T$  parametrized by  $(x_0, y_0)$  the coordinates of its top-left corner, and  $w$  and  $h$  respectively the width and height of the face bounding box. The face bounding box generated for frame  $n + 1$  is thus defined as:

$$rect(\mathcal{I}^{n+1}) = \frac{1}{2}((x_0^n + \delta t_x^n, y_0^n + \delta t_y^n, w^n + \delta \alpha_x^n, h^n + \delta \alpha_y^n)^T + f(\mathbf{s}^n)) \quad (6.9)$$

Where  $\delta \alpha_x^n$ ,  $\delta \alpha_y^n$ ,  $\delta t_x^n$  and  $\delta t_y^n$  are the regressed rigid parameter updates for frame  $n$  respectively for scaling and translation ( $x$  and  $y$  coordinates).  $f$  is a function that is estimated directly from the aligned shape for frame  $n$  using linear regression:



Figure 6.6: Examples of face alignment on still images from the LFPW database, with 1, 2, 3 levels parametric cascade, and 4 levels semi-parametric cascade.

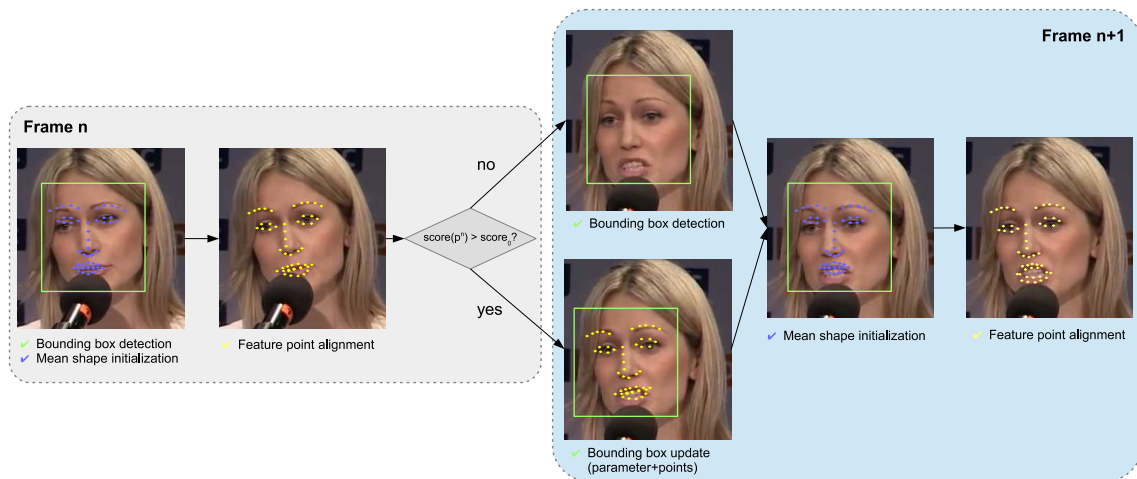


Figure 6.7: Outline of feature point tracking from video. First, the face is detected and a bounding box is generated accordingly. The mean shape is then initialized within the bounding box and the alignment is performed using the CSP-dGNDF method from that mean shape. A confidence score is thus evaluated. From this point, if that score is superior to some threshold the bounding box is updated using a combination of two techniques. Conversely, if the threshold is not reached we perform redetection to retrieve the face bounding box. The shape is then aligned from the mean shape centered within that bounding box.

$$f(\mathbf{s}^n) = \mathbf{A}\mathbf{s}^n + \mathbf{b} \quad (6.10)$$

Where  $\mathbf{A}$  and  $\mathbf{b}$  are respectively the  $4 \times 102$  and  $4 \times 1$  regression matrix and bias. These values are obtained by using least-squares regression on the whole dataset. Thus, the generated bounding box is obtained by averaging two estimates. The first of this terms is obtained by applying the rigid parameters to “follow” the motion of the face, and the second one is obtained by explicitly regressing the face rectangle from the shape using least square regression coefficients learned on the training set. In practice, we found this generation process to provide a bounding box estimate that is much more stable, which allows to more efficiently follow the deformation of the shape.

**6.3.2.3.2 Error case detection.** Another important concern for face alignment on video is to know exactly when the mesh is degenerating so that we can perform re-detection quickly. In order to do that, we define an alignment score defined as the average score over all the levels of the cascade.

$$score_i = 1 / \sum_{lv} \sum_j \sum_t (\delta p_{i,j,t}^{(lv)} - \hat{\delta p}_{i,j}^{(lv)})^2 \quad (6.11)$$

The score is defined as the average standard deviation of the values regressed by each tree for all rigid and non-rigid model parameters (averaged across all the landmarks). The rationale behind this is that the amount of agreement between the trees of each collection (each one corresponding to a shape model parameter) provides an information of how easily it is for the GNDF models at each layer to retrieve correct values for the parameters or displacements.

#### 6.3.2.4 Evaluation on the 300-W video challenge

We use the data from the 300-W video challenge data to evaluate our video alignment procedure for feature point detection and tracking on video sequences. More specifically, we use the framework outlined on Figure 6.7 to track the points for each video. Namely, in order to perform fully-automatic feature point alignment, for each frame, we generate a



new bounding box using the algorithm proposed in Paragraph 6.3.2.3.1. Moreover, we set a threshold of 0.05 on the score function proposed in Paragraph 6.3.2.3.2 to detect misalignment. In case the score falls below that threshold, the bounding box is regenerated using OpenCV Viola and Jones algorithm, then feature points are aligned from the generated bounding box. We then measure the average point-to-point alignment error (PPE), normalized by the inter-ocular distance between the retrieved points and the ground truth labels. In order to assess the validity of the proposed score function, we also measure the correlation coefficient (CC) between the score function and the PPE. In order to remove outlier data (that can be due to false detections from Viola and Jones face detector), we do not consider the frames for which the (normalized) PPE is above 1.

Using those settings, our system was able to correctly align 197844 out of the 218595 frames from the video corpus, making a recall rate of 90.5%. Out of those frames, the median PPE was 0.049 which is similar to the above discussed case of the aligning feature points on still images. The little drop in performance is likely to be due to the difficulty of the benchmark, with one third of the videos containing a lot of non-frontal head poses and partial occlusions. Furthermore, the techniques we use for bounding box regeneration are geared towards mimicking the method used to preprocess the images during the training step, *i.e.* aligning the feature points starting from an initial mean shape centered on Viola and Jones bounding box. Likewise, the methods used in this respect (bounding box regression from the feature points and update from model parameters) seem to perform quite well but can lead to some errors.

Last but not least, the CC between the score and distance to the ground truth is  $-0.48$ , which indicates a significant correlation between the alignment error and the score. Hence, this measure can be used to assess the alignment quality throughout the sequences, as well as to perform redetection of the face when the aligned shape drifts too much from the face. Figure 6.8 shows an example of plot containing both the (scaled) average PPE and the  $1 - \text{score}$  function observed on one sequence from the 300-W video challenge. One can see qualitatively how low score values indicate bad alignment cases. Also notice that we chose a somewhat “pessimistic” setting, which sometimes causes correctly aligned frames to have relatively low scores. Such setting was adopted in order to mini-

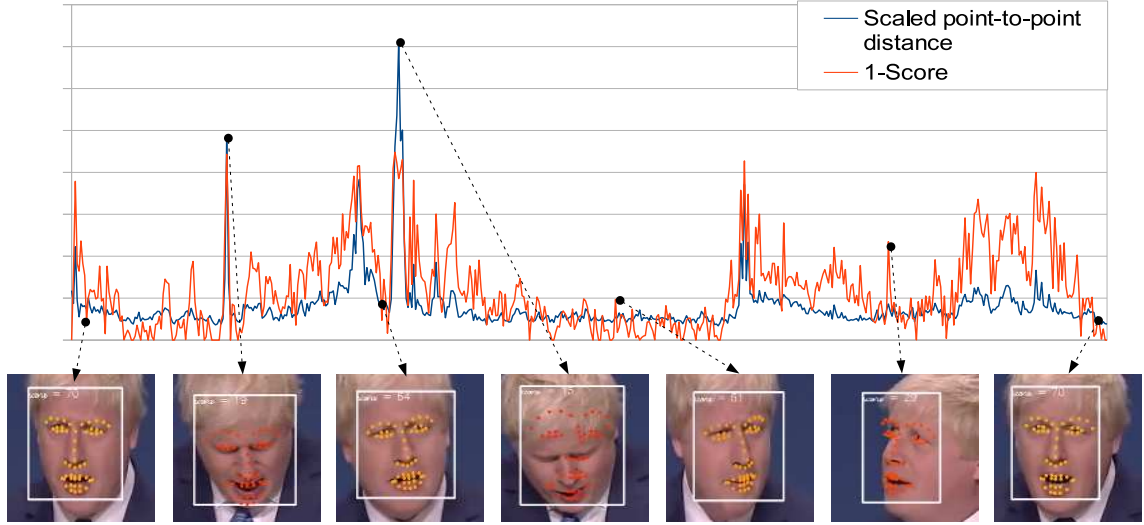


Figure 6.8: Example of an aligned sequence and correlation between the scaled average PPE and score function.

mize the false positive number -in our case, the number of badly aligned frames with high scores. The reason is that bad alignments can cause the model to drift from the face to the background, and take some time to recover, which may result in a lot of frames being lost. On the contrary, if the feature points are correctly aligned, the face detection algorithm should provide a satisfying bounding box location.

Overall, those results show that the framework illustrated on Figure 6.7, which mainly consists in two adaptations (bounding box regeneration and alignment quality control) provides satisfying results for feature point alignment on video sequences.

## 6.4 Discussion

In this Section, we introduced improvements over the recent Neural Decision Forest framework, which mainly consist in a simplified training procedure as well as a faster greedy evaluation procedure. Furthermore, we study two main applications for the proposed algorithms: First, we use the proposed GNDF for FER involving hyperparameter setting, learning deep representations using CNN features, and runtime comparison between GNDF and a classical NDF. Secondly, we show that GNDF can be successfully



applied to feature point alignment within a cascaded regression framework. By doing so, we obtain very satisfying results for feature point alignment on still images and propose a few methods to perform alignment on video.

Nevertheless, the proposed contributions leads to a number of interesting questions for future research. The first of them would be to know if we could further reduce the evaluation runtime of a NDF by forcing the splits to be axis-aligned instead of oblique. For that matter, an interesting take on that problem would be to use  $\mathcal{L}_1$  regularization on the oblique split weights (as well as the neural network's weights) during training to reduce the numbers of non-zero coefficients. Furthermore, an interesting option would be to fine-tune the NDF trees in an online fashion using a few specific examples for case-specific calibration. Examples of this would be person-specific categorical FER, or pose-specific feature point alignment.

# Chapter 7

## Discussion and Conclusions

### 7.1 Conclusion

Throughout this thesis, we propose solutions to address multiple sub-problems of face analysis, and facial expression recognition in particular. The algorithms we developed find a broad range of applications, including classifying both categorical facial expressions and action units, as well as regressing facial feature points.

In the second chapter of this thesis, we drew an outline of facial expression recognition. We first focused on describing how facial expressions can be modelled and present a generic pipeline of an automatic framework. We then introduced a number of challenges for successful automatic expression recognition, namely morphological factors, environmental changes, as well as head pose variation and occlusion handling.

In the third chapter, we described a number of pattern recognition tools that were later adapted for the purpose of facial expression recognition. We started by describing the representations that can be extracted from the raw face images and fed to the subsequent machine learning layer. We then explained the generic problem of learning a predictive model for classification and regression, and introduce some of the most widely used models that were used in our experiments. Namely, we empathize on describing (deep) neural networks and random forests, as well as recent in-between models called neural decision forests.

The fourth chapter focuses on describing our pairwise conditional random forests

method, which is an adaptation of the random forest framework to learn trees using pairwise differential cues. Those pairwise trees can then be averaged over time to flexibly perform facial expression recognition from videos. Moreover, we also extend our approach to multi-view scenarios to significantly increase robustness to head pose variations.

In the fifth chapter, we describe another adaptation of the random forest framework to learn local expression prediction by spatially restricting the subspaces upon which each tree is learned. Those representations can be weighted by local confidence measurements outputted by an autoencoder network for occlusion-robust categorical expression recognition. Furthermore, it can successfully be used to predict facial action units activation, as the latter are intrinsically local and are closely related to categorical expressions.

Last but not least, the sixth chapter introduces adaptations of the recent neural decision forest algorithm. More specifically, we show that our approach can be used for on-line learning of categorical expression predictors involving deep feature representations. Moreover, it is also suitable for locating facial feature points in the frame of a cascaded regression framework.

Those contributions led to a number of publications in international venues (see Section 1.2.1) as well as a C++ code framework, that will be released open source, for performing end-to-end face analysis, from feature point alignment and feature extraction to expression recognition and AU activation prediction. Noteworthy, the algorithms for training and testing the predictive models can be used for other applicative tasks as well as to address other problems relative to face analysis.

## **7.2 Future works**

### **7.2.1 Using all the labelled data**

As pointed out in Chapter 2, labelled data available to train algorithms for face analysis is relatively scarce, as compared to e.g. image classification or semantic segmentation. Thus, it is important to use all the available data to train the algorithms. The local expression prediction features introduced in Chapter 5 are an example of how categorical expression data can be used to predict a closely related task (e.g. action unit occurrence).

In addition to that, a number of solutions were considered in the frame of this thesis, such as semi-supervised training in the context of training frame-based classifiers upon video sequences, with the idea of “extending” the neutral and apex frame annotations to others images in the sequence. We also considered learning representations with convolutional autoencoders using privileged information (by trying to reconstruct a neutral face from an expressive one, hence capturing features that somewhat represent a difference between those two). Unfortunately, mostly due to time requirements, we were not able to pull up interesting results using those approaches. However, those directions might still lead to interesting conclusions.

### **7.2.2 Exploiting JEMImE database**

As the annotation data for the JEMImE database is currently not available, in this thesis we focused on evaluating our algorithms on state-of-the-art databases for FER. However, it is non-trivial to infer from those results the generalization capabilities of the presented approaches on the JEMImE database, as both the domain (mostly adults for state-of-the-art datasets vs. children for JEMImE) and the tasks (7 categorical expressions or action units vs. quality measurements for a subset of 4 expressions). For that matter, it can be interesting to consider domain adaptation and knowledge transfer techniques. A good survey of those approaches can be found in [66]. A basic example of this would be to pre-train a deep GNDF classifier on state-of-the-art datasets and to fine-tune the model on the JEMImE database.

### **7.2.3 Tuning the algorithms for more efficiency**

Besides the raw accuracy measurement, runtime evaluation is another important factor for the evaluation of a predictive model. Throughout this thesis, we stressed out that our algorithms could run in real-time on a standard computer. Nevertheless, there are a number of approaches that should be considered to further accelerating the processing. For instance, in Chapter 6 we applied  $\mathcal{L}_1$ -regularization on the weights of the neuron layers to reduce the number of non-zero connexions. The same algorithm could in theory

be applied to regularize the weights of the NDF split nodes, though additional testing would be necessary to ensure that the greedy evaluation procedure could hold in such a case.

Another aspect that may be crucial when running the algorithms on low-power devices is the memory usage reduction. For this we are currently considering tuning the neural decision forest framework to train decision jungles [81]. Decision Jungles are similar to decision forests, except the prediction (or leaf) nodes are factorized, greatly diminishing the memory usage.

#### **7.2.4 Adapting the proposed methods to other problems**

Most of the ideas presented in this thesis are not specific to facial expression recognition and could be adapted to other problems in the field of computer vision and pattern recognition. As such, the LS-RF method introduced in Chapter 5 could be used to design an occlusion-robust face alignment system. Furthermore, the PCRf framework (Chapter 4) could be applied to other problems that involve classification of time series, such as gesture recognition, interpersonal synchrony or role prediction in an interactive context. Last but not least, GNDF 6 could be used as a predictive model for any classification or regression task. Noteworthy, as it enables both deep learning of intermediate representations (e.g. CNN) as well as fast evaluation, it could theoretically be used as a replacement to fully-connected layers to speed up semantic segmentation and object recognition systems.

# Bibliography

- [1] M. Abd El Meguid and M. Levine. Fully automated recognition of spontaneous facial expressions in videos using random forest classifiers. *Transactions on Affective Computing*, 5:151–154, 2014. [83](#), [84](#), [85](#), [127](#), [129](#)
- [2] A. Asthana, S. Zafeiriou, S. Cheng, and M. Pantic. Robust discriminative response map fitting with constrained local models. In *International Conference on Computer Vision and Pattern Recognition*, pages 3444–3451, 2013. [139](#)
- [3] A. Asthana, S. Zafeiriou, S. Cheng, and M. Pantic. Incremental face alignment in the wild. In *International Conference on Computer Vision and Pattern Recognition*, pages 1859–1866, 2014. [131](#), [139](#)
- [4] S. A. Bakar, M. S. Hitam, W. Yussof, and W. N. J. Hj. Content-based image retrieval using sift for binary and greyscale images. In *International Conference on Signal and Image Processing Applications*, pages 83–88, 2013. [36](#)
- [5] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. In *European Conference on Computer Vision*, pages 404–417. 2006. [36](#)
- [6] B. Ben Amor, H. Drira, S. Berretti, M. Daoudi, and A. Srivastava. 4-D facial expression recognition by learning geometric deformations. *Transactions on Cybernetics*, 44(12):2443–2457, 2014. [85](#)
- [7] Y. Bengio. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009. [57](#)
- [8] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, et al. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19:153, 2007. [40](#)

- [9] K. S. Bhat, R. Goldenthal, Y. Ye, R. Mallet, and M. Koperwas. High fidelity facial animation capture and retargeting with contours. In *ACM SIGGRAPH/eurographics Symposium on Computer Animation*, pages 7–14, 2013. 9
- [10] L. Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996. 45, 55
- [11] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001. 19, 39, 50, 55, 57, 58, 81, 91
- [12] X. P. Burgos-Artizzu, P. Perona, and P. Dollár. Robust face landmark estimation under occlusion. In *International Conference on Computer Vision*, pages 1513–1520, 2013. 139
- [13] T. Bylander. Estimating generalization error on two-class datasets using out-of-bag estimates. *Machine Learning*, 48(1-3):287–297, 2002. 59
- [14] C. Chen, A. Liaw, and L. Breiman. Using random forest to learn imbalanced data. *University of California, Berkeley*, 2004. 58
- [15] W.-S. Chu, F. Torre, and J. Cohn. Selective transfer machine for personalized facial action unit detection. In *International Conference on Computer Vision and Pattern Recognition*, pages 3515–3522, 2013. 21
- [16] T. F. Cootes, G. J. Edwards, and C. J. Taylor. Active appearance models. In *European Conference on Computer Vision*, pages 484–498, 1998. 131
- [17] T. F. Cootes, C. J. Taylor, D. H. Cooper, and J. Graham. Active shape models-their training and application. *Computer vision and Image Understanding*, 61(1):38–59, 1995. 131
- [18] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995. 19
- [19] S. F. Cotter. Sparse representation for accurate classification of corrupted and occluded facial expressions. In *International Conference on Acoustics, Speech and Signal Processing*, pages 838–841, 2010. 26
- [20] A. Criminisi and J. Shotton. Manifold forests. In *Decision Forests for Computer Vision and Medical Image Analysis*, pages 79–93. Springer, 2013. 45
- [21] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition*, volume 1, pages 886–893, 2005. 23, 36

- [22] M. Dantone, J. Gall, G. Fanelli, and L. Van Gool. Real-time facial feature detection using conditional regression forests. In *International Conference on Computer Vision and Pattern Recognition*, pages 2578–2585, 2012. 65, 76
- [23] D. F. Dementhon and L. S. Davis. Model-based object pose in 25 lines of code. *International Journal of Computer Vision*, 15(1-2):123–141, 1995. 74, 78
- [24] A. Dhall, R. Goecke, S. Lucey, and T. Gedeon. Static facial expression analysis in tough conditions: Data, evaluation protocol and benchmark. In *International Conference on Computer Vision Workshops*, pages 2106–2112, 2011. 30, 108
- [25] P. Dollár, Z. Tu, P. Perona, and S. Belongie. Integral channel features. In *British Machine Vision Conference*, 2009. 36
- [26] P. Ekman. Basic emotions. *Handbook of cognition and emotion*, 98:45–60, 1999. 16
- [27] P. Ekman and W. Friesen. Constants across cultures in the face and emotion. *Journal of personality and social psychology*, 17(2):124, 1971. 16, 28
- [28] P. Ekman and W. V. Friesen. Facial action coding system. 1977. 17
- [29] S. Eleftheriadis, O. Rudovic, and M. Pantic. Discriminative shared gaussian processes for multiview and view-invariant facial expression recognition. *Transactions on Image Processing*, 24(1):189–204, 2015. 24, 108, 109
- [30] Y. Freund, R. Schapire, and N. Abe. A short introduction to boosting. *Japanese Society For Artificial Intelligence*, 14(771-780):1612, 1999. 55
- [31] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980. 40
- [32] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006. 58
- [33] G. Ghiasi and C. C. Fowlkes. Occlusion coherence: Localizing occluded faces with a hierarchical deformable part model. In *International Conference on Computer Vision and Pattern Recognition*, pages 1899–1906, 2014. 26, 120
- [34] S. Ghosh, E. Laksana, S. Scherer, and L.-P. Morency. A multi-label convolutional neural network approach to cross-domain action unit detection. In *Affective Computing and Intelligent Interaction*, 2015. 118, 119, 120



- [35] M. K. Greenwald, E. W. Cook, and P. J. Lang. Affective judgment and psychophysiological response: Dimensional covariation in the evaluation of pictorial stimuli. *Journal of psychophysiology*, 3(1):51–64, 1989. 16
- [36] S. Happy and A. Routray. Automatic facial expression recognition using features of salient facial patches. *IEEE Transactions on Affective Computing*, pages 1–13, 2014. 82, 83
- [37] M. Hayat, M. Bennamoun, and A. El-Sallam. Evaluation of spatiotemporal detectors and descriptors for facial expression recognition. In *International Conference on Human System Interactions*, pages 43–47, 2012. 21, 83, 108
- [38] T. K. Ho. The random subspace method for constructing decision forests. *Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998. 55
- [39] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989. 19, 39
- [40] X. Huang, G. Zhao, W. Zheng, and M. Pietikäinen. Towards a dynamic expression recognition system under facial occlusion. *Pattern Recognition Letters*, 33(16):2181–2191, 2012. 26, 110
- [41] L. Jeni, D. Takacs, A. Lorincz, et al. High quality facial expression recognition in video streams using shape related information only. In *International Conference on Computer Vision Workshops*, pages 2168–2174, 2011. 86
- [42] L. A. Jeni, J. F. Cohn, and T. Kanade. Dense 3d face alignment from 2d videos in real-time, 2015. 97
- [43] I. Jolliffe. *Principal component analysis*. Wiley Online Library, 2002. 45
- [44] H. Jung, S. Lee, S. Park, I. Lee, C. Ahn, and J. Kim. Deep temporal appearance-geometry network for facial expression recognition. *International Conference on Computer Vision*, 2015. 22, 36, 127
- [45] M. Khademi and L.-P. Morency. Relative facial action unit detection. In *Winter Conference on Applications of Computer Vision*, pages 1090–1095, 2014. 21
- [46] R. A. Khan, A. Meyer, H. Konik, and S. Bouakaz. Human vision inspired framework for facial expressions recognition. In *International Conference on Image Processing*, pages 2593–2596, 2012. 19
- [47] A. Klaser, M. Marszałek, and C. Schmid. A spatio-temporal descriptor based on 3D-gradients. In *British Machine Vision Conference*, pages 995–1004, 2008. 21, 84, 90

- [48] P. Kotschieder, M. Fiterau, A. Criminisi, and S. Rota Bulo. Deep neural decision forests. In *International Conference on Computer Vision*, pages 1467–1475, 2015. 50, 60, 61, 62, 121, 122
- [49] I. Kotsia, I. Buciu, and I. Pitas. An analysis of facial expression recognition under partial facial image occlusion. *Image and Vision Computing*, 26(7):1052–1067, 2008. 26
- [50] B. Lakshminarayanan, D. M. Roy, and Y. W. Teh. Mondrian forests: Efficient online random forests. In *Advances in Neural Information Processing Systems*, pages 3140–3148, 2014. 126
- [51] J. Langford, L. Li, and T. Zhang. Sparse online learning via truncated gradient. *Journal of Machine Learning Research*, 10(Mar):777–801, 2009. 135
- [52] Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995. 48
- [53] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989. 43
- [54] H. Linusson. Multi-output random forests. 2013. 106, 116
- [55] M. Liu, S. Li, S. Shan, and X. Chen. Enhancing expression recognition in the wild with unlabeled reference data. In *Asian Conference on Computer Vision*, pages 577–588. 2013. 20, 108, 109
- [56] M. Liu, S. Li, S. Shan, and X. Chen. Au-inspired deep networks for facial expression feature learning. *Neurocomputing*, 159:126–136, 2015. 20, 108
- [57] G. Louppe. *Understanding Random Forests: From Theory to Practice*. PhD thesis, University of Liège. 87
- [58] D. G. Lowe. Object recognition from local scale-invariant features. In *International Conference on Computer Vision*, volume 2, pages 1150–1157, 1999. 23, 24, 36
- [59] P. Lucey, J. F. Cohn, T. Kanade, J. Saragih, Z. Ambadar, and I. Matthews. The extended cohn-kanade dataset (CK+): A complete dataset for action unit and emotion-specified expression. In *International Conference on Computer Vision and Pattern Recognition Workshops*, pages 94–101, 2010. 16, 20, 28
- [60] A. Makhzani and B. Frey. k-sparse autoencoders. *arXiv preprint arXiv:1312.5663*, 2013. 47

- [61] B. Martinez and M. F. Valstar. L 2, 1-based regression and prediction accumulation across views for robust facial landmark detection. *Image and Vision Computing*, 47:36–44, 2016. 139
- [62] S. M. Mavadati, M. H. Mahoor, K. Bartlett, P. Trinh, and J. F. Cohn. DISFA: A spontaneous facial action intensity database. *Transactions on Affective Computing*, 4(2):151–160, 2013. 31
- [63] M. Mohammadi, E. Fatemizadeh, and M. Mahoor. Non-negative sparse decomposition based on constrained smoothed l0 norm. *Signal Processing*, 100:42–50, 2014. 82, 83
- [64] S. Moore and R. Bowden. Local binary patterns for multi-view facial expression recognition. *Computer Vision and Image Understanding*, 115(4):541–558, 2011. 24
- [65] J. Nicolle, K. Bailly, and M. Chetouani. Facial action unit intensity prediction via hard multi-task metric learning for kernel regression. In *International Conference on Automatic Face and Gesture Recognition*, 2015. 103
- [66] S. J. Pan and Q. Yang. A survey on transfer learning. *Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010. 149
- [67] J. R. Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986. 53
- [68] M. A. Ranzato, J. Susskind, V. Mnih, and G. Hinton. On deep generative models with applications to recognition. In *International Conference on Computer Vision and Pattern Recognition*, pages 2857–2864, 2011. 26, 129
- [69] S. Ren, X. Cao, Y. Wei, and J. Sun. Face alignment at 3000 fps via regressing local binary features. In *International Conference on Computer Vision and Pattern Recognition*, pages 1685–1692, 2014. 25, 88, 95
- [70] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *International Conference on Machine Learning*, pages 833–840, 2011. 47
- [71] H. Robbins and S. Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951. 43
- [72] J. V. d. W. Ruiz Adria and X. Binefa. From emotions to action units with hidden and semi-hidden-task learning. In *International Conference on Computer Vision*, 2015. 118, 119

- [73] A. Saffari, C. Leistner, J. Santner, M. Godec, and H. Bischof. On-line random forests. In *International Conference on Computer Vision Workshops*, pages 1393–1400, 2009. 126
- [74] G. Sandbach, S. Zafeiriou, M. Pantic, and D. Rueck. A dynamic approach to the recognition of 3D facial expressions and their temporal models. In *International Conference on Automatic Face and Gesture Recognition*, pages 406–413, 2011. 108
- [75] W. S. Sarle. Stopped training and other remedies for overfitting. In *Proc. of the 27th symposium on the interface of computing science and statistics*, pages 352–360, 1995. 45
- [76] A. Savran, N. Alyüz, H. Dibeklioglu, O. Çeliktutan, B. Gökberk, B. Sankur, and L. Akarun. Bosphorus database for 3d face analysis. In *Biometrics and Identity Management*, pages 47–56. 2008. 120
- [77] T. Senechal, V. Rapp, H. Salam, R. Segulier, K. Bailly, and L. Prevost. Facial action recognition combining heterogeneous features via multikernel learning. *Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 42(4):993–1005, 2012. 19, 36
- [78] C. Shan, S. Gong, and P. W. McOwan. Facial expression recognition based on local binary patterns: A comprehensive study. *Image and Vision Computing*, 27(6):803–816, 2009. 19, 108, 129
- [79] S. Shojaeilangari, W.-Y. Yau, J. Li, and E.-K. Teoh. Multi-scale analysis of local phase and local orientation for dynamic facial expression recognition. *Journal of Multimedia Theory and Application*, 1:1–10, 2014. 22, 82, 83
- [80] J. Shotton, T. Sharp, A. Kipman, A. Fitzgibbon, M. Finocchio, A. Blake, M. Cook, and R. Moore. Real-time human pose recognition in parts from single depth images. *Communications of the ACM*, 56(1):116–124, 2013. 58
- [81] J. Shotton, T. Sharp, P. Kohli, S. Nowozin, J. Winn, and A. Criminisi. Decision jungles: Compact and rich models for classification. In *Advances in Neural Information Processing Systems*, pages 234–242, 2013. 150
- [82] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014. 40, 45, 61
- [83] A. Suga, K. Fukuda, T. Takiguchi, and Y. Ariki. Object recognition and segmentation using sift and graph cuts. In *International Conference on Pattern Recognition*, pages 1–4. IEEE, 2008. 36

- [84] M. Sun, P. Kohli, and J. Shotton. Conditional regression forests for human pose estimation. In *International Conference on Computer Vision and Pattern Recognition*, pages 3394–3401, 2012. 65, 76
- [85] Y. Sun and L. Yin. Facial expression recognition based on 3D dynamic range model sequences. In *European Conference on Computer Vision*, pages 58–71. 2008. 83, 108
- [86] Y. Tang. Deep learning using linear support vector machines. *arXiv preprint arXiv:1306.0239*, 2013. 50
- [87] U. Tariq, J. Yang, and T. S. Huang. Multi-view facial expression recognition analysis with generic sparse coding feature. In *European Conference on Computer Vision Workshops and Demonstrations*, pages 578–588, 2012. 24, 76, 86
- [88] G. Tzimiropoulos. Project-out cascaded regression with an application to face alignment. In *International Conference on Computer Vision and Pattern Recognition*, pages 3659–3667, 2015. 139
- [89] G. Tzimiropoulos and M. Pantic. Gauss-newton deformable part models for face alignment in-the-wild. In *International Conference on Computer Vision and Pattern Recognition*, pages 1851–1858, 2014. 139
- [90] R.-L. Vieriu, S. Tulyakov, S. Semeniuta, E. Sangineto, and N. Sebe. Facial expression recognition under a wide range of head poses. In *International Conference on Automatic Face and Gesture Recognition*, 2015. 24, 76, 86
- [91] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11:3371–3408, 2010. 47
- [92] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Conference on Computer Vision and Pattern Recognition*, volume 1, pages 511–518, 2001. 33, 77, 131, 138
- [93] F. Wallhoff. Database with facial expressions and emotions from technical university of munich (feedtum). <http://cotesys.mmk.e-technik.tu-muenchen.de/waf/fgnet/feedtum.html>, 2006. 29, 120
- [94] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning*, pages 1058–1066, 2013. 45

- [95] J. Wang, S. Wang, and Q. Ji. Early facial expression recognition using hidden markov models. In *International Conference on Pattern Recognition*, pages 4594–4599, 2014. 22
- [96] Z. Wang, S. Wang, and Q. Ji. Capturing complex spatio-temporal relations among facial muscles for facial expression recognition. In *International Conference on Computer Vision and Pattern Recognition*, pages 3422–3429, 2013. 22
- [97] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *Transactions on Evolutionary Computation*, 1(1):67–82, 1997. 37
- [98] X. Xiong and F. De la Torre. Supervised descent method and its applications to face alignment. In *International Conference on Computer Vision and Pattern Recognition*, pages 532–539, 2013. 25, 33, 78, 96, 134, 135, 139
- [99] X. Xiong and F. De la Torre. Global supervised descent method. In *Conference on Computer Vision and Pattern Recognition*, pages 2664–2673, 2015. 25, 90
- [100] L. Xu and P. Mordohai. Automatic facial expression recognition using bags of motion words. In *British Machine Vision Conference*, pages 1–13, 2010. 108
- [101] H. Yang, X. Jia, C. C. Loy, and P. Robinson. An empirical study of recent face alignment methods. *arXiv preprint arXiv:1511.05049*, 2015. 130, 138
- [102] Y. Yao, L. Rosasco, and A. Caponnetto. On early stopping in gradient descent learning. *Constructive Approximation*, 26(2):289–315, 2007. 45
- [103] L. Yin, X. Chen, Y. Sun, T. Worm, and M. Reale. A high-resolution 3D dynamic facial expression database. In *International Conference on Automatic Face and Gesture Recognition*, pages 1–6, 2008. 16, 20, 25, 29, 76
- [104] L. Yin, X. Wei, Y. Sun, J. Wang, and M. J. Rosato. A 3D facial expression database for facial behavior research. In *International Conference on Automatic Face and Gesture Recognition*, pages 211–216, 2006. 24, 76, 83
- [105] M. D. Zeiler and R. Fergus. Stochastic pooling for regularization of deep convolutional neural networks. *arXiv preprint arXiv:1301.3557*, 2013. 49
- [106] J. Zhang, S. Shan, M. Kan, and X. Chen. Coarse-to-fine auto-encoder networks for real-time face alignment. In *European Conference on Computer Vision*, pages 1–16, 2014. 139
- [107] L. Zhang, D. Tjondronegoro, and V. Chandran. Random Gabor based templates for facial expression recognition in images with facial occlusion. *Neurocomputing*, 145:451–464, 2014. 26, 110, 111, 112, 113

- [108] X. Zhang, L. Yin, J. F. Cohn, S. Canavan, M. Reale, A. Horowitz, P. Liu, and J. M. Girard. BP4D-spontaneous: a high-resolution spontaneous 3D dynamic facial expression database. *Image and Vision Computing*, 32(10):692–706, 2014. 16, 30, 84, 85, 118, 119
- [109] G. Zhao and M. Pietikainen. Dynamic texture recognition using local binary patterns with an application to facial expressions. *Transactions on Pattern Analysis and Machine Intelligence*, 29(6):915–928, 2007. 21, 84, 90
- [110] X. Zhao, T.-K. Kim, and W. Luo. Unified face analysis by iterative multi-output random forests. In *International Conference on Computer Vision and Pattern Recognition*, pages 1765–1772, 2014. 20, 108
- [111] W. Zheng, H. Tang, Z. Lin, and T. S. Huang. Emotion recognition from arbitrary view facial images. In *European Conference on Computer Vision*, pages 490–503. 2010. 24
- [112] L. Zhong, Q. Liu, P. Yang, B. Liu, J. Huang, and D. N. Metaxas. Learning active facial patches for expression analysis. In *International Conference on Computer Vision and Pattern Recognition*, pages 2562–2569, 2012. 20, 108

# **Appendices**





# Appendix A

## A lower bound for the depth of randomly initialized trees with constant prediction nodes

### A.1 Random uniform initialization for classification

**proof:** Let  $\mathcal{A}$  denote the following proposition: “For every class  $c \in [1, \mathcal{C}]$  tree  $t$  contains at least one leaf that predicts class  $c$ ”.

For one specific class  $c$ , we denote  $\mathcal{A}_c$  the following event “tree  $t$  contains at least one leaf that predicts class  $c$ ”. The opposite event  $\bar{\mathcal{A}}_c$  then reads “no leaf of tree  $t$  predicts class  $c$ ”. We thus have:

$$p(\mathcal{A}) = (p(\mathcal{A}_c))^{\mathcal{C}} = (1 - p(\bar{\mathcal{A}}_c))^{\mathcal{C}} \quad (\text{A.1})$$

Furthermore, for one specific prediction (leaf) node  $l$ , as the prediction values are randomly sampled from a uniform distribution, the probability not to predict class  $c$  is  $1 - \frac{1}{\mathcal{C}}$ . Thus, for a balanced tree of depth  $\mathcal{D}$ , *i.e.* that contains  $2^{\mathcal{D}}$  prediction nodes we have:

$$p(\bar{\mathcal{A}}_c) = (1 - \frac{1}{\mathcal{C}})^{2^{\mathcal{D}}} \quad (\text{A.2})$$

By using equations A.1 and A.2 we obtain:

$$p(\mathcal{A}) = (1 - (1 - \frac{1}{c})^{2^{\mathcal{D}}})^c \quad (\text{A.3})$$

We then want to set the tree depth  $\mathcal{D}$  so that  $p(\mathcal{A}) > 1 - \epsilon$  (*i.e.* to have all classes covered by at least one leaf of tree  $t$  with a probability superior to  $1 - \epsilon$ ). Using A.2 this is equivalent to:

$$\mathcal{D} > \mathcal{D}_0 \quad (\text{A.4})$$

With

$$\mathcal{D}_0 = \frac{1}{\ln(2)} \ln\left(\frac{\ln(1 - (1 - \epsilon)^{1/\mathcal{C}})}{\ln(1 - 1/\mathcal{C})}\right) \quad (\text{A.5})$$

This setting is of tremendous importance if we choose not to adapt the prediction nodes during training. For instance, if one tree do not contain class  $c$ , it will always make erroneous prediction w.r.t. this class, which will result in more noise in the predicted values after averaging the multiple tree predictors. Furthermore, we can write:

$$1 - (1 - \epsilon)^{1/c} \underset{c \rightarrow \infty}{=} -\frac{1}{\mathcal{C}} \ln(1 - \epsilon) + o\left(\frac{1}{\mathcal{C}}\right) \quad (\text{A.6})$$

Hence

$$\ln(1 - (1 - \epsilon)^{1/\mathcal{C}}) \underset{\mathcal{C} \rightarrow \infty}{\sim} -\ln(\mathcal{C}) \quad (\text{A.7})$$

Moreover,

$$\ln(1 - \frac{1}{\mathcal{C}}) \underset{\mathcal{C} \rightarrow \infty}{\sim} -\frac{1}{\mathcal{C}} \quad (\text{A.8})$$

Thus:

$$\mathcal{D}_0 \underset{\mathcal{C} \rightarrow \infty}{\sim} \frac{\ln(\mathcal{C})}{\ln(2)} \quad (\text{A.9})$$

*i.e.* The lower bound depth  $\mathcal{D}_0$  for trees of a NDF with constant leaf nodes that are randomly sampled from uniform distribution across the classes grows as the logarithm of the number of classes  $\mathcal{C}$ .

## A.2 Gaussian initialization for regression

In the case of regression, we aim at showing that, provided the tree is deep enough, for each value in the range that shall be covered by the tree, one can find at least one leaf prediction that is close to that value. Let  $\mathcal{A}$  denote the following proposition: “For every value  $y \in [\bar{\delta}_k - \sigma_k, \bar{\delta}_k + \sigma_k]$  tree  $t$  contains at least one leaf such that the prediction  $y_l$  of that leaf satisfies  $|y_l - y| < \epsilon$ ”.

We also define  $\mathcal{A}_y$  the following proposition “For value  $y$  there is at least one leaf  $l$  of tree  $t$ , such that  $|y_l - y| < \epsilon$ ”. We have:

$$p(\mathcal{A}) = \prod_{\bar{\delta}_k - \sigma_k}^{\bar{\delta}_k + \sigma_k} p(\mathcal{A}_y)^{dy} \quad (\text{A.10})$$

Which can also be written

$$p(\mathcal{A}) = \exp\left(\int_{\bar{\delta}_k - \sigma_k}^{\bar{\delta}_k + \sigma_k} \ln(p(\mathcal{A}_y)) dy\right) \quad (\text{A.11})$$

Let's then denote  $\bar{\mathcal{A}}_y$  the proposition: “for every leaf of tree  $t$ ,  $|y_l - y| > \epsilon$ . Clearly we have

$$p(\mathcal{A}_y) = 1 - p(\bar{\mathcal{A}}_y) \quad (\text{A.12})$$

Moreover, as a tree of depth  $\mathcal{D}$  shall contain  $2^{\mathcal{D}}$ , we have:

$$p(\bar{\mathcal{A}}_y) = p(|y_l - y| > \epsilon)^{2^{\mathcal{D}}} \quad (\text{A.13})$$

Furthermore, as the leaf predictions  $y_l$  are randomly initialized, for one specific leaf node  $l$  we can write:

$$p(|y_l - y| > \epsilon) = 1 - \int_{y-\epsilon}^{y+\epsilon} \frac{1}{\sqrt{2\pi}\sigma_k} e^{-\frac{(z-\delta_k)^2}{2\sigma_k^2}} dz \quad (\text{A.14})$$

We can use a lower bound of the gaussian function on the interval  $[\delta_k - \sigma_k - \epsilon, \delta_k + \sigma_k + \epsilon]$  to provide an upper bound on this probability:

$$p(|y_l - y| > \epsilon) < 1 - \int_{y-\epsilon}^{y+\epsilon} \frac{1}{\sqrt{2\pi}\sigma_k} e^{-\frac{(\sigma_k+\epsilon)^2}{2\sigma_k^2}} dz \quad (\text{A.15})$$

thus

$$p(|y_l - y| > \epsilon) < 1 - \frac{2\epsilon}{\sqrt{2\pi}\sigma_k} e^{-\frac{(\sigma_k + \epsilon)^2}{2\sigma_k^2}} \quad (\text{A.16})$$

Moreover, using Equations A.11, A.12 and A.13 we have:

$$p(\mathcal{A}) = \exp\left(\int_{\bar{\delta}_k - \sigma_k}^{\bar{\delta}_k + \sigma_k} \ln(1 - p(|y_l - y| > \epsilon)^{2^D}) dy\right) \quad (\text{A.17})$$

Thus, as both  $\ln$ ,  $\exp$  and  $\int$  are increasing functions, using Equations A.17 and A.16 provides a lower bound of  $p(\mathcal{A})$ :

$$p(\mathcal{A}) > \exp\left(\int_{\bar{\delta}_k - \sigma_k}^{\bar{\delta}_k + \sigma_k} \ln\left(1 - \left(1 - \frac{2\epsilon}{\sqrt{2\pi}\sigma_k} e^{-\frac{(\sigma_k + \epsilon)^2}{2\sigma_k^2}}\right)^{2^D}\right) dy\right) \quad (\text{A.18})$$

Which we can write

$$p(\mathcal{A}) > \left(1 - \left(1 - \frac{2\epsilon}{\sqrt{2\pi}\sigma_k} e^{-\frac{(\sigma_k + \epsilon)^2}{2\sigma_k^2}}\right)^{2^D}\right)^{2\sigma_k} \quad (\text{A.19})$$

Thus, a sufficient condition to ensure  $p(\mathcal{A}) > 1 - \epsilon'$  (with  $\epsilon'$  close to 0) is to have  $\mathcal{D} > \mathcal{D}_0$  with

$$\mathcal{D}_0 = \frac{1}{\ln(2)} \ln \frac{\ln\left(1 - \left(1 - \epsilon'\right)^{\frac{1}{2\sigma_k}}\right)}{\ln\left(1 - \frac{2\epsilon}{\sqrt{2\pi}\sigma_k} e^{-\frac{(\sigma_k + \epsilon)^2}{2\sigma_k^2}}\right)} \quad (\text{A.20})$$

The lower bound  $\mathcal{D}_0$  is somewhat similar to the one in the classification case. Furthermore, we show that:

$$\mathcal{D}_0 \underset{\epsilon \rightarrow 0}{\sim} -\frac{\ln(\epsilon)}{\ln(2)} \quad (\text{A.21})$$

As in the classification case, given the regression range  $\sigma_k$  the lower bound depth  $\mathcal{D}_0$  grows as the logarithm of the desired “resolution” (which is the inverse of  $\epsilon$ ).

# Appendix B

## implementation details

One of the focus of the PhD was to produce a reusable, open-source C++ implementation to perform real-time expression recognition on video streams, as well as to provide a framework for learning architectures for face analysis. To this aim, this section describes the main features of the proposed source code.

### B.1 Dependencies

The proposed C++ solution have been developped on a Windows environment with a few dependencies that are listed below. Note that for the projects to be compiled properly you will need to set the environment variables to the values indicated in italics.

- **Boost** v. 1.54 or higher (*BOOST*) - libraries *system, filesystem*
- **OpenCV** v. 2.4.6 or higher (*OPENCV*) - libraries *core, highgui, imgproc, objdetect*
- **0MQ** v. 4.0.4 or higher (*ZMQ*)
- **Intraface** (*INTRAFACE*)
- **pThread** (*PTHREAD*)

## B.2 Data structures

The basic data structures are declared in header `HMimproc.h`. The most useful ones are described below.

**imgdesc:** as suggested by its name, this class implements an image descriptor that is composed of an image (`image_curr`), a set of aligned facial feature points (`featurePoints_curr`), the corresponding head pose estimate (`headPose_curr`), and a set of integral feature channels (`pcIntegralChannels_curr`) with the correspondingly scaled feature points (`pcFeaturePoints_curr`). Each `imgdesc` also contains a class label (or a set of labels for multi-output training) that is used only for training/evaluating the ensembles of randomized trees. Also, for PCRF and MVPCRF training and evaluation, a pairwise version of the descriptor contains pairs of images (`image_past`) with the corresponding feature points, head pose, integral channels and scaled feature points thereof.

`imgdesc` objects are most of the time initialized through constructors

```
imgdesc( shared_ptr<Mat> & img, const vector<Point2f> & fpts  
        , const unsigned int lbl )
```

or

```
imgdesc( shared_ptr<Mat> & img, const vector<Point2f> & fpts  
        , const HeadPose & hp, const unsigned int lbl )
```

Those constructors are generally called through overloaded functions such as `loadImageDesc` from the `database.h` header, in order to load all the images/feature points/class labels from a specific folder and generate image descriptors accordingly. Note that integral feature channels have to be generated separately, using overloaded function `generateIntegralChannels`. Then, for pairwise RF training and testing, pairwise `imgdesc` have to be generated using the `generateTransitionDescriptor` function.

**BagOfFeatures:** The `imgdesc` class is mainly used for training/evaluating ensembles of trees with on-the-fly candidate feature generation. However it is also useful in

many case to allow RF induction with a more generic pipeline, *i.e.* with feature descriptors represented as raw vectors (`features_`) of the `BagOfFeatures` class. This class also contains fields for class labels (`label_`) or multi-output training labels (`MO_labels`). In either cases, `BagOfFeatures` objects have to be initialized using constructors

```
BagOfFeatures(const vector<float> & features , unsigned int
    lbl)
```

or

```
BagOfFeatures(const vector<float> & features , const vector<
    unsigned int> & lbls)
```

Note that the feature vector has to be generated beforehand for each object.

**HMSQMat:** This class is used as a basic square matrix class. It implements a number of basic operations for image processing, such as addition, subtraction, scalar and element-wise multiplication. As it is used for storing CNN inputs and feature maps, it also has high-level operations such as convolution (*full* or *valid*, *flip* or *upsample*).

## B.3 Low-level operations

Most of the low-level image processing and machine learning functions are implemented in headers `HMimproc.h` and `HMmlfun.h`. The most useful ones are described below.

- `loadImageDesc`: load image descriptor from the provided database path. Namely, it generates a vector of `imgdesc` objects from the provided `trainingImagePath_` folder by recursively searching using the `recLoadImages`, `recLoadPoints` and `recLoadImageLabels` functions
- `extractROI`: extract face ROI using the provided facial feature points
- `generateIntegralChannels`: generate integral HOG channels with an 8-bit gradient quantization, with a specified output size



- `generateTransitionDescriptor`: generate pairwise `imgdesc` descriptor from two separate descriptors

## B.4 Solution architecture

The proposed solution consists of 3 independant projects:

- `Decision Forests`: project for RF induction, including a number of contributions from this PHD thesis: PCRf training and testing, as well as static and dynamic multi-view extentions (MVRF/MVPCRf). It also features the LS-RF for occlusion handling, multi-output multi-class prediction and regression, and ND-F/GNDF training and evaluation.
- `NeuralNets`: project for generic neural network training/testing. Also features unsupervised learning of autoencoder nets and convolutional neural networks.
- `SDM`: project for real-time automated facial expression recognition from live or pre-recorded video stream.

### B.4.1 The Decision Forests project

#### B.4.1.1 Project overview

Class diagram for the `Decision Forests` project can be seen on Figure B.1, summarizing its architecture as well as its most relevant features.

Specifically, the entry point of the project lies in the `demoemclassifier.cpp` file, which creates a `RFTrainer/RFTester` object from the `config.txt` configuration file that shall be placed in the project folder. Section B.4.1.2 explains the syntax of the configuration file. Section B.4.1.3 explains how a classic RF predictor is generated from the proposed code. Finally, Section B.4.1.4 describes a number of methods that have been implemented for RF induction and testing for face analysis.

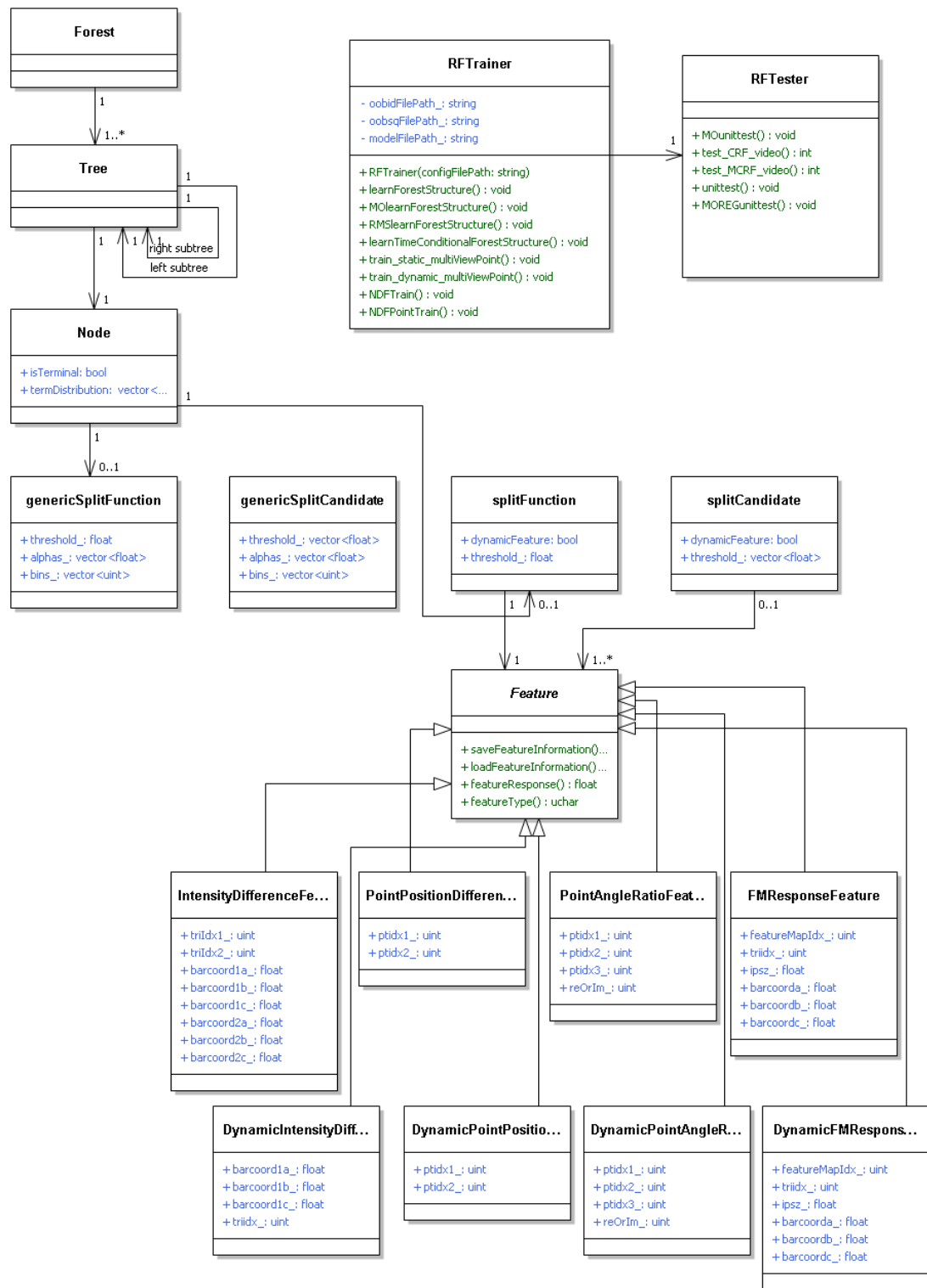


Figure B.1: Class diagram for the Decision Forests project

#### B.4.1.2 Configuration options and class constructors

The `config.txt` file is parsed through the functions written in the `parser.cpp` file within the constructor of `RFTrainer/RFTester`. Be careful while tweaking the `config.txt` file, as verifications may not be implemented in all the methods, causing the program to crash (for example, avoid specifying features from the dynamic templates while training a static RF, or using on-the-fly candidate features in addition of pre-recorded ones).

You can specify the following main options through the `config.txt` file:

- `modeVerbose_`: (boolean) activating/deactivating console feedback during RF training.
- `nbXXXFeatures_`: (integer) specifying the number of on-the-fly features for a specific static template (3 static templates: `PointDiff`, `AngleRatio`, `FM`).
- `nbDynamicXXXFeatures_`: (integer) specifying the number of on-the-fly features for a specific dynamic template (3 dynamic templates: `PointDiff`, `AngleRatio`, `FM`).
- `nbPreRecordedFeatures_`: (integer) specifying the number of pre-recorded features.
- `nbCandidateThresholdsPerFeature_`: self-explanatory!
- `numFeaturePoints_`: number of aligned feature points (49 provided by the SDM tracker).
- `numFeatureTri_`: number of aligned feature triangles (79 provided by the SDM tracker). the description of the facial mesh is loaded in the `trisummits49.txt` file. If you wish to use a different number of facial feature points or a different mesh you will need to manually re-enter a facial mesh.
- `treeNum_`: self-explanatory!
- `numClasses_`: self-explanatory!

- `treeMaxDepth_`: maximum depth of the trees, above which no more split is calculated and a leaf node is set.
- `datasizeRatioPerTree_`: size of the bootstraps for each tree (relatively to the total data size).
- `labelImbalancyTolerance_`: accepted absolute ratio of unbalance for each bootstrap.
- `trainingImagePath_`: path of the training database that will be loaded by the `RFTrainer` class constructor
- `testingImagePath_`: path of the training database that will be loaded by the `RFTester` class constructor
- `modelFilePath_`: path to the saved RF model file
- `oobidFilePath_`: path to the saved OOB elements indexes record file
- `oobsqFilePath_`: path to the saved OOB sequence indexes record file

#### B.4.1.3 Delving into the code: an example of RF induction

After the configuration file is loaded, the function `train( )` from the `RFTrainer` class loads the database from the provided `trainingImagePath_` folder using the `loadImageDesc` function. The database folder shall contain a number of data point, each of which consisting in three files:

- a `.ann` file containing the labelling information. The base templating is `<subject id (unsigned int)> <expression label (unsigned int)> <recording session id (unsigned int)>`. By default, the available expression labels are: 0 for *neutral*, 1 for *happy*, 2 for *angry*, 3 for *sad*, 4 for *fear*, 5 for *disgust* and 6 for *surprise*. Alternatively, 8-class FER can be performed using also the *contempt* label from the CK+ database. Also, the `.ann` file may contain additional information, such as the frame index and total number of frame in the recording session, or the head pose information (pitch/yaw/roll).

- a .pts file containing information on the 49 feature points (x-y coordinates) previously aligned on the face using a feature point tracker.
- a .png face image

Once the database is loaded in the form of a vector of `imgdesc` pointers (shared pointers from the `std` library are used within the whole solution to prevent any kind of memory leaks), the integral HOG feature channels are computed if the FMFeatures are used (that is, if the `nbFMFeatures_` parameter in the `config.txt` file is non-zero). Finally, the `learnForestStructure` function is called to generate a RF prediction model.

The `learnForestStructure` function first computes the bounds for each feature template (point distance and angle ratio, and HOG feature maps) by calling `findFeatureBounds` on the provided data. After that, the model and OOB files are reinitialized. Then, for each new tree a data bootstrap is generated at the subject level using the `randomSample2` function using the provided `seqref` and `datasizeRatioPerTree_` parameter. Optionally, the bootstrap is balanced by downsampling the majority classes using the `balancedataset` function. Then `learnTreeStructure` is called on the generated bootstrap. Note that the accuracy of the tree collection can be tested at any time using OOB estimate by pressing the “a” key while the program has the focus (which toggles the global boolean `SHOW_ACCURACY`).

The `learnTreeStructure` is the main function for to grow a randomized decision tree on a provided data collection. It works in a recursive fashion starting from a root node, by first checking if the termination criterion is met (*i.e.* current tree depth `currDepth` has reached the maximum allowed depth `treeMaxDepth_`, or the provided data is homogeneous in term of class labels). If not, a split node is set, increasing the mean purity in left and right subtrees (function `splitAtCurrentNode`). Then `learnTreeStructure` is recursively called on the left and right subtrees. Note that the nodes are stored in the model file (`modelFilePath` argument) as and when the split and leaf nodes are set with corresponding parameters and terminal distributions. Thus, the proposed implementation is memory efficient as exactly one node is stored in memory at

any time. The `splitAtCurrentNode` method works by (a) randomly generating a list of split candidates (function `generateRandomCandidates`), (b) concurrently evaluating the Shannon entropy for those candidates using `entropyOfPartition_MT` and (c) returning the data partition at current node that is induced by the candidate for which the entropy is the lowest.

#### **B.4.1.4 Main methods for face analysis**

In this Section, we present a number of methods that correspond to implementations of variants of the above described classical RF induction procedure that provided interesting results for face analysis and affect recognition.

- `MOlearnForestStructure`: method for learning RF for multi output classification (`MULTI_OUTPUT_MODE=true`, `USE_REGRESSION=false`) and regression (`USE_REGRESSION=true`). For using this function, the `MO_labels` field of the `imgdesc` objects have to be pre-emptively filled.
- `RMSlearnForestStructure`: method for learning RF using spatially-defined local face subspaces, that are generated using the `RMSgenerateRandomMask` method. The mask informations are saved within the provided `rmsmaskFilePath` file.
- `learnTimeConditionalForestStructure`: method to train a PCRf. It outputs `numClasses` pairwise RF models that are saved within the provided `rootSavePath` folder. Optionally, those models can be defined on local subspaces (`USE_LOCAL_SUBSPACE=true`), in which case the masks are also saved within the folders.
- `train_static_multiViewPoint`: method to train a MVRF from the multi-view database root folder `rootDatabasePath`. The pose-conditional models are saved in subfolders of the provided root save path `rootSavePath`. For each pose bin the models are saved along with the local subspace masks (optional) and the pose distribution file.

- `train_dynamic_multiViewPoint`: method to train a MVPCRF from the multi-view database root folder `rootDatabasePath`. The pose-conditional pairwise models are saved in subfolders of the provided root save path `rootSavePath`. For each pose bin the models are saved along with the local subspace masks (optional) and the pose distribution file.
- `NDFTrain`: method for learning a NDF for facial expression classification (or any provided categorical data) using pre-computed features (in the sample code provided, distances between feature points and/or CNN and DAG-CNN features).
- `NDFPointsTrain`: method for learning a CSP-dGNDF for facial feature point alignment.

## B.4.2 The NeuralNets project

### B.4.2.1 Project overview

This project is mainly used to train the autoencoder networks proposed in Section 5.3.2. However, the network training procedure and architecture specification is also used in the Decision Forests and Real Time projects respectively when training NDFs and using these for real-time face alignment. The class diagram of the NeuralNets can be seen in Figure B.2. The main class of the project is the `NeuralNet` class. In the following paragraph, we explain the architecture of this class, as well as its most relevant features.

**B.4.2.1.1 The NeuralNet class.** This class is mainly composed of a vector of pointers towards elements of class `NeuronLayer`. The `NeuronLayer` class is virtual, so elements must be specified as either `SigmoidLayer` (tag 0), `Autoencoder` (tag 1), `SoftMaxLayer` (tag 2), `TanhLayer` (tag 3) or `SparseLayer` (tag 4), depending on which activation function is used (Section 3.3.3.1). The connexions between the neuron layer are stored into the `connexions_matrix`. This matrix row and columns numbers are equal to the number of neuron layers  $M$ : a 1 value at the  $(i, j)$  position indicates that the  $i^{th}$  layer has a forward connexion to the  $j^{th}$  layer. For instance,

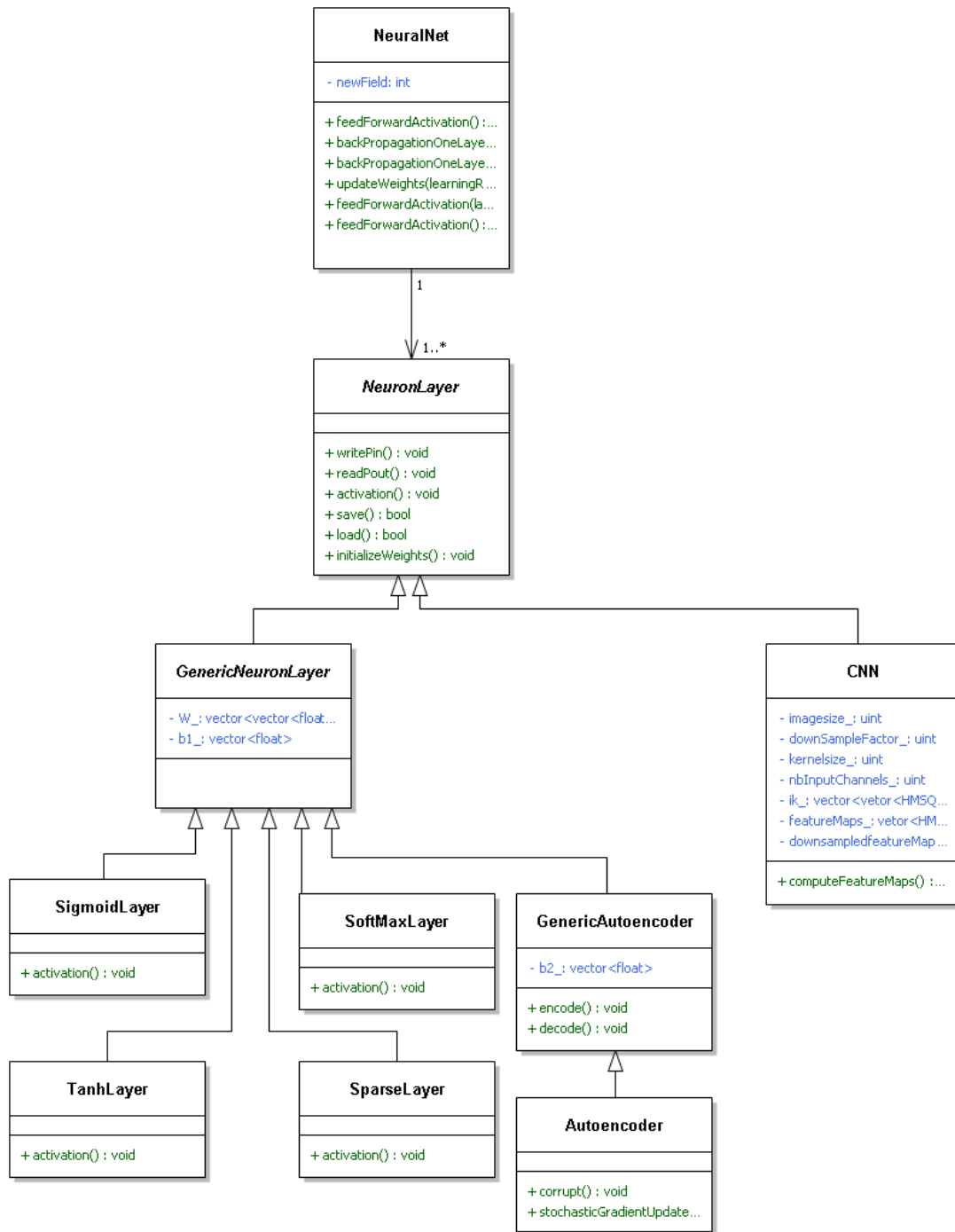


Figure B.2: Class diagram for the NeuralNets project

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad (B.1)$$



represents a network with 3 layers, the first being connected to the second, and the second being connected with the third. With this in mind, backward connexions (for performing backpropagation) can be found very quickly by simply looking at the transpose of the connexion matrix.

**B.4.2.1.2 The NeuronLayer class.** This class implements the individual neuron layers that compose the networks. As shown on Figure B.2, all the different neuron classes inherit from the virtual NeuronLayer class. Generally speaking, all neurons have to implement the following methods:

- `writePin`: copy the argument input to the neuron layer input.
- `readPout`: returns the output of the neuron.
- `activation`: computes the activation of the neuron, depending on the neuron type (all the activation functions presented in Section 3.3.3.1 have been implemented, except the ReLU function. It can however be implemented straightforwardly if necessary).
- `save/load`: self-explanatory.
- `initializeWeights`: initialize the weights of the neurons by random uniform sampling in the  $[-0.01, 0.01]$  interval. It is also called by the default constructor.

#### **B.4.2.2 How to train a NeuralNet**

A neural net has first to be initialized through the function *loadNetwork*. This function takes as input the absolute adress of a root path as well as the relative address of the network architecture file (.txt file) in that folder. The information that shall be specified in that file are:

- `neuronnumber`: self-explanatory.
- `connexions`: the connexion matrix (specified by square brackets and lines separated by vertical bars).

- `numberofbottomlayers`: self-explanatory (usually 1).
- `bottomlayerids`: a vector containing the indexed of the bottom layers (specified by square brackets).
- `inputdimension`: the total input dimension of the network.
- `outputdimension`: the total output dimension of the network.
- `neurontags`: a vector of size `neuronnumber` indicating the nature of the neuron layers (SigmoidLayer (tag 0), Autoencoder (tag 1), SoftMaxLayer (tag 2), TanhLayer (tag 3), SparseLayer (tag 4) or CNN (tag 5)).
- `loadneuronsfromfile`: a boolean vector of size `neuronnumber` indicating if the neurons shall be initialized from file (in which case a `.nnl` file with correct syntax (see below) and input/output dimensions shall be provided in the same folder). Otherwise, the neurons shall be randomly initialized.
- `neuroninputdimensions`: an integer vector containing the neurons input dimensions.
- `neuronoutputdimensions`: an integer vector containing the neurons input dimensions.

After the network is initialized, training is performed by applying a number of updates to the network (See Section 3.3.3.1). This is implemented in the body of the `train` function. This function is overloaded, as different inputs are required depending on if the first layers are convolutional (in which case the input has to be of type `HMSQMat`) or not (`float`). For each of these updates (to which randomly sampled examples are drawn), the `train` function successively calls `writePin`, `feedForwardActivation`, `backPropagation` and `updateWeights`. Then, once a number of updates (usually a number of epochs through the whole training set) are applied, the network is saved using `saveNetwork`. Note that there is an automatic checking, at each layer, of the correspondence of the provided input and expected format and size for that layer. for example, it is theoretically possible to feed a CNN layer a vector of floating point values,

or to feed a 100-dimensional layer a 1000-dimensional input vector. The layer simply won't process the data (if readPout is used on release mode, the output vector will be filled with zeros).

## B.4.3 The Real Time project

### B.4.3.1 Project overview

Class diagram for the SDM project can be seen on Figure B.3. The main elements composing this project are the Tracker, emAnalyzer, Displayer and GNDFCascade classes.

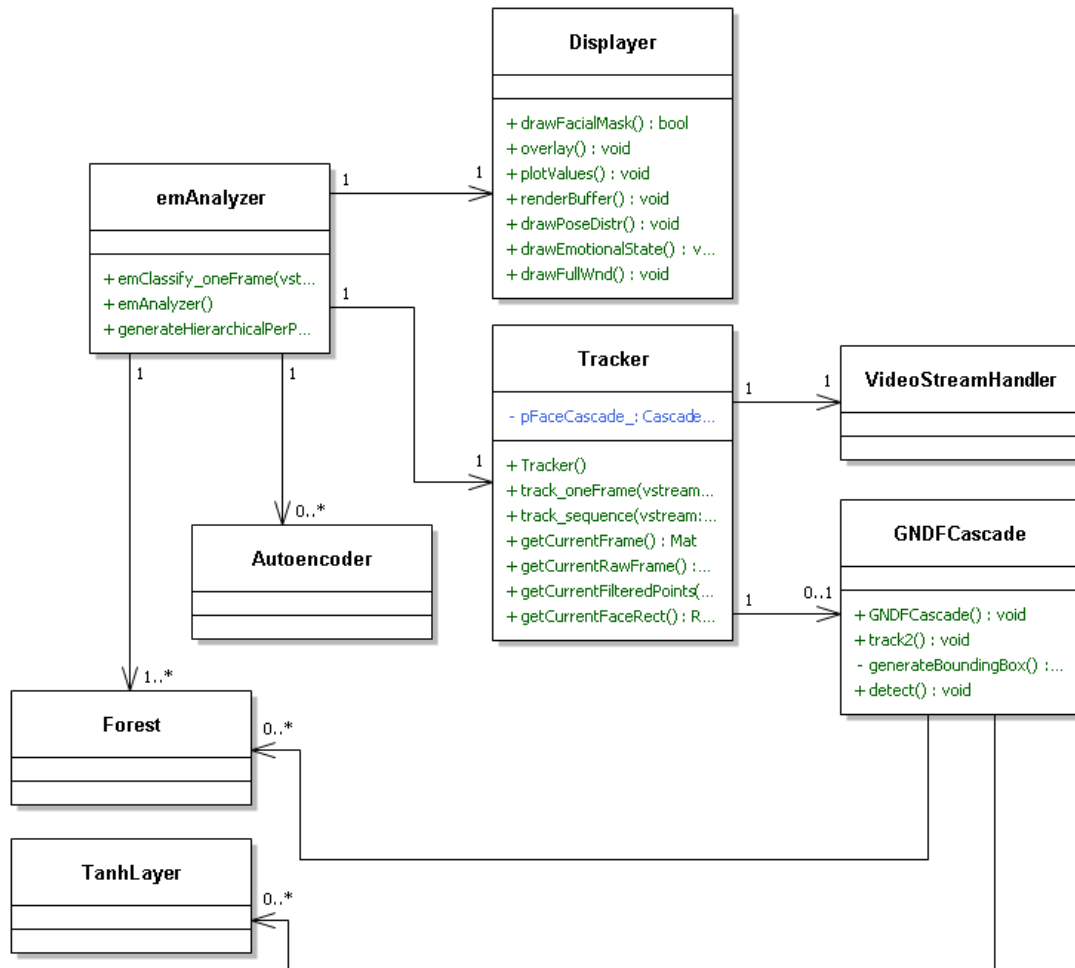


Figure B.3: Class diagram for the Real Time project

The `Tracker` class contains methods for detecting the faces within the images using OpenCV `CascadeClassifiers`. It also encapsulates an instance of class `GNDFCascade`, whose `detect` and `track2` methods essentially performs face alignment using the CSP-dGNDF method introduced in Section 6.3.2, respectively using a provided face bounding box and previous feature point estimation. `track2` also returns the score function that is used to control the alignment quality to re-perform face detection, if necessary. Also note that all low-level operation (camera driver handling and image retrieval from video stream) are encapsulated within the `VideoStreamHandler` class.

Another important class is the `emAnalyzer` class, which stores the Random Forests models for categorical expression recognition (using PCRF 4 and WLS-RF 5 - the per point confidence measurements can be obtained from the image and feature point location by calling function `generateHierarchicalPerPointConfidence`). AU occurrence prediction and confidence measurements can also be obtained using the method described in Section 5.4. All the rendering functions are encapsulated in class `Display`.