Implementation and Study of Cascaded-Regression Methods for Facial Feature Points Detection

BEng Individual Project Report

Author: Andrej Maris

Supervisor: Dr. Stefanos Zafeirou

Second Marker: Dr. Ben Glocker

June 2015
Abstract

Detecting facial feature points, or Face Alignment, is the problem of detecting semantic facial points in an image or a video, such as points around eyes, nose, mouth or jaw. Cascaded-regression methods for Face Alignment are regression-based methods that work in a stage-by-stage cascade, iteratively improving an initial shape estimate in a coarse-to-fine manner. The goal of this work is to provide an open source implementation of various cascaded regression-based Face Alignment methods in Python and make them available within the i-BUG research group at Imperial College London, but also for the general public via GitHub. My implementations achieved superior performance on the LFPW and Helen datasets compared to implementations of two other state-of-the-art techniques, namely an Active Appearance Model-based approach and the Supervised Descent Method. Analysis and comparison of these algorithms are provided in this report.
Acknowledgements

I would like to thank Dr Stefanos Zafeirou for supervising my project, as well as his PhD students Patrick Snape and Joan Alabort for their guidance and help throughout my work. Without their willingness and ability to help, I could not have finished this project successfully. Lastly, I would like to thank my parents and girlfriend Elizabeth for their continuous support and unconditional love.
# Contents

1 Introduction ............................................. 3  
1.1 Problem definition ..................................... 3  
1.2 Motivation and objectives for this work ................. 6  

2 Technical preliminaries .................................. 7  
2.1 Image processing and Computer Vision .................. 7  
2.2 Machine Learning preliminaries .......................... 9  
  2.2.1 Supervised learning .................................. 9  
  2.2.2 Feature extraction .................................. 10  
  2.2.3 Problem classes ...................................... 10  
  2.2.4 Model construction .................................... 11  
  2.2.5 Machine Learning for Face Alignment ................. 13  
  2.2.6 Decision trees ....................................... 14  
  2.2.7 Boosting ............................................ 15  
  2.2.8 Random Fern ......................................... 16  
2.3 Face detection ......................................... 18  
  2.3.1 Viola-Jones Face detector ............................ 18  
    2.3.1.1 Feature extraction ............................... 19  
    2.3.1.2 Integral image .................................. 19  
    2.3.1.3 AdaBoost for feature selection and training ..... 21  
  2.3.4 Cascaded framework .................................. 22  
  2.3.5 OpenCV implementation ............................... 22  

3 Face alignment methods .................................. 23  
3.1 Active Appearance Model-based methods .................. 23  
  3.1.1 Active Appearance Models ............................ 24  
  3.1.2 AAM for Face Alignment .............................. 25  
3.2 Regression-based methods ................................ 27  
  3.2.1 Cascaded shape regression ............................ 27
## 4 Implemented methods

4.1 Face alignment by Explicit Shape regression ........................................ 29
  4.1.1 Training a stage regressor ......................................................... 29
  4.1.2 Shape-indexed local features ..................................................... 32
  4.1.3 Two-level boosted regression .................................................... 33
  4.1.4 Correlation-based feature selection .......................................... 34

4.2 One millisecond Face Alignment with an Ensemble of Regression Trees ... 35
  4.2.1 Selecting splits ........................................................................... 35
  4.2.2 Exponential prior distribution of selected features ....................... 37

4.3 Face alignment by regressing Local Binary Features ............................ 38
  4.3.1 Feature extraction ....................................................................... 38
  4.3.2 Global linear regression ............................................................... 40

4.4 Implementation details ........................................................................ 41
  4.4.1 Class hierarchy ............................................................................ 41
  4.4.2 Third party libraries ..................................................................... 42
  4.4.3 Testing ........................................................................................ 43

## 5 Evaluation

5.1 Alignment accuracy analysis ............................................................... 45

5.2 Computational Efficiency ................................................................. 49

5.3 Challenges ......................................................................................... 50

## 6 Conclusion and future work

6.1 Contributions ................................................................................... 51

6.2 Further extensions ............................................................................. 52
Chapter 1

Introduction

1.1 Problem definition

The problem of face alignment concerns localising facial feature points (also called facial landmarks) in an image or a video. Typically, 17, 29 or 68 such points are elected to be searched for. Examples of such landmarks are points located around eyes, nose, lips or the jaw. These areas carry the most amount of semantic information for discriminative and generative purposes. [1]

The sought-after facial feature points are typically represented as a shape vector \( s = (x_1, y_1, x_2, y_2, \ldots, x_n, y_n) \) where \((x_i, y_i)\) is the position of the \(i\)-th landmark within an image and \(n\) is the number of landmarks that we wish to detect. The objective of face alignment is to produce such a shape vector from a given image.

Figure 1.1: The famous picture of Lenna annotated with facial landmarks. Image source: built-in menpo[2] assets.
1.1. PROBLEM DEFINITION

Various real world and research applications would benefit from accurate and efficient facial feature point detection methods [1]. These include face recognition of pre-learnt faces in unseen images, face hallucination (synthesising a high-resolution facial image from a lower resolution one [3]), face animation (real-time animation of a graphical avatars expression based on real human facial expression [4]), as well as affect analysis and facial expression recognition, which are widely studied within the Intelligent Behaviour Understanding Group (iBUG) at Imperial College London. Finding fiducial points on objects is not limited to faces - for example techniques such as Active Appearance Models and Active Shape Models are widely utilised in the medical imaging community [1].

Detecting facial landmarks is an extremely challenging problem due to the high variability in facial appearance (everyone has a unique identity), positions of faces in the image, rigid transformations (scale, rotation and translation) as well as non-rigid facial deformation (variability in person’s expression in the image). Furthermore, illumination conditions, image resolution or partial occlusion of faces by other objects greatly influence the complexity of the task.

Figure 1.2: Examples of challenging images (annotated) with significant occlusion, poor illumination conditions, rigid and non-rigid deformations. Source: Helen dataset[5].
Typically the problem of face alignment assumes an image with an annotated bounding box which has been detected to contain a face. These can be found using an off-the-shelf face detector, such as the one implemented in OpenCV based on Viola-Jones[6] or a HOG-based (Histogram of Oriented Gradient) detector found in the dlib library [7].

![Original input image](image1) ![Face detection](image2) ![Face alignment](image3)

Figure 1.3: Illustration of an original image, result of a face detector and the result of face alignment. Source: Helen dataset[5].

Thus, a face alignment pipeline normally starts with a face detector, which takes the input image and outputs bounding boxes containing faces. The image together with bounding boxes are then fed to the face alignment component, that returns a facial shape.

The problem of face alignment is typically approached by supervised machine learning, whereby a model is trained from a large amount of human-labeled images and can then be used for facial shape estimation on unseen images.

There are popular state-of-the-art approaches for face alignment currently studied. In section 3, we will describe some of the most used ones, but we shall only have a closer look at regression-based methods 3.2, namely those that use a cascaded shape regression framework first proposed by [8]. As opposed to other methods, these progressively refine an initial shape estimate in several stages directly from appearance, without learning any parametric shape or appearance models.
1.2 Motivation and objectives for this work

In academia, research papers often claim significant improvements over state of the art without offering a reference implementation of the proposed methods. The objective of my project was to study, implement and analyse the following research papers in Face Alignment that lack original implementations:

1. Face Alignment by Explicit Shape Regression by Cao et. al. [9]
2. One Millisecond Face Alignment with an Ensemble of Regression Trees by Kazemi et. al. [10]
3. Face Alignment at 3000 FPS via Regressing Local Binary Features by Ren et. al. [11]

These will be made available open source for the i-BUG and GitHub community. Furthermore, I shall run various experiments analysing the efficiency of the proposed methods.

To achieve good performance, it is important to find good parameters which are often not sufficiently described in the research papers. For example, in my project, I managed to improve mean alignment error by 9% just by changing the way of data set augmentation. Thus, apart from writing an implementation, the goal of this project is also to find the best possible such parameters.
Chapter 2

Technical preliminaries

2.1 Image processing and Computer Vision

Informally, an image is a two-dimensional matrix of pixels. A pixel is a numerical representation of a colour sampled in a particular place.

A colour model is a way of describing colours. These are usually represented as tuples of numbers, typically as three or four colour components, also called channels. In practice, an RGB model is used most often, whereby each colour is encoded as a composition of a red, green and blue channel. This would be represented as triple (r, g, b), where r, g, b are within a certain range and correspond to the intensities of red, green and blue respectively.

Figure 2.1: Illustration of how an image is represented as a grid of pixels. Individual pixels can be seen on the right. Source of image: LFPW dataset[12].
2.1. IMAGE PROCESSING AND COMPUTER VISION

**Image processing** deals with extracting low-level information from images by treating them as signals. In fact, image processing is often regarded as a subfield of signal processing and uses a large amount of the same techniques (e.g. convolution, ...). This kind of processing can be used to detect edges or perform blurring.

![Figure 2.2: Examples of an LFPW[12] image processed by convolution filters. Original image is depicted on the left, the middle one contains detected edges and the last one has been blurred by Gaussian blurring.](image)

**Computer Vision** is a subfield of computer science that deals with techniques for processing, analyzing, and understanding images in order to extract numerical and symbolic information in order to make decisions [13]. This usually involves finding more high-level information than image processing, although image processing techniques are often used before applying computer vision techniques - e.g. Gaussian blurring is often applied as a pre-processing step to smooth out random spikes in pixel intensities within the image. Face alignment is a computer vision problem, as it involves finding facial points with semantic (high-level) information.
2.2 Machine Learning preliminaries

Machine learning is a field of computer science that explores the construction and study of algorithms that can learn from and make predictions on data [14]. It extensively uses ideas and techniques from statistics, mathematical optimisation, linear algebra and logic. Machine learning is used in both research and industry, examples including search engines, spam detection, computer vision, object detection, weather prediction etc. Depending on the presence of training data, machine learning is broadly classified as supervised and unsupervised. As face alignment typically uses supervised learning, we shall explore the topic more closely in the following section.

2.2.1 Supervised learning

The goal of supervised learning is to take advantage of large amounts of training data to create statistical data models which can then be used for prediction, inference or classification.

Generally, we study a relationship between some independent (observable) variables, or features, and target variables. Features are extracted from the input data and are meant to be discriminative descriptors of the data. Those are typically grouped to form a feature vector, which represents the input data. For an image, i.e. a grid of pixels, an example feature could be simply number of white pixels, although this would be a poor descriptor of the image for most purposes.

The training data, consisting of a set of tuples of the form (feature vector, target), is analysed by the supervised learning algorithm to infer a function (also referred to as “model”) capable of mapping new examples. If the model generalises well, it can correctly predict correct values for unseen instances in most cases.

Based on how the models work, they can be divided to [15]:

- **generative models** that model the joint probability distribution $p(\text{features, target})$ and thus are capable of generating new data points by sampling from this distribution.

- **discriminative models** model the conditional probability distribution $p(\text{target} | \text{features})$, which is what we usually care about, thus are preferred to generative models.

- **discriminant function** directly learns a model $\text{target} = F(\text{features})$ without constructing probability models. This category includes the methods introduced in 1.2.
2.2. MACHINE LEARNING PRELIMINARIES

2.2.2 Feature extraction

In the case of face alignment, the raw input to the problem are images, i.e. large matrices of pixel intensities. In reality, these are rarely worked with directly, thus different kinds of features might be extracted such as differences in pixel intensities within a subset of pixels or average intensities in certain regions etc. This is to achieve invariance to common forms of image variation, such as illumination, rigid transformations and camera noise or to simply reduce input space. After extraction, all the computations are performed on the features rather than the original input data, thus it is desirable for them to be as descriptive as the original data with respect to the problem.

The choice of features greatly influences the predictive power of a model and thus is a crucial decision before designing the model. It is often challenging to derive features which are both informative and non-redundant (e.g. merely functions of other features), which is important for performance reasons.

Popular features for face alignment are differences in pixel intensities, HOG (Histograms of Oriented Gradients) and SIFT (Scale-Invariant Feature Transform) features. In the methods presented in this work, we shall only discuss pixel-intensity differences and features derived from them.

2.2.3 Problem classes

Supervised learning problems can be classified with respect to the target variable they are solving for to several classes. In this thesis, we will be mentioning:

- **classification** problems, in which the goal is to classify a given data point to two or more discrete classes (e.g. classifying an email message as spam/not-spam). An algorithm which implements a mapping from independent variables to the corresponding category is called a classifier.

- **regression problems**, whereby a continuous target variable is being predicted from a set of observed variables (e.g. predicting a price of a house based on its size, age, proximity to the city centre etc.). Similarly to the previous case, a regressor is an implementation of a solution to the regression problem.
2.2.4 Model construction

The pre-labeled dataset available during the design of a machine learning model is usually divided into a training and testing set. The former one is used to construct the model during training phase and the latter to benchmark its performance (testing phase).

To illustrate construction of a machine learning model, let us think of the problem of predicting house prices. The features might include size of the house in \( m^2 \), age of the house in years and proximity to the nearest city centre. Thus, the data points in our training set are of the form \( x = (\text{size}, \text{age}, \text{proximity}) \). The target variable would be the actual price in pounds sterling. Based on a data set with already estimated prices (e.g. by a real estate agent), this could be modelled using a linear regression model which has the form:

\[
t = w_1 \times \text{size} + w_2 \times \text{age} + w_3 \times \text{proximity} + \epsilon = w \cdot x + \epsilon
\]

where \( \epsilon \) is an error term. Our goal now would be to find a vector of parameters \( w = (w_1, w_2, w_3) \) such that the price model would be a good approximation of the real prices of houses. One way of doing it would be maximum likelihood estimation, whereby we find parameters which maximise the likelihood of observing the training data.

We start with an assumption about the distribution of the error term \( \epsilon \), namely that it is distributed normally with variance \( \beta \). Thus, the value of the target variable \( t \) has a Gaussian distribution centred around the value of \( y(x, w) = w_1 \times \text{size} + w_2 \times \text{age} + w_3 \times \text{proximity} \), i.e.

\[
t = N(y(x, w), \beta)
\]

for unknown parameters \( w \) and \( \beta \) (where \( \beta \) is the variance of the distribution). To find the unknown parameters, we shall use maximum likelihood estimation.

Assuming independent observations within the training set, the likelihood of observing \( (x_1, t_1), (x_2, t_2), \ldots (x_n, t_n) \) is given by

\[
p(t|x, w) = \prod_{i=1}^{N} p(t = t_i|x_i, w)
\]

(2.2)
2.2. MACHINE LEARNING PRELIMINARIES

Figure 2.3: Illustration of data points sampled from a polynomial with normally distributed error.

To find parameters that maximise the likelihood, we shall maximise its logarithm:

\[
\ln p(t \mid x, w) = \sum_{i=0}^{n} \ln p(t = t_i \mid x_i, w) = \sum_{i=0}^{n} \ln N(t_i \mid y(x_i, w), \beta) = -\frac{\beta}{2} \sum_{i=1}^{n} (y(x_i, w) - t_i)^2 + \frac{n}{2} \ln \beta - \frac{n}{2} \ln (2\pi) \tag{2.3}
\]

Clearly, this function can be maximised by minimising \( \sum_{i=1}^{n} (y(x_i, w) - t_i)^2 \) which is a familiar formula for “sum of square errors”.

To simplify calculations, we shall regard all feature vectors in the training set as columns of a matrix \( X = (x_1, x_2, \ldots, x_n) \) and all corresponding targets variables into vector \( t = (t_1, t_2, \ldots, t_n) \). Now, our objective is to minimise:

\[
S(w) = \|t - X \cdot w\| = t^T t - 2w^T X^T t + w^T X^T X w \tag{2.4}
\]
In order to find the minimum, the equation 2.4 can be differentiated w.r.t. $w$ and set to 0.

$$S'(w) = -X^T t + (X^T X)w = 0$$
\hspace{1cm} (2.5)

Rearranging the equation 2.5 we obtain

$$w = (X^T X)^{-1} + X^T t$$
\hspace{1cm} (2.6)

Now, the vector of parameters $w$ can be used in our linear regression model to predict prices for any house with features $x$.

### 2.2.5 Machine Learning for Face Alignment

In the context of face alignment, the training phase consists of extracting feature vectors out of images and learning a model using the feature vectors and human-labeled true shapes, as depicted on the following diagram. The model can be then used for computing facial shapes for unseen images.

![Diagram of training and testing a model in the context of face alignment.](image)

Figure 2.4: Illustration of training and testing a model in the context of face alignment.
2.2.6 Decision trees

Decision trees are usually used to implement classifiers. The input to the tree is a feature vector, which is fed into the tree’s root and proceeds all the way down into a leaf node. Each internal node of a tree comprises a “question” and the branch to be followed depends on the answer. Note this is not necessarily a binary tree (i.e., the question is not necessarily a yes/no question). An example of a decision tree is depicted below, which models the probability of a customer buying a certain product on an e-commerce website:

![Decision Tree Diagram](image)

Figure 2.5: Example of a decision tree in an e-commerce setting. The internal nodes of the tree test particular features of a customer and the leaf nodes contain probabilities whether he/she will or will not purchase a product from the e-store.

Each internal node contains a feature to be tested and each leaf contains a probability histogram, that assigns to each data point falling into the leaf the chance of being of each class. The probability histograms are inferred from the frequencies of observing each class during training time. To train a decision tree, the important decision to make is which features to split on and thresholds to compare them with. In the methods described in this thesis, the objective for splitting is maximising variance reduction, that chooses the split in each internal node so that the sum of variances in the children nodes is as small as possible. This will be further described in section 4.2.1.
2.2.7 Boosting

Boosting was first applied to the problem of classification, whereby using an ensemble of weak classifiers could “boost” them to produce a strong classifier. Weak classifiers are broadly defined to be “better than random guessing”, i.e. very simple models which work with probability more than 0.5. However, combination of their outputs turns out to work well in practice.

An example of a classification problem that can be efficiently solved using ensemble methods is spam detection in electronic mail systems. Given a training set comprising emails labeled as spam/non-spam, we wish to classify unseen messages into one of those categories.

Consider a weak classifier that classifies a given email message as spam if it contains the
2.2. MACHINE LEARNING PRELIMINARIES

string “money”. Obviously, this would catch a great amount of spam messages, but would also have a large false positive error rate, marking a non-spam message as spam.

However, combining enough weak classifiers of this kind can give provable guarantees of good results.

Figure 2.7: Diagrammatic explanation of spam detection using an ensemble of weak decision trees. Each decision tree performs classification based on a simple feature test and the results are composed to produce the final class.

2.2.8 Random Fern

A fern is another type of classifier usually used as a weak classifier in an ensemble model. A random fern works in a similar way as a decision tree, with two important distinctions:

1. All “questions” are binary tests that compare a feature \( F_i \) with a threshold \( T_i \).

2. Each input data point is asked all the questions. This results in a boolean string which represents the class of the input, or a bin. Thus, a fern with \( N \) questions bins its input into one of \( 2^N \) bins.

Formally, a fern is a sequence of \( N \) feature-threshold pairs \((F_i, T_i)\). A fern can be visualised as balanced binary decision tree, whereby nodes at the same levels contain the same questions.
CHAPTER 2. TECHNICAL PRELIMINARIES

Figure 2.8: Fern can be viewed as a degenerate decision tree, whereby each sample is tested on the very same set of questions.

The construction of a random fern consists of selecting the feature-threshold pairs \((F_i, T_i)\) and having each data point from the training set evaluated with the fern. The output of the evaluation is a binary string which represents a bin. Originally, the fern was designed to be used for classification purposes. In such a case, each bin trained to store a histogram of frequencies of each class falling into the bin. During the test phase, these are used as probabilities that the a datapoint falling into the particular bin is of a particular class.

The fern can also be used as a regressor. In this manner, each bin could be trained to store the average value of all target variables falling into the bin.
2.3 Face detection

Face detection is most of the times a necessary preprocessing step for face alignment, as most of the regression models assume a bounding box as an input, although there are now methods that perform detection and alignment jointly, such as [16].

A notable face detector developed in the 2000’s is the seminal work by Viola and Jones [6], which is also implemented in OpenCV. In my experiments, I used both OpenCV’s face detector as well as the one found in the dlib library [7].

2.3.1 Viola-Jones Face detector

The Viola-Jones detection framework was the first object detection framework to provide competitive detection rates in real-time [17]. Although it can be applied for general object detection, the primary motivation behind the method was face detection.

The detection algorithm uses a sliding window approach - the algorithm essentially asks “Does this region contain a face?” for many possible regions within the image. As there are many candidate regions, this classification task has to be handled extremely fast.

Figure 2.9: The Viola-Jones face detector is a sliding-window algorithm that classifies rectangles of different sizes as either “face” or “non-face”.
2.3.1.1 Feature extraction

Due to the need for high-speed classification of each region, the features has to be simple and fast to extract. Viola-Jones uses Haar-like features that involve sums of pixel intensities within rectangular regions. They exploit the fact that certain regions of a face have higher intensity than others, e.g. the eye region is generally darker than upper part of the cheeks.

Different Haar features are depicted as rectangles of black and white areas. To calculate the value of a particular feature, sum of all pixels within white regions is subtracted from sum of pixels within black regions. These features can be computed extremely fast using a pre-processing technique called integral image.

![Haar-like features illustration](image)

Figure 2.10: Illustration of Haar-like features on an image from LFPW[12]. The value of the shown feature equals the sum of pixel intensities in the lower (white) region subtracted from the upper (black) region. This feature relies on the fact that the eye region is darker than the region containing upper cheeks and nose.

2.3.1.2 Integral image

For a given image of interest $I$ of dimensions $W \times H$, an associated integral image is a two-dimensional array $P$ of dimensions $(W + 1, H + 1)$ that contains 2D cumulative sums of pixel intensities at each element, i.e.

$$P[i,j] = \sum_{k=0}^{i-1} \sum_{l=0}^{j-1} I(k,l)$$  \hspace{1cm} (2.7)
2.3. FACE DETECTION

Figure 2.11: Haar-like feature taking advantage of the fact that the eye region is generally
darker than the bridge of the nose. Source: Helen dataset[5].

Figure 2.12: Illustration of the definition of the integral image $P$, that contains 2D
cumulative sums of pixel intensities. The value $P[m, n]$ contains the sum of pixel intensities
in the grey region, while $P[i, j]$ contains the sum of intensities in the white dotted rectangle.
Source of background image: LFPW dataset[12].

where $I(k, l)$ is the pixel intensity at coordinates $[k, l]$ (assuming a greyscale image).
This array can be computed in linear time w.r.t. the number of pixels, i.e. using only one pass through $I$. The following recursive formula can be used:

\begin{align}
P[i, 0] &= 0 \\
P[0, j] &= 0 \\
P[i, j] &= P[i][j - 1] + P[i - 1][j] - P[i - 1][j - 1] + I[i - 1][j - 1]
\end{align}

(2.8)

The formula uses the concept of dynamic programming - we calculate the value of $P[i, j]$ using solutions to subproblems $P[i][j - 1]$, $P[i - 1][j]$ and $P[i - 1][j - 1]$.

The integral image $P$ can now be used for quickly calculating sums in any rectangular region in the original image. In fact, this can be achieved in $O(1)$ time as follows.

$$\text{sum}([a, b], [c, d]) = P[c, d] - P[a, d] - P[c, b] + P[a, b]$$

Figure 2.13: Pictorial representation of how sum inside rectangle $(a, b), (c, d)$ can be computed.

### 2.3.1.3 AdaBoost for feature selection and training

As the number of possible Haar-like features is large, only relevant ones have to be selected. For feature selection, Viola-Jones uses the AdaBoost architecture, or **AdaBoost**, a method proposed in [18] to improve performance of classification algorithms.
As described in section X, boosting involves training a large number of weak classifiers which are assembled to create a strong one. Each weak classifier compares a particular Haar-like feature with a threshold value to decide the class. It consists of a feature $f_j$, a threshold $\theta_j$ and a parity $p_j$ that indicates the direction of the sign of the inequality [6]:

$$
h_j(x) = 1 \text{ if } f_j(x) < p_j \theta_j \\
h_j(x) = 0 \text{ otherwise}
$$

Training one weak classifier involves selecting a feature $f_j$ and finding a suitable threshold $\theta_j$. The algorithm produces an ensemble of $T$ weak classifiers in $T$ steps. In each step, a separate weak classifier is trained for each possible Haar-like feature. The one with the lowest detection error is chosen as the result for that step.

The detection error is a sum of detection errors of all images, but each detection error is weighted differently. Every image in the training set has a different weight, which is maintained and calculated throughout the whole training process. The is the adaptive part of the AdaBoost approach - later weak classifiers in the ensemble are tweaked to account for those input instances that were misclassified by former weak classifiers. In order words, previously misclassified images are given more weight so that subsequent weak classifiers will “concentrate more to get those right”.

### 2.3.1.4 Cascaded framework

The weak classifiers are grouped into gradually more complex (stronger) cascades. At testing time, if a window does not pass a strong cascade, it is immediately classified as negative. As these strong groups of weak classifiers are sorted with respect to their complexity, the very first ones can quickly discard windows not containing faces. Thanks to this early-stopping approach, the whole ensemble of cascades is passed through only for positive windows, which makes this method runnable on videos in real-time.

### 2.3.1.5 OpenCV implementation

The OpenCV library contains functionality to implement a face detector based on the Viola-Jones method. Pre-trained models are available in XML format which can be loaded.

```python
import cv2
face_detector = cv2.CascadeClassifier("frontalface_model.xml")
image = cv2.imread(IMAGE_PATH)
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
boxes = face_detector.detectMultiScale(gray, 1.3, 5)
```
Chapter 3

Face alignment methods

There are two main streams of methods for face alignment - methods based on Active Appearance Models [19] that build parametric models of appearance and regression-based models, that directly model a mapping from appearance to shape.

The three implemented methods are based on regression. In this chapter, I will briefly describe Active Appearance Models and regression-based methods with the emphasis on cascaded shape regression.

3.1 Active Appearance Model-based methods

Methods based on Active Appearance Models [19] jointly build generative models of both texture and shape. Thus, a deformable model is learnt from training data, which is capable of generating/synthesising (even unseen) faces by tweaking the parameters of the model. When aligning a face in an unseen image, the model is deformed to fit the observed face. This is often referred to as “analysis-by-synthesis”.

In other words, for a given image, AAM-based methods seek to find a vector of parameters \( \theta \) such that the output of the model \( M(\theta) \) best matches the appearance of the given face, where the “best match” is defined as minimising the texture difference between the synthesised face and the given one. This set of parameters can be then used as a descriptor of the observed face and can be used for further processing (recognition, classification ...).
3.1. ACTIVE APPEARANCE MODEL-BASED METHODS

Figure 3.1: Example of a synthesised face (on the right) by an AAM, given an image on the left. Source: [19]

3.1.1 Active Appearance Models

The Active Appearance Model combines a model of shape variation with a model of appearance variations. These are learnt from a training set of face images annotated with main facial landmarks.

Before the shape model is built, all shapes are first aligned into a common coordinate frame using Procrustes Analysis. The shape model is then found by applying Principal Component Analysis on the normalised shapes and has the following form:

\[ s = \bar{s} + \sum_{i=1}^{n} \alpha_i s_i = \bar{s} + Ps\alpha \] (3.1)

where \( \bar{s} \) is the mean of all normalised training shapes, and \( s_i \) are principal components. As the model generates normalised shapes, to synthesise one in the image frame rigid transformation must be applied to \( s \).

To build a model of the greyscale texture, each image in the training set is warped to the mean shape. From the shape-normalised image, appearance is sampled from the region covered by the mean shape. Again, principal component analysis is applied to obtain the following model:

\[ a = \bar{a} + \sum_{i=1}^{n} \beta_i a_i = \bar{a} + Pa\beta \] (3.2)

where \( \bar{a} \) is the mean of appearance vector and \( a_i \)-s are principal components.
The relationship between the shape and appearance model is modelled by applying PCA on concatenated vectors of the form:

\[
\begin{bmatrix}
W_s \alpha \\
\beta 
\end{bmatrix} = \begin{bmatrix}
W_s P_s^T (s - \bar{s}) \\
P_a^T (a - \bar{a})
\end{bmatrix} = \begin{bmatrix}
Q_s \\
Q_a
\end{bmatrix} c \tag{3.3}
\]

where \( W_s \) is a diagonal matrix of parameters that captures the difference in units of the shape and texture models. Such concatenated vectors as in 3.3 are generated for each training sample. Applying PCA on these yields a further model:

\[
b = Q c \tag{3.4}
\]

where

\[
Q = \begin{bmatrix}
Q_s \\
Q_a
\end{bmatrix} \tag{3.5}
\]

The shape and texture models can be expressed directly as functions of \( c \)

\[
s = \bar{s} + P_s W_s Q_s c \tag{3.6}
\]

\[
a = \bar{a} + P_a Q_a c \tag{3.7}
\]

A face can be synthesised by varying the parameter vector \( c \). Plugging a particular \( c \) into equations 3.6 and 3.7 will result in a normalised shape and texture image in a mean-shape position. This has to be further warped using the points of the generated shape.

Synthesising a face that would match a given image is an optimisation problem with the objective of minimising square texture difference. To solve this, a linear regression model \( A \) is learnt to correct the parameters of the generative model based on the current alignment difference:

\[
\delta c = A (\hat{a} - a) \tag{3.8}
\]

where \( \hat{a} \) is the given image and \( a \) is the generated appearance. At test time, the equation 3.8 is applied in an iterative manner until the alignment manner is below a certain threshold.

### 3.1.2 AAM for Face Alignment

Generally, generative methods like AAM can achieve robust performance by using the model to constrain solutions to be valid examples of a face[19]. Also, another advantage is that an appearance model describes a given image with respect to a small set of model parameters.
3.1. ACTIVE APPEARANCE MODEL-BASED METHODS

Figure 3.2: Illustration of fitting an AAM model to an image. From left-to-right, the figure contains the initial estimate and estimates after 2, 8, 14 and 20 iterations respectively. The rightmost image shows the converged solution. Source: [19]

Figure 3.3: Comparison of the converged solution from figure 3.2 and the true appearance. Source: [19]

This representation can be then further used in other applications.

However, when it comes to face alignment, regression-based techniques are often preferable to AAMs, due to AAM’s sensitivity to initialisation and performance issues. The AAM method really solves a different, more general problem of matching a model with an image. However, they are widely used in many fields, such as medical image analysis.
3.2 Regression-based methods

Regression-based methods do not build any parametric models of shape/appearance, but merely study the correlations between image features to infer a facial shape. These methods directly learn a regression function from image features to the target facial shape [1]:

\[ M : \phi(Image) \rightarrow s \in \mathbb{R}^{2N} \]  

where \( M \) is the model, \( \phi(Image) \) is a function which extracts features from an image instance and \( s \) is the resultant facial shape. Examples of generally used features include pairwise pixel differences, Haar-like[6], SIFT[20] or HOG[21] features.

There are various face alignment methods based on regression. As the 3 methods studied in this thesis ([9, 10, 11]) are all based on cascaded shape regression, we shall study this framework more closely in the following section.

3.2.1 Cascaded shape regression

Many face alignment methods work in a cascaded framework whereby an ensemble of \( N \) regressors works in a stage-by-stage manner, which are referred to as stage regressors. This approach was first explored by [8].

At test time, the input to a regressor \( R_t \) at stage \( t \) is a tuple \((I, S_{t-1})\) where \( I \) is an image and \( S_{t-1} \) is the shape estimate from the previous stage (the initial shape \( S_0 \) is typically the mean shape of the training set). The stage regressor extracts features w.r.t to the current shape estimate and regresses a vectorial shape increment:

\[ S_t = S_{t-1} + R_t(\phi_t(I, S_{t-1})) \]  

where \( \phi_t(I, S_{t-1}) \) are referred to as shape-indexed features, i.e. they depend on the current shape estimate.

The cascade progressively infers the shape in a coarse-to-fine manner - the early regressors handle large variations in shape, while the later ones ensure small refinements. After each stage, the shape estimate resembles the true shape closer and closer.
3.2. REGRESSION-BASED METHODS

Figure 3.4: Diagrammatic representation of a cascaded shape regression framework. The first regressor is initialised with a mean shape. The alignment then progresses through the cascade of regressors in a coarse-to-fine manner. Source of included photo: LFPW dataset [12]
Chapter 4

Implemented methods

4.1 Face alignment by Explicit Shape regression

The Explicit Shape Regression method by Cao et al [9] uses a cascade of regressors to infer the shape as a whole and explicitly minimises the alignment error over the training data.

Each regressor in the cascade returns a vector which is used to update the current shape estimate in an additive manner. To achieve invariance to scale, the shape increment is returned normalised and has to be first transformed before the current shape estimate is updated.

4.1.1 Training a stage regressor

Before the training takes place, the mean shape is calculated out of all training shapes, which is rescaled and centred at the origin.

To train a regressor at one stage, each ground truth shape in the training set is first centred at the origin and then aligned with the mean shape using a similarity alignment $M_{t,i}$ (consisting of rotation and scaling only) that minimises the point-to-point alignment error between the two. Such a similarity transformation can be found by Generalised Procrustes Analysis. Operating in this “mean shape frame” is necessary to ensure scale-invariance.

For a regressor at stage $t$, the (normalised) target shape increment is

$$y_{t,i} = M_{t,i} \cdot (S_i - S_{t,i}) \quad (4.1)$$

where $S_i$ is the ground truth shape of training image $i$ and $S_{t,i}$ is the estimate at stage $t$. 

29
4.1. FACE ALIGNMENT BY EXPLICIT SHAPE REGRESSION

Figure 4.1: The average shape calculated from all training ground truth shapes, normalised and centred at origin.

Thus, each stage regressor is trained using tuples \((I_i, S_{t,i}, y_{t,i})\) where the target variable is the normalised shape difference \(y_{t,i}\). The objective of the training is to explicitly minimise the L2 alignment error, which is the same objective that we have at testing:

\[
R_t = \arg\min_R \sum_{i=0}^{N} ||y_i - R(I_i, S_{i-1})||_2^2 \quad (4.2)
\]

To ensure better generalization, the whole training dataset is augmented by perturbing the initial estimates of the initial stage regressor. In my implementations, I performed 20 perturbations of each image.

When testing, the current shape estimate is updated at each stage by the regressed normalised shape increment, which is transformed to the global coordinates using the corresponding inverse similarity alignment \(M_{t,i}^{-1}\):

\[
S_{t,i} = S_{t,i-1} + M_{t,i}^{-1} \cdot R_t(I_i, S_{i-1}) \quad (4.3)
\]
Figure 4.2: A pass through one stage regressor. Each stage regressor extracts shape-indexed pixel difference features from the given image and returns a normalised shape increment. The current shape estimate is updated with the regressed shape increment transformed to the frame of current shape using $M_{t;i}^{-1}$. Source of included photo: LFPW dataset [12]
4.1.2 Shape-indexed local features

To ensure invariance to illumination conditions, the features used in each regressor are differences in pixel intensities, extracted from the image based on the current shape estimate at each stage.

![Image](image_url)

Figure 4.3: The pixel-difference features are extracted locally w.r.t the nearest landmark on the mean shape. This makes these features invariant to pose and expression variations.

At training time, each stage regressor generates a random set of normalised pixel coordinates indexed relative to the nearest landmark on the mean shape. To extract features from a given image $I_i$ with current shape estimate $S_{t,i}$, each of the local pixel coordinates $(x_{l,i}, y_{l,i})$ (where $l$ is an index of the nearest landmark), is transformed by $M_{t,i}^{-1}$ to global coordinates of the image.

The way feature extraction in machine learning problems is done significantly impacts the predictive power of a constructed model. In this case, the reasons for choosing the aforementioned features are as follows:

- Local features close to facial landmarks are more discriminative than global ones.
- Generating pixel coordinates with respect to the mean shape achieves geometric invariance.
- Using differences in pixel intensities rather than absolute values achieves invariance to illumination conditions.
4.1.3 Two-level boosted regression

Typically, each regressor within the cascade is a weak regressor, such as a random fern as suggested by [8]. The Explicit Shape Regression framework, however, suggests another cascade of regressors (ferns in this case) at each stage of the outer cascade. The proposed model has 10 stages, each consisting of 500 random ferns of depth 5.

The pixel-difference features are extracted at each outer stage and kept constant at each fern. However, each fern selects only 5 of these features to infer a vectorial offset, based on the method outlined in the next section. The final output of each primary regressors is the sum of outputs of each fern.

Figure 4.4: Illustration of two-level boosted regression with an ensemble of ferns. Each fern within a stage receives the same shape-indexed features and performs correlation-based feature selection as described in section 4.1.4
4.1.4 Correlation-based feature selection

Ferns are trained using merely a subset of pixel-difference features extracted in the preceding regression level. In fact, ferns use only $F = 5$ out of $P^2$ ($P = 400$ in the implementation) features. There are two requirements for the $F$ features selected:

- Features carry as much discriminative information as possible
- Features are as independent to one another as possible

The suggested method is based on calculating the correlation between each feature and the regression targets (ground truth shapes). This is achieved by generating a random unit vector, projecting each target onto it and finding the Pearson correlation coefficient between feature values and lengths of projections.
4.2 One millisecond Face Alignment with an Ensemble of Regression Trees

This method outlined in [10] uses a very similar approach to the previous one ([9]) with a notable difference of using a decision tree as a primitive regressor instead of a random fern.

Thus, in this method, each stage regressor in the cascaded shape regression framework is an ensemble of regression trees (also called a Random Forest). The objective of building a decision tree is explicitly minimising the alignment error in the least squares sense, which is the same goal as in testing. The training of the decision tree is governed by three rules:

1. The optimal split in each internal node of the decision tree is chosen from a random pool of candidate splits such that it maximises the \textit{variance reduction} in the child nodes.
2. Each leaf node contains the mean of all training samples falling into the leaf multiplied by a regularization parameter \( \lambda = 0.1 \) in a multiplicative manner.
3. When choosing splits at internal nodes, rather than performing correlation-based feature selection, a pool of features are selected at random with an exponential prior distribution, biased towards pixel-pairs that are closer together. From this pool, features are further selected to maximise the variance reduction, as stated in point 1.

4.2.1 Selecting splits

Formally, a split is a tuple \((p, p', t)\) where \((p, p')\) are coordinates of two pixels whose difference is a feature, and \(t\) is a threshold this feature will be compared against. For example, the splitting decision at an internal node with a split \((p, p', t)\) might be that

\[
\text{if } (\text{image_intensity}[p_x, p_y] - \text{image_intensity}[p'_x, p'_y] < t) \\
\text{then continue in the left child} \\
\text{else continue in the right child}
\]

To determine a split at an internal node, a pool of candidate splits is generated \(\theta_{pool} = \{(p_1, p'_1, t_1), (p_2, p'_2, t_2), \ldots (p_K, p'_K, t_K)\}\) the split \(\theta^*\) which maximises the variance reduction in the child nodes is selected. This is simply the difference in variances between a node and its children. As the value stored in each leaf is the mean of all training samples within the leaf, we can note that maximising variance reduction is equivalent to minimising sum of square errors, which is also the objective at testing.
4.2. ONE MILLISECOND FACE ALIGNMENT WITH AN ENSEMBLE OF REGRESSION TREES

Suppose a sample of regression targets \( Y = \{y_1, y_2, y_3 ... y_N\} \) falling into a node and a candidate split \( \theta \) splitting the sample into sets \( L_\theta = \{l_{\theta,1}, l_{\theta,2}, ... l_{\theta,|L_\theta|}\} \) and \( R_\theta = \{r_{\theta,1}, r_{\theta,2}, ... r_{\theta,|R_\theta|}\} \). The variance within the node is, by definition:

\[
\sigma^2_v = \frac{1}{N} \sum_i^N (y_i - \bar{y})^2 \tag{4.4}
\]

and the variances of left and right children w.r.t. to split \( \theta \) are:

\[
\sigma^2_{\theta,l} = \frac{1}{|L_\theta|} \sum_i^{|L_\theta|} (l_{\theta,i} - \bar{l}_\theta)^2 \tag{4.5}
\]

\[
\sigma^2_{\theta,r} = \frac{1}{|R_\theta|} \sum_i^{|R_\theta|} (r_{\theta,i} - \bar{r}_\theta)^2 \tag{4.6}
\]

The problem of maximising variance reduction can be then simply formulated as

\[
\theta^* = \arg\min_{\theta} \left( \sigma^2_v - (\sigma^2_{\theta,l} + \sigma^2_{\theta,r}) \right) \tag{4.7}
\]

As the variance of the sample in node \( v \) is fixed, the problem is equivalent to

\[
\theta^* = \arg\min_{\theta} \left( \sigma^2_{\theta,l} + \sigma^2_{\theta,r} \right) = \frac{1}{|L_\theta|} \sum_{i=0}^{|L_\theta|} (l_{\theta,i} - \bar{l}_\theta)^2 + \frac{1}{|R_\theta|} \sum_{i=0}^{|R_\theta|} (r_{\theta,i} - \bar{r}_\theta)^2 \tag{4.8}
\]

After rearranging this equation to omit terms independent of \( \theta \), the final form of the splitting objective function is:

\[
\theta^* = \arg\min_{\theta} \left( |L_\theta| \bar{l}_\theta^T \cdot \bar{l}_\theta + |R_\theta| \bar{r}_\theta^T \cdot \bar{r}_\theta \right) \tag{4.9}
\]

The expression 4.9 has to be evaluated for each candidate split in \( \theta_{pool} \) at every internal node of the decision tree. To reduce computation time, we can observe that the average of the samples falling into the right child can be calculated from the average in the left child and the overall average, i.e.:

\[
r_\theta = \frac{|Y| \bar{y} - |L_\theta| \bar{l}_\theta}{|R_\theta|} \tag{4.10}
\]
4.2.2 Exponential prior distribution of selected features

As mentioned in the previous section, the pool of features $\theta_{pool}$ is selected at random. However, as features consisting of pixel pairs that are closer together tend to be more discriminative as of those that are further away, the pixel pairs are sampled from an exponential distribution that favours closer pixels:

$$P(p, p') = ke^{-\lambda||p-p'||}$$  \hfill (4.11)
4.3 Face alignment by regressing Local Binary Features

The third approach that I have studied is Face alignment at 3000fps via regressing Local Binary Features[11]. There are a few approaches in this method that significantly differ from the previous two:

1. This method proposes the idea of **tree-induced binary features** using a random forest. These so-called “local binary features” induced by an ensemble of regression trees are extracted at each stage of the cascade.

2. The random forest samples pixel-difference features within a region of “adaptive” radius - the radius gets progressively smaller in each stage.

3. To model the mapping between local binary features and normalised shape deltas, **multivariate linear regression** is used - at stage t, a a regression matrix $W^t$ is learnt which can then predict shape increments as follows:

$$\delta S_t = S_{t-1} + W^t \phi_t(I, S_{t-1})$$  \hspace{1cm} (4.12)

where $\phi_t(I, S_{t-1})$ is a function that extracts shape-indexed local binary features from image $I$ at stage $t$.

### 4.3.1 Feature extraction

As mentioned before, a random forest is used to induce local binary features. In this method, **each landmark is regressed independently**, i.e. there is a separate forest for each landmark of the facial shape. Each forest maps pixel-difference features to normalised landmark increments and is trained in the same way as described in 4.2 - to explicitly minimise sum of square alignment errors.

Once a forest is trained for each landmark, the local binary features can be extracted. For one landmark, a vector of local binary features is a D dimensional vector, where D is the overall number of leaves in the forest. The $i$-th element of the feature vector is 1 if the sample falls in $i$-th leaf of the forest, otherwise it is zero. Features of each landmark are then concatenated to produce the final feature descriptor - a large sparse vector of local binary features. This contains a large amount of information about the local texture around each landmark, which makes it a very discriminative descriptor.

At each stage of the cascade, each tree in the forest stores a vector offset in its leaf and thus
CHAPTER 4. IMPLEMENTED METHODS

Figure 4.5: Illustration of a regressor at one stage of the cascade. In this method, a random forest (in fact, a multiple of them - one per each landmark) is used to induce local binary features which are then regressed using a global linear regression matrix. This one returns a normalised shape increment as before.

is capable of regression. However, these vectors are discarded - the ensemble of trees is only used for inducing local binary features. These are then fed into a global linear regression matrix which returns a normalised vectorial increment for the overall shape. This is to take advantage of the global correlations between individual landmarks and enforce the shape constraint.

As opposed to the previous two methods (4.1, 4.2), each landmark is regressed independently by a separate random forest. This is because the features local to the landmark are less noisy than then in global learning, which makes it easier to select good features. As they are independent, the resulting features are more diverse and complementary to each other and thus are more appropriate for global learning in the second step [11].
4.3. FACE ALIGNMENT BY REGRESSING LOCAL BINARY FEATURES

Figure 4.6: The feature extractor consists of one random forest per each landmark. Pixel-difference features sampled around a landmark are fed to the corresponding random forest and depending on the paths taken, a binary string is constructed. Local binary features of all landmarks are then concatenated to produce the final feature vector. Source of included photo: LFPW dataset [12]

4.3.2 Global linear regression

The objective of learning the linear regression matrix $W^t$ at stage $t$ is to minimise the following loss function:

$$W^t = \arg\min_W \sum_{i=0}^{N} ||\delta\hat{S}_{t,i} - W\phi_t(I, S_{t,i-1})||_2^2 + \lambda||W||_2^2$$

(4.13)

where $\delta\hat{S}_{t,i}$ is the true increment for image $i$. As the number of local binary features is very large, overfitting can become a significant problem. Thus, the last term is a penalty term with a regularisation parameter $\lambda$.

At test time, the dot product $W^t\phi_t(I, S_{t,i-1})$ can be computed extremely fast - as the binary features are sparse, it is sufficient to directly index into the matrix $W^t$ to obtain the columns where the corresponding binary feature is 1 and sum these up. This will yield the regression output without having to explicitly calculate matrix multiplication. This allows the method to run in real-time at 3000 frames per second.
4.4 Implementation details

4.4.1 Class hierarchy

As all 3 methods are based on cascaded shape regression, I created a reusable framework which can be easily extended to implement all the algorithms. These are all distributed as a Python package called facefit. The core part comprises the Regressor and FeatureExtractor interfaces, as well as a CascadedShapeRegressor class representing a cascade of regressors of any type. Each class that represents a machine-learnable object also has a corresponding builder class - for CascadedShapeRegressor this would be CascadedShapeRegressorBuilder. The constructor of a builder object takes all free parameters of the model as arguments, and can train a model from given training data using the build(features, targets) methods.

![UML diagram](image.png)

Figure 4.7: A UML diagram containing the relationships between core classes in the facefit framework.

With this modular design, it is easy create cascaded regressors with feature extractors and different stage regressors. For example, to construct a model described in 4.3, it is enough to compose a CascadedShapeRegressor with the appropriate stage-regressor builder and feature-extractor builder:

```python
from facefit import lbf, CascadedShapeRegressorBuilder
lbf_extractor_builder = lbf.LocalBinaryFeaturesExtractorBuilder(tree_depth=5, ...) # Set params.
```
4.4. IMPLEMENTATION DETAILS

```
4.4. IMPLEMENTATION DETAILS

lbf_regressor_builder =
    lbf.GlobalRegressionBuilder(lbf_extractor_builder, ...) # Set params.

model_trainer = CascadedShapeRegressorBuilder(n_stages=10,
    weak_builder = lbf_regressor_builder)

# Train on pre-loaded training data.
model = model_trainer.build(images, ground_truth_shapes, bounding_boxes)

# Apply on an unseen image.
bounding_boxes = face_detector(image)
shape = model.apply(image, bounding_boxes)
```

Here, the LocalBinaryFeaturesExtractorBuilder implements the FeatureExtractorBuilder interface and GlobalRegressionBuilder implements the RegressionBuilder interface. As all the implementations only assume these interfaces, it is easy to swap these for other implementations of FeatureExtractorBuilder and RegressionBuilder.

This modular design allowed me to reuse a lot of code and properly write unit tests for each unit. It will also allow me to extend the repository by implementing other cascaded regression methods.

### 4.4.2 Third party libraries

I tried to keep the number of dependencies to the minimum. I extensively used the menpo[2] ecosystem developed within i-BUG as well as the Numerical computing for Python library (numpy). To solve the large-scale linear regression problem described in 4.3, I used Liblinear[22].

For face alignment, I started off with the OpenCV implementation of the Viola-Jones[6] method. However, this gave very inconsistent results with a lot of false positives, so I opted for the face detection implementation in dlib [7]. This was more accurate but about 1.5 times slower.
CHAPTER 4. IMPLEMENTED METHODS

4.4.3 Testing

Naturally, I wrote unit tests to thoroughly test each module in my code. Also, I included some end-to-end tests which both train and test all three regressors on a very small sample of annotated images and ensure that the alignment error is zero.

For manual testing, I created a webcam client that loads a serialised model and annotates the live video feed with facial landmarks, as can be seen below. On the live feed, all methods gave fairly good performance, although further optimisation is required to obtain full real-time performance. Notably, the LBF method by far did not achieve 3000fps - this would require a highly optimized version in a lower-level language. Computational performance will be further analysed in the next chapter.

Figure 4.8: A screen shot of real time annotation via a webcam feed.
4.4. IMPLEMENTATION DETAILS
Chapter 5

Evaluation

In this chapter, I am going to analyse the performance of all three methods and compare them to implementations of other state of the art methods - an AAM-based method and a method based on Supervised Descent Method (SDM). Later in this chapter I will talk about challenges I faced while working on this project. The Explicit Shape Regression method will be referred to as ESR, the One millisecond method using an Ensemble of Regression Trees will be referred to as ERT and the last one will be LBF (Local Binary Features).

5.1 Alignment accuracy analysis

I trained all regression methods on a combined dataset consisting of training images from both LFPW[12] and Helen[5]. I performed tests on the corresponding testing datasets and measured point-to-point alignment errors normalised by the diagonal size of each shape. Normalisation of errors is important for consistency - without normalisation, large images would have inherently fairly large errors compared to smaller ones, even though the actual alignment might be reasonable.

The SDM implementation comes from the menpo library (part of menpo [2]). The AAM-based method builds a HOG-based Active Appearance Model and use the Alternating Inverse-Compositional (AIC)[23] algorithm to perform fitting. This set up performed the best among the ones experimented with in [24]. In my experiments, I used the reference implementation of HOG-AIC provided by [24].

As can be seen from figure 5.2, the three methods give comparable performance when tested on LFPW and Helen with ERT slightly outperforming the other two. These findings are consistent with the results from the original papers[9, 10, 11].
As the only difference between ERT and ESR is using a random forest instead of an ensemble of ferns, the increase in accuracy (around 6% and 2% lower mean error on LFPW and HELEN respectively) must be from a better generalisation ability of decision trees. This corresponds to intuition - the decision trees pick different features in different split nodes and explicitly maximise variance reduction at each split. On the contrary, ferns compare the same features across each level, which might not necessarily maximise variance reduction (note as each leaf node outputs the mean of all training shapes falling into that leaf, maximising variance reduction is equivalent to minimising sum of squares of alignment errors at training. Minimising square of alignment error is also our objective at test time.)

Although LBF also uses a random forest (albeit per single landmark and combined with a linear regression matrix), its accuracy is slightly lower than the one of ERT. This is because in my tests, I opted for the faster and less accurate version of LBF, that is referred to as LBF-fast in the original paper and consists of 5 stages of regression, 300 decision trees per stage, each of depth 5. The more accurate (but less efficient) version has 5 stages and 1200 decision trees of depth 7 per each stage. Unfortunately, I did not have enough computational resources to run this version and thus cannot comment on the accuracy. However, I expect the accuracy to be comparable to ERT - as there will be a greater number of local binary features, they will be a more discriminative descriptor of each shape. Also, the global regression matrix will have larger dimensions, thus will be capable of capturing more variation.

Note that in all experiments, I used the default parameters as stated in the original papers. These are visualised in the following figure.

<table>
<thead>
<tr>
<th>parameter</th>
<th>ESR</th>
<th>ERT</th>
<th>LBF</th>
</tr>
</thead>
<tbody>
<tr>
<td>num. of stages</td>
<td>10</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>num. of primitive regressors</td>
<td>500</td>
<td>500</td>
<td>300</td>
</tr>
<tr>
<td>num. of perturbations at train time</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>num. of initialisations at test time</td>
<td>5</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>num of pixel difference features</td>
<td>16000</td>
<td>16000</td>
<td>16000</td>
</tr>
<tr>
<td>radius of sampling region on mean shape</td>
<td>0.3</td>
<td>0.3</td>
<td>adaptive</td>
</tr>
<tr>
<td>tree depth</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

Figure 5.1: Parameter settings used in training and testing.
Figure 5.2: CED curves of all tested methods together with a distribution of initial errors, as well as mean errors in each experiment.
5.1. ALIGNMENT ACCURACY ANALYSIS

Figure 5.3: Examples of most challenging images from the testing set of Helen, fitted using ESR, ERT and LBF.
5.2 Computational Efficiency

I performed measurements of fitting time for one image with one initial shape initialisation. This was done on a 2.4GHz Intel Core i5 machine with 8GB of RAM. In terms of computational complexity, I achieved the following results:

In terms of fitting time, my implementation of ESR achieves very good performance - much superior to SDM and AAM. Higher fitting times in the ERT and LBF methods are probably due to a slow implementation of random forest. This will certainly require significant amount of optimization.

Note the slow fitting time of Active Appearance Models - as they solve a difficult texture-matching optimisation problem, the fitting time for a single image takes 2.21s. SDM gives a reasonable performance at 53.9ms per one image.

On the other hand, LBF is fairly efficient in terms of storage space. As part of ESR, I also implemented a sparse-coding based compression algorithm based on Orthogonal Matching Pursuit (OMP) [25], which resulted in a model of size only 60MB without hurting accuracy. Note that this was using the HDF-5 format in a very naive way - simply using the hickle Python package. Serializing directly into binary would yield much smaller model sizes. As part of my next steps, I want to employ OMP in the other two methods as well to decrease sizes of serialised models.

<table>
<thead>
<tr>
<th>Method</th>
<th>time</th>
<th>storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESR</td>
<td>38.4ms</td>
<td>452MB</td>
</tr>
<tr>
<td>ERT</td>
<td>110ms</td>
<td>234MB</td>
</tr>
<tr>
<td>LBF</td>
<td>73.5ms</td>
<td>92MB</td>
</tr>
<tr>
<td>AAM</td>
<td>2.2s</td>
<td>1GB</td>
</tr>
<tr>
<td>SDM</td>
<td>53.9ms</td>
<td>74 MB</td>
</tr>
</tbody>
</table>

Figure 5.4: Table showing alignment time required to fit one image using one initialisation. The storage column refers to the size of a serialised model in the HDF-5 format.
5.3 Challenges

The biggest challenges I faced when implementing the project were the following two:

- The face detector sometimes outputs inconsistent bounding boxes.
- The training phase took an enormous amount of time (for LBF, it took more than 36 hours to train it on LFPW+Helen).

The first one turned out to be a significant problem, which I resolved by manually analysing the face detector output and tried to derive a good strategy for generating perturbations / initialisations. This included rescaling and translating the mean shape when fitting it to a bounding box. By changing my initialisation strategy from my original one, I improved the mean alignment error by 9%.

I combatted the second issue by using vectorised operations in Python using the numpy library - these operate on matrices as a whole and are evaluated within fast C code. Using these, I managed to bring down the training time on ESR to around 30% of the training time of my original implementation. However, it still takes a couple of hours to train each model (on a 2.4GHz Intel Core i5 computer with 8GB of memory). This is longer than stated in the original papers. Obviously, as I am using Python in a single-threaded manner, there are certain limitations, but further attempt at optimisation is required.

As stated above, LBF took around 36 hours to train. However, a lot of this time was spent in the Liblinear [22] solving for the global regression matrix. This is understandable, as the size of the matrix was $136 \times 5440$ and the augmented dataset consisted of 60k training samples. Unfortunately, there might not be much space for further improvement here, but some experimentation might be necessary.

A related issue is memory consumption - due to the large size of the augmented training set, the memory consumption could get up to 30GB. As I had access to only limited computational resources, this might be the cause for long training periods.

Obviously, these performance issues had negative impact on my development efficiency. However, I was able to partially mitigate these by training different methods in parallel on separate virtual machines in the departmental cloud.
Chapter 6

Conclusion and future work

In this chapter I summarise the achievements of this project as well as provide suggestions for future work and experiments.

6.1 Contributions

A summary of the core contributions of this project is as follows:

- Successful implementation of a cascaded shape regression framework, distributed as a Python package called facefit.
- Implementation of Face Alignment by Explicit Shape Regression [9] paper within facefit, achieving state of the art accuracy results on both LFPW and Helen datasets.
- Comparison and analysis of all three techniques on LFPW and Helen datasets with an AAM-based and SDM-based method.
6.2 Further extensions

In the future, I have been considering the following extensions to the project:

- **Implementation of a C++ client**, which would merely load a serialised model and use it for fitting. Compared to the current implementation in Python, a C++ implementation could take advantage of low-level optimisations and combined with multi-threading, this would definitely yield superior performance in terms of fitting time.

- **Implementation of Joint Cascade Face Detection and Alignment**\[16\]. This paper further builds up on \[11\] and describe a method that performs both face detection and alignment in a joint way. The paper claims state-of-the art results on both detection and alignment in terms of accuracy and efficiency.

- **Integration into menpofit**. Although this work extensively uses the menpo framework, it is created as a standalone project. Menpofit is a library from the menpo ecosystem that provides implementations of some alignment algorithms, such as SDM, AAM and CLM (Constrained Local Models). With further refactoring, my project could be integrated into the framework.

- **Experiments with affect analysis in real time**. The facial feature points could be used to further learn a classifier for emotional analysis.

- **Experiments with regressing shapes with these methods - not necessarily faces.**

- **Implement a nice web interface for the real-time webcam fitter.**
Bibliography


BIBLIOGRAPHY


