



Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG

---

# Diversity-driven Hopfield Neural Network Ensembles for Face Detection

**Dissertation**

zur Erlangung des Doktorgrades

an der Fakultät für Mathematik, Informatik und Naturwissenschaften

Fachbereich Informatik

der Universität Hamburg

eingereicht beim Fach-Promotionsausschuss Informatik von

**Dipl.-Inf. Nils Meins**

Hamburg, 2019

---

---

Submission of the thesis:

7 of January of 2019

Date of oral defense:

5 of June of 2019

Dissertation Committee:

Prof. Dr. Stefan Wermter (reviewer)

Dept. of Computer Science

Universität Hamburg, Germany

Prof. Dr. Leonie Dreschler-Fischer (reviewer)

Dept. of Computer Science

Universität Hamburg, Germany

Prof. Dr.-Ing. Wolfgang Menzel (chair)

Dept. of Computer Science

Universität Hamburg, Germany

---

©2019 by Nils Meins

All the illustrations, except where explicitly stated, are work by Nils Meins and are licensed under the Creative Commons Attribution-ShareAlike 4.0 International (CC BY-SA 4.0).

To view a copy of this license, visit <https://creativecommons.org/licenses/by-sa/4.0/>





---

## Abstract

A key factor in creating ensemble classifiers, which combine several classifiers to obtain a result, is diversity of the combined classifiers. If we combine many classifiers that react in the same way, we do not achieve any benefit by combining them. Thus, the application we built focuses on diversity. We use simple features and simple base classifiers, but increase diversity to create a strong ensemble classifier. Thus, the features, classifiers and hybrid architectures we create are designed to increase diversity instead of sophisticating them to achieve greater accuracy.

This thesis arises out of the European research project KSERA which examines a social assistance robot created to help the elderly. To communicate with a person, a robot needs to know whether it has the attention of the person, which is given if the person looks at the robot. Therefore, an essential first step is face detection, which is why we use it to examine our application. Further, our application shall be swift to train and execute because it has been able to run with (limited) robot hardware.

As features, we use the pixel sum of several rectangles, arranged in different geometrical structures. The value of one feature is a vector using the pixel sum of one rectangle as an element. We arrange the rectangles into different systematics, i.e. different types of features with different appearances are grouped into sets based on their similarity. We developed two classifier models that use these features. One classifier uses template matching to compare a feature with a learned pattern. The second classifier works the same way, but uses a Hopfield Neural Network (HNN) to learn the feature. Together with the Viola/Jones threshold classifier, we combine our classifiers to create a hybrid ensemble and cascade classifier.

The hybrid classifiers use a small but more diverse set of classifiers for training, and we show that our new features and classifiers improve the single type ensemble classifiers. Both classifier models that we introduced only differ in terms of whether or not they use the HNN. We introduced the HNN to achieve a diversity benefit from its dynamic and to use its recreation ability of patterns from a noisy input. The HNN leads to different behavior among both classifier models. We analyze the differences between both classifier types and show that the HNN increases diversity.

In our second combination, the Forced Hybrid Architecture, we adapt the training to increase diversity by forcing it to select different features first, instead of the best. Further, we introduce more different appearances of features, also with the aim of increasing diversity. We show that most sets create also good results on their own. However, through the Forced Hybrid Architecture and the extended

---

features, we can further improve our application. Combining a good set with an inferior set can improve the resulting classifier compared to training both classifiers with only one set. Although we are forced to use an inferior set, its influence is often beneficial all the same. Thus, the Forced Hybrid Architecture improves our classifiers.

Finally, we applied our features and classifier models to the multi-class problem of head-pose estimation. There, we showed that our approach of features and the HNN to learn multiple patterns is also able to solve multi-class problems.

---

## Zusammenfassung

Ein Schlüsselfaktor beim Erstellen von Ensemble-Klassifikatoren, die mehrere Klassifikatoren kombinieren, um ein Ergebnis zu erhalten, ist die Diversität der kombinierten Klassifikatoren. Wenn wir viele Klassifikatoren kombinieren, die auf dieselbe Weise reagieren, erzielen wir durch die Kombination keinen Vorteil. Daher konzentriert sich die von uns entwickelte Anwendung auf Diversität. Wir verwenden einfache Merkmale und einfache Basisklassifikatoren, erhöhen jedoch die Diversität, um einen starken Ensembleklassifizierer zu erstellen. Daher sind die von uns erstellten Merkmale, Klassifikatoren und Hybridarchitekturen darauf ausgelegt, die Diversität zu erhöhen anstatt diese zu verfeinern, um eine höhere Genauigkeit zu erreichen.

Diese Dissertation entstand aus dem europäischen Forschungsprojekt KSERA, das den Einsatz von Robotern zur sozialen Unterstützung von älteren Menschen untersuchte. Um mit einer Person zu kommunizieren, muss ein Roboter wissen, ob er die Aufmerksamkeit einer Person hat, was der Fall ist, wenn die Person den Roboter ansieht. Ein wesentlicher erster Schritt ist daher die Gesichtserkennung, weshalb wir Gesichtserkennung gewählt haben, um unser Model zu untersuchen. Außerdem sollte unsere Anwendung schnell trainiert und ausgeführt werden können, da sie mit (begrenzter) Roboterhardware ausgeführt werden können soll.

Als Merkmale nutzen wir die Pixelsumme mehrerer Rechtecke, die in unterschiedlichen geometrischen Strukturen angeordnet sind. Der Wert eines Merkmals ist ein Vektor, der die Pixelsumme eines Rechtecks als Element verwendet. Wir ordnen die Rechtecke in verschiedene Systematiken ein, d.h. verschiedene Arten von Merkmalen mit unterschiedlichem Aussehen werden in Gruppen nach ihrer Ähnlichkeit gruppiert. Wir entwickelten zwei Klassifikatoren, die diese Merkmale verwenden. Ein Klassifikator nutzt Template Matching, um ein Merkmal mit einem erlernten Muster zu vergleichen. Der zweite Klassifikator funktioniert auf die gleiche Weise, verwendet jedoch ein Hopfield Neural Network (HNN), um das Merkmalsmuster zu lernen. Wir kombinieren unsere Klassifikatoren mit dem Viola/Jones Schwellwertklassifikator, um hybride Ensemble- und Kaskadenklassifikatoren zu erstellen.

Die hybriden Klassifikatoren verwenden eine kleinere Menge von Klassifikatoren mit erhöhter Diversität für das Training. Wir zeigen, dass unsere neuen Merkmale und Klassifikatoren die homogenen Ensemble-Klassifikatoren verbessern. Beide Klassifikationsmodelle, die wir eingeführt haben, unterscheiden sich nur darin, ob sie das HNN verwenden oder nicht. Wir haben das HNN eingeführt, um durch sein

dynamisches Verhalten die Diversität zu erhöhen und um die Fähigkeit zu nutzen, Muster aus unscharfen Eingangsdaten wieder herzustellen. Das HNN führt zu unterschiedlichem Verhalten beider Klassifikationsmodelle. Wir analysieren die Unterschiede zwischen den beiden Klassifikationstypen und zeigen, dass das HNN die Diversität erhöht.

In unserer zweiten Kombination, der Forced Hybrid Architecture, passen wir das Training an, um die Diversität zu erhöhen, indem wir dazu zwingen, erst andere Merkmale auszuwählen, anstatt die besten. Außerdem führen wir weitere unterschiedlich aussehende Merkmale ein, mit dem Ziel, die Diversität zu erhöhen. Wir zeigen, dass die meisten Merkmale auch für sich alleine gute Ergebnisse erzielen. Durch die Forced Hybrid Architektur und die erweiterten Merkmale können wir unsere Anwendung weiter verbessern. Die Kombination guter Merkmale mit schlechteren Merkmalen kann den resultierenden Klassifikator verbessern, verglichen mit dem Training beider Klassifikatoren mit nur einem Merkmalstyp. Obwohl wir dazu zwingen, schlechtere Merkmale zu verwenden, ist deren Einfluss oft gleichwohl von Vorteil. Daher verbessert die Forced Hybrid Architecture unsere Klassifikatoren.

Schließlich haben wir unsere Merkmale und Klassifikationsmodelle auf das Problem der nicht binären Abschätzung von Blickrichtungen angewendet. Dort haben wir gezeigt, dass unser Ansatz von Merkmalen und das HNN zum Erlernen mehrerer Muster zu verwenden, auch in der Lage ist, Probleme mit mehreren Klassen zu lösen.

# Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Introduction</b>   | <b>1</b>  |
| 1.1      | Motivation . . . . .  | 1         |
| 1.2      | Contribution and Research Question . . . . .                                    | 2         |
| 1.3      | Structure of this Thesis . . . . .  | 3         |
| <b>2</b> | <b>Related and Underlying Methods</b>   | <b>5</b>  |
| 2.1      | Early Approaches of Face Detection . . . . .                                    | 5         |
| 2.2      | Detailed Viola and Jones Methods and Algorithm . . . . .                        | 6         |
| 2.2.1    | Introduction and Overview . . . . .   | 7         |
| 2.2.2    | The Weak Classifier and its Components . . . . .                                | 7         |
| 2.2.3    | Combining Weak Classifiers: Training Ensemble and Cascade classifiers . . . . . | 11        |
| 2.2.4    | Localisation within an Image . . . . .  | 17        |
| 2.3      | Ensembles . . . . .   | 18        |
| 2.3.1    | Ensembles Motivation . . . . .  | 18        |
| 2.3.2    | Ensemble Output Combining Methods . . . . .                                     | 19        |
| 2.3.3    | Ensemble Creation Methods . . . . .   | 22        |
| 2.4      | Diversity . . . . .   | 23        |
| 2.5      | Hopfield Neural Network . . . . .   | 26        |
| 2.6      | Extensions of Features and Algorithms . . . . .                                 | 30        |
| 2.7      | Training and Test Sets . . . . .  | 33        |
| <b>3</b> | <b>The Haar-Feature-Like Patch and Hybrid Diversity Approach</b>                | <b>37</b> |
| 3.1      | Introduction . . . . .  | 37        |
| 3.2      | Methods . . . . .   | 37        |
| 3.2.1    | Introducing Haar-Feature-Like Patches . . . . .                                 | 37        |
| 3.2.2    | Classification using Haar-Feature-Like Patches . . . . .                        | 43        |
| 3.2.3    | Combining the Hopfield Neural Network and Haar-Feature-Like Patches . . . . .   | 45        |

|          |  |           |
|----------|--|-----------|
| 3.2.4    | Training the Hybrid Ensemble and Cascade Classifier . . . . .        | 53        |
| 3.3      | Comparing Hybrid Architecture . . . . .                              | 56        |
| 3.3.1    | Introduction to Experiments . . . . .                                | 56        |
| 3.3.2    | Findings . . . . .   | 58        |
| 3.3.3    | Discussion and Conclusion . . . . .                                  | 64        |
| 3.4      | Overlap Detection Merge and Samples . . . . .                        | 69        |
| 3.5      | Summary . . . . .  | 71        |
| <b>4</b> | <b>Diversity and Common Characteristics of our Classifier Models</b> | <b>73</b> |
| 4.1      | Introduction . . . . .   | 73        |
| 4.2      | Diversity and Difference of our Classifier Models . . . . .          | 73        |
| 4.2.1    | Introduction to Experiments . . . . .                                | 73        |
| 4.2.2    | Results comparing HFP and HaarNN classifiers . . . . .               | 75        |
| 4.2.3    | Discussion and Conclusion . . . . .                                  | 81        |
| 4.3      | Increased Diversity by the Hopfield Neural Network . . . . .         | 83        |
| 4.3.1    | Introduction to Experiments . . . . .                                | 83        |
| 4.3.2    | Methods - Parameters for Increasing Diversity . . . . .              | 84        |
| 4.3.3    | Results . . . . .  | 86        |
| 4.3.4    | Discussion and Conclusion . . . . .                                  | 91        |
| 4.4      | Required Number of Features . . . . .                                | 92        |
| 4.4.1    | Introduction to Experiments . . . . .                                | 92        |
| 4.4.2    | Results of Number of Features . . . . .                              | 92        |
| 4.4.3    | Discussion and Conclusion . . . . .                                  | 95        |
| 4.5      | Summary . . . . .  | 95        |
| <b>5</b> | <b>Increasing Diversity by Features and Forced Architecture</b>      | <b>97</b> |
| 5.1      | Introduction . . . . .   | 97        |
| 5.2      | Methods . . . . .  | 98        |
| 5.2.1    | More Types of Haar-Feature-like patches (HFP) . . . . .              | 98        |
| 5.2.2    | Forced Hybrid Architecture . . . . .                                 | 102       |
| 5.3      | The Feasibility of HFP Sets . . . . .                                | 105       |
| 5.3.1    | Introduction to Experiments . . . . .                                | 105       |
| 5.3.2    | Results for the Feasible Cascade Classifiers . . . . .               | 105       |
| 5.3.3    | Discussion and Conclusion . . . . .                                  | 110       |
| 5.4      | The HFP sets' Diversity . . . . .                                    | 110       |
| 5.4.1    | Introduction to Experiments . . . . .                                | 110       |
| 5.4.2    | Results for the Diversity of the HFP Sets . . . . .                  | 110       |
| 5.4.3    | Discussion and Conclusion . . . . .                                  | 119       |

|          |  |            |
|----------|--|------------|
| 5.5      | Forced Hybrid Architectures . . . . .                    | 120        |
| 5.5.1    | Introduction to Experiments . . . . .                    | 120        |
| 5.5.2    | Results for the Alternating Feature Sets . . . . .       | 121        |
| 5.5.3    | Results for the Hybrid Alternating Classifiers . . . . . | 125        |
| 5.5.4    | Discussion and Conclusion . . . . .                      | 129        |
| 5.6      | Summary . . . . .  | 130        |
| <b>6</b> | <b>Multi-Class Ability</b>                               | <b>131</b> |
| 6.1      | Experiments Introduction . . . . .                       | 131        |
| 6.2      | Methods . . . . .  | 135        |
| 6.3      | Findings . . . . .                                       | 138        |
| 6.4      | Discussion and Conclusion . . . . .                      | 139        |
| <b>7</b> | <b>Thesis Summary and Conclusion</b>                     | <b>141</b> |
| 7.1      | Thesis Summary . . . . .                                 | 141        |
| 7.2      | Future Work . . . . .                                    | 142        |
| 7.3      | Conclusion . . . . .                                     | 143        |
| <b>A</b> | <b>Appendix</b>  | <b>145</b> |
| A.1      | Glossary of Acronyms and Abbreviations . . . . .         | 145        |
| A.2      | Classification Samples . . . . .                         | 147        |
| A.3      | Publications . . . . .                                   | 153        |
|          | <b>Bibliography</b>                                      | <b>154</b> |





# List of Figures

|     |   |    |
|-----|---|----|
| 2.1 | Figures a) to d) are examples of Haar-like features. The value of the Haar-like feature is the difference between the pixel sum of the gray rectangles and the pixel sum of the white rectangles. Considering Haar-like feature d) in more detail, the value is $v = a_2 + a_3 - (a_1 + a_4)$   | 8  |
| 2.2 | Calculation of the integral image according to Viola and Jones [69]   | 9  |
| 2.3 | This figure describes the calculation for the pixel sum of an arbitrary rectangle by using the integral image following Viola and Jones [69]. Any specific location value is the sum of the pixels left of and above this location (the sum of the pixels of the rectangle from the top left image position to the considered location). Writing $A = 1$ means that the sum of the pixels in $A$ is the value in location 1 of the integral image. The sum of rectangle D will be $D = (4 + 1 - (2 + 3))$ , which is the whole rectangle (the reference in 4) minus rectangles $A, B$ and $C$ . $A = 1$ , $A + B = 2$ and $A + C = 3$ . . . . . | 10 |
| 2.4 | Pseudo-code of the AdaBoost algorithm (see Viola and Jones [71]) .  | 15 |
| 2.5 | Every node decides whether the examined frame contains a face or is background. The complexity of the ensemble classifiers is increased for every subsequent node to speed up the rejection of background while maintaining a high level of accuracy. . . . .   | 16 |
| 2.6 | Pseudo-code for training a cascade classifier (see Viola and Jones [71])  | 17 |
| 2.7 | Voting, perhaps the most natural way to combine different opinions. In parliament, it will mostly be yes, no or abstention. The yes and no votes will be counted and the one with the higher number determines the decision. . . . .  | 20 |
| 2.8 | Training one base classifier $h_i$ on subset $i$ to create the ensemble $H$ .   | 23 |
| 2.9 | Pseudo-code of the bagging algorithm by Breimann [5, 6] . . . . .   | 24 |

|      |   |    |
|------|---|----|
| 2.10 | Every neuron is connected to every other neuron inside the Hopfield Neural Network except itself. It works as an auto-associator and can recall a learned pattern from a noisy input. . . . .   | 27 |
| 2.11 | One neuron calculates the weighted sum of the input $x$ . . . . .   | 28 |
| 2.12 | CBCL Face Database Train Images Samples . . . . .   | 34 |
| 2.13 | CMU Test Set A by Rowley et al. [49] contains 42 Images . . . . .   | 35 |
| 2.14 | Test Set by Sung and Poggio [60] contains 23 Images . . . . .   | 35 |
| 2.15 | CMU Test Set C by Rowley et al. [49] - contains 65 Images . . . . .   | 36 |
| 3.1  | Modeling one Haar-feature-like patch (E) instead of four Haar-like features (A-D). If we wanted to use a corner shape model, we would have to create four Haar-like features to get a feature for every possible corner. Using every value on its own, we can model all corner shapes using one feature. . . . .  | 39 |
| 3.2  | Figures A-1 and A-2 have no difference if we calculate the Haar-like feature value, but they are very different used as Haar-feature-like patch. The same applies to B-1, B-2 and C-1,C-2 . . . . .   | 39 |
| 3.3  | Haar-like features used by Viola and Jones [69]. . . . .  | 40 |
| 3.4  | These Tetris-like and Block shapes consist of either six or nine boxes with a width and height of 2x3, 3x2, and 3x3 rectangles. . . . .   | 41 |
| 3.5  | This figure shows the BlockN HFP set. We create these blocks by systematically increasing the number of rectangles in a row and a column. . . . .   | 41 |
| 3.6  | Systematically, we create BlockN HFPs by increasing the number of rows and columns of a source HFP. . . . .   | 42 |
| 3.7  | We create different appearances of an HFP source shape by stretching it horizontally or vertically. As illustrated in this figure, we create several HFPs by expanding one source HFP. We then incrementally increase the overall width and height, thereby creating every possible incremental increase in height for each incremental increase in width until a maximum height or width is reached. . . . | 43 |
| 3.8  | Every HFP has a position within the corresponding frame. The pixel sums within the Haar-Feature-like patch (HFP) constitute the input for classification. First, we calculate the HFP values, which are the pixel sums of the rectangles from which the HFP is built. Second, we compare this vector with a previously learned pattern. . . .   | 44 |
| 3.9  | Pseudo-code for learning HFP classifiers for all Haar-Feature-like patches and relative positions within the training images. . . . .   | 46 |

|      |   |    |
|------|---|----|
| 3.10 | Summary and short description of the HFP classifier . . . . .   | 46 |
| 3.11 | The pixel sums within the Haar-Feature-like patch (HFP) are the input for a Hopfield Neural Network (HNN). Executing the HNN leads to a stable state, i.e. a vector that does not change anymore. This final vector is the input for classification or the output of the HaarNN if used as a learnable filter. . . . .  | 48 |
| 3.12 | To balance the pattern and input between positive and negative values, we shift the average of the maximum and minimum to zero. . . . .   | 50 |
| 3.13 | Activation function using different parameters to adapt to different feature value ranges. . . . .  | 51 |
| 3.14 | After execution, the HNN will converge to a stable state ( $S_i$ ) which is assigned to a set of face and none-face probability values. . . . .   | 52 |
| 3.15 | Execution overview of an HaarNN . . . . .   | 53 |
| 3.16 | Training overview of an HaarNN . . . . .  | 54 |
| 3.17 | Hybrid Alternating Ensemble Architectures, combining threshold classifiers and HFP or HaarNN classifiers. Instead of choosing one classifier from a homogeneous set, here, AdaBoost uses a hybrid set of different classifier types. . . . .  | 55 |
| 3.18 | Determining whether a found region $F$ is similar to the expected region $E$ can be calculated by its overlapping $\frac{E \cap F}{E \cup F}$ . . . . .   | 57 |
| 3.19 | All of the red rectangles are false positives, while the green rectangles represent correct detections and the blue rectangles are the labeled faces. In general, no detection is likely to match the labeled region exactly. Hence, a detection is deemed a correct detection if the overlapping takes place within a proper range (see equation 3.11). However, there will mostly be more than one correct detection, all of which will be counted as one detection, whereas all of the nearby false detections are counted individually. . . . . | 57 |
| 3.20 | This distribution shows the detection and false-positive rate of cascade classifiers. All of the classifiers are trained using the same parameters except for the HFP sets and the classifier models. The single classifiers are trained using 200 features, i.e. 200 classifiers in each iteration. The hybrid classifiers are all trained by uniting the single classifiers. They are trained with 100 features each. Through uniting, they were also 200 classifiers for each iteration as for the single classifier training. . . . .           | 63 |

|      |   |    |
|------|---|----|
| 3.21 | Sample images of two hybrid cascade classifiers. The first image is a detection result of a NCC-HaarNN/threshold hybrid cascade classifier using the TLB69 and base features (table 3.5). The second image is a detection result of an NCC-HFP/threshold hybrid cascade classifier using the base features (table 3.6). . . . .   | 66 |
| 3.22 | The image on the left shows the result without any merging, i.e. just the detections as they are. The image in the middle shows the results after merging the nearby detections, while the image on the right shows the result after removing regions. This image sequence is a perfect example. First, there are some false detections; after merging this was reduced to only one, and in the final image there were no false detections at all. . . . .                                | 69 |
| 3.23 | The above-left image is without merging, above-right is with merging, and the lower image with removing of regions. There, we see an example of losing one correct detection due to merging. While in the first image all of the faces are found, the second image lost one correct detection. Finally, we can reduce a lot of the false detections.  | 72 |
| 4.1  | Comparing HaarNN and HFP classifier models. The images describe the classification process. It starts with the input, followed by applying the Haar-Feature-like patch (HFP). The HaarNN classifier (top) uses a Hopfield Neural Network (HNN) to filter the Haar-Feature-like patch values. In contrast, the HFP classifier (bottom) uses these values directly. Finally, both classifiers compare the result with a learned patch to decide whether the input is a face or not. . . . . | 74 |
| 4.2  | These sample images visualize the effect of the merge cascade. The top left image belongs to the NCC-HaarNN classifier, while the top right image belongs to the NCC-HFP classifier. The bottom image is the result of the merge cascade classifier consisting of the above HaarNN and HFP classifiers. We can see a reduction in false positives (red rectangles) because both classifiers react differently to most inputs; hence, they are diverse. . . . .                            | 80 |
| 4.3  | The logistic activation function with different values for the parameters <i>stretchFactor</i> ( $\beta$ ) and <i>threshold</i> ( $\theta$ ) . . . . .  | 86 |

|     |   |     |
|-----|---|-----|
| 4.4 | This figure shows the NCC-HaarNN cascade classifiers with their detection and false-positive rates. All classifiers are trained using the TLB69 HFP-set. The cascade classifiers are trained using 100, 500, 2000 and all features in each iteration with up to five node classifiers. . . . .  | 94  |
| 5.1 | Original Haar-like features (HF) and more complex Haar-feature-like patches (HFP). The feature value of the original features is the pixel sum difference of the gray and white rectangles. Instead of a single value, the HFP uses the vector defined by the values of all dedicated rectangle pixel sums. A) represents the features used by Viola and Jones, while TL6 and TL9 describe the group of Tetris-like features derived from Block6 and Block9. The final row consists of a few examples of the Symmetrical HFP (Sym). . . . . | 98  |
| 5.2 | The Tetris-like Six (TL6) shapes are composed of six boxes with a width and height of 2x3 and 3x2. . . . .  | 99  |
| 5.3 | The Tetris-like Nine (TL9) shapes are composed of nine boxes with a width and height of 3x3. . . . .  | 100 |
| 5.4 | CB4x4 is the abbreviation of "Cutting Blocks of 16" boxes with a width and height of 4x4. . . . .   | 100 |
| 5.5 | Here are four HFP sets. The three sets on the left are Block6, Block9, and Block16. They consist of six (two rows and three columns), nine (three rows and three columns), and sixteen rectangles (four rows and four columns). The three blocks on the right belong to the BlockN set. We create these blocks by systematically increasing the number of rectangles in a row and a column. . . . .   | 101 |
| 5.6 | Sym2 is the abbreviation for Symmetrical Shape 2. It is created by mirroring (horizontally and vertically) two rectangles with a different width, height, and position. . . . .   | 101 |
| 5.7 | All of the parameters that determine the Random-N HFP set are randomly chosen with a different width, height and position of a rectangle and also the number of rectangles. . . . .   | 102 |
| 5.8 | Forced hybrid architecture is used to create an ensemble; combining threshold classifiers and HaarNN classifiers. This figure describes the forced hybrid AdaBoost which first chooses a threshold classifier, followed by a HaarNN classifier (and so on). . . . .   | 103 |

|      |   |     |
|------|---|-----|
| 5.9  | Forced hybrid architecture is used to create a cascade; combining ensembles of threshold classifiers and ensembles of HaarNN classifiers. This figure describes the cascade classifier creation process. One node of a threshold classifier ensemble is followed by a node of a HaarNN classifier ensemble. . . . .   | 104 |
| 5.10 | Samples of cascade classifiers trained with up to ten ensemble nodes. The first four images belong to the NCC-HaarNN cascade trained with the HFP set TL6. The upper six images belong to the NCC-HFP cascade classifier trained with the HFP set CB4x4. . . . .  | 106 |
| 5.11 | This figure shows the NCC-HFP cascade classifiers with their detection and false-positive rate. All of the classifiers are trained using the same parameters. The cascade classifiers are trained using 100 features during each iteration with up to five node classifiers. We trained ten classifiers for each described HFP set. . . . .   | 108 |
| 5.12 | This figure shows the NCC-HaarNN cascade classifiers with their detection and false-positive rates. All of the classifiers are trained using the same parameters, except the HFP set. The cascade classifiers are trained using 100 features during each iteration with up to five node classifiers. We trained ten classifiers for every HFP set described. The classifiers using the HFP sets TL9 and CB4x4 are removed from the graphic for presentation purposes to show a similar scale as for the HFP cascades. TL9 does not reach suitable false-positive rates. The best false-positive rate for the five-node HaarNN cascades and CB4x4 set is 0.02. . . . . | 109 |
| 5.13 | The left images of both rows are the results of the NCC-HaarNN cascade classifier trained with the Block6 HFP set, while the images in the middle were trained with the TL6 HFP set. The right images show the results of the merge cascade classifier. The detection and false-positive rates of the classifiers are depicted in tables 5.5 and 5.6. . . . .   | 113 |
| 5.14 | The left image is the result of the NCC-HFP cascade classifier trained using the Block6 HFP set, while the image in the middle was trained using the BlockN HFP set whose diversity and performance are shown in tables 5.8 and 5.6. The right image is the result of the merge cascade classifier created from the other two classifiers to illustrate their diversity. . . . .  | 116 |

|      |   |     |
|------|---|-----|
| 5.15 | Comparing 550 cascade classifiers with their detection and false-positive rate. The blue triangles are the single classifier results, while the green squares represent the results of the hybrid classifiers that alternate the classifier type within the ensemble creation, and the red squares belong to the hybrid classifiers where the classifier type is altered after creating an ensemble node. . . . . | 126 |
| 5.16 | We compared 50 cascade classifiers with their detection and false-positive rate. They were chosen by taking the three best classifiers for every detection rate out of 550 classifiers. The only difference in parameters during training was whether they are single or hybrid classifiers and the HFP set. . . . .  | 129 |
| 6.1  | Samples of the training set. Three source images (left). Randomly created additional images with different width, height, rotation and gamma corrections (middle). Randomly cropped negative samples of landscape, office and apartment images taken from the internet. .   | 134 |
| 6.2  | Samples of the test set. Cropped images without any changes (left). Randomly created additional images with different width, height, rotation and gamma corrections (middle). Randomly cropped negative samples of landscape, offices and apartment images taken from the internet. . . . .   | 134 |
| 6.3  | Pseudo-code K-means algorithm to group objects in a cluster by their similarity. . . . .  | 135 |
| 6.4  | Instead of using a threshold to distinguish faces and background, head-pose estimation involves the KM-HFP and KM-HaarNN classifiers which use probabilities to create the four different outcomes: left, frontal, right, and background. . . . .   | 136 |
| 6.5  | The execution process of the HaarNN for multi-class solutions is basically the same as for the binary solution. However, instead of one positive input, we use three different positive classes for training.   | 137 |





# List of Tables

|     |  |    |
|-----|--|----|
| 2.1 | Possible results of two different classifiers. . . . .   | 25 |
| 3.1 | Single and hybrid ensembles trained using the same training parameters and feature sets except for the classifier model. The ensemble is trained using Adaboost. The second row shows the first features of the first ensemble. The single type cascades used 200 features in each iteration. The hybrid cascades also used 200 features, but the set is split into 100 features for each classifier model. The test was performed done with all three of the described test sets. . . . . | 59 |
| 3.2 | Single and Hybrid ensembles trained using the same training parameters and feature sets except for the classifier model. The ensemble training is done using asym-Adaboost. The second row shows the first features of the first ensemble. The single type cascades used 200 features in each iteration. The hybrid cascades also used 200 features, but the set is parted in 100 features for each classifier model. The test is done with all three described test sets. . . . .         | 60 |
| 3.3 | This table shows the diversity measure of the trained classifiers, the result of which are provided in table 3.1. . . . .  | 61 |
| 3.4 | This table shows the diversity measure of the trained classifiers, the results of which are presented in table 3.2. . . . .  | 62 |
| 3.5 | The table shows the two best classifiers for a certain detection rate. 5 times the hybrid classifier wins and 3 times the single classifier. The rows are grouped by detection rate and sorted by false-positive rate; thus, the lower false-positive rate comes first. . . . .  | 64 |
| 3.6 | The table shows the two best classifiers for a certain detection rate. 5 times the hybrid classifier wins and 2 times the single classifier. The rows are grouped by detection rate and sorted by false-positive rate; thus, the lower false-positive rate comes first. . . . .  | 65 |

|     |  |    |
|-----|--|----|
| 3.7 | Results of hybrid cascade classifiers to show the merging effect. The first row shows the results if no merging is done, the second if nearby regions are merged, and the third if every region not consisting of at least two merged regions is removed. Merging nearby regions reduced the false-positives by about half, while maintaining a detection rate with a minor loss of one percent. Instead, removing regions reduces the detection rate from 6 to 11 percent and the false-positive rate, again, to about half. . . . .  | 70 |
| 3.8 | Results of cascade classifiers to show the merging effect. The first row shows the results if no merging is done, the second if nearby regions are merged, and the third if every region not consisting of at least two merged regions is removed. The effect is the same as that of the hybrid classifiers in table 3.7. . . . .  | 71 |
| 4.1 | Trained with Asym-AdaBoost, 500 random features per iteration, to a size of 50 ensemble members using Normalized Cross-Correlation. Thereby, DR is the detection rate, and FPR is the false-positive rate. Every second row shows the first Haar-Feature-like patches that are chosen through the training algorithms. The last picture in each second row shows the Summed Picture (SP). It is developed by putting the learned patterns on top of each other and normalizing the result to create an image. Although they are equally trained, they differ in their chosen features and performance. . . . . | 75 |
| 4.2 | Correlation diversity of the above classifiers (see table 4.1). Fewer values mean greater independence and, therefore, higher diversity. Both classifier models are trained using the same parameters. . . .   | 76 |
| 4.3 | Classification results of the trained cascades. The HFP and HaarNN classifiers are trained using the same configurations. . . . .  | 77 |
| 4.4 | We can paint the learned pattern as a summed image which is an aggregation of all chosen features. We show ten pictures for the ten ensembles that make up the cascade classifiers. All of the summed pictures look different, meaning that no ensemble consists of the same set of features. . . . .  | 77 |

|      |  |    |
|------|--|----|
| 4.5  | Here, we show the difference of the chosen features for both classifier models. The only difference when training the NCC-HaarNN and NCC-HFP models are the classification method used. The column Node depicts the ensemble nodes of the cascade classifier in their sequential order. The painted features consist of the first ten features chosen during training. . . . .                             | 78 |
| 4.6  | Diversity correlation measure of the classifiers shown in table 4.5. .   | 78 |
| 4.7  | The second method to show diversity involves merging classifiers. The merge cascade classifier is the union of both of the above classifiers. It classifies an input as a face if both classifiers classify this input as a face. If both classifiers are independent, the resulting detection rate is $0.9 \times 0.9 = 0.81$ . . . . .   | 79 |
| 4.8  | Cascades trained with up to five ensemble nodes using asym-AdaBoost and BlockN as the HFP set. While all cascade classifiers have comparable results, the ED-HFP classifier has a very high false-positive rate. . . . .   | 80 |
| 4.9  | Cascades trained with up to five ensemble nodes using AdaBoost and BlockN as the HFP set. The ED-HFP cascade classifier has a very high false-positive rate, while all other cascades are feasible. .  | 81 |
| 4.10 | NCC-HaarNN ensembles using BlockN features trained to 80 base classifiers only differing in the scaling factor ( <i>normTo</i> ) and the threshold ( <i>fixThr</i> ) of the activation function. We trained all classifiers using the binary activation function 3.9 ( <i>fixThr = true</i> ) and one threshold for all neurons ( <i>usePT = false</i> ) as described in the method section 4.3.2. . . . . | 87 |
| 4.11 | Diversity measure of classifiers from table 4.10. The final row contains the average diversity value for each classifier. . . . .  | 88 |
| 4.12 | NCC-HaarNN ensembles using the BlockN HFP set, trained to 50 members. These classifier ensembles use the logistic activation function 3.8 and a learned threshold for all neurons. . . . .   | 88 |
| 4.13 | Diversity of the ensemble classifiers from table 4.12. . . . .   | 89 |
| 4.14 | NCC-HaarNN cascade classifiers using the BlockN HFP set trained with up to 10 nodes using several varied parameters. . . . .   | 89 |
| 4.15 | Paired diversity of the three cascade classifiers, the results of which are shown in table 4.14. . . . .   | 89 |
| 4.16 | 10er CC-HaarNN cascades trained with several varied parameters.  | 90 |

|      |   |     |
|------|---|-----|
| 4.17 | Paired diversity of the three cascade classifiers, the results of which are shown in table 4.16. . . . .  | 90  |
| 4.18 | Results of the merging NCC-HaarNN cascade classifiers in tables 4.14 and 4.16. . . . .  | 91  |
| 4.19 | HaarNN cascade classifiers trained with several different number of features and the BlockN HFP set. . . . .  | 93  |
| 4.20 | Median values of cascade classifiers in figure 5.12. These are not trained classifiers, but a center-like median of the distribution. . .   | 94  |
| 5.1  | This table shows the performance, first features, and the summed pattern image of several cascade classifiers. All of the cascade classifiers were trained with up to ten ensemble classifiers. . . . .   | 107 |
| 5.2  | HaarNN cascade classifiers trained with the HFP set TL9. The number of base classifiers is very small because the ensemble creation breaks if errors in the base classifiers are too high. . . . .  | 107 |
| 5.3  | Here, the HaarNN cascade classifiers are trained with up to ten node classifiers, each using the same parameters except for the HFP set. They have different detection and false-positive rates, and the different feature types cover different areas. Note the fact that the summed pictures of all features are different. . . . .                 | 111 |
| 5.4  | Diversity correlation values of the HaarNN cascade classifiers used in table 5.3. The cross-point depicts the correlation value of two classifiers. The smaller the value, the higher the diversity. Here, we compare HaarNN classifiers. The table key is the respective HFP set with which the classifier is trained. . . . .                       | 112 |
| 5.5  | The results of several cascade classifiers and their merge cascade classifiers so as to underline their diversity. Every merge cascade is depicted by <i>Merge</i> within the model column and is the result of both cascade classifiers above. Merge cascade means that both classifiers have to return a positive result for the given input. . . . | 113 |
| 5.6  | This table shows the HFP cascade classifiers also used in table 5.3. These results also show their difference in terms of performance and feature appearance. . . . .   | 114 |
| 5.7  | Diversity correlation values of the HFP cascade classifiers in table 5.6. The cross-point depicts the correlation value of two classifiers. . . . .   | 115 |

|      |   |     |
|------|---|-----|
| 5.8  | The results of several cascade classifiers and their merge cascade classifiers to prove their diversity. Every merge cascade is depicted by <i>Merge</i> within the model column and is the result of both cascade classifiers above. Merge cascade means that both classifiers have to react with a positive result for the given input. . . . .   | 115 |
| 5.9  | Cascade classifiers trained to five nodes, using 500 features. These are the results of a subset of all described HFP sets and only involving HFP classifiers. . . . .  | 116 |
| 5.10 | Correlation values for the cascade classifiers in table 5.9. . . . .  | 117 |
| 5.11 | Several ensembles trained only with different HFP sets. They are tested with the CBCL test set. The ensemble classifiers are trained with up to 50 members. The last two ensembles use the same HFP set type but different subsets. . . . .   | 117 |
| 5.12 | Correlation diversity for the classifiers in table 5.11 . . . . .   | 118 |
| 5.13 | Several ensembles trained only with different HFP sets. They are tested with the CBCL test set. The ensemble classifiers are trained with up to 50 members. The final two ensembles use the same HFP set type but different subsets. . . . .  | 118 |
| 5.14 | Correlation diversity for the classifiers in table 5.13 . . . . .   | 118 |
| 5.15 | The classifiers are trained with a fixed (random) set of 100 for every iteration. FH-AdaBoost means that we force alternation of the set during ensemble creation. Accordingly, the first five features of the FH-AdaBoost classifier consist of Block6 and 2Sym HFP features. FH-Cascade means forcing alternation after creating an ensemble node. Since we show the first five features of the first ensemble node classifiers, and the ensembles are trained using the same random sequence, the present features of the FH-Cascade classifiers are the same as those for the single classifiers. . . . . | 122 |
| 5.16 | The forced hybrid cascade improves both single classifiers in terms of false-positive rate by maintaining the detection rate of a single classifier. . . . .  | 123 |
| 5.17 | These results show that forcing the use of one set with a bad false-positive rate also leads to a feasible combined classifier. Note the reduction of the false-positive rate compared to the single CB4x4 classifier. . . . .  | 124 |
| 5.18 | The forced hybrid cascade classifier improves the false-positive rate of both single classifiers as well as the detection rate of one classifier.   | 124 |

|      |   |     |
|------|---|-----|
| 5.19 | This table shows the potential benefit of using two different HFP sets. Comparing accuracy (Acc(w)), all of the combinations outperform the classifier trained with a single set. The correlation value of both classifiers is 0.208. . . . .   | 125 |
| 5.20 | This table shows the results of two ensembles trained with the HFP sets BlockN and 2Sym as well as three ensembles trained with a combination of the two sets. The correlation value of both ensembles trained with a single HFP set is 0.237. . . . .  | 125 |
| 5.21 | Both ensembles using the Block6 and BlockN HFP sets for training have a correlation value of 0.629. The hybrid-trained ensembles slightly improve upon the single-trained ensembles. . . . .  | 126 |
| 5.22 | Direct comparison of single and hybrid classifiers. We chose two classifiers for the NCC-HaarNN, and BlockN set, and the NCC-HFP Block9 set, compared with four hybrid classifiers that use the same classifier models and sets. The single and hybrid classifiers are trained with asym-AdaBoost and 100 randomly chosen features during each iteration. . . . . | 127 |
| 5.23 | We took some samples that show the tendency of the classifiers. . .   | 128 |
| 6.1  | Results of the KM-HaarNN ensembles, trained with up to 50 base classifiers. . . . .   | 139 |
| 6.2  | Results of the BP-HaarNN ensembles, trained with up to 50 base classifiers. . . . .   | 139 |
| 6.3  | Results of the KM-HFP ensembles, trained with up to 50 base classifiers. The KM-HFM ensembles achieve high detection rates, but incorrectly classify about half of all negative samples. . . . .  | 140 |
| A.1  | Single cascade classifier using as base classifier NCC-HFP, the HFP set RandomN, trained with 100 features per iteration. . . . .   | 148 |
| A.2  | Forced Hybrid cascade classifier (FH-AdaBoost) using as base classifiers NCC-HaarNN and Threshold, the HFP sets Block6 and Base, trained with 100 features per iteration. . . . .   | 149 |
| A.3  | Hybrid cascade classifier (Unite) using as base classifiers NCC-HFP and Threshold, the HFP set Base, trained with 100 features per iteration. . . . .   | 150 |
| A.4  | Forced Hybrid cascade classifier (FH-AdaBoost) using as base classifiers NCC-HaarNN and NCC-HFP, the HFP sets 2Sym and Block6, trained with 100 features per iteration. . . . .   | 151 |

|     |   |     |
|-----|---|-----|
| A.5 | Forced Hybrid cascade classifier (FH-AdaBoost) using as base classifier NCC-HFP, the HFP sets BlockN and Base, trained with 100 features per iteration. . . . . | 152 |
|-----|---|-----|





# Chapter 1

## Introduction

### 1.1 Motivation

This work started as part of the European research project KSERA, which examines the role of robots in assisting elderly people. Face detection is an ability robots need to interact with humans. Face detection is the first step for many subsequent tasks, such as pose estimation, face tracking, emotion recognition or face recognition, which are all relevant for robot-human interaction. Gaze direction, for example, is needed to check whether a human pays attention to the robot and, therefore, is ready for communication.

While cameras are the robot’s visual system, we have to find faces in images, which is challenging. If we consider faces in images that are easy to recognize for a human, it could be very difficult for robots. For example, even in passport images where the pose does not vary a lot, there are differences like lighting conditions or the effect of different camera sensors. In natural images, we have differences caused by technical and environmental aspects. Image appearances depend on whether we take the image inside a room or outside. When outside, it makes a difference if the sun is shining or if it is cloudy. Also, shadow and reflections influence the image. These aspects also play a role indoors and in the event of additional artificial light. The relative camera position and orientation play a role, as is the case if parts of the face are covered with other items in the given surroundings. Another challenge in terms of differences are the faces themselves. They differ in structural elements, e.g. men in the images wearing beards, mustaches or glasses. Also, emotional reactions change facial appearance. However, all these different faces have to be detected.

The idea of our classifier model presented in this thesis starts with reading an article in a popular scientific journal which claims that the eyes “throw” a

lot of geometric figures to the brain. The brain, in turn, derives meaning from these figures. This “throwing of geometric structures” was the initial impulse for questioning whether it is possible to create a strong classifier that only uses simple geometric structures for object classification in images rather than using more complex features. Haar-like features, used by a simple threshold classifier and combined to create an ensemble (Viola and Jones [69]), are such simple features. This method, which inspired our work, has the drawback that as the ensemble grows, fewer accurate single classifiers will be added. However, we intend to overcome this problem by using an increased set of shapes as features, more accurate patch-like classifiers including a Hopfield Neural Network, and increasing diversity. Diversity plays a key role for ensemble methods, especially the diversity of the features and the classifiers which use these features and form the ensemble by way of combination. Another important aspect is the computational complexity. We want to train our ensembles as quickly as possible. While training often is performed with a large set of possible classifiers, it is crucial to have easy and fast models to train. Therefore, our proposed models focus on simplicity, speed, and increased diversity.

Summarizing, we want to create a strong classifier for face detection that uses simple features and classifiers by exceedingly increasing diversity. Therefore, we create novel features, classifier models, and hybrid architectures.

## 1.2 Contribution and Research Question

In this work, we introduce and analyze novel features, classifier models and hybrid architectures that focus on simplicity and increased diversity. We have created some novel feature shapes, explored their combination with classifiers and the aggregation of this combination to create a hybrid architecture. Our approach is driven by increasing diversity but retaining simplicity and computational performance, which examines an alternative way of developing classifiers by primarily thinking of diversity instead of more complex and accurate classifiers.

The main question relating to our approach is: Is our model, using increased diversity, able to create a well-performing classifier? Are our classifiers, which only use the pixel sums of several geometric shapes, able to distinguish faces from non-faces in arbitrary natural scenes? While we want to overcome this challenge mainly through diversity, we have to question whether our method can increase and create a feasible amount of diversity. Therefore, we explore the following questions:

- What is the role of our feature sets?

- Are the sets diverse to each other?
  - Are all of them beneficial?
- The role of combination:
  - Does the hybrid architecture create better results than the single classifiers?
  - Can we create a benefit from the forced hybrid architecture and its inherit diversity?
- What is the role of the Hopfield Neural Network?
  - Does the HNN create further diversity within the classification model?
  - Does the HNN really make a difference? Or do both classifier models have an equal effect within a bigger ensemble and cascade classifier?
  - Can we create a benefit out of the HNN?
- Finally
  - Does our model have the ability to become a strong classifier?

## 1.3 Structure of this Thesis

### Chapter 2: Related and Underlying Methods

We give an overview of state of the art and the methods which form the basis of our approach in the related work chapter 2. There, we start by providing an overview of methods used in the field of face detection, followed by an introduction to the relevant methods that we use or extend in this thesis. Finally, we introduce the train and test sets which we use in our experiments.

### Chapter 3: The Haar-Feature-Like Patch and Hybrid Diversity Approach

In chapter 3, we introduce our approach and analyze the results of our models. The method section introduces our Haar-Feature-like patches and diversity approach. There, we explain our features, the Haar-Feature-like patches (HFP) and how they are related to Haar-like features. Following that, we describe our classifier model that uses the HFP sets and our extended classifier model, which includes a Hopfield Neural Network. Finally, we show and analyze the results of the hybrid approach.

## **Chapter 4: Diversity and Common Characteristics of our Classifier Models**

In chapter 4, we explore the characteristics of both of our classifier models. First, we compare the diversity and differences of both models; second, we examine the role of the Hopfield Neural Network within the HaarNN classifier. Finally, we justify the use of small random sets for training.

## **Chapter 5: Increasing Diversity by Features and Forced Architecture**

In chapter 5, we introduce our Forced Hybrid Architecture approach and further HFP sets for increasing diversity. In the first section, we analyze the results of different HFP sets. We examine their ability to create good classifiers and their diversity compared with one another. Finally, we examine the use of the extended sets by the Forced Hybrid Architecture.

## **Chapter 6: Multi-Class Ability**

In chapter 6, we present our findings and attend to the use of our HFP and HaarNN classifiers as a multi-class solution. We explore this question by applying our models to the head-pose estimation problem.

## **Chapter 7: Thesis Summary and Conclusion**

In the final chapter, we summarize the findings of our thesis, discuss possible future work, and offer a final conclusion.

# Chapter 2

## Related and Underlying Methods

In this chapter, we describe the underlying and related work of our model. We start with an overview of early approaches in the field of face detection (section 2.1) and follow up on this with a detailed description of AdaBoost (section 2.2) which is an essential part of the work in this thesis. The model we suggest extends the features Viola and Jones [71] used and other Ensemble Method concepts (section 2.3) to increase diversity (section 2.4). Further, we use a Hopfield Neural Network for our classifier model (section 2.5). We follow this with an overview of the features and algorithms extending the underlying models and state-of-the-art methods (section 2.6). Finally, we present the training and test set we use (section 2.7).

### 2.1 Early Approaches of Face Detection

Face detection aims to find faces and their respective positions in images. In their survey from 2002, Yang et al. [86] separated face detection techniques into four groups: a) knowledge-based approaches, b) feature invariant approaches, c) template matching methods, and d) appearance-based methods. Knowledge-based methods use predefined rules to recognize faces based on the relative positions of facial features like the eyes, nose, mouth and so on. Such methods search for facial features first and then add the results to the researcher's manually coded rules. The problem here is the generality of the rules. If these rules are too detailed, they fail to detect faces. If they are too general, they lead to many false positives. Yan and Huang [85] built a hierarchical knowledge-based application to address this problem. They use a three-rule layer where the rules were applied to every location during a systematic scan of the considered image. The rules were executed beginning with the highest, most general layer and progressing through to the lowest detailed layer. The highest layer uses a general description of what

describes a face so as to select every possible face. The lowest layer represents low-level detailed facial features. The idea is to select everything that could be a face in the first step so they do not miss any faces while removing many non-faces. The other two layers distinguish faces from non-faces based on these preconditioned areas. With their application, the authors located faces correctly in 50 out of 60 test images and returned false positives in 28 test images.

The motivation for the feature-invariant method stems from the observation that humans can easily detect faces in scenes with difficult lighting, pose, and so forth. The idea is to find invariant features or properties that most faces have in common, including in cluttered scenes. However, these variations of lighting, noise or occlusion are also challenging when looking to recognize facial features. Sirohey [56] reaches 80% accuracy in 48 test images involving cluttered scenes. The author uses an edge map built using the Canny edge detector [9]. Through a heuristic of grouping and removing edges, Sirohey [56] extracts face contours, places an ellipse over the found region, and, finally, separates the faces from the background.

Template matching is performed by way of face patterns which are manually specified or given by parameterized functions. Classification is achieved by correlating all of the facial features and calculating and comparing a template. This solution is easy to implement, but the drawback lies in classifying faces in different poses or scalings. To handle this difficulty, some authors suggested multiscale sub-templates and deformable templates. An early attempt was published by Sakai et al. [51], who use line segments to model several sub-templates for the eyes, nose, mouth, and face contour. In 1993 and 1994, Tsukamoto et al. [64], [63] introduced a qualitative model where every input image is split into separate blocks. Then, qualitative features are calculated for every block. A face was subsequently classified if the block values were higher than a predefined threshold. In 1999, Miao et al. [42] proposed a hierarchical template matching approach. First, they rotate the image from -20 to 20 degrees, then they create a multiresolution hierarchy and extract edges through a Laplacian operator.

## **2.2 Detailed Viola and Jones Methods and Algorithm**

In this section we provide an overview of the Viola and Jones face detection framework [69, 71, 72] which is essential for our work because we use it as a basis and then extend upon their the ideas.

### 2.2.1 Introduction and Overview

Another group of Ensemble Methods emerged in 2002 with the work of Viola and Jones [69] and dominates the following work in this field as Zhang noted [90].

In 2001, Schapire [53] gave an overview of the boosting approach to machine learning. In that overview, he stated that the first demonstrable polynomial time boosting algorithm was published in 1989 in his work [52]. One year later, Freund [14] presented an improved variation. In 1995, Freund and Schapire [15] jointly proposed the AdaBoost algorithm. Building on that, in 2001 Viola and Jones published “Rapid object detection using a boosted cascade of simple features” [69] and later “Robust Realtime Object Detection” and “Robust Realtime Face Detection” [71, 72]. Their model outperformed other state-of-the-art applications in terms of accuracy. It distinguished itself with a high detection and very low false-positive rate. Further, it had (and still has) a very high computational detection speed which also outperformed the other methods that existed at that time.

### 2.2.2 The Weak Classifier and its Components

#### Haar-like features

Motivated by the work of Papageorgiou et al. [47], Viola and Jones did not use pixel values directly, but worked with functions related to Haar basis functions. Further, they mentioned two main reasons for using feature instead of pixel values. On the one hand, this provides the benefit of ad-hoc domain knowledge inherent to the feature, while on the other hand it enables a higher computational speed.

Haar-like features are the difference between several rectangle pixel sums. They used four different shapes of Haar-like features with a different number of rectangles, as depicted in figure 2.1. Their base features consist of between two and four rectangles. Considering figure 2.1, we calculate the sum of the pixel values inside the gray and the white rectangle and then normalize this sum to the underlying area. Independent of the number of rectangles, at the end of this process we have one sum value for the gray rectangles and another sum value for the white rectangles. Afterwards, we calculate the difference between both these sum values, which is then our Haar-like feature value. One Haar-like feature can be scaled to many different widths and heights, leading to a huge number of possible features.

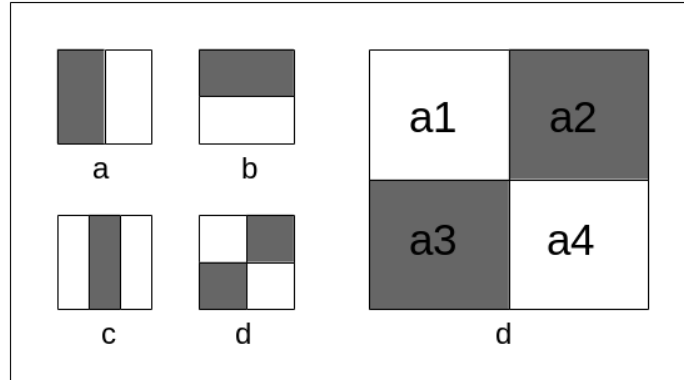


Figure 2.1: Figures a) to d) are examples of Haar-like features. The value of the Haar-like feature is the difference between the pixel sum of the gray rectangles and the pixel sum of the white rectangles. Considering Haar-like feature d) in more detail, the value is  $v = a2 + a3 - (a1 + a4)$

### Calculating Haar-like features: The Integral Image

Viola and Jones use AdaBoost to combine several weak classifiers into one strong classifier. In the context of ensembles like AdaBoost, this involves combining a number of features. In general, fast feature computation is beneficial, but it is essential to features used by ensembles because a large number of them have to be created.

To calculate the Haar-like feature, we have to compute the pixel sum of several rectangles again and again, which is computationally very expensive. Consequently, it is important to have a fast method to calculate the pixel sums. Viola and Jones achieved this by using another image representation called “integral image”, which is motivated by the work of Crow [10] and is similar to Crow’s “summed area table (SAT)”. Because of this integral image, every arbitrary rectangle pixel sum can be calculated using just four arithmetic operations. As an image contains pixels, the integral image contains the sum of the pixels up to the considered location. The integral image can be created from the original image in a single loop (see figure 2.2).

This image representation is key to achieving high performance in the resulting system. At the beginning of the detection process, the integral image is calculated once. Afterwards, every Haar-like feature is created in every required scaling using just four accessions of the integral image (figure 2.3).



The integral image values in every location  $(x, y)$  are the summed pixel values up to and left of the considered location. For visualizing, we can draw a rectangle, starting at the top left point of the image and ending at the bottom right point, which is our considered location  $(x, y)$ . Now, the value of the integral image at location  $(x, y)$  is the sum of the pixels within the painted rectangle, as expressed in equation 2.1.

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y') \quad (2.1)$$

where  $ii(x, y)$  represents the integral image and  $i(x, y)$  the original image. The calculation, requiring just one loop, is performed using the following two recursive equations.

$$s(x, y) = s(x, y - 1) + i(x, y) \quad (2.2)$$

$$ii(x, y) = ii(x - 1, y) + s(x, y) \quad (2.3)$$

Figure 2.2: Calculation of the integral image according to Viola and Jones [69]

### Threshold Classifier using Haar-like features

The Haar-like feature values used as an input for the classification process are the difference between several rectangle pixel value sums, as described in figure 2.1. Thereby, one classifier uses exactly one Haar-like feature value as an input. Viola and Jones [69, 72] proposed a threshold classifier that distinguishes between positive and negative samples, which are faces or no faces (background). The threshold classifier is trained by calculating a threshold which provides the greatest error reduction based on the training set. This is performed via the following steps: For one given Haar-like feature and a given position in the image, they calculated the values of the feature for all training images. Thereby, the training images consist of positive samples that show faces, and negative samples that do not show any faces. With all the values of one feature and one specific position for all training images, we then choose one value within the range of all values as our threshold. We therefore want to have a threshold that distinguishes perfectly between faces and non-faces. Hence, the samples with a lower value than the threshold should all be faces, while the samples with higher values are all non-faces. However, this is not usually the case. Irrespective of which threshold we choose, there will be faces

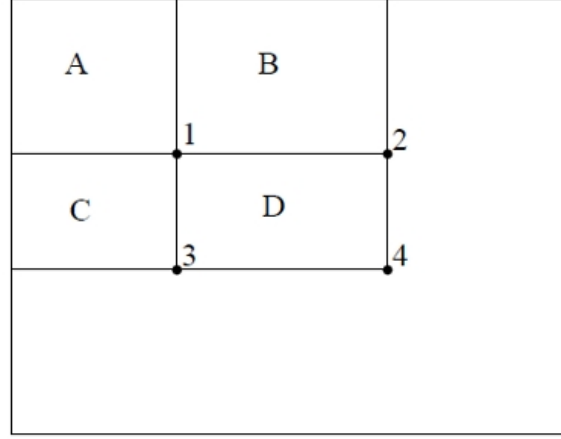


Figure 2.3: This figure describes the calculation for the pixel sum of an arbitrary rectangle by using the integral image following Viola and Jones [69]. Any specific location value is the sum of the pixels left of and above this location (the sum of the pixels of the rectangle from the top left image position to the considered location). Writing  $A = 1$  means that the sum of the pixels in  $A$  is the value in location 1 of the integral image. The sum of rectangle  $D$  will be  $D = (4 + 1 - (2 + 3))$ , which is the whole rectangle (the reference in 4) minus rectangles  $A, B$  and  $C$ .  $A = 1$ ,  $A + B = 2$  and  $A + C = 3$ .

and non-faces on both sides of the threshold. Therefore, the algorithm searches for the threshold that reaches the lowest error, considering the training images. Normally, every threshold classifier has errors; thus, Viola and Jones called them “weak classifiers”. Nonetheless, the AdaBoost ensemble method achieved a high level of accuracy by combining a set of weak classifiers.

$$h_j(x) = \begin{cases} 1 & p_j f_j(x) < p_j \theta_j \\ 0 & \text{otherwise} \end{cases} \quad (2.4)$$

Thereby,  $h_j(x)$  in equation 2.4 is the result of the threshold classifier. It is given by multiplying  $f_j$ , which is the value of the used Haar-like feature and parity  $p_j$ . The mathematical construct  $p_j$  indicates whether the inequation means less than or greater than; hence, the value is greater or smaller than the threshold  $\theta_j$ . With threshold  $\theta_j$ , parity  $p_j$  will be learned during training.

As mentioned for Haar-like features, when it comes to ensembles and their high number of possible weak classifiers, computational speed is important. Calculating the output of the threshold classifier (see equation 2.4) is quite inexpensive. However, it is also not very expensive to search for the best threshold that minimized

the error most, also considering different weighting of the training samples as used by AdaBoost (see the next section 2.2.3). Therefore, we choose one Haar-like feature and calculate its values for a set of training samples. Then we sort this list by its value, keeping the link to the corresponding training image. This link is needed to determine whether it is a face or a non-face example. Now, we can go through the sorted list to create a threshold between the current and the next value. By knowing whether the samples are faces or non-faces, we can calculate the current error. The only left to do now is to remember for which value the error has its lowest value, which, ultimately, is the threshold we are looking for.

### 2.2.3 Combining Weak Classifiers: Training Ensemble and Cascade classifiers

#### AdaBoost

Beside Haar-like features and the integral image, an additional important part of the system from Viola and Jones [71] is AdaBoost.

#### Selecting and Weighting - ensemble core creation

To describe the AdaBoost [71] algorithm, we start with an informal summary of the original and detailed formulation shown in figure 2.4. AdaBoost creates the ensemble by repeating the following steps:

1. The weak classifier with the lowest error over the training set will be selected, taking into account the current weights of the training images.
2. The selected classifier is added to the ensemble, and a classifier weight is calculated according to its current error.
3. The training set will be re-weighted so that the weights of correctly classified samples will be decreased, meaning that misclassified samples are more relevant in the next iteration.

#### Lowest error for given sample weights

AdaBoost calculates one classifier for all possible features and its error. Afterward, it selects the classifier with the lowest error over a weighted training set. The relevant part of the AdaBoost algorithm is the weighting of the training samples, which we will describe in the next paragraph. The algorithm starts by setting initial weights to  $1/n$ , where  $n$  is the number of samples. If we take four training

images and a classifier that correctly classified three of them, we get an error of  $1/4$  respectively  $\frac{n_{ic}}{n}$ , where  $n_{ic}$  is the number of incorrect classified samples and  $n$  the number of all samples. Describing this process using the weight  $1/n$  for every sample, we get:

$$error = \frac{n_{ic}}{n} = \frac{\sum_1^{n_{ic}} 1}{n} = \sum_{i=1}^{n_{ic}} \frac{1}{n} = \sum_{i=1}^{n_{ic}} w_i \quad (2.5)$$

which also leads to  $1/4$ . Using this formulation, we have to normalize the weights of our training set to ensure that  $\sum_{i=1}^n w_i = 1$ . We have already mentioned that the classifier with the lowest error is chosen, but by repeating this, we want to emphasize that this error is the lowest, given the current weighting of the samples, while the unweighted error could be very different. To illustrate this, we change the weight of one of our four samples to  $0.7$ . Now we consider two classifiers, one that misclassifies the  $0.7$  sample, and another that misclassifies all the other samples. The second classifier misclassifies three samples, nevertheless this is the one with the lowest error ( $\sum_{i=1}^{n_{ic}} w_i = \sum_1^3 0.1 = 0.3$  versus  $\sum_1^1 0.7 = 0.7$ ) and therefore will be chosen.

To simplify the explanation of what the weighting means, we used an equal weighting for all examples. The same weight means that every image is equally important, irrespective of whether it is a positive or negative example. However, this would lead to a misbalance if the negative examples were larger than the positive ones (or vice versa). Especially when training a cascade of ensembles (see the following section 2.2.3), the amounts of negative and positive examples are often entirely different. If the weighting of the training set is equal for all examples, we arrive at an implicit higher value of the part that has more members. For example, we have fifty of one hundred correctly classified negative examples and nine of ten correctly classified positive examples. With an equal weighting of all samples, we get an overall error of about  $0.46$  ( $\frac{1+50}{10+100}$ ). If we consider the parts on their own, we get an error of  $\frac{1}{10} = 0.1$  for the positive part and  $\frac{50}{100} = 0.5$  for the negative part, leading to an averaged error of  $\frac{0.1+0.5}{2} = 0.3$ . This is not as bad as  $0.46$ .

Following this example, looking for the classifier with the lowest error would only lead to classifiers with good results for the negatives, so we would primarily end up with classifiers that have a low false positive rate. The implicit influence of the negative examples in percent is higher than that for the positives. However, this is not what we want, which is why we do not use equal weighting. The initial weighting is set to  $1/2n_n$  and  $1/2n_p$ , where  $n_n$  are the number of negative samples and  $n_p$  are the number of positive samples, thus leading to a balance of

the negative and positive parts. While we consider two parts, we use the factor  $1/2$  for normalization.

### Reweighting the training set

In the last paragraph, we discussed the vital role of sample weighting. Now, we consider how AdaBoost uses this weighting to create a balanced ensemble. We call it a balanced ensemble because, over a period of time, the ensemble will have classifiers that fit every sample. Hence, the ensemble handles every aspect of the different samples.

This important mechanism is the reweighting of the training examples. For other ensemble methods such as bagging, it is crucial to use different training sample subsets. For AdaBoost, this involves reweighting. Reweighting is the second step, performed after one weak classifier is selected. Then, the training set will be re-weighted in such a way that a misclassified image becomes more valuable for the classifier selection process, requiring an increase in the weights of the misclassified sample as described in the following equation 2.6.

$$w_{t+1,i} = w_{t,i} \beta_t^{1-e_i} \quad (2.6)$$

where  $e_i = 0$  if example  $x_i$  is classified correctly,  $e_i = 1$  otherwise, and  $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$ .

We achieve a revaluation of the misclassified samples by increasing their weights compared to the correctly classified examples. It is similar to a voting system where some images have more votes than others. If we have an equal weighting of  $1/n$  for training images  $n$ , every sample has the same “voting relevance”. Through the reweighting step, an example becomes more relevant. The more a sample was misclassified, the more its weight will be increased. Thus, this example becomes more important to the selection process and a classifier that distinguishes this example correctly has a better chance of having the lowest error.

During the creation process, the reweighting mechanism focuses selection of the weak classifiers on different aspects of the considered objects. If we examine training images with faces, we can imagine having  $n$  persons, but only one of them has a beard. If we had an equal weighting of the training images, it would not matter whether the bearded man was rightly classified or not. It would not make any difference because the bearded man is only one sample of  $n$ . Moreover, it is not unlikely that only classifiers are constantly chosen which ignore the beard. This would not happen by reweighting the samples. If the bearded man is misclassified again and again, after a while this bearded training sample will become so important that only a classifier that recognizes this example as a face has a chance

of being the one with the lowest error and, thus, will be selected for addition to the ensemble. Describing the reweighting in such detail emphasizes the fact that reweighting of the images within the training process is a fundamental and very relevant mechanism of AdaBoost. It is an essential part of the ensemble creation process used to increase diversity.

### **Weighting the chosen classifier for the final ensemble**

The ensemble creation process finishes by reaching a predefined number of weak classifiers  $T$  or a given ensemble error. The classification of the ensemble is performed by weighted voting of its members. The weighting is bound to every dedicated weak classifier and is calculated after its selection in respect to its classification error of the current sample weighting.

### **Asym-AdaBoost**

With cascade classifier learning, the focus is on a high detection rate. Therefore, Viola and Jones introduced Asym-AdaBoost which prefers classifiers with a higher detection rate. Asym-AdaBoost got its name because the choice of the “best” classifier is asymmetric. As described, AdaBoost selects this classifier that has the lowest error for adding to the ensemble. As a result, the error is calculated such that the detection rate and the false-positive rate is equally weighted.

In their work, Viola and Jones [70] showed that they can improve the ensemble by using an asymmetric weighting of the detection rate and the false-positive rate. They increase the weights of the positive samples that make them more important for choosing a classifier and, therefore, the classifier selecting process prefers classifiers that have a good detection rate.

### **Early rejection of background - The Cascade Structure**

Viola and Jones [71] used the exhaustive search over a whole image to find the considered object. This is known as rare event detection meaning that most of the examined frames do not contain the object we are looking for. Crawling through an image largely means that our classifier has to examine a lot of background. Hence, the goal is to quickly decide whether or not the considered frame is background.

However, ensembles often have many base classifiers. Every base classifier needs some time to get its classification result and the more base classifiers an ensemble consists of, the more execution time the ensemble needs. The cascade classifier solved this by starting with simple classifiers to speed up the classification process

Pseudo-code AdaBoost:

- Given example images  $(x_1, y_1) \dots (x_n, y_n)$ , where  $y_i = 0, 1$  for negative and positive examples respectively.
- Initialize weights  $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$  for  $y_i = 0, 1$  respectively, where  $m$  and  $l$  are the number of negatives and positives respectively.
- for  $t = 1, \dots, T$ :
  1. Normalize the weights,  $w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^n w_{t,j}}$  so that  $w_t$  is a probability distribution.
  2. For each feature,  $j$ , train a classifier  $h_j$  which is restricted to using a single feature. The error is evaluated with respect to:  
 $w_t, \epsilon_j = \sum_i |h_j(x_i) - y_i|$ .
  3. Choose the classifier  $h_t$  with the lowest error  $\epsilon_j$ .
  4. Update the weights:  $w_{t+1,i} = w_{t,i} \beta_t^{1-e_i}$ , where  $e_i = 0$  if example  $x_i$  is classified correctly,  $e_i = 1$  otherwise, and  $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$ .
- The final strong classifier is: 
$$h(x) = \begin{cases} 1 & \sum_{t=1}^T \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^T \alpha_t \\ 0 & \text{otherwise} \end{cases}$$

where  $\alpha_t = \log \frac{1}{\beta_t}$ .

Figure 2.4: Pseudo-code of the AdaBoost algorithm (see Viola and Jones [71])

at the beginning. Then, the complexity of the classifiers in the following nodes is increased to also increase the accuracy. Increasing the complexity and accuracy is achieved by decreasing the expected false-positive rate for the current ensemble by keeping the high detection rate and increasing the number of negative samples. On top of that, the correctly classified negative samples of the currently trained ensemble will be replaced with new negative samples.

Every node in the cascade decides whether it is background or the object we are looking for (faces in our application). Thus, a cascade classifier only needs a few classifiers to remove most of the background. The cascade structure is a so-called degenerated decision tree. It has only one path, and every node in this path has to react with a positive result to get the final positive result. If any node

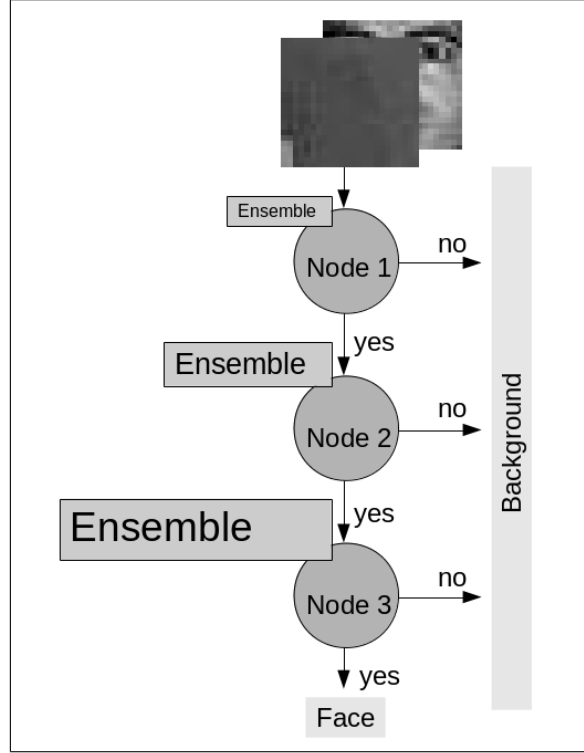


Figure 2.5: Every node decides whether the examined frame contains a face or is background. The complexity of the ensemble classifiers is increased for every subsequent node to speed up the rejection of background while maintaining a high level of accuracy.

reacts with a negative result, the process finishes with a negative result, as shown in figure 2.5.

The equations 2.7 and 2.8 describe the calculation of the final detection and false-positive rate. The equations presume that the classifiers within each node are statistically independent. Both the detection rate and the false-positive rate will decrease, which is what we want for the false-positive rate but not for the detection rate. Therefore, we must have classifiers with a high detection rate but not necessarily a good false-positive rate. If we have different classifiers for every node, it will be more unlikely with every step through the next node that the cumulated result will be false, which is expressed in the following equations 2.7 and 2.8.

$$F = \prod_{i=1}^K f_i \quad (2.7)$$

The  $F$  in equation 2.7 is the overall false-positive rate and  $f_i$  the false-positive rate



Pseudo-code for training a cascade classifier:

- Select all positive samples and a subset of the negative samples
- Set the expected detection rate ( $dr_e$ ) and false-positive rate  $fpr_e$  for the ensemble nodes
- Set the final expected false-positive rate  $fpr_f$  or max number of nodes  $N$  for the cascade classifier
- For  $t = 1, \dots, N$ :
  1. train an ensemble until it reaches  $dr_e$  and  $fpr_e$
  2. add the ensemble to the cascade
  3. replace the correctly classified negative samples with new ones
  4. add additional negative samples
  5. decrease the false-positive rate  $fpr_e = fpr_e - \delta$
  6. calculate the current false-positive rate of the cascade classifier  $fpr_c$
  7. if( $fpr_c < fpr_f$ ) break

Figure 2.6: Pseudo-code for training a cascade classifier (see Viola and Jones [71])

of the  $i$ th classifier.

$$D = \prod_{i=1}^K d_i \quad (2.8)$$

The  $D$  in equation 2.8 is the overall detection rate and  $d_i$  the detection rate of the  $i$ th classifier.

### 2.2.4 Localisation within an Image

Viola and Jones use a sliding window with different scalings to find a face. They put a frame over part of the image and test whether or not there is a face. The test window usually starts in one corner and then slides horizontally to the other side of the image. After that, it slides one step vertically to the next line and repeats this by sliding to the other side. After the test window reaches the corner opposite the starting corner (hence, every possible frame is examined), the process starts

from the beginning using an upscaled test window. Then, the process of the sliding window starts again, beginning at one corner and ending in the opposite corner. Increasing the test window, sliding, and classifying all possible frames within the image will be repeated as long as the upscaled window reaches a maximum width and height. The process will stop at the latest once the test window has the same size as the image to be tested.

## 2.3 Ensembles

In this chapter, we provide an overview of the field of ensemble classifiers. There are different names for this field, such as multiple classifier systems or classifier combiners.

### 2.3.1 Ensembles Motivation

The core of ensembles is a set of classifiers that work together. Kuncheva [31] mentioned that combining suggestions to find a decision is quite natural. In daily life, we often decide by asking several people for their opinion. “Get a second opinion” is a familiar statement used to arrive at a solution by taking the view of other people into account. Transferring this to the field of machine learning, we do not just use a single classifier, but multiple classifiers to create a compound result. The aim of using multiple classifiers together is to compensate errors for single classifiers. The final ensemble classification result should be more accurate than the sum of the single results.

Dietterich et al. [12] mentioned that there are three main reasons why classifier ensembles work.

- The statistical reason:  
This covers the fact that every trained classifier will have uncertainty because we cannot describe a perfect model that comprises everything. If there is so much data that we are not able to handle this amount of information, we will only use a subset to train our classifiers. However, there will be facts that are not known during training. Using more than one classifier can reduce this uncertainty.
- The computational reason;  
Data is not the only source of “noise”. The training algorithms themselves have imperfections. Many algorithms reduce an error for a dedicated set of

properties, but therefore sometimes stick to a local minimum. Running an algorithm with different properties and combining their abilities could help to avoid this.

- The representational reason:

If an algorithm is searching for a function that describes a given problem, it could be very complex, time-consuming or unfeasible to create a single function that solves the problem. Instead, for example, we can train simple linear classifiers and combine them to approximate a nonlinear function.

Kuncheva [31] underlines that it is important to make use of the variations of the training sets, feature sets or other, so that the classifier creation becomes unstable. If they were stable, they would produce similar results for similar training sets, which is not what we want to achieve. They should react as differently as possible to provide independent classifiers.

### 2.3.2 Ensemble Output Combining Methods

If we have one classifier, the output is the final classification result. However, having several classifiers means that we have to interpret the single results and combine them into one final result. Xu et al. [83] and Kuncheva [31] state four classes of individual outputs.

- Class labels:

Every classifier predicts precisely one class label for one input. This prediction comes without any further probabilities or other context information.

- Ranked class labels:

Each classifier returns a list of class labels in order of preference. While there is no value for the likelihood of the output, this result at least confirms a priority.

- Numerical support for classes:

The classifier result is a vector. Every value of this vector describes the support for a dedicated class.

- Oracle:

The only knowledge of the outcome is whether the classifier's result is wrong or false. [31] described the Oracle method as artificial because it ignores classes and only expresses whether an output is wrong or does not involve a labeled data set.

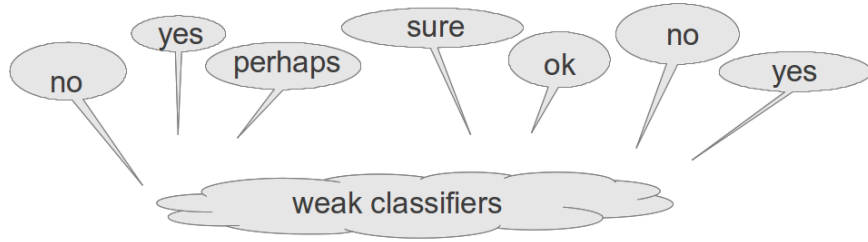


Figure 2.7: Voting, perhaps the most natural way to combine different opinions. In parliament, it will mostly be yes, no or abstention. The yes and no votes will be counted and the one with the higher number determines the decision.

If we have a set of different classifier answers and want to create a benefit from these multiple answers, we have to combine these results. Combining the outputs of the used classifiers is an essential part of every ensemble. The methods, as mentioned below, are based on the needs of this thesis and follow Zhou [92].

## Voting

The perhaps most natural way of combining different predictions is voting. Using democracy as a source of inspiration, we count the different “opinions” and finally take a decision in favour of the person with the maximum votes (plurality voting). Alternatively, we only accept a decision if a minimum number of counts is reached and otherwise reject the decision (majority voting). Another option is soft voting, which involves applying a list of weightings to every decision we make.

## Majority and Plurality Voting

Majority voting, as described in equation 2.9 counts the outcomes of every single classifier. By and large, it is necessary to reach at least a minimum number of votes to claim a positive result. If this minimum is not reached, it leads to a rejection.

$$H(x) = \begin{cases} c_j & \text{if } \sum_i^T h_i^j(x) > \frac{1}{2} \sum_{k=1}^l \sum_{i=1}^T h_i^k(x) \\ \text{rejection} & \text{otherwise} \end{cases} \quad (2.9)$$

Here,  $T$  is the number of classifiers,  $l$  is the number of class labels, and  $h_i^j(x)$  is the  $i$ th classifier that predicts an outcome for the  $j$ th class label  $c_j$  receiving  $x$  as an input. The outcome can be 1 ( $x$  belongs to the class label) or 0 (belongs not). Another possibility is that the outcome is a value between 0 and 1, which can be seen as the probability of the class label being right.

Xu et al. [83] adapt the equation 2.10 to a threshold majority vote. They enlarge the classes through another class  $\omega_n$ , which will be returned if no class reaches the threshold.

$$\begin{cases} \omega_k, & \text{if } \sum_i^T h_i^j(x) \geq \theta \sum_{k=1}^l \sum_{i=1}^T h_i^k(x) \\ \omega_n, & \text{otherwise} \end{cases} \quad \text{with } 0 < \theta \leq 1 \quad (2.10)$$

Plurality voting differs from majority voting in that it will return a result in each case. As described in equation 2.11, the resulting class label will be the class with the most counts. Thereby, the absolute number of counts does not matter as long as a class has the most counts.

$$H(x) = c_{\underset{j}{\operatorname{argmax}} \sum_{i=1}^T h_i^j(x)} \quad (2.11)$$

### Soft Voting

Above, we considered classifiers that create one class as an output. However, we also regard multi-class classifiers, i.e. classifiers that return a vector of class labels. Soft voting (see equation 2.12) is a method which handles such a vector and expects a probability like value for every class label in the vector. Then, analogous to the other methods, the soft voting method calculates the sum of the class labels of every classifier within the ensemble.

$$H^j(x) = \frac{1}{T} \sum_{i=1}^T h_i^j(x) \quad (2.12)$$

### Weighted Voting

Sometimes we consider some classifiers to be more important than other classifiers. Therefore, we use a weighting factor, a real value which is multiplied by the counting of the class labels. If a classifier has a weighting of 2, then its class-label weightiness is doubled. AdaBoost [69] produces an ensemble that creates a weighted voting. During training, these weightings will be set in relation to the error calculated for the training set.

According to the weighting, we have to adapt our equations. The outcome of every single classifier  $h_i^j(x)$  is multiplied by a weight  $w$  so that the final outcome is  $o_i(x) = w_i h_i^j(x)$ . The class with the highest value is the final result class as described in equation 2.13. All of the voting methods discussed above also work with weighted voting.

$$H(x) = c_{\underset{j}{\operatorname{argmax}} \sum_i^T w_i h_i^j(x)} \quad (2.13)$$

### 2.3.3 Ensemble Creation Methods

We use the term ensemble methods to describe the methods that create an ensemble. There are some tasks to consider here. A method that creates an ensemble has to search for and select classifiers or features, train them where necessary, and combine them. Further, it also has to introduce mechanisms to create diversity.

#### Bagging

The early ensemble method “bagging” was introduced by Leo Breimann in his work “Bagging Predictors”, reported in 1994 and 1996 (see [5, 6]). The name is an acronym for Bootstrap AGGREGatING. It is difficult to create independent base classifiers that are all trained using the same training set. Hence, the idea is to use independent training sets to also arrive at independent classifiers through these sets. A drawback here is the overall number of samples in a set. Often, there are not as many samples available to arrive at independent sets that are also large enough to create sufficient base learners. If these sets are too small, the resulting ensemble leads to poor results. To overcome this problem, Efron and Tibshirani [13] introduced bootstrapping as a means of creating a training set by subsampling. There, they take for every learning iteration a (mostly random) subset of samples with replacement. Of course, none of these subsets will be completely independent, but none of them will be identical, either.

Bagging aggregates several classifiers, trained with different, but overlapping, training samples to arrive at a final ensemble. Typically, a bagging ensemble uses voting for classification tasks, as noted in figure 2.9. To create a most diverse classifier of these sets, the instability of the multiple base classifiers is important. Further, besides the benefit of using multiple subsets instead of one set, bagging has computational advantages in that it can be trained as well as executed in parallel. Performance is a major advantage, and with modern multi-core architectures, you can make use of and increase this advantage by using algorithms that run in parallel.

#### Boosting

While describing AdaBoost above, we pointed out how essential it is while emphasizing its difference to bagging. The general idea here and one major difference to

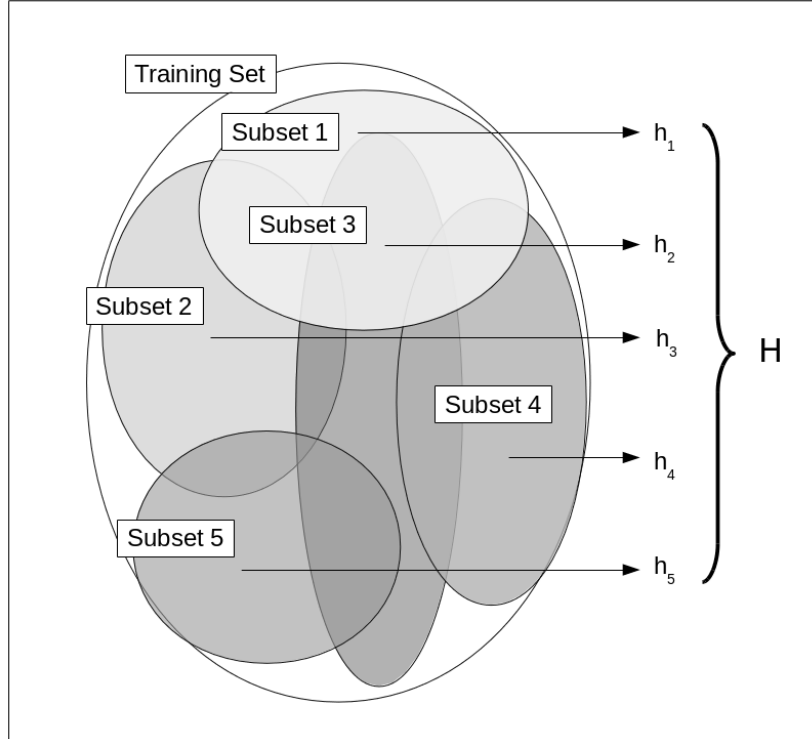


Figure 2.8: Training one base classifier  $h_i$  on subset  $i$  to create the ensemble  $H$ .

bagging is to introduce a weighting for every training sample. These weightings will be changed in every training iteration for every training sample related to the error of the last-added classifier. It is therefore a sequential algorithm, so one classifier will be added after another according to the former classifier. AdaBoost is used only with reweighting or with an additional resampling step. The reweighting variant uses the whole training set and only the weighting of the training samples to generate diversity. Instead, the resampling variant additionally uses a subset of the training samples in every iteration.

## 2.4 Diversity

Diversity means heterogeneity within a system or a group of individuals. Diversity plays a vital role in creating an ensemble. The core idea is to combine classifiers that have individual failures but high accuracy if their results are combined. If the individual errors of the classifiers are the same, we do not arrive at an improved ensemble from these classifiers. This is because several classifiers which produce the same misclassifications for a given input also produce these misclassifications as a combination for the same input. We will only succeed in creating an ensemble

Pseudo-code of the bagging algorithm:

- Given example images  $D = x_1, x_2 \dots x_i$ , where  $i$  is the number of samples.
- $T$  is the number of sample subsets.
- $H$  is the bagging ensemble and  $h_t$  is a base classifier.
- for  $t = 1, \dots, T$ :
  1.  $D_t = \text{createSubSetOf}(D)$
  2.  $h_t = \text{trainClassifierOn}(D)$
  3. add  $h_t$  to  $H$
- The final strong classifier is:  $H(x) = \begin{cases} 1 & \sum_{t=1}^T h_t(x) \geq \theta \\ 0 & \text{otherwise} \end{cases}$ , where, in general,  $\theta = \frac{1}{2}$ .

Figure 2.9: Pseudo-code of the bagging algorithm by Breimann [5, 6]

if the individual errors are different.

Tumer and Gosh [66] show the importance of diversity through the equation 2.14 for simple soft voting.

$$err_{add}^{ssv}(H) = \frac{1 + \theta(T - 1)}{T} err_{add}(h) \quad (2.14)$$

Thereby,  $\theta$  describes the correlation of the different classifiers  $h$ ,  $T$  is the number of ensemble members,  $err_{add}(h)$  the individual error and  $err_{add}^{ssv}(H)$  the error that is added to the ensemble by adding another classifier. If all classifiers have the same individual error  $err_{add}(h)$ , then the classifiers achieve maximum correlation at  $\theta = 1$ , and their errors are added in full to the ensemble error. However, if there is no correlation ( $\theta = 0$ ), then all of the classifiers are independent and the final ensemble error is the smallest.

As we would naturally assume, we need classifiers as independent as possible with a minimum degree of accuracy. However, high diversity does not guarantee a good ensemble. As Kuncheva [31] describes, we can have classifiers with a high diversity, but a poor combined classification result.



|             | $h(x) = +1$ | $h(x) = -1$ |
|-------------|-------------|-------------|
| $h(x) = +1$ | a           | c           |
| $h(x) = -1$ | b           | d           |

Table 2.1: Possible results of two different classifiers.

### Diversity Measurement

Following Kuncheva [31] and Zhou [92], we consider diversity measurements for binary classifiers. These measurements calculate the similarity of two classifiers. A high diversity means that two classifiers create different outputs for the same input. To compare these different outputs, it is necessary to have a set of labeled examples as a basis for calculating the diversity. Table 2.1 describes relationships of classifier outcomes. The variables  $(a, b, c, d)$  describe the number of occurrences where the possible outcome of the two considered classifiers are equal.

- $a$  is the number of examples where both classifiers have a positive outcome
- $d$  is the number of examples where both classifiers have a negative outcome
- $b, c$  are the number of examples where one classifier has a positive outcome and the other a negative (and vice versa)

Here,  $a + b + c + d = m$  and  $m$  is the total number of samples.

The Disagreement Measure [57, 24] of two classifiers  $(h_i, h_j)$  calculates the ratio of the number of classification results where both classifiers differ from the whole number of samples (see equation 2.15). There,  $d_{ij}$  can be at an interval of  $[0, 1]$  with a maximum value of 1 where the classifiers estimate all of the samples differently, therefore the two classifiers have the highest diversity. Accordingly, the lower  $d$  is, the lower the diversity.

$$d_{ij} = \frac{b + c}{m} \quad (2.15)$$

In 1900, Yule [89] proposed the Q-Statistic defined by equation 2.16 where, again,  $Q_{ij}$  describes the similarity of the two different classifiers  $h_i$  and  $h_j$ .

$$Q_{ij} = \frac{ad - bc}{ad + bc} \quad (2.16)$$

Also, a pairwise statistic is the Correlation Coefficient [58] of equation 2.17.

$$\rho_{ij} = \frac{ad - bc}{\sqrt{(a + b)(a + c)(c + d)(b + d)}} \quad (2.17)$$

## Increase Diversity

As already mentioned, diversity is essential to creating an ensemble and, therefore, increasing diversity is an important aspect. We can consider methods to increase diversity as a “shaking of parameters” to produce randomness or subdivisions within the ensemble-creation process.

- **Sample Manipulation:**

One way to increase diversity is to manipulate the training data. Bagging, for example, creates different subsets of the training data and selects another subset for every iteration. The idea is that the difference between these subsets is transferred to the classifiers and, therefore, causes a difference between the created classifiers which, ultimately, creates proper and diverse classifiers. In contrast, AdaBoost also manipulates the training set. Despite it changing the weighting of the training set, the set as a whole remains unchanged.

- **Feature Manipulation:**

As with sample manipulation, we can create diversity by using different subsets of the feature set. “The Random Subspace method” by Ho [24], for example. With every iteration, this method uses a different randomly chosen subset of the whole feature set for training the classifiers.

- **Learning Parameter Manipulation:**

The creation of base classifiers is determined by several parameters that can have different values. While the parameters affect the base classifier creation process, we can change these parameters which cause different classifiers, therefore increasing diversity. For example, we can vary the momentum or initial weights for creating a neural network.

## 2.5 Hopfield Neural Network

In 1982, John Hopfield introduced a Neural Network model in his paper [25] as an auto-associative memory called Hopfield Neural Network (HNN) in honour of its inventor. The Hopfield Neural Network (HNN) is a single layer, recurrent neural network with perceptrons as neurons. Here, every neuron has a connection to every other neuron except itself.

The HNN’s ability to rebuild a learned pattern from a noisy input makes it an attractive tool, especially when reconstructing images from learned examples.

Single layer means there are no hidden neurons between the input and the output neurons. Every neuron acts as an input and also as an output unit as depicted in figure 2.10. The relation between two neurons  $n_i, n_j$  is described by a numerical weight  $w_{i,j}$ . This relation can be seen as the importance of one neuron for another. The relation is symmetric, meaning there is only one link between two neurons, hence only one weight. Therefore, considering the weighting, it does not matter whether  $n_i$  activates  $n_j$  or vice versa.

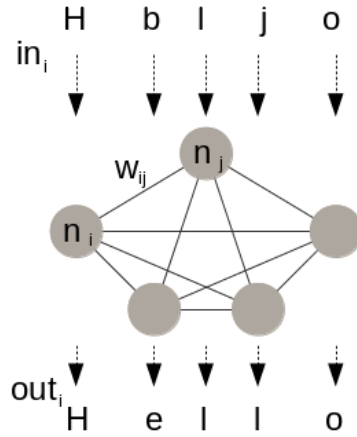


Figure 2.10: Every neuron is connected to every other neuron inside the Hopfield Neural Network except itself. It works as an auto-associator and can recall a learned pattern from a noisy input.

Equation 2.18 depicts the essential structure of the perceptron as a calculation unit. The neurons are composed of an input function  $s(x, j)$  and an activation function  $f(s)$  as described in figure 2.11. Thereby, input  $x$  is a vector  $x = (x_1, x_2, \dots, x_n)$

$$o_j = f(s(x, j)) \quad (2.18)$$

Equation 2.19 describes the input function which calculates the weighted sum of the input as an aggregated result for one neuron.

$$s_j(x) = \sum_{i=1}^N w_{ij} x_i, \quad (2.19)$$

The  $N$  in equation 2.19 is the number of neurons and  $w_{ij}$  is the weight between neuron  $i$  and neuron  $j$ . The other part of equation 2.18 is the activation function  $f$ . Thereby, we can distinguish between the binary and logistic activation function. Initially, a binary activation function (equation 2.20) is used which returns 1 if the sum is greater than a given threshold and  $-1$  otherwise. The other often used

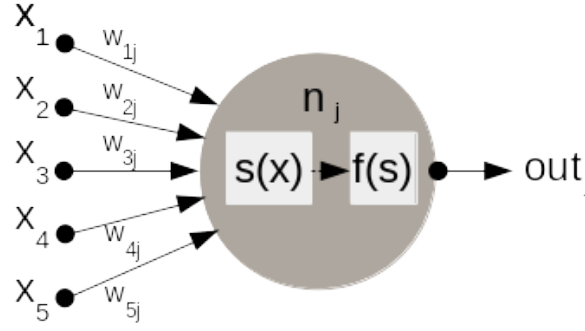


Figure 2.11: One neuron calculates the weighted sum of the input  $x$ .

activation function is the logistic activation function as shown in equation 2.21.

$$o_j = \begin{cases} 1 & s_j > \theta \\ -1 & \text{otherwise} \end{cases}, s_j \text{ see equation 2.19} \quad (2.20)$$

$$o_j = \tanh(s_j) = \frac{1}{2 + e^{-2s_j}} - 1, s_j \text{ see equation 2.19} \quad (2.21)$$

After activating neuron  $n_i$ , the result forms part of the next input  $x_i$ , or, collectively, all of the single neuron results become the new input vector  $x$ .

### Execution of the HNN

The Hopfield Neural Network can be executed synchronously or asynchronously. The calculation of the neuron outcome is the same in both directions. The scalar multiplication of both vectors  $w_j$  and  $x$  will be calculated for every neuron. Executing the activation function creates the outcome which becomes the new input. Thus, after executing the HNN, the input vector  $x^1$  results in output vector  $o^1$ , and therefore in the new input vector  $x^2 = o^1$ . This then provides us with  $x^1, o^1, o^2, \dots, o^T$  as the input/output vector sequence in time where  $T$  is the number of steps.

First, we consider the asynchronous procedure. In biological systems, every neuron acts on its own and sends its result immediately after the result is created. Therefore, we have a sequence in time instead of the synchronous procedure where we freeze the neuron's states and let everything happen in one step. If we consider when each neuron is firing, then the neuron that first arrives at an outcome changes the input vector for neighboring neurons.

The asynchronous procedure respects this time commitment and takes an arbitrary neuron. Then it calculates its outcome vector  $o$ , and immediately updates the input vector  $x$ . Hence, the input vector  $x$  changes once an arbitrary neuron

has finished its calculation. Afterward, we take the next neuron which now receives a different input vector to the previous neuron. As a consequence of the asynchronous executing, every single neuron gets a different input vector. With a

view to the above equations 2.18, 2.19, 2.20, and 2.21, input vector  $x^0 = \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix}$

becomes the new input vector  $x^1 = \begin{pmatrix} x_1 \\ o_2 \\ \dots \\ x_n \end{pmatrix}$  after executing neuron  $n_2$  and arriving

at output  $o_2$ .

The synchronous execution procedure of the HNN ignores this sequence in time for the calculation and assumes that the neurons are all activated at the same time. Accordingly, the synchronous method calculates the current output of the whole network by only considering one input vector and performing the calculation in a single loop across all neurons. The junction of the single outputs of every neuron together becomes the new input vector.

While both methods differ in terms of updating input vector  $x$ , calculating the outcome of one neuron is the same for both methods and also the recursive dynamic. The current vector is called state  $s^t$ . If we did not have criteria to stop the procedure, it would continue endlessly. However, the calculation will stop if the network becomes a stable state, meaning that the output does not change

anymore, in turn meaning that  $s^1 = s^2$ , respectively  $s^1 = \begin{pmatrix} o_1 \\ o_2 \\ \dots \\ o_n \end{pmatrix} = s^2 = \begin{pmatrix} o_1 \\ o_2 \\ \dots \\ o_n \end{pmatrix}$

and therefore  $o_i^1 = o_i^2$  for every element of the vector  $s^t$  or shorter  $s^t = s^{t+1}$  for any  $t > T_s$ , where  $T_s$  is the time at which the HNN becomes stable.  $o_{i1} = o_{i2}$  This stable state is the learned pattern.

### Hebb-learning of the HNN

The HNN is trained using the Hebb-learning rule (equation 2.22) which has a high computational performance. The learning performance only depends upon the dimension of the input vector. Hopfield has shown in his work [25] that an

HNN with a symmetric weight matrix will converge to a final stable state which is ensured by equation 2.22.

$$w_{ij} = \sum_{m=1}^M x_i^m \cdot x_j^m \text{ if } j \neq i, w_{ij} = 0 \text{ otherwise,} \quad (2.22)$$

where  $M$  is the number of patterns and  $x_i, x_j$  are the input pattern ( $x$ ).

## 2.6 Extensions of Features and Algorithms

The original Haar-like features achieve good results in terms of finding near frontal faces. However, not all faces in images are straightforward frontal faces like those of passport photos, for example. Therefore, Lienhard and Mayedt [36] extended the Haar-like features of Viola and Jones to include a 45-degree rotated variant.

Also, Viola and Jones [68] extend their features to diagonal shifted and overlapping rectangles which they call diagonal features. Viola et al. [73] further extended their features to use them for pedestrian detection in videos. Mita et al. [43] extended the Haar-like feature by combining them with a new feature they called a joint Haar-like feature. They measured the co-occurrence of a few Haar-like features to train more accurate features. Another method to improve the features was propounded by Wu et al. [77] who proposed a histogram of the Haar-like feature values. Later, Huang et al. [26] followed their approach and used a tree cascade structure for multi-view face detection. They equally bin the values and use the extended feature as an input for the RealBoost [16] learning algorithm. While the number of bins is important to the success of the classifiers, the system tends to overfit if the number of bins is too high.

Beyond the Haar-like features, another way to improve accuracy is to use improved base classifiers. Therefore, Brubaker et al. [8] proved that Classification and Regression Trees (CART) [7] can be used as base classifiers to improve various boosting ensemble methods. Also, Xiao et al. [81] proposed an improved weak classifier which they call the Bayesian stump. This weak classifier is created by a split and merge approach using three different feature sets, which are Haar-like features, Gabor wavelet features, and EOH features. Levi and Weiss [33] introduced the local edge orientation histograms (called EOH). When training an AdaBoost ensemble, they also use three features. Besides EOH features, they also use Haar-like features and the mean intensity in a given rectangle. The EOH feature is created by the gradients given by convolving Sobel mask and dividing the edges into bins. Xiao et al. [81] also introduced an altered cascade training algorithm

that they called “Dynamic Cascade”. They claim it improves the handling of large datasets as well as learning from a few samples.

To overcome the lack of handling extreme lighting conditions of the original Haar-like features, Fröba and Ernst [18] proposed a modified census transformation to get illumination-intensity features which compare pixel values with an intensity mean in the local neighborhood. The Local Binary Pattern (LBP) proposed by Ojala et al. [45] for texture classification is a set of features unaffected by illumination changes. Amongst others, Jin et al. [29] used a Bayesian Framework and Zhang et al. [91] applied a boosting approach to successfully use LBP for face detection. Inspired by LBP, Yan et al. [84] introduced locally assembled binary features, which provide good results for a wide range of face detection image sets.

Liu and Shum [38] proposed the “Kullback-Leibler boosting”, which uses the Kullback-Leibler divergence of positive and negative histograms. Wang et al. [74] used Fisher discriminant analysis to create a linear feature. Both approaches ([74], [38]) create good features similar to face templates, but handle them with care to avoid overfitting. Similar to EOH from Levi and Weiss [33], Dalal and Triggs [11] proposed their “Histograms of oriented gradients” which became very popular for pedestrian detection. A statistical approach, published by Tuzel et al. [67], used region covariance which involves the fast creation of the integral image for detection and classification. Huang et al. [27] obtained very good results with their proposed sparse features.

Despite better features, the means of selecting these features (found the good one) plays an important part, which was focused on by different authors. Yuan et al. [88] addressed this and used the frequent item-set mining scheme that is often applied in data mining. Han et al. [21] used the Swendsen-Wang Cut algorithm [61] to create partitions with individual subsets of compositional features. This procedure is repeated to train every base classifier, which is subsequently selected and combined by AdaBoost. This approach provides very good results for person detection. Opelt et al. [46] formed a boundary model as the base classifier. Contour-based features are also used by Shotton et al. [55]. These features, built on top of a dictionary of contour fragments, are trained to an object detector by a boosting algorithm. Shapelet features proposed by Sabzmeydany et al. [50] use low-level gradient information from local image regions.

Aside from different features and classifiers, there are many suggestions to improve the original boosting methods of Freund and Schapire [15] and AdaBoost by Viola and Jones [69]. Li et al. [34] proposed FloatBoost. which incorporates the FloatingSearch approach by Pudil et al. [48]. Instead of the greedy sequential

search of AdaBoost, FloatBoost also takes already learned base classifiers or features and removes the least valuable features. The outcome of this are ensembles with fewer base classifier as the authors [34] argue. Jang and Kim [28] presented an evolutionary algorithm to reduce the number of classifiers -by 40%- but keeping the overall ensemble accuracy.

While Viola and Jones [69] trained every cascade node independently, some authors suggest solutions that retain the knowledge of further trained nodes. For example, Xiao et al. [82] created a chain structure that used the further trained classifiers as a prefix and started training up to these. In addition, Wu et al. [77] changed the independent node training by creating a nested structured cascade which took the confidence of further trained classifiers and used them as a feature for the first base classifier. Both claim to achieve better detection results compared with the original Viola and Jones algorithm.

In 1999, Mason et al. [41] proposed a gradient descent boosting which they called AnyBoost. In 2000, Friedmann et al. [17] published a statistical interpretation of boosting. Inspired by these two publications, Masnadi-Shirazi and Vasconcelos [39] proposed another asymmetric boosting which especially minimizes the exponential cost criterion of AnyBoost and achieves a very high detection rate [40]. Wu et al. [79] used the “Forward Feature Selection” method of [75] to obtain a set of features and a linear asymmetric classifier (LAC) which creates the ensemble results by voting among the selected features.

Configuring a cascade is a challenge. A threshold is set to distinguish objects and background. Here, we have to balance the need for speed with the need for accuracy. In fact, both depend upon one another. If the threshold is set to remove more background, the speed will be increased, but this will also decrease accuracy. Inversely, setting a high threshold will reduce speed while improving accuracy. Lienhart et al. [35] addressed this by targeting their cascade creation process to reject 50% background for every node and achieving 0.1% false negatives.

Sochman and Matas followed a different approach [59]. Instead of considering the nodes of the cascade, they approximated the joint likelihood of all base classifiers. An automatic solution was offered by Brubaker et al. [8] by training the node thresholds against validation data.

Bourdev and Brandt [4] proposed a scheme to sequentially increase the number of negative samples for every cascade node. Also, they removed correctly classified negatives and replaced them with new ones. The effect is that every ensemble is trained with a different set of negative samples and a further increased set of samples. Only the misclassified negatives are left over. Every ensemble of a cascade



node is trained with a given maximum of the false-positive rate. By increasing of the number of negative samples, it became more and more difficult to reach the expected false-positive rate, and therefore the ensemble grew and became more complex.

A very time-consuming process involves the selection of features to create an ensemble because there is a need for a large feasible set of features, and base classifiers have to be calculated in every iteration of the (AdaBoost) ensemble creation process for the whole set of features. Notably at an early stage, training a whole cascade classifier of AdaBoost ensembles takes weeks and sometimes months. To overcome this drawback, many authors published variants of the original algorithm to reduce the amount of time needed to create an ensemble. For example, Brubaker et al. [8] tested different ways to reduce the size of feature sets and showed that random selection to create a subset of the original set provides ensembles with comparably good results in terms of accuracy. Wu et al. [80] use “Forward Feature Selection” [75], and claimed that they are much faster than AdaBoost while retaining comparable accuracy. Instead of training every weak classifier again during each iteration, they train them once and add this weak classifier which most improves the ensemble.

Another time-consuming aspect is the detector location test. Many windows must be analyzed to determine whether there the sought object is present, or whether there is just background. Many suggestions are made to improve detector speed. As an example, Schneiderman [54] proposed a feature-centric cascade. There, a set of features is computed in advance so these features can be shared for several window lookups.

Deep neural networks in tandem with face detection exhibit excellent performance for many machine learning fields. Zhu et al. [93] called their method for face detection Contextual Multi-Scale Region-based Convolution Neural Network (CMS-RCNN). Yang et al. [87] trained CNN to find facial parts and scores parts by spatial structure and arrangements. Both methods present impressive results.

## 2.7 Training and Test Sets

Training and test sets are essential to creating classifiers, and especially to revising and reproducing experiments. Thus, we describe the sets we used to train and analyze our models.

### CBCL Face Database

The CBCL Face Database [3] is a set of already cropped training and test images. The images are all gray-valued images with a size of 19x19 pixels. The training set includes 2,429 face examples and 4,548 non-face examples initiated by Sung [60] and Heisele et al. [23]. The test set, created for Heisele et al. [23], comprise 472 faces and 23,573 non-faces and is a subset of the CMU Test Set (see below).

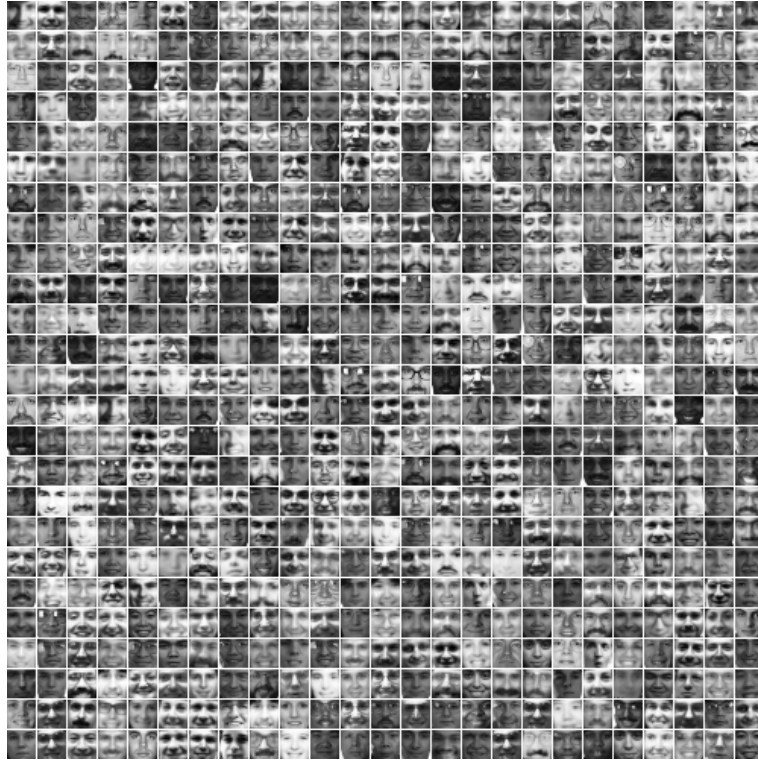


Figure 2.12: CBCL Face Database Train Images Samples

### CMU Test Set A

The CMU Test Set A, introduced by Rowley et al. [49], contains 42 images and 169 faces. The authors Rowley et al. [49] created the image set by scanning photographs and newspapers and then adding images from the internet.

### Test Set Sung and Poggio

Rowley et al. [49] also used the test set by Sung and Poggio [60], which they referred to as CMU Test Set B. It contains 23 images with 157 faces, ranging from high-quality camera pictures to low-quality scans as stated by Sung and Poggio [60].



Figure 2.13: CMU Test Set A by Rowley et al. [49] contains 42 Images



Figure 2.14: Test Set by Sung and Poggio [60] contains 23 Images



Figure 2.15: CMU Test Set C by Rowley et al. [49] - contains 65 Images

### CMU Test Set C

CMU Test Set C, also introduced by Rowley et al. [49], contains 65 images comprising 183 faces. Test Set C is similar to Test Set A, but includes many images with a more complex background while also comprising pictures that show no face at all.

Overall test sets, we have 130 images containing 509 faces and about  $3,67 \cdot 10^7$  non-face frames that have to be tested.

## Chapter 3

# The Haar-Feature-Like Patch and Hybrid Diversity Approach

### 3.1 Introduction

In this chapter, we introduce our approach for increasing the accuracy of ensemble and cascade classifiers. We aim to improve these classifiers by increasing the diversity of the underlying classifier and feature sets. Therefore, we create differently shaped geometrical features and two classifier models to be introduced in the methods section 3.2. In section 3.3, we combine our models and a simple threshold classifier with a hybrid architecture and examine their benefits. In real applications, nearby hits are merged into one region. However, we use the detection results without merging because we want to compare our results without the influence of the merging mechanism. Merging nearby regions reduces the false positives, but also the detections shown in section 3.4 to gain an idea of how such a method changes our results.

### 3.2 Methods

#### 3.2.1 Introducing Haar-Feature-Like Patches

As is the case with Haar-like features, we do not use pixels directly, but the pixel sum of a dedicated rectangle. Instead of Haar-like features, which use the subtraction of several rectangle pixel sums, we use the pixel sums of every single rectangle directly as a feature. Our feature, which we call Haar-Feature-like patch (HFP), can be described as a geometrical group of several rectangles or a template-like structure. The rectangles that compose our feature can differ in terms of their

width, height, and relative positions. We choose the name Haar-Feature-like patch to correspond with Haar-like features, which are the source of inspiration for the HFP.

However, we want to emphasize the main difference to Haar-like features. The HFP does not return one value, but a vector of values. This leads to the following consequences: First, the single rectangles of Haar-Feature-like patches do not have any dependency on other rectangles in the way that Haar-like features do. The rectangles within a Haar-like feature are marked as positive or negative. The value of the Haar-like features is calculated by subtracting the pixel sums of the positive and negative rectangles. As a consequence, different rectangle pixel sums lead to the same subtraction result and, therefore, to the same feature value. To a certain extent, this subtraction is a first interpretation of the source, which we avoid by using Haar-Feature-like patches with the original pixel sums of all rectangles.

### **Pixel sums to handle noise and prevent overfitting**

One reason for using Haar-Feature-like patches is to be able to consider the circumstances of real visual scenes as changes in lighting, scaling, and perspective. Using pixel sums instead of the pixel values themselves will improve the handling of noise and thus prevent overfitting. Overfitting occurs if features reproduce every small detail that exists only in the training images. If a classifier uses this detail to distinguish between the given classes, the classifier perhaps handles the training image perfectly but has a poor generalization ability. Using the accumulated area sums instead of single pixels, the dedicated single pixel within the considered rectangle becomes less important because we use a group of pixels in a greater area of the image what is an average of pixels in the considered rectangle. Therefore, every single pixel is just one part of creating this average. We may encounter problems if one dedicated pixel is learned because images of natural scenes change their appearance and it is unlikely that one dedicated pixel of an image has the same value in the image that was taken a second later. By using an area of pixels, the single pixel value will become fuzzier and less meaningful compared to the sum of all the other pixels. Noise within this region can be balanced.

### **Patch for compact modeling complex realities**

Our main motivation for introducing Haar-Feature-like patches is the ability to perform modeling. The aim of creating these features is to obtain a more flexible feature so as to be able to model more complex scenes using just one feature. The

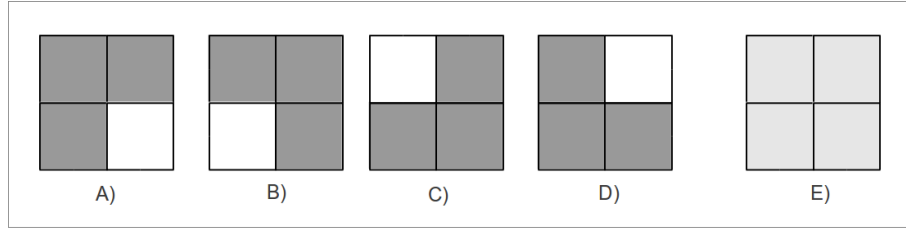


Figure 3.1: Modeling one Haar-feature-like patch (E) instead of four Haar-like features (A-D). If we wanted to use a corner shape model, we would have to create four Haar-like features to get a feature for every possible corner. Using every value on its own, we can model all corner shapes using one feature.

drawback of Haar-like features is that we have to create many features if we want to model such scenes. Figure 3.1 shows a “corner model”. If we want to arrive at a model for every possible corner, we have to create four Haar-like features as depicted in figure 3.1. Instead, our feature uses the structure directly as a patch, and this template-like view on Haar-like features is what we call the Haar-Feature-like patch (HFP). Hence, our HFP is similar to Haar-like features, but we use the vector of pixel sums instead of calculating a single feature value.

In the context of ensemble methods, we often have large feature sets and, therefore, a long training time. Apart from having more freedom to adapt to a given scene, it is beneficial to have a more compact set as well as fewer features by retaining the ability to model natural scenes. However, interpreting Haar-like features as a template patch leads to some differences.

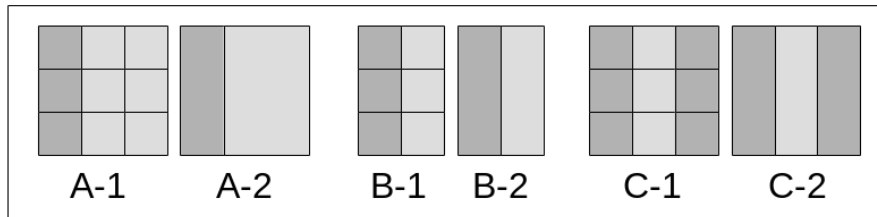


Figure 3.2: Figures A-1 and A-2 have no difference if we calculate the Haar-like feature value, but they are very different used as Haar-feature-like patch. The same applies to B-1, B-2 and C-1, C-2

Some Haar-like features look different given the rectangles on which they are built, but this does not make any difference when calculating their feature values. We illustrate this in figure 3.2. If we calculate the difference between the light gray and the dark gray rectangles, it does not matter how many rectangles there

are; only the whole area matters. If a group of rectangles covers a rectangle area, we can also use the overlaying rectangle to calculate the Haar-like feature value. There is in fact a difference in computational speed, but the value will be the same. While the two left features A-1 and A-2 in figure 3.2 do not make any difference when calculating their Haar-like feature value, these two are very different Haar-Feature-like patches. In comparison, features A-1 and C-1 are the same as HFP, but we will end up with two different values as a Haar-like feature.

While we can create many Haar-Feature-like patches by simply removing or adding rectangles, for the analog Haar-like features we have to decide which are the positive parts and which are the negative parts for the subtraction as described by the corner features in figure 3.1. Introducing many opportunities create a huge number of Haar-like features, thus requiring a lot of computational resources. Haar-Feature-like patches make it easier to handle these complex structures.

However, we have to be aware that the possibilities of modeling Haar-Feature-like patches by adding various rectangles can also create a huge number of features, perhaps more than we actually want to handle. At some point, we have to stop modeling to avoid drowning in the ensuing flood of features. However, finally, with fewer features, we can describe more dedicated scenes compared to Haar-like features. We inherit high computational performance from Haar-like features by using the Integral Image (see section 2.2.2) and achieve some degree of scaling invariance.

## Types of Haar-Feature-like patches

To examine our Haar-Feature-like patches, we use three different sets. The different shapes shown below are prototypes. We can use them to derive the concrete Haar-Feature-like patches which ultimately constitute the HFPs that we use for training and classification.

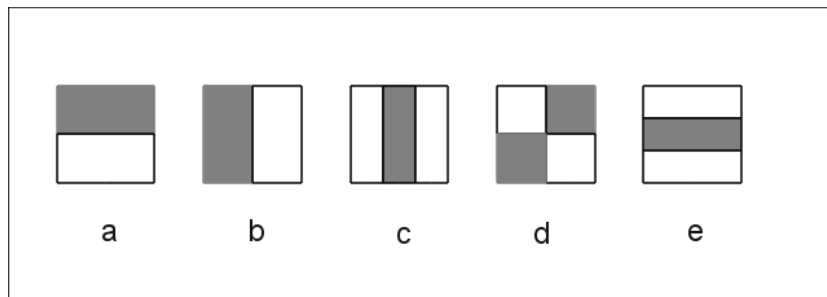


Figure 3.3: Haar-like features used by Viola and Jones [69].



### Haar-Like Features (Base) as HFP

The first group of HFPs are the feature shapes used by Viola and Jones [69] and described in figure 3.3. While these Haar-like features are made to calculate the difference between the two rectangles, we use the dedicated area values of the features as a vector.

### Tetris-like and Blocks 6 and 9 (TLB69) HFP

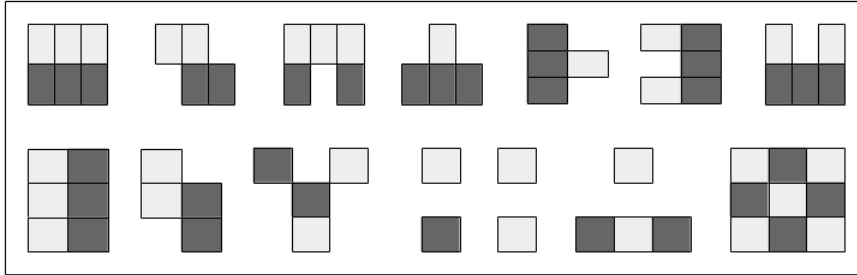


Figure 3.4: These Tetris-like and Block shapes consist of either six or nine boxes with a width and height of 2x3, 3x2, and 3x3 rectangles.

Figure 3.4 shows HFP shapes that are similar to the shapes used in the well-known game Tetris. The maximum dimensions of every shape are 2x3, 3x2, and 3x3 cells. While we also use this HFP set as Haar-like features for training threshold classifiers, we mark the dark gray and light gray areas to calculate the difference. As seen in figure 3.4, we start with the full block and then cut out cells from the whole block to arrive at the final shape, some of which look like the shapes used in Tetris.

### Several blocks of various width and height (BlockN) HFP

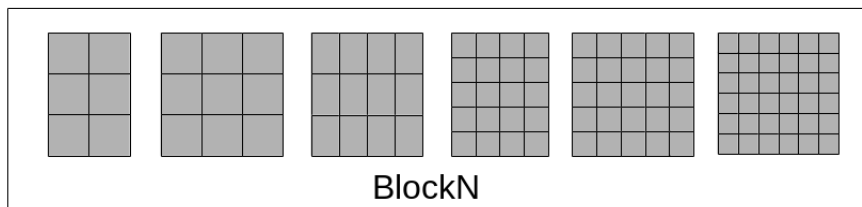


Figure 3.5: This figure shows the BlockN HFP set. We create these blocks by systematically increasing the number of rectangles in a row and a column.

```

 $n_r = startRows$ 

 $n_c = startColumns$ 

until( $width < maxWidth$ )

    until( $height < maxHeight$ )
        createBlock( $n_c, n_r, w_c, w_r$ )
         $n_r = n_r + 1$ 
         $height = n_r h_r$ ;

     $n_c = n_c + 1$ 
     $width = n_c w_c$ ;

 $n_r = startRows$ 

```

Here,  $w_r, w_c$  are the width of a row and column and  $h_r, h_c$  their height.

Figure 3.6: Systematically, we create BlockN HFPs by increasing the number of rows and columns of a source HFP.

In figure 3.5, we show the HFP set BlockN. As the name suggests, the block features are made up of rectangles which take up all of the places in the rows and columns. The number  $N$  at the end of the name "BlockN" stands for the number of rectangles the feature consist of. The number of rectangles within one row must not be the same as the number of rectangles in one column. These widths and heights are chosen systematically, beginning with a size of 2x3, and then increasing the number of rectangles in a row by  $row_{next} = row + 1$  until a given maximum width for the whole feature is reached. The same is done with the columns  $col_{next} = col + 1$  (see pseudo-code 3.6).

## Expansion and Derivation of Haar-like-features patches

All of the described sets are prototypes where the concrete widths and heights of the cells do not matter. From these prototypes, we derive the concrete Haar-Feature-like patches by setting the width and height of the single cells. We create a whole set by systematically creating the features with all possible widths and heights. As a result, all of the cells in a row have the same height, while all of the cells in a column have the same width. The number of derived HFPs is limited by

the maximum width and height of the single HFPs that we want to create. In most cases, the maximum width and height are restricted by the width and height of our training images because we cannot use an HFP that is bigger than the image to which we want the HFP to apply.

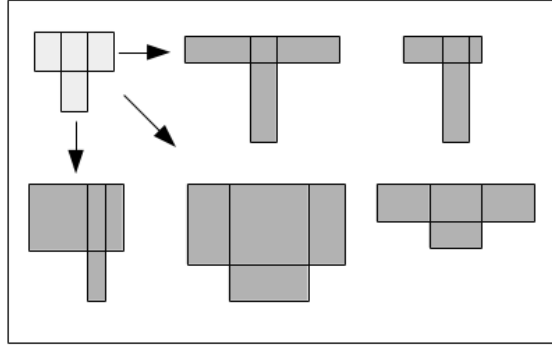


Figure 3.7: We create different appearances of an HFP source shape by stretching it horizontally or vertically. As illustrated in this figure, we create several HFPs by expanding one source HFP. We then incrementally increase the overall width and height, thereby creating every possible incremental increase in height for each incremental increase in width until a maximum height or width is reached.

### 3.2.2 Classification using Haar-Feature-Like Patches

In the following section, we describe in detail how we use the Haar-Feature-like patches as a classifier, and how we can train and execute them.

#### Haar-Feature-Like Patch as a Classification Model

Inspired by Viola and Jones [71] and their combination of threshold classifiers and Haar-like features (see section 2.2.2), we also use one Haar-Feature-like patch as the input for one classifier. We classify an input by calculating and comparing the HFP vector with a learned pattern. Therefore, we first apply the HFP at a relative position within the given input frame, thus resulting in a vector of pixel sums. Each pixel sum will be normalized to their underlying area; hence, we have a vector of values between 0 and 255 (see section 3.2.1). After calculating the distance  $d$  between the HFP vector and the learned pattern, we can finally test whether this distance is less than a given threshold that determines the classification result as

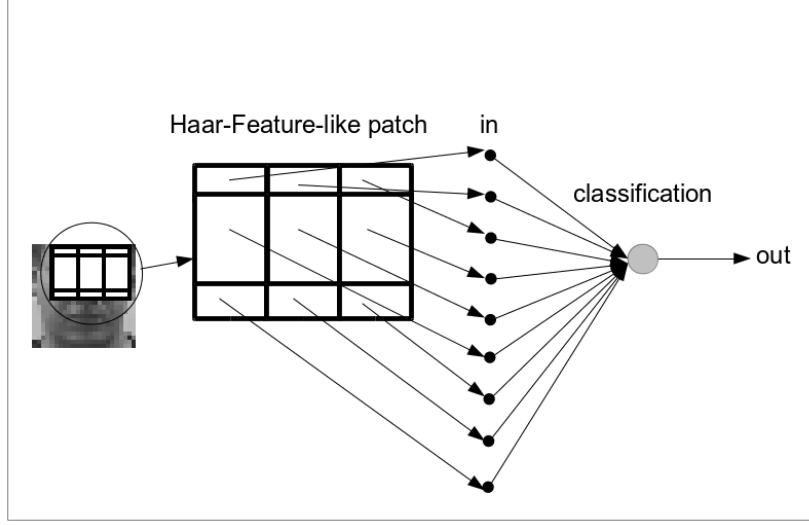


Figure 3.8: Every HFP has a position within the corresponding frame. The pixel sums within the Haar-Feature-like patch (HFP) constitute the input for classification. First, we calculate the HFP values, which are the pixel sums of the rectangles from which the HFP is built. Second, we compare this vector with a previously learned pattern.

described in equation 3.1.

$$h(x) = \begin{cases} 1 & \text{if } d(x, p) < \theta \\ 0 & \text{otherwise} \end{cases} \quad (3.1)$$

Distance measure  $d$  is calculated using the Euclidean distance (equation 3.2) and the Normalized Cross-correlation (equation 3.3).

### Euclidean Distance

$$d_e(x, p) = \sqrt{\sum_{i=1}^N (x_i - p_i)^2} \quad (3.2)$$

where  $N$  is the size of the pattern.

### Normalized Cross-correlation

Another measurement of similarity often used in signal processing is the Normalized Cross-correlation. Normalized Cross-correlation calculates the similarity of two vectors by comparing their single values. The more values of a vector (or a

matrix) that are near each other ( $x_i$  is similar to  $y_i$ ), the more vectors  $X$  and  $Y$  correlate.

$$d_{ncc}(x, p) = \frac{\sum_j \sum_i (x_{ij} - x_a) (y_{ij} - y_a)}{\sqrt{\sum_j \sum_i (x_{ij} - x_a)^2 (y_{ij} - y_a)^2}} \quad (3.3)$$

where  $x_{ij}, y_{ij}$  are the pixel values of image  $X$  respectively  $Y$  at the position  $(i, j)$ , and  $x_a, y_a$  are the averages of all pixel values of image  $X$  respectively  $Y$ .

We add the abbreviation of the distance measurement method as a prefix to the classifier name to show which method a classifier uses. Therefore, we use NCC-HFP for Normalized Cross-correlation and ED-HFP for the Euclidean distance.

## Training the HFP classifier

We train the HFP classifier by calculating the average of the HFP values for a set of positive sample images as described in figure 3.9. Therefore, we calculate the values of one Haar-Feature-like patch of a current training image in a fixed position. We repeat this calculation and then sum all of the values. After iterating all of the positive images, we divide the values by the number of images used, thus arriving at the average values for the entire training set. Finally, we learn the decision threshold of equation 3.1 by minimizing the error according to the training set as per the threshold classifier from Viola and Jones (see section 2.2.2). We repeat this procedure for every possible position by applying the HFP to every step and scale where the HFP stays within the training image size.

### 3.2.3 Combining the Hopfield Neural Network and Haar-Feature-Like Patches

The second classifier model which uses the Haar-Feature-like patches is the HaarNN classifier that includes a Hopfield Neural Network. By introducing the Hopfield Neural Network, we want to use its ability to recreate patterns of noisy input and its dynamic to increase diversity. We have called the combination of a Haar-Feature-like patch and a Hopfield Neural Network "HaarNN". "Haar" of "HaarNN" emphasizes the connection to the Haar-Feature-like patch and, considering the upper cases of HaarNN, we have HNN (Hopfield Neural Network). As is the case with the HFP classifier, we add the used distance measure method as a prefix, thus leading to NCC-HaarNN or ED-HaarNN.

As for the HFP model, we also use one Haar-Feature-like patch for one HaarNN classifier and apply the HFP to arrive at the feature vector described in section

```

i = 0

for EachHFP

    for Eachposition

        i ++
        for EachPositiveTrainingSample
            hfpValuer = calculateHFPValue(sample, hfp, position)
            hfpSum += hfpValue
        hfpAverage = calculateAverage(hfpSum)
        hfpClassifieri = searchForBestThreshold(hfpAverage)

```

Finally, we get a set of HFP classifiers (*hfpClassifier*) with the amount  $N = f_n p_m$ , where  $f_n$  is the number of HFPs and  $p_m$  the number of possible positions.

Figure 3.9: Pseudo-code for learning HFP classifiers for all Haar-Feature-like patches and relative positions within the training images.

The HFP classifier is a compound of:

- Haar-Feature-like patch
- Relative position of the HFP
- Distance Measure Method
- Learned pattern
- Decision threshold

The HFP classifier first calculates the HFP values, followed by the distance to a previously learned pattern. The learned pattern is thus the average of all positive samples. If the distance is less than a given decision threshold, the classification result is positive.

Figure 3.10: Summary and short description of the HFP classifier

3.2.2. Therefore, except for the HNN, both models are equal. However, while an HFP classifier uses the feature vector directly for classification, the HaarNN classifier uses the feature vector to execute the Hopfield Neural Network. As depicted in figure 3.11, one area of the HFP is connected to one input node of the HNN. After executing the HNN, we obtain a new vector, the stable state of the HNN which is our final result pattern used for classification. The HNN can be considered as a transformation towards a learned pattern. Without any classification, we can consider the HaarNN to be a HNN filter which can be learned using the HFP structure.

The Hopfield Neural Network has three characteristics which encourage us to use it.

- Recall pattern of noisy input
- Good performance of the learning method
- High dynamic, non-linear method to increase diversity

Our first motivation for using Hopfield Neural Networks is their ability to reconstruct a learned pattern from a noisy or incomplete input. Restoring a pattern from a noisy input is, besides using rectangle pixel sums, another potential means of preventing overfitting.

The underlying Hebbian learning is a simple, straight-forward learning rule. The HNN provides us with a fast method of training the HaarNN, which is our second reason for using it. Together with the HFP, we can retain the simplicity of the composition, which is an additional motivation for this model. We do not have to learn the weights by employing a costly method such as backpropagation where the weights have to be optimized for error. Instead, by using the HNN, every new pattern is simply an addition to the weight matrix, and especially a fast training method is key to training an ensemble within a reasonable period of time. The HNN shares the same relative fast execution performance as is the case with other neural network approaches.

The final, yet extremely important motivation, is the HNN's nonlinear dynamic which we can increase by slightly modifying the learning parameters. The nonlinear dynamic of base classifiers is used as a basis for diversity and creating feasible ensembles (see Kuncheva [31]).

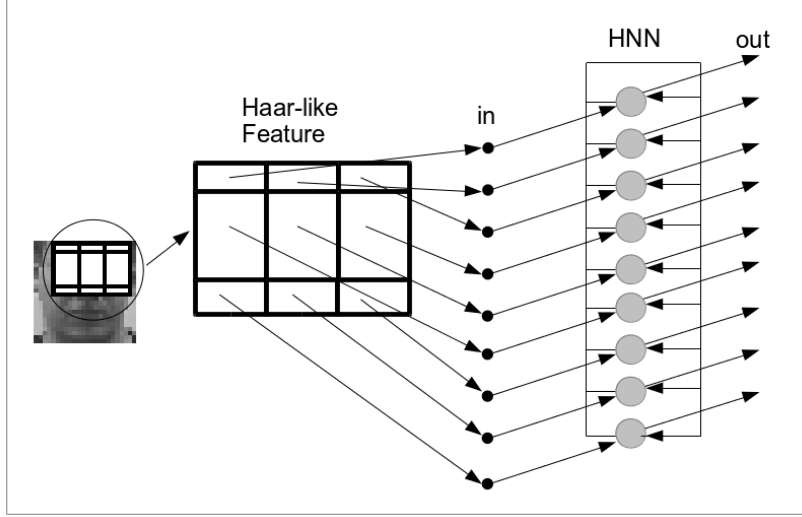


Figure 3.11: The pixel sums within the Haar-Feature-like patch (HFP) are the input for a Hopfield Neural Network (HNN). Executing the HNN leads to a stable state, i.e. a vector that does not change anymore. This final vector is the input for classification or the output of the HaarNN if used as a learnable filter.

### Classification using the HaarNN model

The classification is the same as for the HFP classifier model, except for the intermediate executing of the HNN. We can consider the HaarNN as a learnable filter for preprocessing the current input. This preprocessed input is used to calculate the distance to a learned pattern. As for the HFP classifier model, we use the Euclidean distance (ED-HaarNN) and the Normalized Cross-correlation (NCC-HaarNN). A third classification metric which uses the stable state of the HNN and, therefore, does not work for the HFP classifier, is the Bayes-like Probability (BP-HaarNN).

### Training the HaarNN model

Training the HaarNN model first involves training the weights of the Hopfield Neural Network which is performed by way of equation 3.7. Iterating all of the possible Haar-Feature-like patches and positions, and subsequently applying the training steps to all of the training samples, is the same as for the HFP model (see section 3.2.2). Since we only want the HNN to memorize positive patterns, we only use the positive samples to learn the HNN weights. While the HFP model only calculates the values of the Haar-Feature-like patch, executing the HaarNN requires some additional preparations. In short, we have to do the following:



- Calculate the offset  
shift the input values of  $x$  to equalize between positive and negative so that  $\max(x_i) + \min(x_i) = 0$
- Calculate the HNN weights  
update the weights using equation 3.7
- Calculate the activation threshold

### Calculating the offset

The first step in training and executing the HaarNN is to subtract an offset from every value of the input vector. This shifting is done to balance the input between the negative and the positive values because, otherwise, the execution process of the Hopfield Neural Network as we use it would always result in the maximum positive values. The execution process of the HNN calculates the product of the input vector and the HNN's weight matrix and then by applying the logistic (equation 2.21) or binary (equation 2.20) activation function. As an input we use the values of the pixel sum normalized to the corresponding area, and therefore the values are between 0 and 255.

Without shifting the values, there would only ever be positive values because the HNN's weights are trained by multiplying pattern values (equation 3.7 and 2.22) and execution is performed by multiplying the previously calculated weights with the input vector (equation 2.19). Finally, these results will be summed (see section 2.5), and always starting with positive values would increase all of the values. Therefore, to obtain the HNN dynamic, we create opponents to balance the input values between the positive and negative values by calculating and subtracting the average of the minimum and maximum of the input vector values as depicted in figure 3.12. We arrive at these opponents by calculating an offset  $\phi$  (see equation 3.4) and adjusting the current input vector  $x$  by applying  $x_i^s = x_i - \phi$  (see equation 3.5) to obtain the final vector  $x^s$ , where  $x_i^s$  are the values of the final input vector.

$$\phi = \frac{\max(x) + \min(x)}{2}, \quad (3.4)$$

where  $x$  of equation 3.4 is the input vector given by the Haar-Feature-like patch.

$$x^s = \text{forEach}(x_i - > x_i - \phi) \quad (3.5)$$

$x^s$  is the shifted vector of  $x$ .

For example, if the input contains values in the interval of  $[0, 255]$ , so  $min = 0$  and  $max = 255$ , the offset is 122 and the new interval becomes  $[-122, 123]$ .

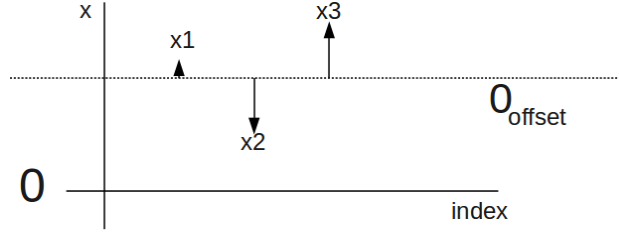


Figure 3.12: To balance the pattern and input between positive and negative values, we shift the average of the maximum and minimum to zero.

$$\phi_a = ave(\phi_m) \quad (3.6)$$

Throughout the training process, we calculate and apply the offset for every training example. The final offset used to execute HaarNN classifier is the average of these offsets as described in equation 3.6.

### Calculating the HNN weights

After calculating the Haar-Feature-like patch and subtracting the offset, we arrive at the feature vector used to calculate the weight matrix of the HNN by applying the Hebbian learning rule. To calculate the weights, we use equation 3.7 (see also equation 2.22 in section 2.5).

$$w_{ij} = \sum_{m=1}^M x_i^m \cdot x_j^m \text{ if } j \neq i, w_{ij} = 0 \text{ otherwise,} \quad (3.7)$$

Originally,  $M$  is the number of different patterns and  $x$  the feature vector. However, as we train the HaarNN,  $M$  is the number of positive training samples. If we train a multi-class classifier, the patterns  $M$  will be orthogonal to one another [1] to recreate the several different patterns best. However, our patterns are very similar, and we consider this a binary classification task. Nonetheless, we use this equation and feed our HaarNN with many similar patterns. In contrast, we could use the average pattern as the only input for the HNN to learn exactly one clear pattern. Finally, the weights will be normalized by the number of samples.

### Calculating the activation threshold

While both equations 2.20 and 2.21 use values in the range of  $[-1, 1]$ , our values vary in terms of their ranges. Therefore, we adapt the equations 2.21 and 2.20 as follows:

$$o_j = \frac{2\theta}{1 + e^{-s_j\beta}} - \theta \quad (3.8)$$

where  $\theta = \max(|x_i^s|)$  of the learned vector  $x^s$ . The parameter  $\theta$  is the activation threshold or just the *threshold* that will be learned by calculating the average of the thresholds of all training samples, as is the case when calculating the offset.

$$o_j = \begin{cases} \theta & \text{if } s_j > \theta \\ -\theta & \text{if } s_j < -\theta \\ s_j & \text{otherwise} \end{cases} \quad (3.9)$$

Here,  $s_j$  is the result of applying the weights for the neuron  $j$  as described in equation 2.19.

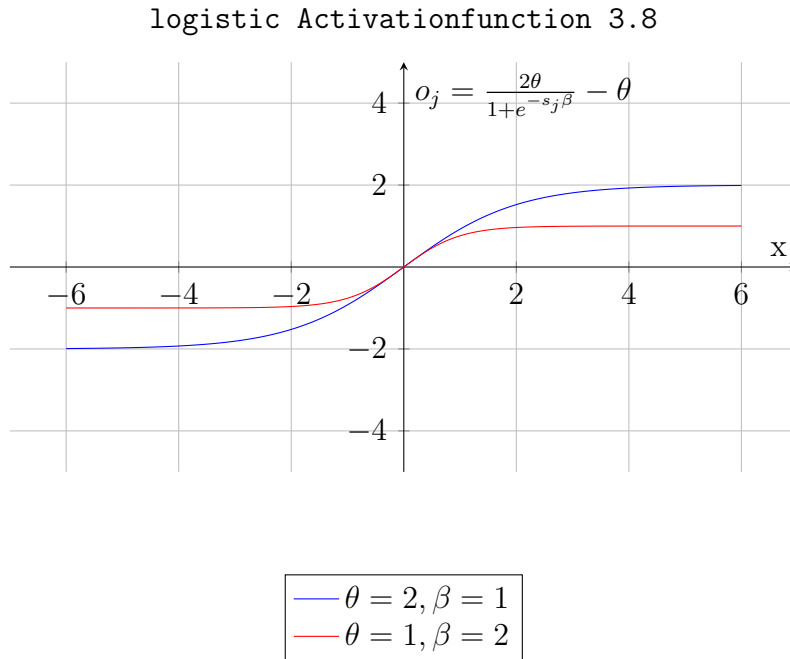


Figure 3.13: Activation function using different parameters to adapt to different feature value ranges.

### Classification using Bayes-like Probability

The Bayes-like method measures the probability of a stable state belonging to a face or background by counting how often a stable state occurs for a positive or negative training sample. The final result is determined by calculating the difference between the face and the none-face probability (see equation 3.10).

$$h(x) = \begin{cases} 1 & \text{if } P(S_i, \text{face}) > P(S_i, \text{none} - \text{face}) \\ 0 & \text{otherwise} \end{cases} \quad (3.10)$$

where  $S_i$  is the current stable state of the HNN and  $P(S_i, \text{face})$  is the probability that  $S_i$  belongs to be a face.

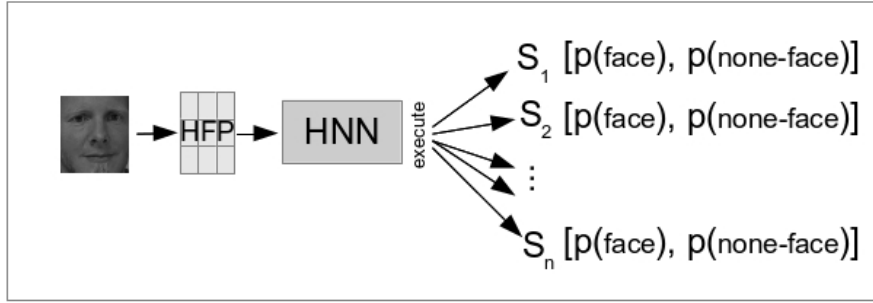


Figure 3.14: After execution, the HNN will converge to a stable state ( $S_i$ ) which is assigned to a set of face and none-face probability values.

### Training the Bayes-like Probability

If the HNN learns exactly one pattern, we can compare whether or not the stable state of the HNN after execution is equal to this pattern. However, the pattern we use for comparison is an average of all training images. Further, we train the HNN with many slightly different patterns, and therefore it cannot be precisely the average pattern which the HNN recreates. There are generally several stable states that the HNN adopts. The idea behind the Bayes-like classification method is to use the stable pattern, but instead of calculating a distance, we use the number of positive and negative samples, thus resulting in the specific stable pattern (compare figure 3.14).

Training is performed in two iterations: The first unsupervised iteration involves learning the weights of the HNN by only using the positive samples. In the second iteration, we execute the trained HNN for all positive and negative samples. For every sample, the HNN creates a stable state. We use this stable

state as a label and measure of how often a face or a non-face produces this state. Hence, we remember the number of occurrences of the specific class. After both iterations, the result is a vector for each stable state of the HaarNN that comprises probability values for every class as described in figure 3.14. Finally, we perform the classification as described above.

### Executing and Training at a Glance

The execution process of the HaarNN is, at its core, a result of the learned offset, the weights, and the threshold used. First, we shift the input vector by subtracting the offset which balances the input between the negative and positive values. Then, the input is repeatedly applied to the weights and the logistic activation function (see equation 3.8), using the learned threshold until it becomes a stable state.

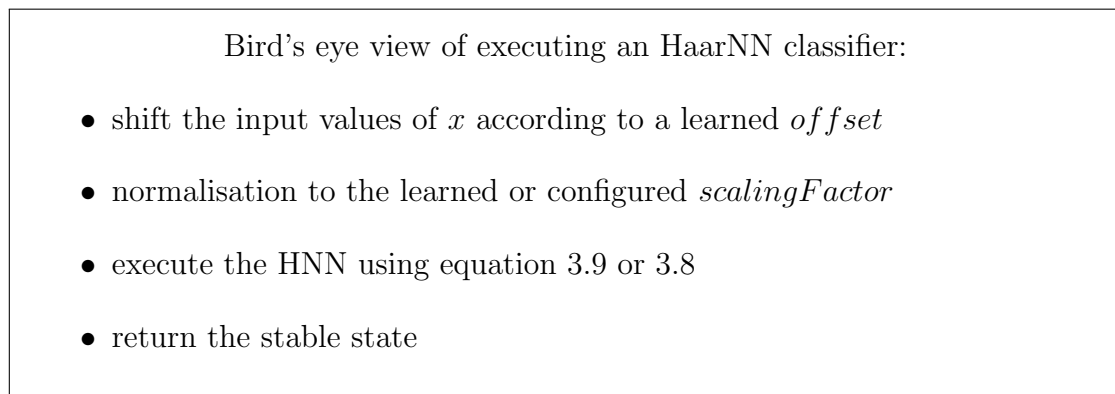


Figure 3.15: Execution overview of an HaarNN

In short, training the HaarNN is performed by adding the outcome of a Haar-Feature-like patch to the HNN weights for all positive training images. The average of this outcome, the later pattern for classification, and the average offset will be calculated during the same iteration.

## 3.2.4 Training the Hybrid Ensemble and Cascade Classifier

### Training the Ensemble

To learn the ensemble, we use AdaBoost as described in section 2.2.3. In our implementation, we can configure the feature sets and base classifiers we want to use. Instead of the original work, we use different sets of features and classifiers as one whole set.

Bird's eye view of training an HaarNN classifier:

```

i = 0

forEach(HFP, position)

    i ++

    forEachtrainingSample

        hfpValues = calculateHFPValues(sample, (HFP, position))
        offset =  $\frac{\max(x_i) + \min(x_i)}{2}$ 
        shiftedValues = forEach(x - > x - offset)
        normalizedValues = normalize(shiftedValues)
        w = updateWeights(normalizedValues)
        avePattern = calculateAverage(normalizedValues)
        haarNN = createModel((HFP, position), avePattern, w, offset)

```

Figure 3.16: Training overview of an HaarNN

Further, to reduce the training time of our classifiers, we use reduced subsets of all of the possible features. According to the bagging idea (see section 2.3.3), we create a new random subset for every boosting iteration. Thus, we use two methods to increase diversity, namely the re-weighting of AdaBoost and the (bagging-like) creation of different subsets.

### Training the Cascade

The training of the cascade classifier is done according to section 2.2.3. As a reminder, the aim of the cascade structure is to reject as much background as possible using fewer base classifiers by keeping most of the objects to find. For training purposes, we want to achieve a 100% detection rate, but only need a moderate false-positive rate for the first nodes. This false-positive rate can be configured and will be reduced during every iteration when training one ensemble classifier for the current cascade node.

For example, we start with a false-positive rate of 0.5 and reduce it for every next node classifier that was trained by AdaBoost. The later trained node ensembles will (mostly) contain more base learners and will therefore become in-

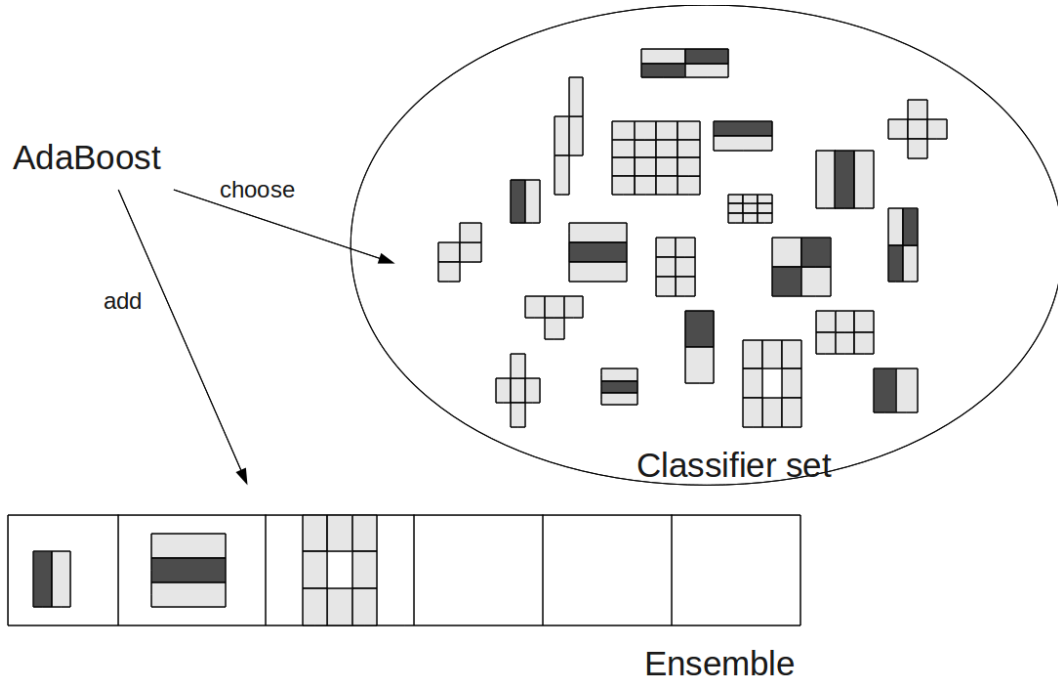


Figure 3.17: Hybrid Alternating Ensemble Architectures, combining threshold classifiers and HFP or HaarNN classifiers. Instead of choosing one classifier from a homogeneous set, here, AdaBoost uses a hybrid set of different classifier types.

creasingly accurate. As described in section 2.2.3, an ensemble classifier classifies an object as positive if more base classifiers than a given threshold classify the object as positive. Hence, we can decrease this threshold to achieve a 100% detection rate with the drawback of an increased false-positive rate. However, due to the cascade architecture, this is precisely what we want. We shift the threshold of the currently trained node ensemble until we reach the aspired high detection rate. Then, if the false-positive rate is less than the current aspired rate, training for the node classifier is completed. Otherwise, training will continue. Besides decreasing the expected false-positive rate, we use two additional mechanisms to increase the number of negative samples during training. One mechanism increases the number of negative samples, while the other replaces the already correct classified negative samples. The final mechanism does not increase the number of negative samples used to train one node classifier, but it does increase the overall number of negative samples used.

### 3.3 Comparing Hybrid Architecture

#### 3.3.1 Introduction to Experiments

With these experiments, we examine whether our new models and their hybrid architecture can improve other ensemble classifiers. As a second base classifier for the hybrid architecture, we use the threshold classifier from Viola and Jones [69, 71, 72]. We train single ensembles and hybrid ensembles using the same training parameters. Hence, the compared ensembles have the same training preconditions, and therefore only differ in terms of their classifiers and features.

First, we compare single and hybrid classifiers which use the same features and the same random sequence of features to analyze the different behavior of the classifier types. The same random sequence means that we use a so-called “seed”, i.e. a number which determines the sequence of random variables and therefore the sequence of features. Thus, using the same seed guarantees that the feature subsets for the training process are the same for all classifiers. We use the CBCL set to train our classifiers and test the resulting classifiers against all of the test sets (CMU A, Sung Poggio, and CMU C) described in section 2.7.

#### Detection Measurement

It is not very likely that a found detection will fit exactly within the labeled region; hence, we have to measure how similar this detection is to the expected region. Therefore, we measure a rate to compare the found region and the expected region. We arrive at this rate by calculating the intersection of the expected and found region to the union of both areas. If this rate is greater than 0.5, we consider the found region to be a hit (see equation 3.11 and the illustration of the equation provided in figure 3.18).

$$r = \frac{E \cap F}{E \cup F} \quad (3.11)$$

Here,  $E$  is the expected and  $F$  the found region and thus considered a hit if  $r > 0.5$  with  $r$  of equation 3.11.

The test sets are labeled with the correct regions marked as blue rectangles (see figure 3.19). The red rectangles are the misclassified detections, i.e. the false positives. The green rectangles indicate a correct detection. The sample figure 3.19 shows multiple nearby false detections (red) and correct detections (green). All of correct detections belonging to one expected area (blue) will be counted as one detection. However, the nearby false detections will be counted



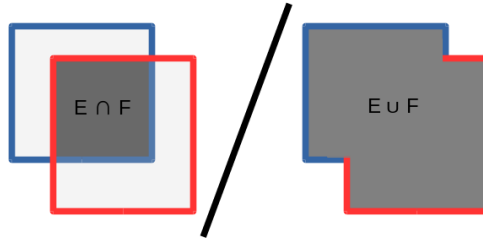


Figure 3.18: Determining whether a found region  $F$  is similar to the expected region  $E$  can be calculated by its overlapping  $\frac{E \cap F}{E \cup F}$

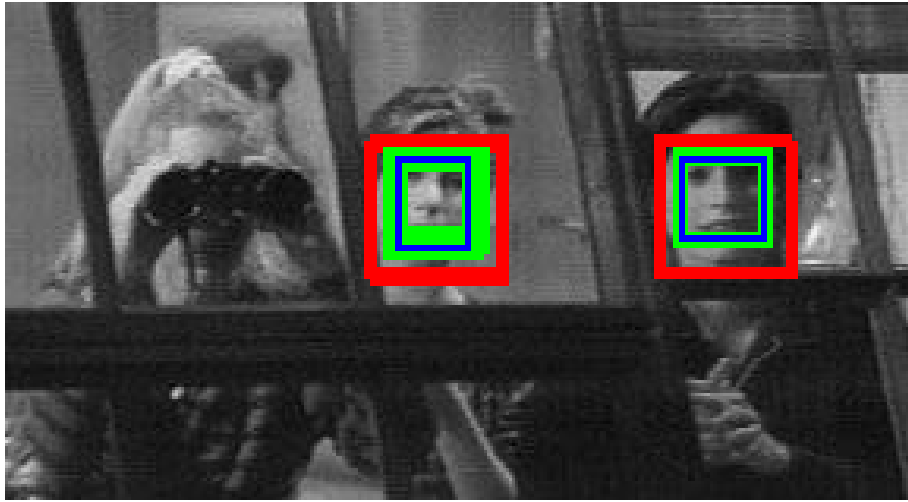


Figure 3.19: All of the red rectangles are false positives, while the green rectangles represent correct detections and the blue rectangles are the labeled faces. In general, no detection is likely to match the labeled region exactly. Hence, a detection is deemed a correct detection if the overlapping takes place within a proper range (see equation 3.11). However, there will mostly be more than one correct detection, all of which will be counted as one detection, whereas all of the nearby false detections are counted individually.

individually, including those that are very near to each other. Alternatively, we can merge the nearby regions into a single region which changes the detections and the number of false positives. In most of the experiments within the scope of this thesis, we do not merge because we do not want to have an influence owing to the merging mechanism. Further, when we compare our own trained classifiers, the measurement is the same for all, meaning that the comparison is fair. However, we will also compare the results with and without merging nearby regions to illustrate the difference.

### 3.3.2 Findings

These experiments aim to emphasize the influence of the hybrid architecture where both single and hybrid cascades were trained using the same parameters. In particular, we used the same HFP feature sets and the same fixed (random) feature sequence. Therefore, the only difference was the use of the hybrid architecture. The results of the hybrid and single cascade classifiers are shown in tables 3.1 and 3.2.

The first table 3.1 shows the result of the hybrid ensembles that are trained using the same training parameters and the same sequence of features. All of the cascade classifiers are trained using up to ten ensemble nodes. The single type classifiers are the homogenous cascade classifiers, which are trained using only one type of base classifier. The single cascades are trained with 200 (random) features in every iteration and the hybrid ensembles with 100 each; hence, the set of classifiers is also 200 for the hybrid ensembles. While we train the classifiers for the hybrid ensembles with 100 features each, the set to choose from is not the same as for the single type ensembles. However, the feature set used by the hybrid classifiers is a subset of the set used to train the single classifiers.

The meaning of the columns in tables 3.1 and 3.2 are as follows: The Type column describes whether it is a single or hybrid cascade classifier. Model and Model2 show which base classifiers are used for training the ensembles. These can be our HFP or HaarNN classifiers or the threshold classifier. The columns HFPSet and HFPSet2 denote the feature set used. The 'Sum-BC' row is the number of base classifiers within an entire cascade classifier. Every second line shows the first chosen features of the classifier above. The features that belong to a threshold classifier have light and dark gray rectangles. Features that only consist of light gray rectangles belong to HFP or HaarNN classifiers.

The first three rows of table 3.1 show the results of the single type cascade classifiers. First, we see the result of the cascade classifier which has simple threshold classifiers as base classifiers. The following two single type classifiers are trained using NCC-HFP and NCC-HaarNN as base classifiers. After the single classifiers, the next two rows show the performance and features of the hybrid classifiers. The final row shows a single cascade classifier that is trained using only 100 features. This is added for comparison because it is trained using exactly the same feature set used to train the hybrid classifiers.

We can see the different detection and false-positive rate of the classifiers. The single threshold cascade classifier has a lower false-positive rate, but the single NCC-HFP cascade has a higher detection rate. The hybrid cascade classifier of











































| Type   | Model   | HFPset  | Model2  | HFPset2   | Sum-BC  | DR   | FPR   |
|--------|---|---|---|---|---|--|---|
| Single | Thres   | Base  |   |   | 462.0   | 0.69   | 1.05E-4   |
|        |    |    |    |    |    |    |    |
| Single | NCC-HFP   | Base  |   |   | 610.0   | 0.75   | 2.03E-4   |
|        |    |    |    |    |    |    |    |
| Single | NCC-HaarNN  | Base  |   |   | 652.0   | 0.72   | 3.74E-4   |
|        |    |    |    |    |    |    |    |
| Hybrid | NCC-HFP   | Base  | Thres   | Base  | 365.0   | 0.75   | 1.09E-4   |
|        |    |    |    |    |    |    |    |
| Hybrid | NCC-HaarNN  | Base  | Thres   | Base  | 485.0   | 0.7  | 9.6E-5  |
|        |   |   |   |   |   |   |   |
| Single | Thres   | Base  |   |   | 525.0   | 0.68   | 6.74E-5   |
|        |  |  |  |  |  |  |  |

Table 3.1: Single and hybrid ensembles trained using the same training parameters and feature sets except for the classifier model. The ensemble is trained using Adaboost. The second row shows the first features of the first ensemble. The single type cascades used 200 features in each iteration. The hybrid cascades also used 200 features, but the set is split into 100 features for each classifier model. The test was performed done with all three of the described test sets.

the HFP and threshold base classifiers (fourth row) retains the higher detection rate of the HFP cascade and has a similar small false-positive rate to that of the threshold cascade classifier. The HFP cascade needs fewer base classifiers than both single cascades. The hybrid HaarNN cascade (fives row) achieves a lower false-positive rate and a higher detection rate than the single threshold cascade classifier. The last row shows a single threshold cascade that is trained using 100 features instead of 200. Therefore, the set of base classifiers only has half of the











































| Type   | Model   | HFPset  | Model2  | HFPset2   | Sum-BC  | DR   | FPR   |
|--------|---|---|---|---|---|--|---|
| Single | Thres   | Base  |   |   | 415.0   | 0.7  | 1.44E-4   |
|        |    |    |    |    |    |    |    |
| Single | NCC-HFP   | Base  |   |   | 546.0   | 0.78   | 4.08E-4   |
|        |    |    |    |    |    |    |    |
| Single | NCC-HaarNN  |   |   | Base  | 649.0   | 0.79   | 6.27E-4   |
|        |    |    |    |    |    |    |    |
| Hybrid | NCC-HFP   | Base  | Thres   | Base  | 373.0   | 0.79   | 2.28E-4   |
|        |    |    |    |    |    |    |    |
| Hybrid | NCC-HaarNN  | Base  | Thres   | Base  | 406.0   | 0.74   | 1.67E-4   |
|        |   |   |   |   |   |   |   |
| Single | Thres   | Base  |   |   | 479.0   | 0.71   | 1.14E-4   |
|        |  |  |  |  |  |  |  |

Table 3.2: Single and Hybrid ensembles trained using the same training parameters and feature sets except for the classifier model. The ensemble training is done using asym-Adaboost. The second row shows the first features of the first ensemble. The single type cascades used 200 features in each iteration. The hybrid cascades also used 200 features, but the set is parted in 100 features for each classifier model. The test is done with all three described test sets.

set used for the hybrid classifiers, but the set is exactly the same as that used to create the hybrid cascade because the hybrid cascade is trained using 100 threshold and 100 HaarNN or HFP base classifiers. This single threshold cascade classifier chooses the same first three features like the hybrid HaarNN classifier and achieves a lower false-positive rate, but also a lower detection rate.

Table 3.2 is much the same as table 3.1, but this time, the cascade classifiers are trained using asym-AdaBoost. First, we show the three single cascades, followed by

the hybrid cascades. The results are similar. The HFP/threshold hybrid cascade improves the detection rate for both single cascades and the false-positive rate for the single NCC-HFP cascade classifier. Again, this hybrid cascade needs fewer base classifiers. The same holds for the NCC-HaarNN/threshold hybrid cascade. However, this hybrid cascade increases the detection rate of the threshold cascade and the false-positive rate of the NCC-HaarNN cascade. The single classifier in the final row has seven out of the eight displayed features in common with its hybrid pendant.

| key           | Single Thres | Single HFP | Single HaarNN | Hybrid HFP | Hybrid HaarNN | Single Thres |
|---------------|--------------|------------|---------------|------------|---------------|--------------|
| Single Thres  | 1.0          | 0.532      | 0.543         | 0.601      | 0.619         | 0.636        |
| Single HFP    | 0.532        | 1.0        | 0.605         | 0.606      | 0.57          | 0.562        |
| Single HaarNN | 0.543        | 0.605      | 1.0           | 0.556      | 0.579         | 0.571        |
| Hybrid HFP    | 0.601        | 0.606      | 0.556         | 1.0        | 0.62          | 0.6          |
| Hybrid HaarNN | 0.619        | 0.57       | 0.579         | 0.62       | 1.0           | 0.654        |
| Single Thres  | 0.636        | 0.562      | 0.571         | 0.6        | 0.654         | 1.0          |

Table 3.3: This table shows the diversity measure of the trained classifiers, the result of which are provided in table 3.1.

Table 3.3 shows the diversity measure of the classifiers, the results of which are provided in table 3.1. The diversity measurement is performed using a subset of the CMU test set as positive samples and 9,000 negative samples. The table shows the diversity value of every classifier combination. A value of 1.0 means the two classifiers are equal, while 0.0 means they are independent. We use the paired correlation coefficient measure of equation 2.17 as described in section 2.4.

Regarding the first row of table 3.3, the highest diversity (lowest values) occurs between the single threshold classifier and the single HFP and HaarNN classifiers, followed by the hybrid classifiers and the other single threshold classifier. The single threshold classifier, which is trained using only 100 features (last row), shows a similar relation. The hybrid cascades are most diverse to the single HFP and

HaarNN classifiers which are not themselves part of the hybrid classifier. In principle, the same holds for the classifiers trained with asym-AdaBoost, although the single values tend to be higher.

| key           | Single Thres | Single HFP | Single HaarNN | Hybrid HFP | Hybrid HaarNN | Single Thres |
|---------------|--------------|------------|---------------|------------|---------------|--------------|
| Single Thres  | 1.0          | 0.54       | 0.56          | 0.635      | 0.609         | 0.662        |
| Single HFP    | 0.54         | 1.0        | 0.595         | 0.609      | 0.541         | 0.565        |
| Single HaarNN | 0.56         | 0.595      | 1.0           | 0.592      | 0.606         | 0.597        |
| Hybrid HFP    | 0.635        | 0.609      | 0.592         | 1.0        | 0.645         | 0.654        |
| Hybrid HaarNN | 0.609        | 0.541      | 0.606         | 0.645      | 1.0           | 0.622        |
| Single Thres  | 0.662        | 0.565      | 0.597         | 0.654      | 0.622         | 1.0          |

Table 3.4: This table shows the diversity measure of the trained classifiers, the results of which are presented in table 3.2.

### Comparing best cascade classifiers

In the previous section, we compared our classifier models within a hybrid architecture. There, we used the same features to train all of the classifiers so that they only differ in terms of their classifier models. However, although the random sequence of features was the same for all classifiers, this remains a random choice. Therefore, the current feature subset may be the only reason for the improvement. Further, we want to test whether our new models work better with the other HFP sets. Hence, we show results for both hybrid and single classifiers that use different sets.

We also use different random subsets, which causes a huge number of different parameters to train different classifiers. Therefore, we use a subset of classifiers to analyze and compare them. We choose this subset by grouping all of trained classifiers by detection rate and sort them with equal detection rate by their false-positive rates. Then, we take the first  $N$  classifiers with the lowest false-positive

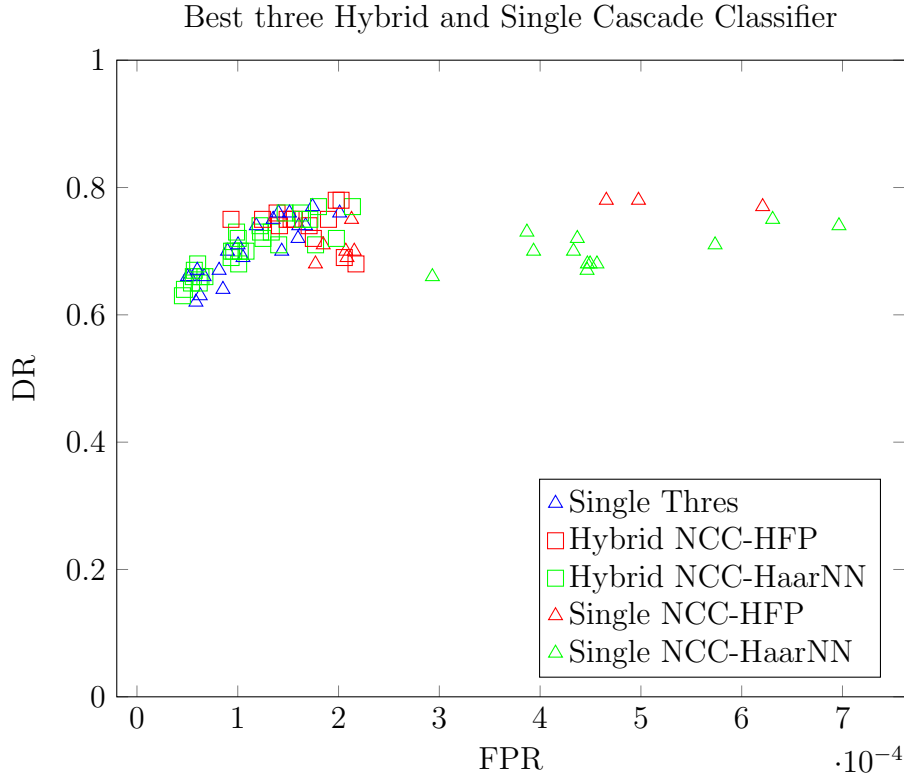


Figure 3.20: This distribution shows the detection and false-positive rate of cascade classifiers. All of the classifiers are trained using the same parameters except for the HFP sets and the classifier models. The single classifiers are trained using 200 features, i.e. 200 classifiers in each iteration. The hybrid classifiers are all trained by uniting the single classifiers. They are trained with 100 features each. Through uniting, they were also 200 classifiers for each iteration as for the single classifier training.

rate independent of the used HFP set and classifier model. If most of the best classifiers are single classifiers, we can question whether it is worth training the hybrid classifiers. The distribution of this process can be seen in figure 3.20.

While 3.20 provides us with an idea of the distribution of the different classifier types, tables 3.5 and 3.6 compare the best single and hybrid classifier for every detection rate. Again, we group all classifiers by detection rate, but then select one hybrid and one single classifier with the lowest false-positive rate in each case. We only take one classifier per type; thus, the second shown classifier is not necessarily the second best because there could be some more classifiers in the classifier type with a lower false-positive rate. To summarize, for every detection rate we take the hybrid winner classifier and the single winner classifier, where the winner has

| Type   | Model      | HFPset | Model2 | HFPset2 | Sum-BC | DR   | FPR     |
|--------|------------|--------|--------|---------|--------|------|---------|
| Hybrid | NCC-HaarNN | TLB69  | Thres  | Base    | 544.0  | 0.63 | 4.51E-5 |
| Single | Thres      | TLB69  |        |         | 561.0  | 0.63 | 6.28E-5 |
| Hybrid | NCC-HaarNN | TLB69  | Thres  | Base    | 519.0  | 0.64 | 4.71E-5 |
| Single | Thres      | TLB69  |        |         | 567.0  | 0.64 | 8.53E-5 |
| Single | Thres      | Base   |        |         | 472.0  | 0.66 | 5.01E-5 |
| Hybrid | NCC-HaarNN | TLB69  | Thres  | Base    | 508.0  | 0.66 | 5.59E-5 |
| Hybrid | NCC-HaarNN | BlockN | Thres  | Base    | 520.0  | 0.67 | 5.69E-5 |
| Single | Thres      | TLB69  |        |         | 552.0  | 0.67 | 5.98E-5 |
| Hybrid | NCC-HaarNN | TLB69  | Thres  | Base    | 496.0  | 0.68 | 6.02E-5 |
| Single | NCC-HFP    | BlockN |        |         | 891.0  | 0.68 | 1.77E-4 |
| Hybrid | NCC-HaarNN | Base   | Thres  | Base    | 436.0  | 0.69 | 9.34E-5 |
| Single | Thres      | Base   |        |         | 462.0  | 0.69 | 1.05E-4 |
| Single | Thres      | Base   |        |         | 447.0  | 0.7  | 8.98E-5 |
| Hybrid | NCC-HaarNN | Base   | Thres  | Base    | 452.0  | 0.7  | 9.39E-5 |
| Single | Thres      | Base   |        |         | 449.0  | 0.71 | 1.0E-4  |
| Hybrid | NCC-HaarNN | TLB69  | Thres  | Base    | 478.0  | 0.71 | 1.41E-4 |

Table 3.5: The table shows the two best classifiers for a certain detection rate. 5 times the hybrid classifier wins and 3 times the single classifier. The rows are grouped by detection rate and sorted by false-positive rate; thus, the lower false-positive rate comes first.

the lower false-positive rate. Considering both tables, 10 out of 15 of these winner classifiers are hybrid classifiers that consist of NCC-HFP and the threshold base classifiers or NCC-HaarNN and the threshold base classifiers.

### 3.3.3 Discussion and Conclusion

We use the detection rate and false-positive rate to evaluate the performance of our classifiers. A problem in using accuracy or error is that an image largely consists of a background. The number of negatives is much higher than the number of positives, meaning that the negatives would dominate the result of the accuracy



| Type   | Model      | HFPset | Model2 | HFPset2 | Sum-BC | DR   | FPR      |
|--------|------------|--------|--------|---------|--------|------|----------|
| Hybrid | NCC-HaarNN | TLB69  | Thres  | Base    | 499.0  | 0.72 | 1.01E-4  |
| Single | Thres      | TLB69  |        |         | 487.0  | 0.72 | 1.6E-4   |
| Hybrid | NCC-HaarNN | Base   | Thres  | Base    | 468.0  | 0.73 | 9.88E-5  |
| Single | NCC-HaarNN | Base   |        |         | 680.0  | 0.73 | 3.87E-4  |
| Single | Thres      | Base   |        |         | 447.0  | 0.74 | 1.18E-4  |
| Hybrid | NCC-HaarNN | TLB69  | Thres  | Base    | 458.0  | 0.74 | 1.22E-4  |
| Hybrid | NCC-HFP    | Base   | Thres  | Base    | 404.0  | 0.75 | 9.33E-5  |
| Single | Thres      | Base   |        |         | 423.0  | 0.75 | 1.351E-4 |
| Hybrid | NCC-HFP    | BlockN | Thres  | Base    | 358.0  | 0.76 | 1.39E-4  |
| Single | Thres      | Base   |        |         | 436.0  | 0.76 | 1.404E-4 |
| Single | Thres      | Base   |        |         | 420.0  | 0.77 | 1.74E-4  |
| Hybrid | NCC-HaarNN | Base   | Thres  | Base    | 445.0  | 0.77 | 1.8E-4   |
| Hybrid | NCC-HFP    | Base   | Thres  | Base    | 367.0  | 0.78 | 1.98E-4  |
| Single | NCC-HFP    | BlockN |        |         | 910.0  | 0.78 | 4.66E-4  |

Table 3.6: The table shows the two best classifiers for a certain detection rate. 5 times the hybrid classifier wins and 2 times the single classifier. The rows are grouped by detection rate and sorted by false-positive rate; thus, the lower false-positive rate comes first.

value (see equation 3.12). To overcome this problem, we can use a weighted accuracy  $acc(w)$  (equation 3.13), which takes the detection and false-positive rate into account in equal measure.

$$acc = \frac{tp + tn}{np + nn} \quad (3.12)$$

As a result,  $tp$  is the number of true positives,  $np$  the number of positive samples,  $tn$  the number of true negatives, and  $nn$  the number of negatives samples.

$$acc(w) = \frac{\frac{tp}{np} + \frac{tn}{nn}}{2} \quad (3.13)$$

The relation to the detection and false-positive rate is  $fpr = 1 - \frac{tn}{nn}$  and  $dr = \frac{tp}{np}$ .

There are also other accuracy measurements (f-score for example) that would also overcome the problem of the high number of negatives through their weighted



Figure 3.21: Sample images of two hybrid cascade classifiers. The first image is a detection result of a NCC-HaarNN/threshold hybrid cascade classifier using the TLB69 and base features (table 3.5). The second image is a detection result of an NCC-HFP/threshold hybrid cascade classifier using the base features (table 3.6).

variations. However, using accuracy or error will just shift the problem because these are also only combinations of detection rate and false-positive rate, thus failing to answer the question of whether a high detection rate is more important than a low false-positive rate. If one classifier has a higher detection rate and a lower false-positive rate than another classifier, then one is clearly better. If a classifier has a higher detection rate, but a higher false-positive rate, then the given application will determine what is more important and which is better. Therefore, we use detection and false-positive rate in our results discussion and group our classifiers by their detection rate. Then, we have the same detection rate, and only the false-positive rate determines which classifier is better.

### Same Feature Sets

Tables 3.1 and 3.2 in our findings section show single and hybrid cascade classifiers that are trained using the same parameters and, especially, with the same features and (random) feature sequences. The sets are not exactly the same because the single classifiers are trained with a set of 200 features and the hybrid classifiers with a set of 100 features each, but the features for training the hybrid classifier are a subset of the features used for training the single cascades. Therefore, the hybrid cascades have fewer different features. However, the last row shows a single threshold cascade trained with only 100 features, but those are the same features used to train both base classifiers of the hybrid classifiers.

The results of the tables emphasize the difference between the models. Every second row shows the first eight chosen features. While the first chosen features are similar for the hybrid and single cascades, they are subsequently more diverse. However, we expect the hybrid architecture to influence both classifier models during training. The HFP and HaarNN models within a hybrid architecture altered the features that are chosen by the threshold classifiers. As a result, the HFP base classifiers influence the hybrid cascade more than the HaarNN base classifiers. If we compare the features of the hybrid and the single threshold cascade that was trained using 100 features, they are equal for the hybrid HaarNN/threshold cascade and the single threshold cascade except for one feature (see table 3.1). Instead, the HFP base classifiers are used more often within the hybrid cascade classifier and also reduce the number of base classifiers to the greatest extent. Further, the diversity tables 3.3 and 3.4 also underline this observation. The single threshold cascade trained with 200 features are most similar (highest value) to the other single threshold cascade, which is trained with 100 features. To summarize, all of the cascade classifiers are different from one another, which illustrates the various chosen features they consist of (see tables 3.1 and 3.2) and their diversity values in tables 3.3 and 3.4.

However, what is more important is the comparison between the detection and false-positive rate of the cascade classifiers. If there is no benefit in using the hybrid architecture, we expect the accuracy of the single classifiers results to be average or worse. The HFP and HaarNN single cascade classifiers have a higher detection rate but also a higher false-positive rate than the threshold cascade classifier. However, compared to the single classifiers, the hybrid classifiers are better in at least one parameter (detection rate or false-positive rate) by keeping the other parameter similar. Thus, the hybrid classifiers improve the single classifiers, although we do not use the full opportunity for our new HFP and HaarNN models, which may use

more diverse feature sets.

Another value that benefits from the hybrid cascade classifiers is the number of base classifiers (Sum-BC). Both hybrid classifiers need fewer base classifiers than the single classifiers, except for the HaarNN/Threshold hybrid cascade shown in table 3.1. From a practical perspective, requiring fewer base classifiers needs less computational speed. From a training point of view, this means there are better base classifiers (regarding the training set). Hence, the configured expectation of detection and false-positive rate is reached earlier during the training process. Regarding the hybrid classifiers, these better base classifiers not only perform better within the training set, but also within the test set. However, we cannot generalize that fewer base classifiers lead to better accuracy, as can be seen if we compare both single threshold cascades. The single threshold cascades in the last row (in both tables) have more base classifiers, but a better detection and false-positive rate in one instance (see table 3.2) and a slightly similar detection and better false-positive rate in another situation (see table 3.1).

After comparing classifiers trained with the same features, we want to compare the hybrid architecture and our new models using different feature sets. To examine whether it is worth training hybrid cascades using the new classifier models and features, we trained several cascade classifiers for every HFP set by using different random subsets. Figure 3.20 shows a distribution of hybrid and single cascade classifiers. We create this distribution by taking these classifiers for every detection rate with the lowest false-positive rate. There, the hybrid classifiers mostly perform best. Tables 3.5 and 3.6 show two cascade classifiers for every detection rate. Here, we take the best (lowest false-positive rate) single and best hybrid cascade classifier for every detection rate. Overall, ten of the fifteen classifiers are hybrid classifiers. The HFP set TLB69 is also useful as a Haar-like feature for the threshold classifier, as can be seen in tables 3.5 and 3.6. There are several threshold classifiers within the list of the best classifiers which are trained using the set TLB69 and not only using the original set base. While all this does not prove that the hybrid architecture is better in every case, it does show that it is likely that the hybrid cascade classifier will improve the single cascade classifier, thus rendering it worthwhile to use the hybrid architecture and the HFP/HaarNN base classifiers and HFP features.

### 3.4 Overlap Detection Merge and Samples

For the results shown previously, we compared results that count every false detection. Mostly, correct detections have more than one nearby detection. Therefore, it is common to merge the nearby regions into one region in a first step. According to Viola and Jones [69], we can proceed with a second step to remove these merged regions consisting of just a few regions. Both steps reduce the false-positive rate, but also the detection rate. However, optimizing the merging method of multiple detections is beyond the scope of this thesis. Nevertheless, we want to show how the results of our classifiers would change their outcome.

Table 3.7 shows the results of several hybrid cascade classifiers without merging, with merging, and when merging and removing regions. The first row of one classifier shows the result without any merging as we used the results stated in the findings section above. The second row shows the result if we simply merge the nearby regions. There, we can reduce the false positives to about half, while the detection rate stays the same or is only slightly reduced. The third row shows the result if we first merge and then remove all these regions consisting of fewer than three regions. Again, we can reduce the false positives by about half compared to merging-only results, but the detection rate is also reduced by six to eleven percent. This observation is the same for single threshold cascade classifiers as shown in table 3.8.



Figure 3.22: The image on the left shows the result without any merging, i.e. just the detections as they are. The image in the middle shows the results after merging the nearby detections, while the image on the right shows the result after removing regions. This image sequence is a perfect example. First, there are some false detections; after merging this was reduced to only one, and in the final image there were no false detections at all.

The images in figure 3.22 are a perfect example. In the image on the left, the classifier has detected all of the faces without any merging. There, the person's face in the middle of the image is perfectly covered by one detection, but the region is too big to count it as a hit because the overlap for the expected region is too small.

| Model      | HFPset | Model2 | HFPset2 | DR   | FPR      |
|------------|--------|--------|---------|------|----------|
| NCC-HFP    | Base   | Thres  | Base    | 0.74 | 1.283E-4 |
| NCC-HFP    | Base   | Thres  | Base    | 0.74 | 6.26E-5  |
| NCC-HFP    | Base   | Thres  | Base    | 0.65 | 2.83E-5  |
| NCC-HaarNN | BlockN | Thres  | Base    | 0.75 | 1.642E-4 |
| NCC-HaarNN | BlockN | Thres  | Base    | 0.74 | 7.4E-5   |
| NCC-HaarNN | BlockN | Thres  | Base    | 0.66 | 3.44E-5  |
| NCC-HaarNN | TLB69  | Thres  | Base    | 0.75 | 1.715E-4 |
| NCC-HaarNN | TLB69  | Thres  | Base    | 0.75 | 8.19E-5  |
| NCC-HaarNN | TLB69  | Thres  | Base    | 0.69 | 3.67E-5  |
| NCC-HFP    | Base   | Thres  | Base    | 0.77 | 2.014E-4 |
| NCC-HFP    | Base   | Thres  | Base    | 0.77 | 9.71E-5  |
| NCC-HFP    | Base   | Thres  | Base    | 0.7  | 4.45E-5  |
| NCC-HaarNN | TLB69  | Thres  | Base    | 0.68 | 6.02E-5  |
| NCC-HaarNN | TLB69  | Thres  | Base    | 0.68 | 2.88E-5  |
| NCC-HaarNN | TLB69  | Thres  | Base    | 0.59 | 1.15E-5  |

Table 3.7: Results of hybrid cascade classifiers to show the merging effect. The first row shows the results if no merging is done, the second if nearby regions are merged, and the third if every region not consisting of at least two merged regions is removed. Merging nearby regions reduced the false-positives by about half, while maintaining a detection rate with a minor loss of one percent. Instead, removing regions reduces the detection rate from 6 to 11 percent and the false-positive rate, again, to about half.

Manually, we would probably count it as a hit. The image in the middle shows the detections after merging the nearby regions into one region. Merging changes the detection of the person in the middle. After merging, the one false detection and the other nearby detections create an average region that meets the overlap criteria 3.11 and, hence, is counted as a correct detection. Lastly, we remove the detections consisting of fewer than two merged regions and have no false detection left, as can be seen in the right-hand image of figure 3.22.

The next image group in figure 3.23 is an example of losing a correct detection through merging. Before merging, we detected all of the faces. In the next image,

| Model | HFPset | DR   | FPR      |
|-------|--------|------|----------|
| Thres | Base   | 0.78 | 1.641E-4 |
| Thres | Base   | 0.77 | 8.22E-5  |
| Thres | Base   | 0.7  | 3.78E-5  |
| Thres | TLB69  | 0.77 | 3.223E-4 |
| Thres | TLB69  | 0.76 | 1.544E-4 |
| Thres | TLB69  | 0.69 | 7.54E-5  |

Table 3.8: Results of cascade classifiers to show the merging effect. The first row shows the results if no merging is done, the second if nearby regions are merged, and the third if every region not consisting of at least two merged regions is removed. The effect is the same as that of the hybrid classifiers in table 3.7.

among others, the top-right regions are merged into one region. While one of these regions is within the correct area before merging, the average of the nearby regions falls outside this area, meaning that we lose one hit. However, the final image shows that we can reduce many of the false detections. With these samples, we want to emphasize that at least via the merging process, we can reduce the false-positive rate by about half while mostly maintaining the detection rate.

## 3.5 Summary

In this chapter, we presented the benefit of our new models within a hybrid architecture. Although there is no guarantee of creating better hybrid classifiers using a small random feature set, there is an increased opportunity for improvement. If we consider the first best cascade classifiers, the majority are hybrid cascade classifiers.

In the overlap and samples section 3.4, we showed that by merging the nearby regions, we decrease the false-positive rate by about half. The drawback here is that for some classifiers, the detection rate is also decreased but mostly by less than one percent. However, in the following sections we will compare the classifiers without any merging and removing so as to compare them without any influence from other methods.



Figure 3.23: The above-left image is without merging, above-right is with merging, and the lower image with removing of regions. There, we see an example of losing one correct detection due to merging. While in the first image all of the faces are found, the second image lost one correct detection. Finally, we can reduce a lot of the false detections.



# Chapter 4

## Diversity and Common Characteristics of our Classifier Models

### 4.1 Introduction

In the previous chapter, we presented our models used as part of a hybrid architecture. In this chapter, we show and analyze the essential characteristics and differences of our HFP and HaarNN classifier models used in homogeneous ensemble and cascade classifiers. We start by examining the diversity and differences of both models in section 4.2. In section 4.3, we analyze the extent to which the HaarNN model is beneficial and provides higher diversity through the included Hopfield Neural Network. Finally, we show that it is justifiable to use a small random subset to train our classifiers in section 4.4.

### 4.2 Diversity and Difference of our Classifier Models

#### 4.2.1 Introduction to Experiments

The Hopfield Neural Network as part of the HaarNN classifier learns the pattern of a Haar-Feature-like patch. The classification of the HaarNN model is done after the HNN is executed. The output of the HNN is the input for the classification method which is the only difference to the HFP classifier. The HFP classifier uses the Haar-Feature-like patch directly as an input for classification (see figure 4.1).

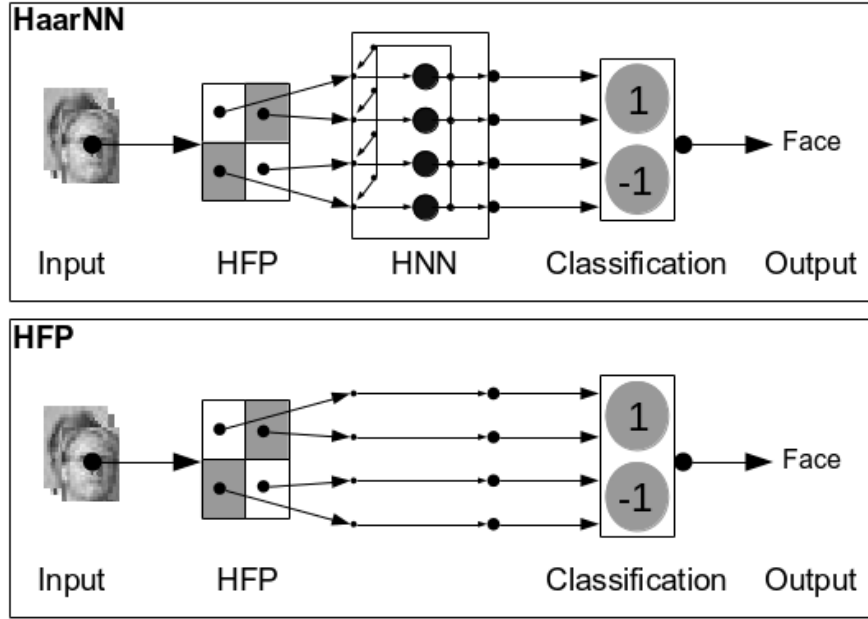


Figure 4.1: Comparing HaarNN and HFP classifier models. The images describe the classification process. It starts with the input, followed by applying the Haar-Feature-like patch (HFP). The HaarNN classifier (top) uses a Hopfield Neural Network (HNN) to filter the Haar-Feature-like patch values. In contrast, the HFP classifier (bottom) uses these values directly. Finally, both classifiers compare the result with a learned patch to decide whether the input is a face or not.

However, if the features, the pattern, and the classification metric are the same for both models, we have to question whether they create significantly different results. If the HNN does not matter, we would expect the same or at least quite similar classification results. To compare both models, we use the same training parameters. Two classifiers can consist of different features, but do not have to be different in their classification ability. If we have two classifiers that differ in terms of the features used, but react to every input with the same result, we would not see any relevant difference in their classification ability. In contrast, having two classifiers showing the same detection and false-positive rates does not necessarily mean they are equal either. We can analyze the single results of dedicated image frames of two classifiers with the same performance to examine whether they act differently for the single input and, therefore, still exhibit diversity and create a benefit as a combined solution.

We have used the CBCL image set for training purposes (see section 2.7). Testing all possible parameter variations would cost so much training time that it would be hard to explore. Therefore, we focus on the following parameters that

determine the classifier training:

- Asym-AdaBoost [true, false]
- The maximum number of cascade nodes; typically five or ten
- The HFP set used
- The number of HFP features used in one iteration

### 4.2.2 Results comparing HFP and HaarNN classifiers

In this section, we compare the HaarNN and the HFP classifier models and analyze their differences. First, we examine their differences in terms of creating ensembles and when it comes to creating cascade classifiers.

#### Comparing HFP and HaarNN ensembles



| Model  | HFPset | DR      | FPR    |
|--|--------|---------|--------|
| NCC-HaarNN   | BlockN | 75.2119 | 0.1959 |
|  |        |         |        |
| NCC-HFP  | BlockN | 55.2966 | 0.0819 |
|  |        |         |        |

Table 4.1: Trained with Asym-AdaBoost, 500 random features per iteration, to a size of 50 ensemble members using Normalized Cross-Correlation. Thereby, DR is the detection rate, and FPR is the false-positive rate. Every second row shows the first Haar-Feature-like patches that are chosen through the training algorithms. The last picture in each second row shows the Summed Picture (SP). It is developed by putting the learned patterns on top of each other and normalizing the result to create an image. Although they are equally trained, they differ in their chosen features and performance.

To compare the HFP and the HaarNN ensemble classifier, we use the same sequence of random features to train them. Therefore, the sets from which the Haar-Feature-like patches are chosen, are equal for both models. Hence, we can analyze how both classifier models differ through the HNN within the HaarNN

classifier. If the classifier models are similar, the chosen features should also be very similar.

The table 4.1 shows the performance of several ensemble classifiers. Each group of two classifiers consists of one HFP and one HaarNN classifier which is trained using the same context; hence, all influencing training parameters are the same. Every second row shows the first ten Haar-Feature-like patches chosen during training. The table shows that the performance and the chosen features are different.

| Feature | Models               | Correlation value |
|---------|----------------------|-------------------|
| BlockN  | NCC-HaarNN / NCC-HFP | 0.441             |

Table 4.2: Correlation diversity of the above classifiers (see table 4.1). Fewer values mean greater independence and, therefore, higher diversity. Both classifier models are trained using the same parameters.

The table 4.2 shows the diversity of both classifier models whose performance and features are depicted in table 4.1. All of the other training parameters, including the feature sets used to build the classifiers, are the same throughout the fixed random sequence. The fixed random sequence guarantees that a chosen set of features consists of the same features. Therefore, the diversity between both classifiers in table 4.2 is a result of the different behavior of the classifier models whose only difference is the HNN.

### Comparing HFP and HaarNN cascades

Cascade classifiers are not diverse simply because their ensembles are. While the cascade classifiers consist of several ensembles, it is possible for all of the single ensembles to be different, while they are equal in their aggregation. Therefore, we analyze whether two cascade classifiers trained using the same sets of Haar-Feature-like patches as above, also show diversity.

The classifiers, whose results are shown in table 4.3, are cascade classifiers containing 10 nodes of ensemble classifiers. The ensembles are trained using asymmetric AdaBoost, with asym-factor  $k = 1.4$ , HFP set BlockN (see section 3.2.1) and distance metrics Normalized Cross-Correlation (NCC). A cascade classifier consists of hundreds of features which makes it difficult to compare them as a whole. To get an idea of the differences of the entire learned set, we use the summed picture which we create by painting all feature patterns to one image. The summed pictures can be seen as a compact representation of all chosen features and learned patterns. We present the summed pictures in table 4.4 to show that every ensemble

| Model      | HFPset | DR  | FPR    |
|------------|--------|-----|--------|
| NCC-HaarNN | BlockN | 0.9 | 0.0052 |
| NCC-HFP    | BlockN | 0.9 | 0.0034 |

Table 4.3: Classification results of the trained cascades. The HFP and HaarNN classifiers are trained using the same configurations.





















| Model      | SP Cascade  |   |   |   |   |   |  |   |   |   |
|------------|---|---|---|---|---|---|--|---|---|---|
| NCC-HaarNN |  |  |  |  |  |  |  |  |  |  |
| NCC-HFP    |  |  |  |  |  |  |  |  |  |  |

Table 4.4: We can paint the learned pattern as a summed image which is an aggregation of all chosen features. We show ten pictures for the ten ensembles that make up the cascade classifiers. All of the summed pictures look different, meaning that no ensemble consists of the same set of features.

within the cascade classifier learned different features. Thereby, each image shows the summed picture of one ensemble where the images are in sequence as they are within the cascade classifiers. Hence, the first image belongs to the ensemble of the first cascade node.

The table 4.5 shows the features of the NCC-HFP and NCC-HaarNN cascade classifiers. We show the features of the first up to ten base classifiers within the first three ensemble nodes of the cascade classifiers. The number of base classifiers chosen during training is determined by the achieved detection and false-positive rate. If the ensemble reaches the demanded detection and false-positive rate, training for that particular ensemble node stops and training of the next ensemble node starts (see section 2.2.3). Therefore, the first nodes in particular may have fewer base classifiers which can be seen in table 4.5. The column *Node* describes the node of the cascade classifier, whereas the sequence of the pictures represents the feature sequence of ensemble members.

Another model used to analyze the diversity of several classifiers involves merging them. Therefore, we create a classifier called “merge cascade”. A cascade classifier consists of nodes of ensemble classifiers. We create the merge cascade by

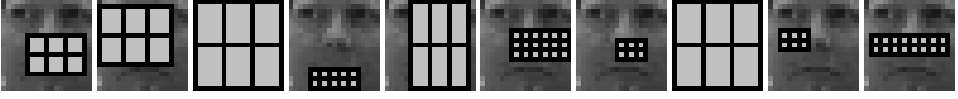

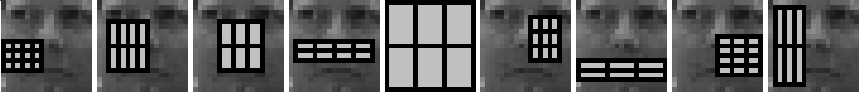
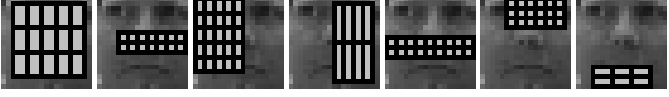
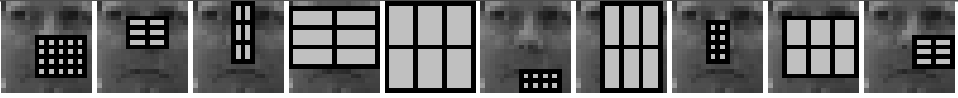
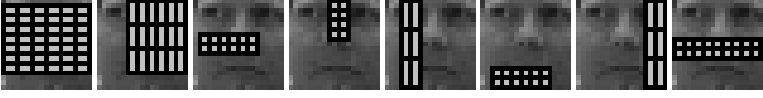
| Model Node    | First HFPs   |
|---------------|--|
| HaarNN Node 1 |    |
| HFP Node 1    |     |
| HaarNN Node 2 |    |
| HFP Node 2    |    |
| HaarNN Node 3 |   |
| HFP Node 3    |  |

Table 4.5: Here, we show the difference of the chosen features for both classifier models. The only difference when training the NCC-HaarNN and NCC-HFP models are the classification method used. The column Node depicts the ensemble nodes of the cascade classifier in their sequential order. The painted features consist of the first ten features chosen during training.

| Models               | HFPset | Correlation value |
|----------------------|--------|-------------------|
| NCC-HaarNN / NCC-HFP | BlockN | 0.61              |

Table 4.6: Diversity correlation measure of the classifiers shown in table 4.5.

adding these nodes one by one to a new cascade classifier. If we have two cascade classifiers  $c^1, c^2$  with their nodes  $c^1_{n_1}, \dots, c^1_{n_5}$  and  $c^2_{n_1}, \dots, c^2_{n_5}$  we arrive at the merge cascade  $mc$  with nodes  $c^1_{n_1}, c^2_{n_1}, \dots, c^1_{n_5}, c^2_{n_5}$ . The effect of using this merged cascade is that we have a logical conjunction of the classification results. The cascade

classifier returns a positive result if every ensemble node does the same. Hence, for the merged cascade classifier, every node classifier of the joined cascade classifiers has to agree in order to achieve a positive result. If we execute the cascade classifiers sequentially instead of merging the nodes to a new classifier, the result is the same, but by merging them, we retain the cascade structure starting with the simpler and moving through to the complex ensembles and, in turn, the performance issue. Recapitulating the equations 4.1 and 4.2 (compare section 2.2.3) we get a second measure of their diversity. If the classifiers are completely independent, we can calculate the final detection and false-positive rate using equations 4.1 and 4.2.

$$F = \prod_{i=1}^K f_i \quad (4.1)$$

$$D = \prod_{i=1}^K d_i \quad (4.2)$$

This conjunction of classifiers confirms and visualizes the diversity because we can easily apply the merge cascade to every test set. While table 4.6 shows the diversity of both classifiers measured using a cropped subset, in table 4.7 we show the results for the merging cascade of both classifiers.

| Model                   | HFPset | DR   | FPR      |
|-------------------------|--------|------|----------|
| Merge of HaarNN + HFP   | BlockN | 0.83 | 0.0015   |
| Theoretical Independent |        | 0.81 | 0.000018 |
| Theoretical Dependent   |        | 0.9  | 0.0034   |

Table 4.7: The second method to show diversity involves merging classifiers. The merge cascade classifier is the union of both of the above classifiers. It classifies an input as a face if both classifiers classify this input as a face. If both classifiers are independent, the resulting detection rate is  $0.9 \times 0.9 = 0.81$ .

The images in figure 4.2 are the results of the NCC-HaarNN, the NCC-HFP classifier, and their merge cascade classifier. To arrive at the results seen in the merge image, both classifiers have to commit. This requires a reduction in the number of false positives.

#### All distance measures work, except ED-HFP

Here, we show examples that emphasize the difference between both models due to the distance measure used. Therefore, we consider the cascade results in tables

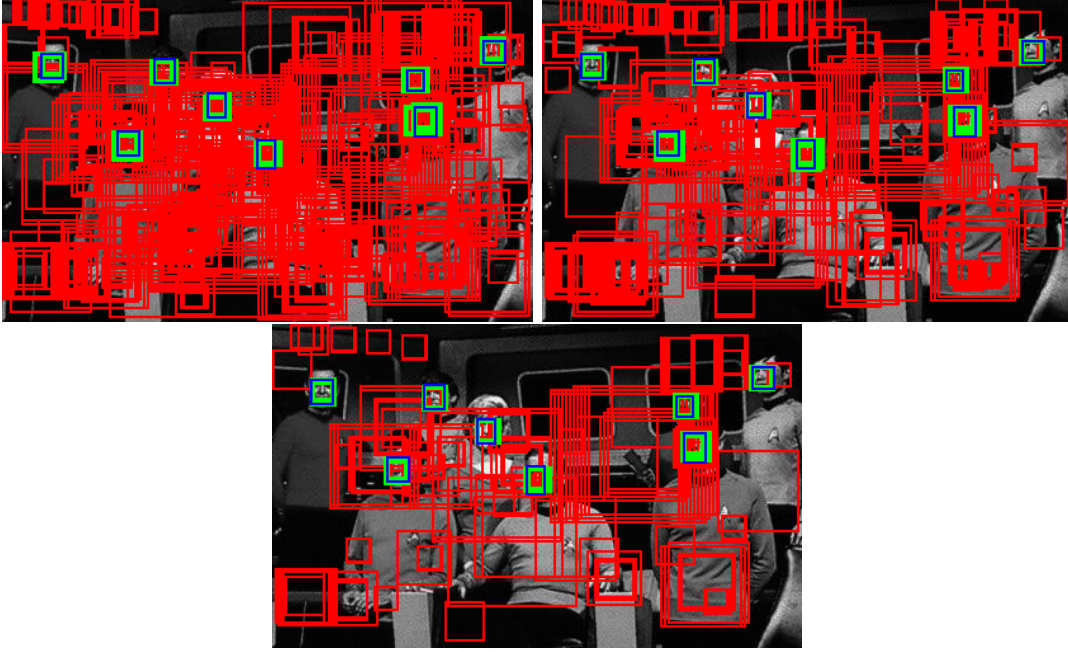


Figure 4.2: These sample images visualize the effect of the merge cascade. The top left image belongs to the NCC-HaarNN classifier, while the top right image belongs to the NCC-HFP classifier. The bottom image is the result of the merge cascade classifier consisting of the above HaarNN and HFP classifiers. We can see a reduction in false positives (red rectangles) because both classifiers react differently to most inputs; hence, they are diverse.

4.8 and 4.9. The cascade classifiers are trained with up to five ensemble nodes and using the same training configurations.

| Model      | HFPset | DR      | FPR    |
|------------|--------|---------|--------|
| ED-HaarNN  | BlockN | 87.037  | 0.0041 |
| NCC-HaarNN | BlockN | 87.037  | 0.0025 |
| ED-HFP     | BlockN | 96.2963 | 0.779  |
| NCC-HFP    | BlockN | 85.1852 | 0.0019 |

Table 4.8: Cascades trained with up to five ensemble nodes using asym-AdaBoost and BlockN as the HFP set. While all cascade classifiers have comparable results, the ED-HFP classifier has a very high false-positive rate.

Table 4.8 compares HFP and HaarNN cascade classifiers trained with asym-AdaBoost and BlockN as the HFP set. All results are feasible, except the results for the ED-HFP classifier.



| Model      | HFPset | DR      | FPR    |
|------------|--------|---------|--------|
| ED-HaarNN  | BlockN | 72.2222 | 0.0017 |
| NCC-HaarNN | BlockN | 75.9259 | 0.002  |
| ED-HFP     | BlockN | 94.4444 | 0.4822 |
| NCC-HFP    | BlockN | 81.4815 | 7.0E-4 |

Table 4.9: Cascades trained with up to five ensemble nodes using AdaBoost and BlockN as the HFP set. The ED-HFP cascade classifier has a very high false-positive rate, while all other cascades are feasible.

While asym-AdaBoost focuses on base classifiers with a higher detection-rate, the false-positive rate is also increased. However, it is not only because of asym-AdaBoost. The cascade classifiers in table 4.9 are trained with AdaBoost, and the ED-HFP classifier also has an unsatisfying false-positive rate.

### 4.2.3 Discussion and Conclusion

#### Comparing HFP and HaarNN ensembles

Table 4.1 shows the results and the features used for several ensembles. The accuracy is different, as are the chosen HFPs and the sum pictures, which are the aggregation of all features and therefore an indication for all of the chosen HFP set. However, a difference in detection and false-positive rates does not necessarily imply a high diversity. The dedicated results for a given input frame could be quite similar if all detections are made for the same objects except for a few. If both classifiers mostly make the same mistakes for the single test frames, it causes a low diversity. Hence, the classifiers could still be very similar in terms of their diversity. Therefore, we consider table 4.2 which shows the correlation of the classifier results. If two classifiers have the same classification answers for every dedicated input frame, the correlation is 1. However, all of the correlation values for two different classifiers in table 4.2 are less than 1.

#### Comparing HFP and HaarNN cascades

If the ensembles are not diverse, the cascades consisting of ensembles should therefore also not be diverse. However, cascade classifiers consist of several ensembles and, therefore, of more base classifiers that could grow to more equality. Two perfect classifiers with no errors would not differ in their results, i.e. there would

be no diversity. However, imperfect classifiers can also lead to a low diversity if they classify most of the regions' equality. Therefore, we consider the cascades in table 4.3 in more detail.

Table 4.3 shows a different performance for the cascade classifiers. While both were trained using the same context with the only difference being the use of the HNN, they achieve different results for the detection and false-positive rate. Therefore, the HFP classifier and HaarNN classifier cannot be exactly the same. However, the difference in accuracy is not as vast. Thus, the classifiers may use mostly the same features and just start to differ towards the end of the training process.

Where the HFP and HaarNN classifiers differ most considerably is in the different features they choose during training, as shown in table 4.5. There, we show the first ten chosen features of the first three cascade nodes of both classifiers. There, none of the feature sequences for the ensemble nodes are equal, despite using the same set of Haar-Feature-like patches for training, which shows more clearly how different they are in their feature-selection behavior. However, this table only depicts the first features of the first ensemble nodes and we want to compare all features to be sure that whole sets do not become equal if we take all features into account. Therefore, we paint the whole chosen set of features as a sum picture as presented in table 4.4. There, all sum pictures are different.

Finally, table 4.6 shows that the HFP model and the HaarNN model are not equal or similar to any large extent. Hence, their correlation value is roughly in the middle of the value range. As double proof, we can consider the merge-models result in table 4.3 as already mentioned, and the figure 4.2 that paints the differences of both classifiers and their merge cascade. If there were no diversity, one classifier would be only a subset of the other classifiers. Given their results and the results of the merge model, this would lead to the minimum HFP and HaarNN classifiers result. This is not the case. The merge classifier reduces the false-positive and detection rate, which is the expected result given equations 4.2 and 4.1. The performance of the merge cascades, which reduces the detection and false-positive rate, verifies the conclusion that the HFP and HaarNN classifiers are in fact different.

### **How the HFP and HaarNN classifiers differ**

In this final passage, we want to analyze how both models differ. One observation is that the HFP models often tend to produce a lower false-positive rate, but also a lower detection rate. This tendency lines up with various different training

parameters, as can be seen, for example, in 4.1.

The more interesting observation is that using the Haar-Feature-like patches and the Euclidean distance for classification, i.e. the ED-HFP model, does not work. This problem arises when using asym-AdaBoost and AdaBoost. While asym-AdaBoost shifted the selection process towards classifiers that favor a good detection rate over a small false-positive rate, we also depicted the results using AdaBoost. Using AdaBoost or the asym variation does not change this observation. We show both results to zoom in on the behavior of the HFP model and to emphasize that it is an inherent problem of the ED-HFP model. Moreover, this is specific to the HFP model. Hence, the ED-HaarNN model has no outlier like this.

We interpret this through the geometry of the different methods. Points that are similar to each other in terms of cross-correlation can be far away when it comes to Euclidian distance. Lighting changes, for example, could lead to similar correlation values, but the Euclidean distance between the new and the changed point could be huge. Therefore, it seems there are many points that belong to the negative samples in the Euclidean range of the positive samples. In contrast, the HaarNN model did not use this value directly, but shifted it to a learned fix point, with the negative points far enough away for the Euclidean distance to work. While we have not currently clarified the research question of whether the HNN is a benefit, we can clarify that it is different using the HNN within the model.

## 4.3 Increased Diversity by the Hopfield Neural Network

In this section, we discover the diversity of the HaarNN model.

### 4.3.1 Introduction to Experiments

Here, we examine the diversity that comes through the Hopfield Neural Network within the HaarNN classifier. In the previous section, we saw that the HaarNN classifier is different from the HFP classifier and, therefore, already increases diversity. However, the Hopfield Neural Network is a dynamic model which changes its behavior along with changing training parameters, and we want to use this dynamic to increase diversity. Since these parameters only affect the HaarNN, we call them “inner” HaarNN parameters. We explore which parameter creates different classifiers and increases diversity, and whether we can create a benefit out of this. Therefore, we train our HaarNN model by fixing all parameters except for the one

we want to analyze. Then, we train several HaarNN cascade classifiers by systematically varying the observed parameters. Summarizing, we question which inner HaarNN parameters can increase diversity and achieve a benefit for the ensembles classification ability.

### 4.3.2 Methods - Parameters for Increasing Diversity

To carve out the parameters that influence the behavior of the HaarNN classifiers and to examine their role in creating diversity, we look deeper at the execution and training process. We explain the parameters in the sequence they are used during execution (see figure 3.15) and the learning process (see 3.16).

As a brief overview, the execution process of the HNN is influenced by the parameters *offset*, *scalingFactor*, *useJustThres*, *threshold*, *stretchFactor* and *maxCount*. During execution, we first subtract the *offset* from the input pattern. Afterwards, the *scalingFactor* is applied, which drags or compresses the pattern. Thereafter, the HNN is executed until it becomes a stable state or until a configured maximum of iterations (parameter *maxCount*) is reached. When executing the HNN, the activation function is applied in every iteration. Thereby, the logistic activation function (equation 3.8) uses the *threshold* and the *stretchFactor*. The binary activation function (equation 3.9) uses the *threshold* and is activated if *useJustThres* is true.

#### Offset

After the input is generated applying an HFP (see section 3.2.3), the first step involves subtracting the offset. We calculate the offset using equation 3.4 for every training sample, and we use the average of all these offsets (equation 3.6) as the threshold for the HaarNN classifier. However, to increase diversity, we can use the offset differently:

- Learned offset  
As described in equation 3.6, we use the average offset.
- None offset  
While the weights are calculated using an offset, we can leave the offset for classification. Then, the input “starting point” is different but also converges to a stable state.
- Final pattern offset  
We can use the final pattern to calculate the offset by applying equation 3.4.

## Scaling Factor

After applying the *offset*, all input values will be normalized to a given range. The value that performs this normalization is *scalingFactor*. The *scalingFactor* is a result of normalizing (*normalizeTo*) of the input after subtracting the *offset*. Therefore the *normalizeTo* parameter determines the *scalingFactor*. Before scaling is performed, the max range of the input is between  $-127$  and  $128$ . For example, normalizing the input to a range of  $[-1, 1]$  will lead to a *scalingFactor*  $s = \frac{1}{128}$ .

## Threshold, Stretch Factor and Activation Function

The *threshold* for the activation function is the max of the final learned pattern as described in section 3.2.3. If the parameter *fixThres* has a value, this is used as the *threshold*. The parameter *stretchFactor* ( $\beta$ ) is part of the logistic activation function that scales the function in the x-direction; thus, the gradient will be smaller.

$$o_j = \frac{2\theta}{1 + e^{-s_j\beta}} - \theta, \text{ (equation 3.8)}$$

*stretchFactor* has a relation to the *threshold*  $\theta$ . With *stretchFactor* 1, the gradient of the function will be higher with a higher  $\theta$ . To repeat,  $s_j$  is the product of the weights and the input as described in equation 2.19.

$$o_j = \begin{cases} \theta & \text{if } s_j > \theta \\ -\theta & \text{if } s_j < -\theta \\ s_j & \text{otherwise} \end{cases}, \text{ (equation 3.9)}$$

Two more parameters that affect the activation function in a broader sense are *useJustThres* and *usePatternThres*, which are both Boolean values. If parameter *useJustThres* is set to true, the binary activation function 3.9 will be used, otherwise the logistic activation function 3.8. The parameter *usePatternThres* changes the usage of the *threshold* for the activation function. While the *threshold* as described above and in section 3.2.3 is one *threshold* for the whole net, a positive *usePatternThres* changes such that every single neuron has its individual *threshold* corresponding to the learned pattern  $p$ . Hence, if we have a pattern of  $p = (-5, 3, -1, 5)$ , we have an HaarNN with four neurons with individual thresholds of  $(-5, 3, -1, 5)$ . Instead, if *usePatternThres* is false, the max of  $p$  is 5, and therefore the threshold is 5 for every neuron.

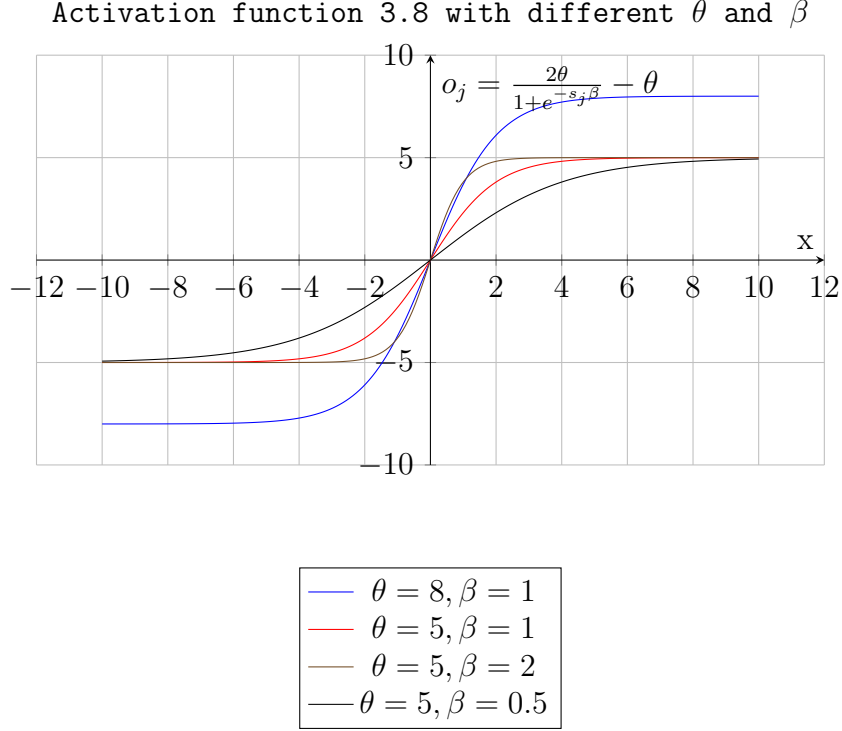


Figure 4.3: The logistic activation function with different values for the parameters *stretchFactor* ( $\beta$ ) and *threshold* ( $\theta$ )

### 4.3.3 Results

First, we compare ensembles and their parameters, which we call “inner” parameters because they influence only how the HNN within the HaarNN works. Hence, the differences in chosen features and diversity only arise from parameters which have no counterpart for the HFP classifier. In this section, we show results of ensembles that are trained varying several parameters as described in sections 3.2.3 and 4.3.2.

Table 4.10 shows the results of ensembles whose training only differs in two parameters. This is the *fixThres* and the *normalizeTo* (short *normTo*) as described in section 4.3.2. As a reminder, the *normTo* is the range the values will be scaled to  $[-normTo, normTo]$ . If *normTo* is marked with  $-1$ , then no normalization is done and we have values in the range of pixels  $[0.255]$ . If fixed threshold (*fixThr*) has a value, the threshold of the activation function 3.9 or 3.8 becomes the value designated in *fixThr*; otherwise, the value will be calculated as described in section 3.2.3.

Table 4.11 shows the diversity between all classifiers, the results of which are shown in table 4.10. The diversity value describes the similarity between two

| Model      | HFPset | DR      | FPR    | fixThr | normTo |
|------------|--------|---------|--------|--------|--------|
| NCC-HaarNN | BlockN | 58.0508 | 0.1077 | 1      | 1      |
| NCC-HaarNN | BlockN | 61.6525 | 0.1082 | 1      | 2      |
| NCC-HaarNN | BlockN | 59.9576 | 0.1124 | 2      | 1      |
| NCC-HaarNN | BlockN | 62.2881 | 0.1124 | 2      | 2      |
| NCC-HaarNN | BlockN | 65.2542 | 0.1611 | 2      | 5      |
| NCC-HaarNN | BlockN | 60.8051 | 0.1198 | 5      | 1      |
| NCC-HaarNN | BlockN | 61.8644 | 0.101  | 5      | 2      |
| NCC-HaarNN | BlockN | 59.9576 | 0.108  | 10     | 1      |
| NCC-HaarNN | BlockN | 61.0169 | 0.1148 | 10     | 2      |

Table 4.10: NCC-HaarNN ensembles using BlockN features trained to 80 base classifiers only differing in the scaling factor (*normTo*) and the threshold (*fixThr*) of the activation function. We trained all classifiers using the binary activation function 3.9 (*fixThr* = *true*) and one threshold for all neurons (*usePT* = *false*) as described in the method section 4.3.2.

classifiers. Despite the fact that only two parameters (*fixThr*, *normTo*) are varied when creating these ensembles, no two ensembles are the same. The minimal diversity value is 0.55 and the maximum is 0.849.

The ensembles in table 4.12 use the BlockN set and are trained with up to 50 members. The other training configuration is the same as for the classifiers above, except for the *fixThr* parameter which is  $-1$ , meaning that the threshold will be learned. This configuration also creates diversity.

The average diversity is 0.505 and diversity values are between 0.475 and 0.561 as depicted in table 4.13. Worthy of note are the chosen features shown in every second row in table 4.12. They are all different, and the aggregated pictures of the features are also different, despite having used a fixed (random) sequence of features, i.e. the same feature sets, for training.

### Several Cascade Classifiers trained with Different Parameters

In this section, we present the results of six HaarNN cascade classifiers, for which we permute several parameters. According to the ensemble results above, we also only change “inner parameters”, hence, only parameters that affect the dynamic of the “inner” Hopfield Neural Net.

| fixThr,<br>normTo | 1, 1  | 1, 2  | 2, 1  | 2, 2  | 2, 5  | 5, 1  | 5, 2  | 10, 1 | 10, 2 |
|-------------------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| 1, 1              | 1.0   | 0.655 | 0.743 | 0.649 | 0.55  | 0.72  | 0.643 | 0.748 | 0.657 |
| 1, 2              | 0.655 | 1.0   | 0.641 | 0.849 | 0.605 | 0.64  | 0.839 | 0.636 | 0.775 |
| 2, 1              | 0.743 | 0.641 | 1.0   | 0.647 | 0.552 | 0.713 | 0.638 | 0.746 | 0.649 |
| 2, 2              | 0.649 | 0.849 | 0.647 | 1.0   | 0.603 | 0.632 | 0.808 | 0.636 | 0.769 |
| 2, 5              | 0.55  | 0.605 | 0.552 | 0.603 | 1.0   | 0.582 | 0.586 | 0.566 | 0.595 |
| 5, 1              | 0.72  | 0.64  | 0.713 | 0.632 | 0.582 | 1.0   | 0.633 | 0.768 | 0.645 |
| 5, 2              | 0.643 | 0.839 | 0.638 | 0.808 | 0.586 | 0.633 | 1.0   | 0.634 | 0.775 |
| 10, 1             | 0.748 | 0.636 | 0.746 | 0.636 | 0.566 | 0.768 | 0.634 | 1.0   | 0.648 |
| 10, 2             | 0.657 | 0.775 | 0.649 | 0.769 | 0.595 | 0.645 | 0.775 | 0.648 | 1.0   |
| 0.672             | 0.671 | 0.705 | 0.666 | 0.699 | 0.58  | 0.667 | 0.695 | 0.673 | 0.689 |

Table 4.11: Diversity measure of classifiers from table 4.10. The final row contains the average diversity value for each classifier.




| Model   | HFPset | DR   | FPR  | fixThr | normTo |
|---|--------|------|------|--------|--------|
| NCC-HaarNN  | BlockN | 0.59 | 0.11 | -1     | 2      |
|  |        |      |      |        |        |
| NCC-HaarNN  | BlockN | 0.56 | 0.11 | -1     | -1     |
|  |        |      |      |        |        |
| NCC-HaarNN  | BlockN | 0.56 | 0.12 | -1     | 5      |
|  |        |      |      |        |        |

Table 4.12: NCC-HaarNN ensembles using the BlockN HFP set, trained to 50 members. These classifier ensembles use the logistic activation function 3.8 and a learned threshold for all neurons.



|        |       |       |       |
|--------|-------|-------|-------|
| normTo | 2     | -1    | 5     |
| 2      | 1.0   | 0.475 | 0.561 |
| -1     | 0.475 | 1.0   | 0.478 |
| 5      | 0.561 | 0.478 | 1.0   |
| 0.505  | 0.518 | 0.476 | 0.52  |

Table 4.13: Diversity of the ensemble classifiers from table 4.12.

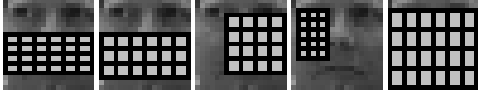



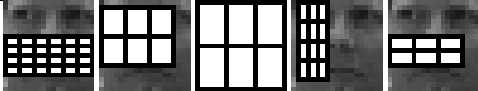

| Model      | HFPset  | DR    | FPR    | fixThr | normTo | jT  | usePT |
|------------|---|-------|--------|--------|--------|---|-------|
| NCC-HaarNN | BlockN  | 74.07 | 0.0093 | 10     | 1.0    | false   | false |
|            |    |       |        |        |        |    |       |
| NCC-HaarNN | BlockN  | 85.19 | 0.0048 | 5      | 1.0    | true  | true  |
|            |  |       |        |        |        |  |       |
| NCC-HaarNN | BlockN  | 87.04 | 0.0061 | 5      | 1.0    | true  | false |
|            |  |       |        |        |        |  |       |

Table 4.14: NCC-HaarNN cascade classifiers using the BlockN HFP set trained with up to 10 nodes using several varied parameters.

|            |       |       |       |
|------------|-------|-------|-------|
| classifier | 1     | 2     | 3     |
| 1          | 1.0   | 0.449 | 0.45  |
| 2          | 0.449 | 1.0   | 0.633 |
| 3          | 0.45  | 0.633 | 1.0   |
| 0.511      | 0.45  | 0.541 | 0.542 |

Table 4.15: Paired diversity of the three cascade classifiers, the results of which are shown in table 4.14.





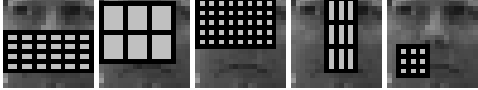

| Model      | HFPset  | DR    | FPR    | fixThr | normTo | jT    | usePT   |
|------------|---|-------|--------|--------|--------|-------|---|
| NCC-HaarNN | BlockN  | 79.63 | 0.0052 | 5      | 1.0    | false | true  |
|            |  |       |        |        |        |       |  |
| NCC-HaarNN | BlockN  | 85.19 | 0.0053 | 10     | 1.0    | false | true  |
|            |  |       |        |        |        |       |  |
| NCC-HaarNN | BlockN  | 81.48 | 0.008  | 10     | 1.0    | true  | false   |
|            |  |       |        |        |        |       |  |

Table 4.16: 10er CC-HaarNN cascades trained with several varied parameters.

| classifier | 1     | 2     | 3     |
|------------|-------|-------|-------|
| 1          | 1.0   | 0.648 | 0.572 |
| 2          | 0.648 | 1.0   | 0.629 |
| 3          | 0.572 | 0.629 | 1.0   |
| 0.616      | 0.61  | 0.639 | 0.6   |

Table 4.17: Paired diversity of the three cascade classifiers, the results of which are shown in table 4.16.

The NCC-HaarNN cascade classifiers, the results of which are shown in tables 4.14 and 4.16, use the BlockN HFP set and are all trained with up to ten cascade nodes. We use  $jT$  as abbreviation for *justThres* and  $usePT$  for *usePatternThres*. We vary both parameters  $fixThr$  and  $normTo$  and the usage of the activation function, i.e. the binary ( $jT = true$ ) or logistic activation function ( $jT = false$ ) (see equations 3.9 and 3.8). Further, we vary whether we use one threshold for all neurons ( $usePT = false$ ) or an individual threshold for every neuron ( $usePT = true$ ).

Corresponding to the cascade classifiers in tables 4.14 and 4.16, we depict diversity in tables 4.15 and 4.17. The diversity of the first three classifiers is higher than the diversity of the other three classifiers. However, all of the classifiers create significant diversity.

Table 4.18 shows the result of the merge cascade classifiers which confirmed the diversity of the classifiers using a different test set. The final row merges all six

| Model         | DR      | FPR    |
|---------------|---------|--------|
| Merge 4.14    | 64.81   | 9.4E-4 |
| Merge 4.16    | 68.518  | 0.001  |
| Merge all six | 59.2593 | 5.0E-4 |

Table 4.18: Results of the merging NCC-HaarNN cascade classifiers in tables 4.14 and 4.16.

classifiers, which further reduces the detection and false-positive rate, while also underlining the diversity of all classifiers.

### 4.3.4 Discussion and Conclusion

#### Inner Parameters create Diversity

Varying several parameters creates diversity, as we showed in table 4.10. There, we created NCC-HaarNN ensembles of up to 80 members using BlockN as the HFP set and by just varying two parameters. Table 4.11 shows the diversity values for each pair of classifiers. No classifier is equal or near equal to another classifier, despite all of the classifiers using exactly the same features (fixed random sequence) for training. These two parameters create different classifiers, hence they increase diversity.

The ensembles in table 4.12 are trained with up to 50 base classifiers and use a learned threshold instead of a fixed threshold as before. Also, these ensembles create diversity as depicted in table 4.13. There, we vary the *normTo* parameter. The threshold is learned during training. Every second row of table 4.12 shows the first five chosen features. Here, we can observe that all chosen features are different, although the feature set to choose from is the same in every training iteration. Only varying these HNN parameters creates these differences and, thus, the diversity.

However, we questioned whether a cascade classifier would retain this diversity. A cascade classifier contains several ensembles. It uses far more features, meaning there is a greater chance of the features being repeated. Thus, it may be the case that two cascade classifiers will have the same features as a whole, i.e. these features are in different sequences and have different weightings. Hence, we have trained a cascade classifier that only changes the inner HaarNN parameter. In addition, we varied two more inner parameters. The results of this are shown in tables 4.14 and 4.16.

As seen in diversity measure tables 4.15 and 4.17, the cascade classifiers also create diversity. Again, every second row shows the first five chosen features, and again, the chosen features are all different. However, the second method for analyzing diversity creates the merge cascade classifiers, the results of which are shown in table 4.18. There, the detection rate and the false-positive rate are decreased, which would not happen if they were very similar. Also, the other parameters create diversity, but we choose these as examples. Finally, we can create diversity by varying parameters which determine how the Hopfield Neural Network works within the HaarNN.

## **4.4 Required Number of Features**

### **4.4.1 Introduction to Experiments**

The training time is a key factor, and it corresponds to the number of features used. Therefore, it is beneficial to have small feature sets, or to use feasible small subsets. During training, we want to use as few features as possible which, nonetheless, create useful ensembles. For many applications, it is not feasible to have a very long training time. Analyzing many different parameters is difficult and, in fact, unfeasible for online learning. Hence, it is an advantage to train with as few features as possible. Brubaker et al. [8] showed that the random selection of subsets creates ensembles with comparable accuracy. However, we have different sets and classifiers, so we can prove whether this observation is also valid for our approach. Mostly in this thesis, we have trained cascade classifiers with a low number of features. In this section, we examine whether this is a useful method or whether a higher number of features provides better cascades.

### **4.4.2 Results of Number of Features**

In table 4.19 we consider a differing number of features in the sequence: 50, 100, 250, 500, 1000, 2500, 5000. The features chosen during training are different. The detection rate varies between 0.92 and 0.88. With increasing features, the false-positive rate decreases at first, but then increases for 2500 and 5000 features. The number of base classifiers decreases constantly, while the number of features increases.

Figure 5.12 shows several trained NCC-HaarNN cascades trained with up to 5 nodes using 100, 500, 2000, as well as all of the features with their detection and false-positive rate.


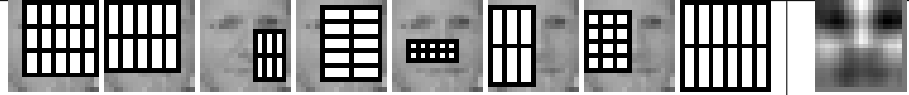
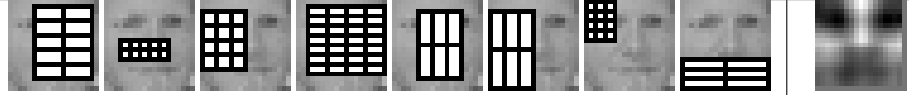




| Model  | HFPset | DR   | FPR       | NF   | Sum-BC |
|--|--------|------|-----------|------|--------|
| NCC-HaarNN   | BlockN | 0.92 | 0.0060073 | 50   | 474.0  |
|    |        |      |           |      |        |
| NCC-HaarNN   | BlockN | 0.9  | 0.0041099 | 100  | 433.0  |
|    |        |      |           |      |        |
| NCC-HaarNN   | BlockN | 0.88 | 0.0032505 | 250  | 402.0  |
|    |        |      |           |      |        |
| NCC-HaarNN   | BlockN | 0.88 | 0.0029399 | 500  | 364.0  |
|    |        |      |           |      |        |
| NCC-HaarNN   | BlockN | 0.88 | 0.0025103 | 1000 | 323.0  |
|   |        |      |           |      |        |
| NCC-HaarNN   | BlockN | 0.91 | 0.004103  | 2500 | 275.0  |
|  |        |      |           |      |        |
| NCC-HaarNN   | BlockN | 0.89 | 0.0034924 | 5000 | 223.0  |
|  |        |      |           |      |        |

Table 4.19: HaarNN cascade classifiers trained with several different number of features and the BlockN HFP set.

Table 4.20 shows the median values of the distribution in figure 5.12. The results are similar to the previous table 4.19. Using a higher number of features reduces the false-positive rate, but also the detection rate. We also see a clear reduction in the number of base classifiers.

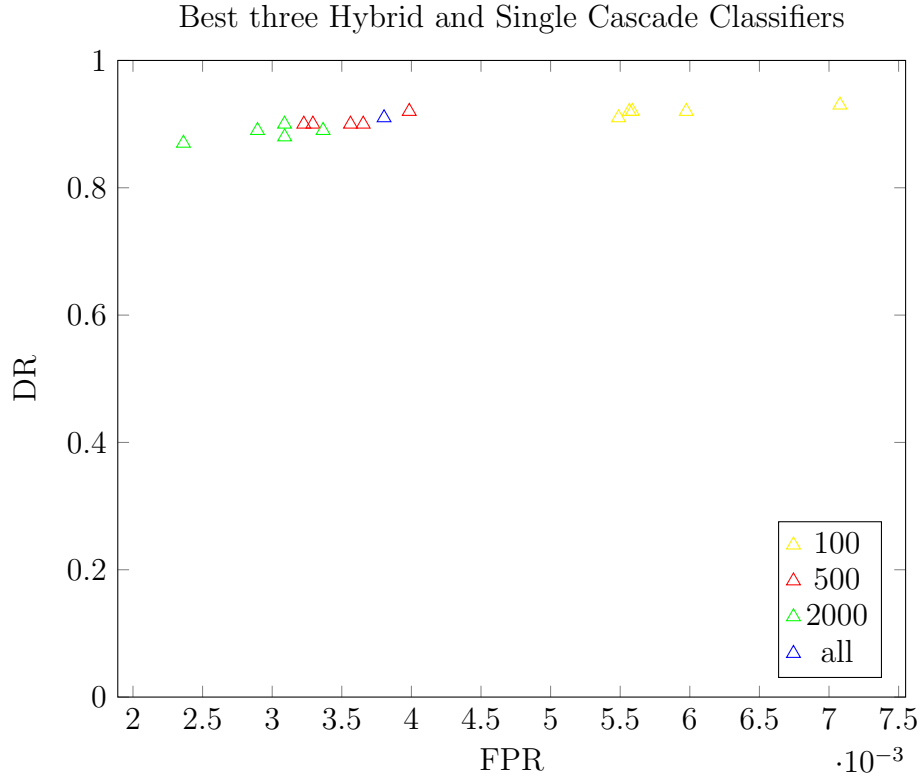


Figure 4.4: This figure shows the NCC-HaarNN cascade classifiers with their detection and false-positive rates. All classifiers are trained using the TLB69 HFP-set. The cascade classifiers are trained using 100, 500, 2000 and all features in each iteration with up to five node classifiers.

| Model             | HFPset | DR   | FPR       | NF   | Sum-BC |
|-------------------|--------|------|-----------|------|--------|
| Median NCC-HaarNN | TLB69  | 0.92 | 0.0055654 | 100  | 585.0  |
| Median NCC-HaarNN | TLB69  | 0.9  | 0.0035625 | 500  | 464.0  |
| Median NCC-HaarNN | TLB69  | 0.89 | 0.0030893 | 2000 | 384.0  |
| Median NCC-HaarNN | TLB69  | 0.91 | 0.0038037 | all  | 311.0  |

Table 4.20: Median values of cascade classifiers in figure 5.12. These are not trained classifiers, but a center-like median of the distribution.

### 4.4.3 Discussion and Conclusion

The results of table 4.19 show that the cascades trained with more features achieve a lower false-positive rate, but also a lower detection rate. The same can be seen in figure 5.12 and their median values provided in table 4.20. However, irrespective of how we evaluate the different detection rates and false-positive rates, the improvement is not as high, so we can conclude that training with a small number of features is worse than with a larger number of features. Further, there is no guarantee of achieving a better classifier using more features, yet it leads to a pronounced increase in training time. A cascade classifier with five nodes consists of a few hundred base classifiers. For every base classifier, the number of features is proven during training which is a few hundred times. Therefore, we have a few hundred iterations for 100 features or 500 or 2000, which leads to a huge increase in training time. By using more features, we create cascades with fewer base classifiers, and training will stop earlier. However, this advantage in training time also does not change the fact that training with smaller sets is faster because, given our results, the number of iterations is lower for training with 100 features in a set ( $100 \text{ features} \cdot 585 \text{ classifiers} < 500 \text{ features} \cdot 464 \text{ classifiers}$ ). These cascades are only trained with up to five nodes, and a longer training time will decrease, but not remove, the difference.

Considering all the classifiers we have trained, there are many good results only using 100 features. Finally, training the cascade classifier with this bagging-like feature selection is a correct way of decreasing training time while maintaining a similar level of accuracy but getting classifiers with a higher number of base classifiers.

## 4.5 Summary

In this chapter, we examined in more detail the difference of our classifier models. The only difference between the HFP and the HaarNN classifier is the Hopfield Neural Network that learns the pattern instead of using an average. Both classifier models provide different results and behavior, therefore making both models diverse.

In the following section, we analyzed whether the HNN within the HaarNN classifier increases diversity. We show that the HaarNN creates diversity by changing several parameters that only affect how the HNN works.

Finally, we examined whether we can use our models with a small feature set to maintain a fast training process. We showed that we can create good results

with significantly smaller feature sets and only slightly inferior accuracy.



# Chapter 5

## Increasing Diversity by Features and Forced Architecture

### 5.1 Introduction

In this chapter, we introduce more different HFP sets and another hybrid architecture to increase diversity further. Our aim in developing all the different HFP sets is to increase diversity. However, the overall question is whether all of these sets are useful? Is there one set that outperforms all the others? Or will the most diverse set be best? Is there a set that includes the other sets? Do they only differ in terms of geometry but not with a view to their classification ability? Simply being diverse is not a guarantee that we can create feasible classifiers out of those sets. Hence, analyzing which sets we can use to create feasible classifiers is the first question in this section, followed by an analysis of how diverse the different HFP sets are.

Finally, we analyze our forced hybrid architecture. The training process for our hybrid architecture described earlier chooses from a single set of heterogeneous classifiers. Having a hybrid set of classifiers to choose from does not guarantee a hybrid ensemble in the end. Some features or classifiers might be better than the others, thus offering better results on the training set and leading to them being chosen more often. However, a training set is only a fraction of reality, and adapting to this set does not mean describing all appearances, but providing the option of overfitting. Having more diversity could create more suitable classifiers. The notion of forcing the training process to choose classifiers alternately from different sets achieves greater diversity by choosing the more diverse classifiers instead of focusing on the best, therefore creating better classification results.

## 5.2 Methods

In the following section, we describe the extended Haar-Feature-like patches. The aim of introducing more sets is to increase diversity further.

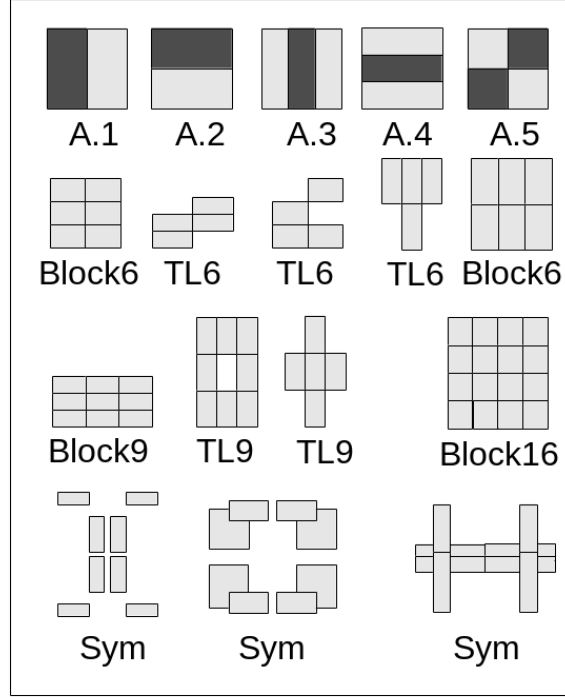


Figure 5.1: Original Haar-like features (HF) and more complex Haar-feature-like patches (HFP). The feature value of the original features is the pixel sum difference of the gray and white rectangles. Instead of a single value, the HFP uses the vector defined by the values of all dedicated rectangle pixel sums. A) represents the features used by Viola and Jones, while TL6 and TL9 describe the group of Tetris-like features derived from Block6 and Block9. The final row consists of a few examples of the Symmetrical HFP (Sym).

### 5.2.1 More Types of Haar-Feature-like patches (HFP)

In our experiments, we used different shaped HFP sets because we need a high diversity for our classifiers and our features. We have developed different HFP shapes as summarized in figure 5.1. Below, we will describe these several types in detail. The different shapes shown below are prototypes. We used these prototypes to derive several concrete Haar-Feature-like patches which form the HFPs we use for training and classification, as described in section 3.2.1.

### Haar-like features as HFP

The first group of HFP sets is the Haar-like features used by Viola and Jones. As described in section 3.2.1, we use the dedicated area values as a vector of the features depicted in the first row of figure 5.1,

The next HFP sets are similar to the previously introduced TLB69 set (see section 3.2.1). Here, we create systematically different options for one dimension and separate the features in different sets by their shape; two TL and two block sets.

### Tetris-like 6 (TL6)

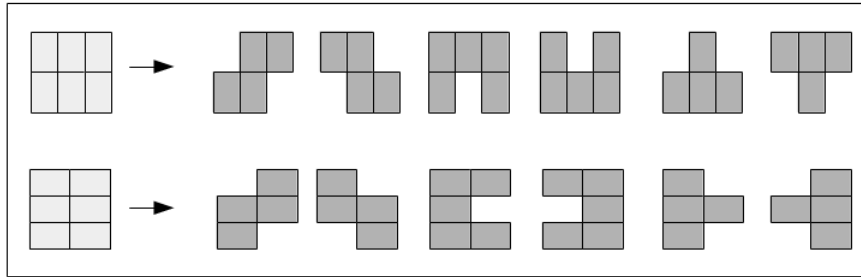


Figure 5.2: The Tetris-like Six (TL6) shapes are composed of six boxes with a width and height of 2x3 and 3x2.

Figure 5.2 shows the shapes of HFP sets, which we have called Tetris-like Six (TL6). The six in the name describes the number of boxes. The width and height of the whole figure are two rows and three columns (2x3) and, vice versa, three rows and two columns (3x2). As seen in figure 5.2, we start with the full block, and then we cut out boxes from the whole block to arrive at the final shapes, which become our resulting prototypes and look like Tetris blocks.

### Tetris-like 9 (TL9)

The features in figure 5.3 look similar to the TL6 set in figure 5.2 above, but in contrast to the TL6 set, they are cut out from a 3x3 block. We called this set Tetris-like Nine (TL9) because it also has similarities to Tetris blocks. Based on nine blocks, TL9 has more variations than TL6.

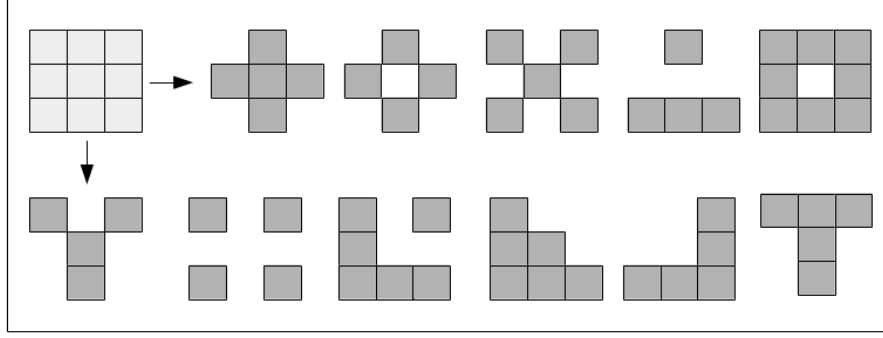


Figure 5.3: The Tetris-like Nine (TL9) shapes are composed of nine boxes with a width and height of 3x3.

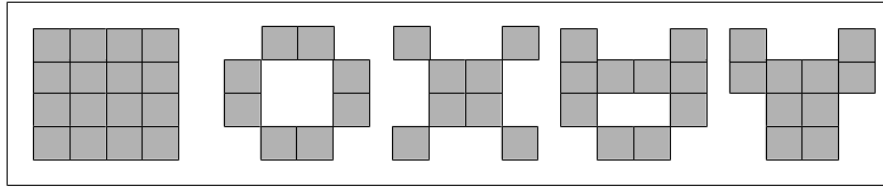


Figure 5.4: CB4x4 is the abbreviation of "Cutting Blocks of 16" boxes with a width and height of 4x4.

### Cutting Blocks 4x4 (CB4x4)

The HFP set "Cutting Blocks 4x4 (CB4x4)", shown in figure 5.4, is created in the same way as TL6 and TL9. We have a rectangle of 16 blocks with a width and height of four blocks. Again, we cut out several blocks to create the final shape.

### Block6, Block9, Block16

Figure 5.5 presents four HFP sets. The first is the Block6 set, second Block9, third Block16 set, and fourth the BlockRN set. As the name suggests, they are all block features, made of rectangles which fill all of the places in the rows and columns. Instead of the Tetris-like sets, where we cut out some rectangles in various positions, the Block features have no removed cells. The number  $N$  at the end of the names "Block $N$ " is the number of rectangles that determines the dimension ( $width \cdot height$ ) of the feature prototype.

Next are the Block9 sets, which consist of three rectangles in one column and three in one row. i.e. each block contains nine rectangles. The final block set with a fixed width and height is Block16. It consists of four blocks in one column and four blocks in one row, in the same pattern as the other block sets.

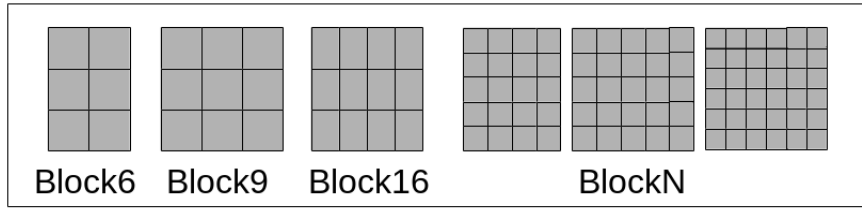


Figure 5.5: Here are four HFP sets. The three sets on the left are Block6, Block9, and Block16. They consist of six (two rows and three columns), nine (three rows and three columns), and sixteen rectangles (four rows and four columns). The three blocks on the right belong to the BlockN set. We create these blocks by systematically increasing the number of rectangles in a row and a column.

The BlockN set differs somewhat from the three previously described block sets in that the number of rectangles is not fixed. The dimension of the BlockN set is created systematically, beginning with three rectangles in a row and three in a column, then increasing the number of rectangles in a row by  $row_{next} = row + 1$  (and equally in a column) until a given max width (height) for the whole block is reached (see figure 3.6).

### Symmetrical Shape 2 (Sym2)

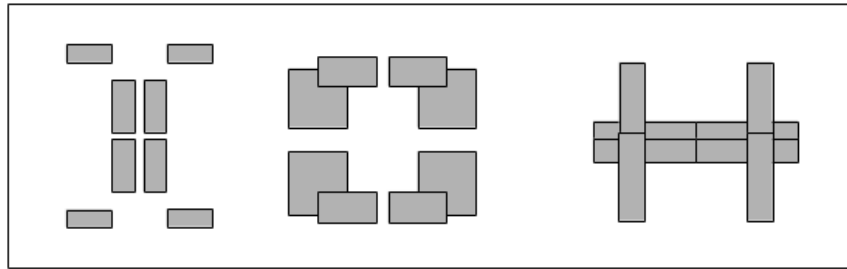


Figure 5.6: Sym2 is the abbreviation for Symmetrical Shape 2. It is created by mirroring (horizontally and vertically) two rectangles with a different width, height, and position.

Another set of features is the Symmetrical Shape 2 (Sym2) set which can be seen in figure 5.6. We create them by randomly creating two rectangles and mirroring them to the other side vertically and horizontally. The two source rectangles are of a different width, height, and position, and they can overlap each other. The

set is created systematically, as with the other sets, by creating and increasing all possible rectangles.

### Random-N

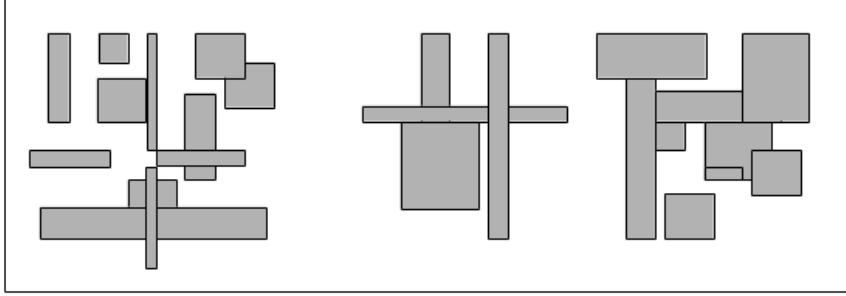


Figure 5.7: All of the parameters that determine the Random-N HFP set are randomly chosen with a different width, height and position of a rectangle and also the number of rectangles.

We also created an HFP set that contains randomly chosen rectangles as depicted in figure 5.7. The width, height, and position of every single rectangle is randomly chosen, and also the number of rectangles for one feature is a random decision.

### 5.2.2 Forced Hybrid Architecture

As mentioned in 2.4, diversity plays a prominent role in the success of ensembles. One way of increasing diversity is to use different base classifiers. In previous experiments, we used different classifier types and combined them into a single set. In the following experiments, we do not combine the sets but force the algorithm to alternate the sets. If we use one entire set with all different types, one classifier type can dominate the selection process. By forcing the algorithm to alternate, every classifier type will form an equal part of the final ensemble.

We apply the forced hybrid architecture to the cascade creation and the ensemble creation. The underlying idea is the same for both. We use two different classifiers and alternate their selections, but when creating an ensemble, we alternate the base classifiers, whereas when creating a cascade, we alternate the ensemble classifiers.

### Forced Hybrid AdaBoost (FH-AdaBoost)

We apply the forced hybrid architecture to AdaBoost. AdaBoost chooses the best classifier for a given set of classifiers. There, the best classifier is the one with the lowest error on a set of training images. Every training image has a weight and, once AdaBoost has chosen a classifier, the training images will be re-weighted according to the error of the chosen classifier. As a result of re-weighting, the misclassified training images become more important to the next iteration (see section 2.2.3).

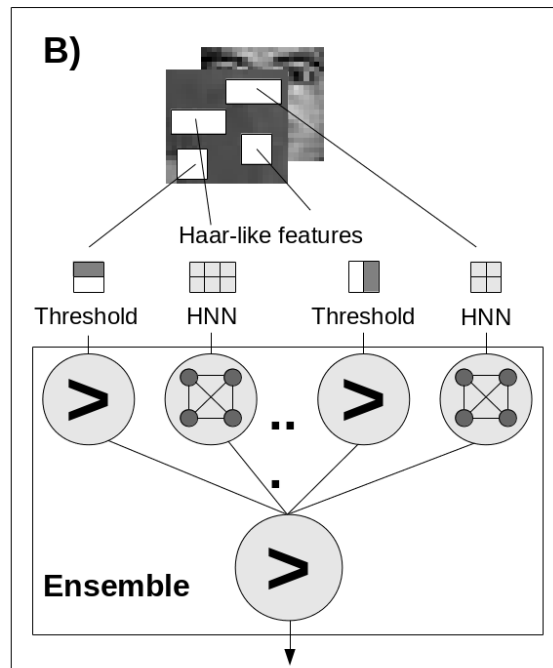


Figure 5.8: Forced hybrid architecture is used to create an ensemble; combining threshold classifiers and HaarNN classifiers. This figure describes the forced hybrid AdaBoost which first chooses a threshold classifier, followed by a HaarNN classifier (and so on).

While AdaBoost uses one set of base classifiers, the forced hybrid architecture uses two (or more) sets of base classifiers and forces the selection process to alternate between these sets, which also differs from the previously discussed hybrid architecture that also uses several sets, but aggregates them. Afterwards, for example, a threshold classifier is chosen and AdaBoost re-weighted the training images, while the forced hybrid architecture then chooses a HaarNN classifier as depicted in figure 5.8. Both classifier types were trained using the same training

images and their weightings. If we aggregated both sets, AdaBoost would choose the best classifier, irrespective of whether it is an HaarNN classifier or a threshold classifier. With aggregation of both sets, it would be possible to create an ensemble that only contains one type of classifier. Forcing the algorithm to alternate the sets leads to a higher diversity, but it also has the drawback of not using the “best” classifier and, therefore (perhaps), creates an ensemble that is less accurate.

### Forced Hybrid Cascade (FH Cascade)

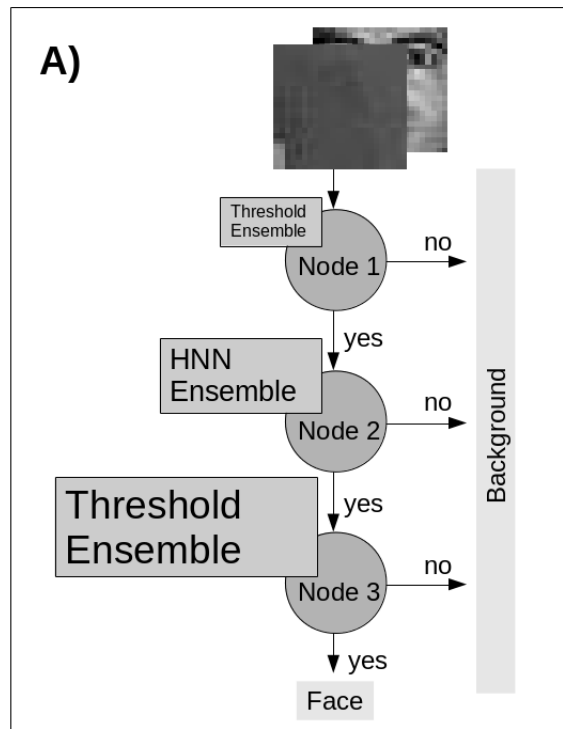


Figure 5.9: Forced hybrid architecture is used to create a cascade; combining ensembles of threshold classifiers and ensembles of HaarNN classifiers. This figure describes the cascade classifier creation process. One node of a threshold classifier ensemble is followed by a node of a HaarNN classifier ensemble.

Here, we apply the forced hybrid approach to create a cascade classifier. To increase computational speed, Viola and Jones wrapped several ensembles inside a degenerative decision tree, the so-called cascade, which contains one ensemble in every node. The cascade nodes are ordered from ensembles with low complexity (few members) to high complexity (many members) as suggested in figure 5.9. If the current node classifier classifies the considered sub-window as a face, it will be



passed on to the successive node. Only if the sub-window reaches the last node is it finally classified as “face”; otherwise, it will be rejected as “background” (see section 2.2.3).

To train the cascade, we also force the algorithm to alternate the two classifier types. Hence, instead of alternating base classifiers, we alternate ensemble classifiers. During cascade creation, this means we train one ensemble consisting of one classifier type (for example NCC-HFP), while the next ensemble consists of the other classifier type (for example ED-HaarNN). Here, the ensembles only contain one type of base classifier. Figure 5.9 illustrates alternating ensembles of threshold classifiers and ensembles of HaarNN classifiers.

## 5.3 The Feasibility of HFP Sets

### 5.3.1 Introduction to Experiments

In this first section, we examine whether the newly introduced sets are feasible for creating good cascade classifiers. Thus, we trained several cascade classifiers for every HFP set. Since we do not compare the sets, but only show that the sets are in fact feasible, it does not matter which training parameters we use. Further, we cannot create good cascade classifiers from bad ensemble classifiers, which is why we did not explicitly analyze the feasibility of single ensembles using the new sets. Finally, we compare the performance of the different sets. Here, we use the same training and test sets as for the other experiments.

### 5.3.2 Results for the Feasible Cascade Classifiers

Table 5.1 shows the results of the trained cascade classifiers for every new HFP set. The first three rows show sample cascade classifiers trained with the present sets, i.e. Base, TLB69, and BlockN.

The classifiers in table 5.1 show comparable results to the cascade classifiers which are trained using the sets Base, BlockN and TLB69. The range of detection rate and false-positive rate is similar, as is the case with the number of base classifiers used (Sum-BC). Figure 5.10 shows sample images of the above classifiers.

#### **TL9 does not work for HaarNN**

While most sets provide similarly good results, TL9 does not. Irrespective of the parameters used, the HaarNN cascade classifier did not achieve a good perfor-

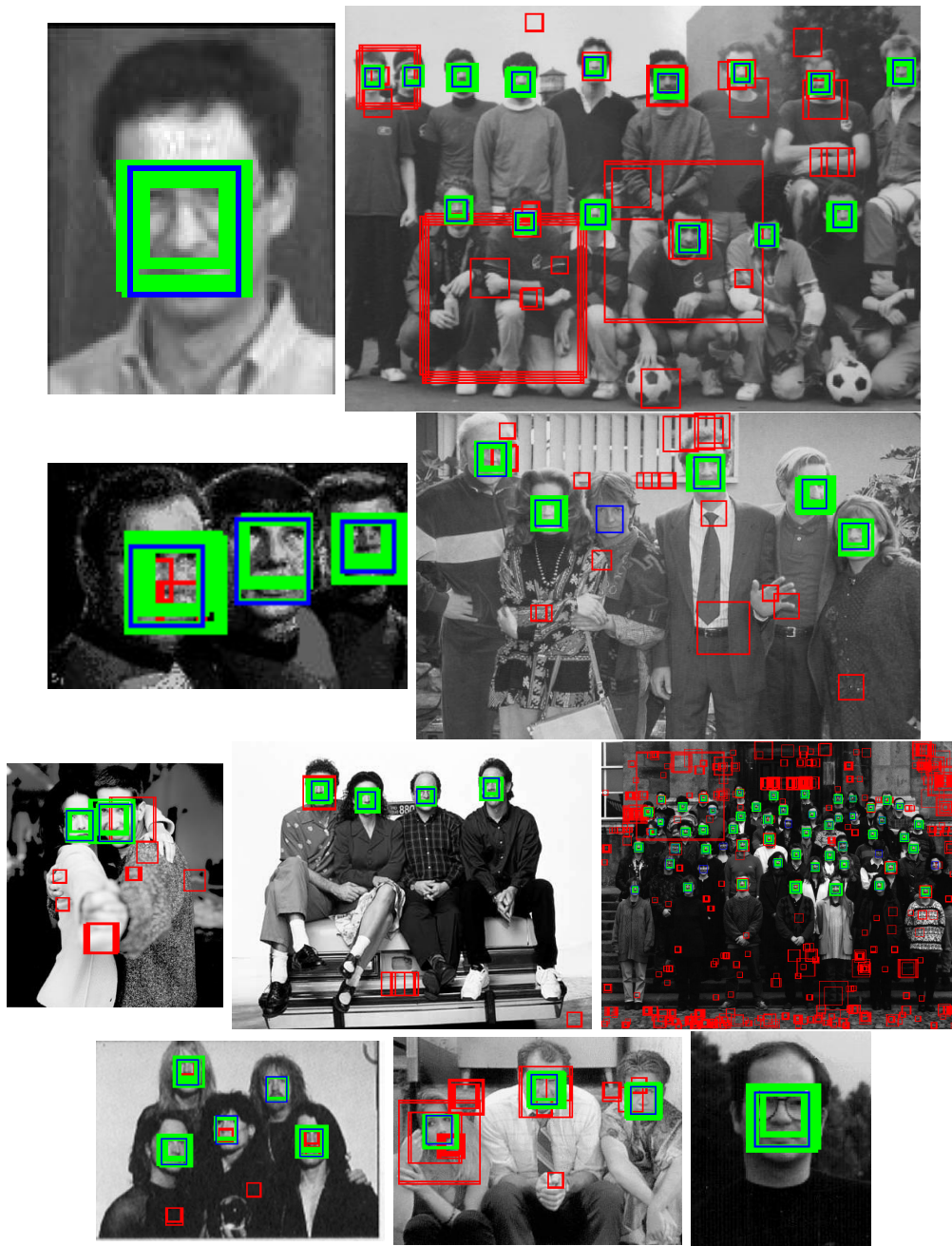


Figure 5.10: Samples of cascade classifiers trained with up to ten ensemble nodes. The first four images belong to the NCC-HaarNN cascade trained with the HFP set TL6. The upper six images belong to the NCC-HFP cascade classifier trained with the HFP set CB4x4.

| Model      | HFPset  | DR   | FPR      | NF  | Sum-BC |
|------------|---------|------|----------|-----|--------|
| NCC-HFP    | Base    | 0.74 | 2.125E-4 | 200 | 620.0  |
| NCC-HaarNN | TL6B9   | 0.66 | 2.931E-4 | 200 | 1205.0 |
| NCC-HFP    | BlockN  | 0.78 | 4.657E-4 | 200 | 910.0  |
| NCC-HaarNN | TL6     | 0.65 | 3.825E-4 | 100 | 586.0  |
| NCC-HFP    | TL9     | 0.73 | 9.638E-4 | 100 | 1160.0 |
| NCC-HFP    | CB4x4   | 0.69 | 6.615E-4 | 100 | 1175.0 |
| NCC-HaarNN | Block6  | 0.76 | 5.606E-4 | 100 | 929.0  |
| NCC-HaarNN | Block9  | 0.72 | 6.949E-4 | 100 | 1195.0 |
| NCC-HFP    | 2Sym    | 0.69 | 9.065E-4 | 100 | 1203.0 |
| NCC-HaarNN | RandomN | 0.79 | 5.417E-4 | 100 | 696.0  |

Table 5.1: This table shows the performance, first features, and the summed pattern image of several cascade classifiers. All of the cascade classifiers were trained with up to ten ensemble classifiers.

| Model      | HFPset | DR   | FPR   | NF   | Sum-BC | Number BC per Node             |
|------------|--------|------|-------|------|--------|--------------------------------|
| NCC-HaarNN | TL9    | 0.84 | 0.042 | 100  | 27.0   | [3, 2, 3, 2, 3, 2, 2, 4, 2, 4] |
| NCC-HaarNN | TL9    | 0.5  | 0.004 | 1500 | 35.0   | [5, 4, 8, 2, 2, 2, 5, 3, 2, 2] |
| ED-HaarNN  | TL9    | 0.74 | 0.009 | 1500 | 36.0   | [2, 3, 3, 2, 6, 4, 5, 2, 5, 4] |

Table 5.2: HaarNN cascade classifiers trained with the HFP set TL9. The number of base classifiers is very small because the ensemble creation breaks if errors in the base classifiers are too high.

mance. If a base classifier's performance is too low, the creation process breaks. The results in table 5.2 show what happens in this case. The 'Number BC per node' column shows the number of base classifiers for every ensemble node. None of the ensembles reaches a higher number. The result remains the same if we use all of the features.

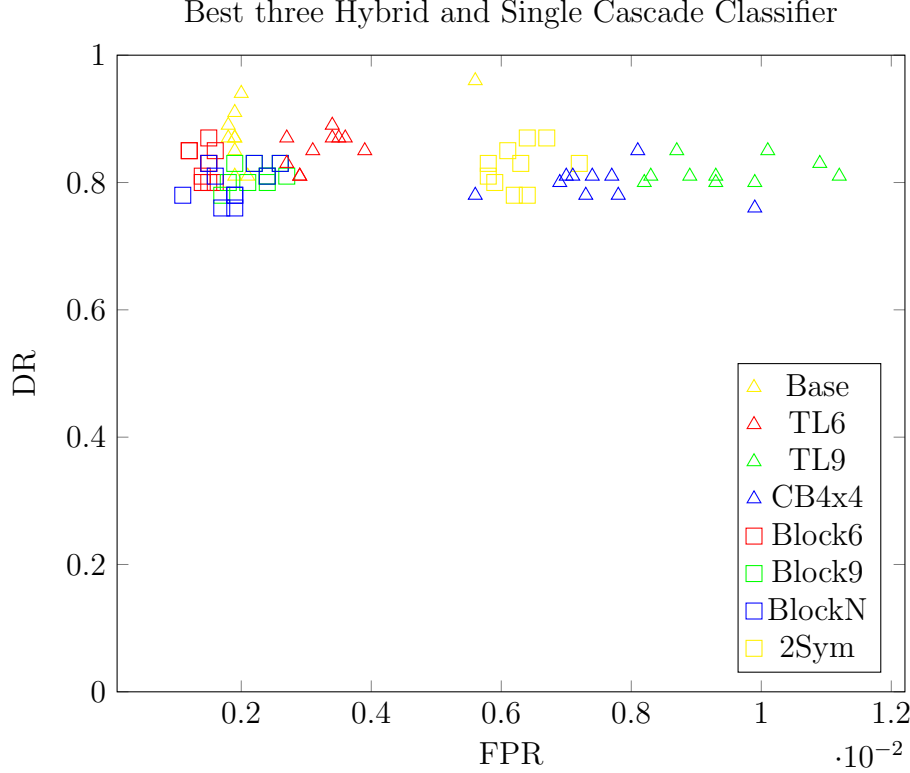


Figure 5.11: This figure shows the NCC-HFP cascade classifiers with their detection and false-positive rate. All of the classifiers are trained using the same parameters. The cascade classifiers are trained using 100 features during each iteration with up to five node classifiers. We trained ten classifiers for each described HFP set.

### Differences of HFP sets

While we considered whether the sets are feasible for creating a good classifier, we now consider how the accuracy of the sets differs. Figure 5.11 shows the distribution of the cascade classifiers in terms of their detection and false-positive rate. We use asymmetric AdaBoost to train the cascade classifiers with up to five nodes and 100 randomly chosen features per iteration. The classifiers are all trained using the same parameters except for the HFP sets randomly chosen during every iteration. Figure 5.11 shows that the different sets form clusters. Here, the Base, TL6, and the Block sets provide the best results compared to the other sets.

The distributions of the classifiers using the different sets are shown in figure 5.12. The HaarNN cascade classifiers, which also show clustered results, are trained with the same parameters as the HFP cascade classifiers. However, we removed the classifiers trained with the TL9 and CB4x4 sets from the figure because of

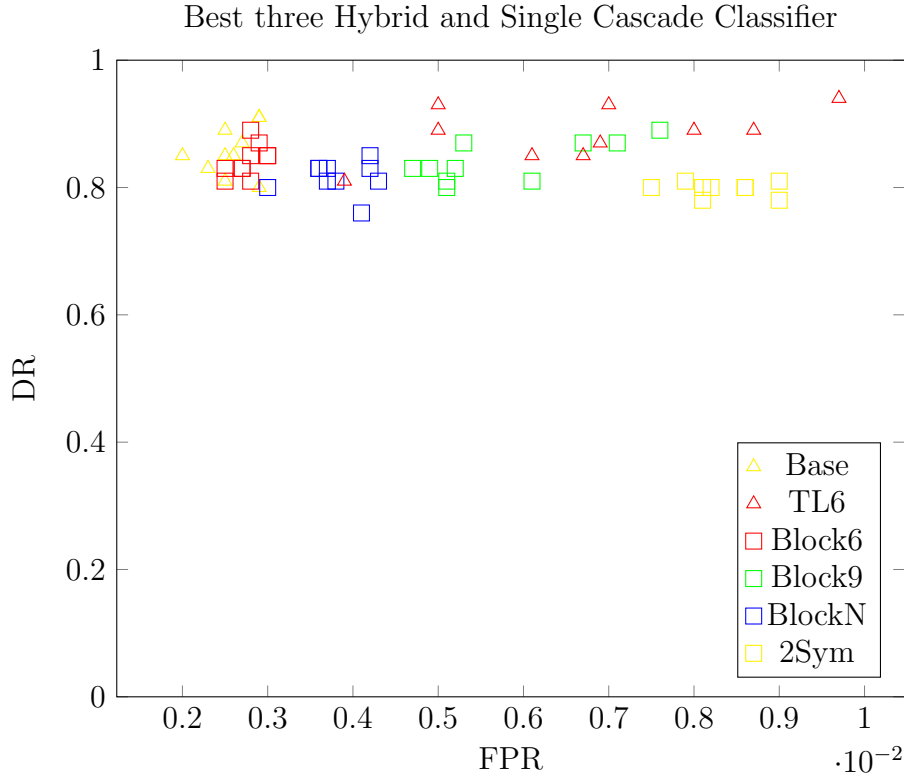


Figure 5.12: This figure shows the NCC-HaarNN cascade classifiers with their detection and false-positive rates. All of the classifiers are trained using the same parameters, except the HFP set. The cascade classifiers are trained using 100 features during each iteration with up to five node classifiers. We trained ten classifiers for every HFP set described. The classifiers using the HFP sets TL9 and CB4x4 are removed from the graphic for presentation purposes to show a similar scale as for the HFP cascades. TL9 does not reach suitable false-positive rates. The best false-positive rate for the five-node HaarNN cascades and CB4x4 set is 0.02.

their high false-positive rate. As mentioned above, the TL9 set does not reach acceptable false-positive rates, and the best false-positive rate for the HaarNN cascade classifier using CB4x4 is 0.02. Using AdaBoost (instead of asymmetric AdaBoost) also achieves the intended results with the CB4x4 HFP set. CB4x4 was removed for presentation purposes; otherwise the scale of the graphic would have been too large to demonstrate the distribution of the other sets. Without the CB4x4 HFP set, the graphic has the same scale as for the HFP classifier.

### 5.3.3 Discussion and Conclusion

A good cascade classifier consists of good ensemble classifiers. We therefore opted not to analyze the performance of single ensemble classifiers. Creating a feasible cascade classifier using the individual sets is possible as can be seen in the results provided in table 5.1. The cascade classifiers in table 5.1 provide results comparable with those for the classifiers we trained in the section 3. They are similar in their range of detection and false-positive rates as well as in their number of base classifiers. The HFP sets Base, Block6, and BlockN provide the best results for both classifier types. Block9 and TL6 create better HFP classifiers, while the other sets are feasible but not as good as the former sets. However, the HFP set TL9 used by a HaarNN classifier is an exception. While the cascade classifiers trained with the other sets reach a total number of about 600 to 1200 base classifiers, the HaarNN cascade classifiers trained with the TL9 set reach a number of 27 to 36 base classifiers because the training algorithm stops early if the error for the base classifiers is too high. The TL9 set does not create feasible base classifiers.

We can conclude that all of the created HFP sets are feasible for further usage. However, it makes no sense to train HaarNN cascade classifiers using only the HFP set TL9.

## 5.4 The HFP sets' Diversity

### 5.4.1 Introduction to Experiments

In this section, we explore the diversity of the individual sets. Thus, we train several cascade classifiers with different parameters for each of the sets we want to compare. To ensure that only the HFP sets influence diversity, we set all of the other parameters involved in the experiments, while also repeating the experiments with other parameters to be sure that the results are not only produced due to one specific parameter configuration.

### 5.4.2 Results for the Diversity of the HFP Sets

Table 5.3 lists the results of the HaarNN cascade classifiers. Every second row contains the five first chosen features as an example. At the end of each second row, we show the face-like aggregation image which combines all of the features. These features are all different because the underlying sets are different, yet the aggregation image could be the same. However, without measuring the distance















| Model   | HFPset | DR   | FPR      | NF  | Testset    |
|---|--------|------|----------|-----|------------|
| NCC-HaarNN  | Base   | 0.71 | 3.745E-4 | 100 | CMU-AC, SP |
|       |        |      |          |     |            |
| NCC-HaarNN  | TL6    | 0.67 | 4.291E-4 | 100 | CMU-AC, SP |
|       |        |      |          |     |            |
| NCC-HaarNN  | CB4x4  | 0.66 | 0.0013   | 100 | CMU-AC, SP |
|       |        |      |          |     |            |
| NCC-HaarNN  | Block6 | 0.77 | 5.185E-4 | 100 | CMU-AC, SP |
|     |        |      |          |     |            |
| NCC-HaarNN  | Block9 | 0.73 | 8.267E-4 | 100 | CMU-AC, SP |
|   |        |      |          |     |            |
| NCC-HaarNN  | BlockN | 0.73 | 6.54E-4  | 100 | CMU-AC, SP |
|   |        |      |          |     |            |
| NCC-HaarNN  | 2Sym   | 0.73 | 0.0016   | 100 | CMU-AC, SP |
|   |        |      |          |     |            |

Table 5.3: Here, the HaarNN cascade classifiers are trained with up to ten node classifiers, each using the same parameters except for the HFP set. They have different detection and false-positive rates, and the different feature types cover different areas. Note the fact that the summed pictures of all features are different.

of the images, it can be seen that they are all different. Aside from the different accuracy, the different aggregation images provide a further indication that the sets create diverse classifiers.

However, table 5.4 shows the diversity measure of the classifiers which are trained with the different sets. There, we added the HFP sets which are used to create the classifiers. A diversity value of 1.0 means that both classifiers are equal in their dedicated results. Hence, the crossing cell value for equal HFP sets (which have the same classifier) is 1.0. All of the other values vary between 0.31 and 0.53. This means that none of the classifier pairs is independent (value of 0.0), which we did not really expect, but none of them are equal or near equal either. The final row depicts the average of the correlation values for the corresponding classifier.

| HFP set   | Base  | TL6   | CB4x4 | Block6 | Block9 | BlockN | 2Sym  |
|-----------|-------|-------|-------|--------|--------|--------|-------|
| Base      | 1.0   | 0.401 | 0.31  | 0.513  | 0.418  | 0.477  | 0.331 |
| TL6       | 0.401 | 1.0   | 0.319 | 0.432  | 0.407  | 0.41   | 0.32  |
| CB4x4     | 0.31  | 0.319 | 1.0   | 0.327  | 0.433  | 0.337  | 0.341 |
| Block6    | 0.513 | 0.432 | 0.327 | 1.0    | 0.481  | 0.529  | 0.38  |
| Block9    | 0.418 | 0.407 | 0.433 | 0.481  | 1.0    | 0.455  | 0.365 |
| BlockN    | 0.477 | 0.41  | 0.337 | 0.529  | 0.455  | 1.0    | 0.391 |
| 2Sym      | 0.331 | 0.32  | 0.341 | 0.38   | 0.365  | 0.391  | 1.0   |
| Ave 0.399 | 0.408 | 0.381 | 0.344 | 0.443  | 0.426  | 0.433  | 0.355 |

Table 5.4: Diversity correlation values of the HaarNN cascade classifiers used in table 5.3. The cross-point depicts the correlation value of two classifiers. The smaller the value, the higher the diversity. Here, we compare HaarNN classifiers. The table key is the respective HFP set with which the classifier is trained.

The HFP cascade classifiers in table 5.6 are all created using the same training parameters as the ones used to create the HaarNN cascade classifiers in table 5.3. Table 5.7 depicts the paired diversity of the HFP cascade classifiers.

The correlation values in tables 5.4 and 5.7 show the paired diversity calculated using the CBCL test set. Another way to prove whether two classifiers are diverse is to create their merge cascade classifier, which is a conjunction of classifiers such that both cascade classifiers have to produce a positive result to create an overall positive result. If the two classifiers are independent, the resulting detection and false-positive rate of their merge cascade classifier are the product of the single values. The results of the two classifiers and their merge cascade are provided in



| Model      | HFPset | DR   | FPR      | NF  |
|------------|--------|------|----------|-----|
| NCC-HaarNN | Block6 | 0.77 | 5.185E-4 | 100 |
| NCC-HaarNN | BlockN | 0.73 | 6.543E-4 | 100 |
| Merge      |        | 0.62 | 1.801E-4 |     |
| NCC-HaarNN | Block6 | 0.77 | 5.185E-4 | 100 |
| NCC-HaarNN | TL6    | 0.67 | 4.291E-4 | 100 |
| Merge      |        | 0.53 | 8.38E-5  |     |

Table 5.5: The results of several cascade classifiers and their merge cascade classifiers so as to underline their diversity. Every merge cascade is depicted by *Merge* within the model column and is the result of both cascade classifiers above. Merge cascade means that both classifiers have to return a positive result for the given input.



Figure 5.13: The left images of both rows are the results of the NCC-HaarNN cascade classifier trained with the Block6 HFP set, while the images in the middle were trained with the TL6 HFP set. The right images show the results of the merge cascade classifier. The detection and false-positive rates of the classifiers are depicted in tables 5.5 and 5.6.

table 5.8. Both merge cascade classifiers have higher values than the product in terms of detection and false-positive rates. These values are therefore not independent, but lower than the single values, which illustrates the diversity of the different classifiers and, therefore, the diversity of the sets.

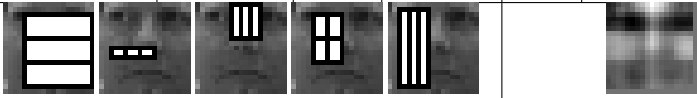






| Model  | HFPset | DR   | FPR    | NF  | Testset    |
|--|--------|------|--------|-----|------------|
| NCC-HFP  | Base   | 0.69 | 2.0E-4 | 100 | CMU-AC, SP |
|    |        |      |        |     |            |
| NCC-HFP  | TL6    | 0.76 | 6.0E-4 | 100 | CMU-AC, SP |
|    |        |      |        |     |            |
| NCC-HFP  | CB4x4  | 0.66 | 8.0E-4 | 100 | CMU-AC, SP |
|    |        |      |        |     |            |
| NCC-HFP  | Block6 | 0.74 | 3.0E-4 | 100 | CMU-AC, SP |
|    |        |      |        |     |            |
| NCC-HFP  | Block9 | 0.63 | 4.0E-4 | 100 | CMU-AC, SP |
|  |        |      |        |     |            |
| NCC-HFP  | BlockN | 0.66 | 2.0E-4 | 100 | CMU-AC, SP |
|  |        |      |        |     |            |
| NCC-HFP  | 2Sym   | 0.6  | 0.001  | 100 | CMU-AC, SP |
|  |        |      |        |     |            |

Table 5.6: This table shows the HFP cascade classifiers also used in table 5.3. These results also show their difference in terms of performance and feature appearance.

Figure 5.14 shows an example of two classifiers and their resulting merge cascade. The images accurately convey the diversity between the two classifiers.

The number of base classifiers differs with the number of cascade nodes. However, if the HFP set causes the diversity, we also expect to see a different correlation for cascade classifiers with a lower number of nodes. Therefore, we list the results of cascade classifiers trained to five nodes and their correlation values in tables 5.9 and 5.10. For this, we only use a subset of HFP sets and compare them to the

| HFP set  | Base  | TL6   | CB4x4 | Block6 | Block9 | BlockN | 2Sym  |
|----------|-------|-------|-------|--------|--------|--------|-------|
| Base     | 1.0   | 0.463 | 0.404 | 0.604  | 0.53   | 0.59   | 0.419 |
| TL6      | 0.463 | 1.0   | 0.677 | 0.546  | 0.598  | 0.535  | 0.452 |
| CB4x4    | 0.404 | 0.677 | 1.0   | 0.491  | 0.587  | 0.507  | 0.482 |
| Block6   | 0.604 | 0.546 | 0.491 | 1.0    | 0.631  | 0.658  | 0.468 |
| Block9   | 0.53  | 0.598 | 0.587 | 0.631  | 1.0    | 0.645  | 0.533 |
| BlockN   | 0.59  | 0.535 | 0.507 | 0.658  | 0.645  | 1.0    | 0.512 |
| 2Sym     | 0.419 | 0.452 | 0.482 | 0.468  | 0.533  | 0.512  | 1.0   |
| Ave 0.54 | 0.502 | 0.545 | 0.525 | 0.566  | 0.587  | 0.574  | 0.478 |

Table 5.7: Diversity correlation values of the HFP cascade classifiers in table 5.6. The cross-point depicts the correlation value of two classifiers.

| Model   | HFPset | DR   | FPR    | NF  |
|---------|--------|------|--------|-----|
| NCC-HFP | Block6 | 0.74 | 3.0E-4 | 100 |
| NCC-HFP | BlockN | 0.66 | 2.0E-4 | 100 |
| Merge   |        | 0.61 | 1.0E-4 |     |
| NCC-HFP | Block6 | 0.74 | 3.0E-4 | 100 |
| NCC-HFP | TL6    | 0.76 | 6.0E-4 | 100 |
| Merge   |        | 0.65 | 1.0E-4 |     |

Table 5.8: The results of several cascade classifiers and their merge cascade classifiers to prove their diversity. Every merge cascade is depicted by *Merge* within the model column and is the result of both cascade classifiers above. Merge cascade means that both classifiers have to react with a positive result for the given input.

classifiers shown before and also to the HFP classifier.

A configured detection rate and false positive rate, among others, control the training ensemble node classifiers of a cascade classifier, which causes a different number of base classifiers for most of the ensemble node classifiers. By comparing cascade classifiers, we compare classifiers mostly of different sizes. However, we want to prove that diversity does not depend on size, but on the HFP sets. Therefore, we also compare ensemble classifiers trained with different HFP sets and with a fixed size.



Figure 5.14: The left image is the result of the NCC-HFP cascade classifier trained using the Block6 HFP set, while the image in the middle was trained using the BlockN HFP set whose diversity and performance are shown in tables 5.8 and 5.6. The right image is the result of the merge cascade classifier created from the other two classifiers to illustrate their diversity.

| Model   | HFPset | DR   | FPR    | NF  | Testset    |
|---------|--------|------|--------|-----|------------|
| NCC-HFP | Base   | 0.87 | 0.0018 | 500 | CMU-AC, SP |
| NCC-HFP | CB4x4  | 0.83 | 0.0086 | 500 | CMU-AC, SP |
| NCC-HFP | Block6 | 0.83 | 0.0016 | 500 | CMU-AC, SP |
| NCC-HFP | Block9 | 0.8  | 0.0019 | 500 | CMU-AC, SP |
| NCC-HFP | BlockN | 0.81 | 0.0016 | 500 | CMU-AC, SP |

Table 5.9: Cascade classifiers trained to five nodes, using 500 features. These are the results of a subset of all described HFP sets and only involving HFP classifiers.

### Results for the HFP Ensemble Diversity

In the following section, we analyze whether the diversity of the different cascade classifiers is caused by the cascade training process with different ensemble sizes or by the bootstrapping mechanism. When training an ensemble, we can control these parameters. Therefore, we calculate and compare the diversity of ensemble classifiers trained with several HFP sets. We collated our results into three groups. In each group, we consider both HaarNN classifiers and HFP classifiers.

In table 5.11 we show the performance results for several HaarNN ensembles.

| HFPset    | Base  | CB4x4 | Block6 | Block9 | BlockN |
|-----------|-------|-------|--------|--------|--------|
| Base      | 1.0   | 0.397 | 0.585  | 0.493  | 0.561  |
| CB4x4     | 0.397 | 1.0   | 0.446  | 0.519  | 0.465  |
| Block6    | 0.585 | 0.446 | 1.0    | 0.612  | 0.663  |
| Block9    | 0.493 | 0.519 | 0.612  | 1.0    | 0.608  |
| BlockN    | 0.561 | 0.465 | 0.663  | 0.608  | 1.0    |
| Ave 0.535 | 0.509 | 0.457 | 0.576  | 0.558  | 0.574  |

Table 5.10: Correlation values for the cascade classifiers in table 5.9.

| Model      | HFPset     | DR   | FPR  |
|------------|------------|------|------|
| NCC-HaarNN | 2Sym-9     | 0.68 | 0.3  |
| NCC-HaarNN | CB4x4      | 0.86 | 0.36 |
| NCC-HaarNN | BlockN     | 0.75 | 0.2  |
| NCC-HaarNN | Block9-2th | 0.77 | 0.22 |
| NCC-HaarNN | Block9     | 0.77 | 0.23 |

Table 5.11: Several ensembles trained only with different HFP sets. They are tested with the CBCL test set. The ensemble classifiers are trained with up to 50 members. The last two ensembles use the same HFP set type but different subsets.

They are trained using identical configuration parameters except for the HFP set. All of the ensembles are trained to have 50 base classifiers. We also listed the results of two classifiers that are trained with the same HFP set type (Block9), but a different subset. The Block9 sets used differ in terms of the number of features. Block9-2 is about half the amount of Block9.

Table 5.12 shows the diversity of the classifiers in table 5.11. The final row of table 5.12 highlights the average correlation values. The classifier, which is trained with the 2Sym HFP set, has the lowest correlation value and therefore results in the highest diversity, followed by the ensemble using the CB4x4 set. A similar average correlation involving the other sets shows both ensembles trained with the HFP set Block9. However, the correlation of these two ensembles is also smaller than 1.0.

Table 5.11 shows the results for HaarNN ensemble classifiers, while table 5.13 depicts similar results for the HFP model ensembles, and, accordingly, table 5.14

| HFPset   | 2Sym  | CB4x4 | BlockA | Block9-2 | Block9 |
|----------|-------|-------|--------|----------|--------|
| 2Sym     | 1.0   | 0.434 | 0.429  | 0.449    | 0.433  |
| CB4x4    | 0.434 | 1.0   | 0.507  | 0.56     | 0.593  |
| BlockA   | 0.429 | 0.507 | 1.0    | 0.612    | 0.603  |
| Block9-2 | 0.449 | 0.56  | 0.612  | 1.0      | 0.658  |
| Block9   | 0.433 | 0.593 | 0.603  | 0.658    | 1.0    |
| 0.528    | 0.436 | 0.523 | 0.538  | 0.57     | 0.572  |

Table 5.12: Correlation diversity for the classifiers in table 5.11

| Model   | HFPset   | DR   | FPR  |
|---------|----------|------|------|
| NCC-HFP | 2Sym-9   | 0.6  | 0.18 |
| NCC-HFP | CB4x4    | 0.7  | 0.16 |
| NCC-HFP | BlockN   | 0.55 | 0.08 |
| NCC-HFP | Block9   | 0.65 | 0.12 |
| NCC-HFP | Block9-2 | 0.73 | 0.11 |

Table 5.13: Several ensembles trained only with different HFP sets. They are tested with the CBCL test set. The ensemble classifiers are trained with up to 50 members. The final two ensembles use the same HFP set type but different subsets.

| HFPset   | 2Sym  | CB4x4 | BlockA | Block | Block9-2 |
|----------|-------|-------|--------|-------|----------|
| 2Sym     | 1.0   | 0.408 | 0.466  | 0.535 | 0.477    |
| CB4x4    | 0.408 | 1.0   | 0.503  | 0.523 | 0.481    |
| BlockA   | 0.466 | 0.503 | 1.0    | 0.587 | 0.578    |
| Block9   | 0.535 | 0.523 | 0.587  | 1.0   | 0.643    |
| Block9-2 | 0.477 | 0.481 | 0.578  | 0.643 | 1.0      |
| 0.52     | 0.472 | 0.479 | 0.533  | 0.572 | 0.545    |

Table 5.14: Correlation diversity for the classifiers in table 5.13

lists their correlation values. Again, the HFP ensemble trained with the 2Sym HFP set shows the highest diversity, followed by the CB4x4 HFP ensemble, much like the HaarNN ensembles in tables 5.11 and 5.12.

### 5.4.3 Discussion and Conclusion

It is obvious from the different shapes that the HFP sets themselves are not equal, but it is not obvious that these differences also lead to different classification results.

Table 5.4 shows the diversity of the trained HaarNN cascade classifiers from table 5.3. No correlation value is 1.0 (except for the cross-points of the classifiers). Therefore, none of the classifiers are identical. As all of the other training parameters are equal, we can conclude that the HFP sets are the reason for the diversity of the trained cascade classifiers. The same is true for the HFP cascade classifiers in table 5.6 whose diversity is shown in table 5.7. There, the average diversity of the HaarNN classifiers is higher than that of the HFP classifiers, although not much.

To verify that the diversity is not only caused by the smaller CBCL test set, we confirm the correlation of the classifiers, also by creating the merge cascade. We tested the resulting cascade classifiers with all three test sets (CMU-A, CMU-C and Sung and Poggio as described in section 2.7). Let us consider the first two HaarNN classifiers from table 5.5. By multiplying the detection rate, we achieve a value of  $0.77 \cdot 0.67 = 0.52$ , which would be the detection rate of the merge cascade if both classifiers were independent of each other. Otherwise, the detection rate would be 0.67 if one classifier were a subset of the other. Our merge classifier reaches a detection rate of 0.62 for the CMU-A test set, which represents a low correlation. The merge classifier shows similar results, which underlines our conclusion that the HFP sets cause the diversity. Regarding the HFP cascade classifiers in table 5.8, we have different values but arrive at the same observation, namely that all cascade classifiers are diverse.

To double-check whether this is not attributable to randomness due to the ten cascade node, we compared cascade classifiers using a subset of HFP sets and trained them to five nodes instead of ten and 500 features instead of 100. While their performance results (see table 5.9) differ as expected (higher DR and FPR because of fewer ensemble nodes), the correlation values are similar (see 5.10). They are all slightly lower, but show similar characteristics. For example, both tables show the highest correlation between BlockN and Block6, and the lowest correlation between Base and CB4x4.

Looking at the diversity of the HFP sets, we started with tables 5.12 and 5.14,

which show exemplary results for HaarNN and the HFP model. As examples, we used HFP sets that are quite different in their appearance, as well as 2Sym, CB4x4 sets and three very similar sets which are BlockN and two Block9 sets. Here, Block9-2 is a subset of the Block9 HFP set. Tables 5.12 and 5.14 show the correlation values between all of the single classifiers.

The average correlation is about 0.53, and none is 1.0, meaning that all the sets are diverse. Hence, not only is their appearance different, they also create different results for same input. The classifiers with the highest similarity are those that trained using the HFP sets Block9, Block9-2, and BlockN. However, although the appearance is very similar or even the same for Block9 and its subset, they create diversity. Their correlation is higher than that of the other sets, but not much higher. By using the HFP sets Block9 and Block9-2, we can conclude that the difference in appearance does not have to be large to create diversity.

In section 4.2 we show that both models, trained using the same parameters and, in particular, the same HFP set, create diverse classifiers. However, despite being different in terms of their classification behavior, we can observe that the diversity is similar in that a larger difference in the appearance of the HFP set leads to a higher diversity for the classifier that used the HFP set. Hence, we conclude that there is a correlation between the geometrical similarity and the resulting diversity of the classifiers. We reached this conclusion without considering or searching for an exact description of the relation between HFP set appearance and the diversity of the classifier. This correlation is beyond the scope of this dissertation. However, we note that creating different geometrical sets causes what we have in mind: diversity.

Taking everything into account, we conclude that diversity is caused by the different HFP sets.

## **5.5 Forced Hybrid Architectures**

### **5.5.1 Introduction to Experiments**

In the previous chapters, we showed that we could increase diversity using our HFP sets and classifier models. In this section, we examine whether we can create a further benefit through our forced hybrid architecture which forces the training algorithms to choose classifiers out of different sets, therefore forcing diversity. In the first chapter, we also used a hybrid architecture which draws on different classifier types for training, but the different types are together in one set and the



decision as to which classifier type is chosen is left up to the AdaBoost selection process to choose the best. Our forced hybrid architecture changes this selection process by forcing it to alternate between the different classifier types or feature sets. The hybrid architecture, which uses a single set, offers a passive option to achieve greater diversity, while the forced architecture actively fosters more diversity.

Here, we pursue two methods. One involves alternating during ensemble training. If we force alternation within the ensemble creation process, it leads to ensemble classifiers consisting of different base classifiers or HFP feature types. The combined hybrid architecture also leads to ensembles of different classifier types, but the number of classifiers belonging to the different types is up to the selection process and can be any fraction. Also, only one classifier type is possible. Instead, ensembles that are forced to alternate will be of equal size for every classifier type. The second method entails alternating during cascade training. There, we change the classifier type for every ensemble subsequently added to the cascade classifier and consisting of only one classifier type. Thus, every ensemble node of the cascade classifier is trained with a different HFP set or classifier type, but the ensemble classifiers themselves are homogenous.

To examine whether the forced architecture creates a benefit, we trained several cascade classifiers using different parameters and then analyzed the results. In the results section below, we first examine the effect of altering only the HFP sets. Afterwards, we only alter the classifier types. Finally, we compare the distribution without restrictions.

### 5.5.2 Results for the Alternating Feature Sets

Below we present the results of our forced hybrid architecture where we only alternated the HFP sets.

#### Cascade Classifier Results

The following table 5.15 shows the advantage of the forced hybrid architecture, but also indicates that the results are ambiguous. The classifiers in table 5.15 are trained with a fixed random feature sequence where all of the classifiers are trained using the same subset of features with the aim of analyzing the effect of alternating the HFP sets.

FH-AdaBoost in the type column shows that we forced alternation of the sets when training an ensemble classifier. Instead, FH-Cascade means forcing alterna-











| Type        | Model   | HFPset | Set2   | DR   | FPR    | Testset   |
|-------------|---|--------|--------|------|--------|---|
| Single      | NCC-HFP   | Block6 |        | 0.75 | 0.0014 | CMU-AC, SP  |
| Single      | NCC-HFP   | 2Sym   |        | 0.66 | 0.003  | CMU-AC, SP  |
| H-Unite     | NCC-HFP   | 2Sym   | Block6 | 0.75 | 0.0016 | CMU-AC, SP  |
|             |    |        |        |      |        |    |
| FH-AdaBoost | NCC-HFP   | Block6 | 2Sym   | 0.77 | 0.0017 | CMU-AC, SP  |
|             |    |        |        |      |        |    |
| FH-AdaBoost | NCC-HFP   | 2Sym   | Block6 | 0.76 | 0.002  | CMU-AC, SP  |
|             |   |        |        |      |        |   |
| FH-Cascade  | NCC-HFP   | 2Sym   | Block6 | 0.7  | 0.0027 | CMU-AC, SP  |
|             |  |        |        |      |        |  |
| FH-Cascade  | NCC-HFP   | Block6 | 2Sym   | 0.65 | 0.001  | CMU-AC, SP  |
|             |  |        |        |      |        |  |

Table 5.15: The classifiers are trained with a fixed (random) set of 100 for every iteration. FH-AdaBoost means that we force alternation of the set during ensemble creation. Accordingly, the first five features of the FH-AdaBoost classifier consist of Block6 and 2Sym HFP features. FH-Cascade means forcing alternation after creating an ensemble node. Since we show the first five features of the first ensemble node classifiers, and the ensembles are trained using the same random sequence, the present features of the FH-Cascade classifiers are the same as those for the single classifiers.

| Type        | Model      | HFPset | Set2   | DR   | FPR    |
|-------------|------------|--------|--------|------|--------|
| Single      | NCC-HaarNN | BlockN |        | 0.89 | 0.0036 |
| Single      | NCC-HaarNN | Block9 |        | 0.91 | 0.0056 |
| FH-AdaBoost | NCC-HaarNN | Block9 | BlockN | 0.89 | 0.0032 |

Table 5.16: The forced hybrid cascade improves both single classifiers in terms of false-positive rate by maintaining the detection rate of a single classifier.

tion after creating an ensemble node, as described in section 5.2. The first two rows show the single classifiers, trained with the Block6 and 2Sym HFP sets, followed by several alternating variations. The classifier, trained with the Block6 set, provides a better detection and false-positive rate than the one trained with the 2Sym HFP set. However, all of the hybrid classifiers offer better performance than the 2Sym classifier. The FH-AdaBoost classifiers have the best detection rate and outperform the Block6 classifier, although half of the base classifiers are trained with the 2Sym set. The final two rows with the FH-Cascade type show the results of forcing alternation of the HFP set after every cascade node. There, the diversity of the different sets causes its clearest impact in reducing the false-positive rate, but also the detection rate.

### Comparing Best Results

The previous classifiers, the results of which we have shown previously, use the same random sequence of features for training. However, there are too many potential random sequences to compare them all. Hence, we have trained several single and forced hybrid cascade classifiers to analyze the distribution of the detection and false-positive rates of the different classifiers. All of the classifiers are tested using all three test sets (CMU-A, CMU-C, SP).

We trained several cascade classifiers using the same training parameters. In table 5.16, we compare the single and alternating classifiers with the highest detection rates. The alternating cascade classifier improves both single classifiers in terms of false-positive rate, while offering a poorer detection rate than the Block9 classifier and the same detection rate as the BlockN classifier.

The following table shows an alternating classifier which uses the CB4x4 HFP set and the BlockN set. Although, the false-positive rate of the single CB4x4 cascade classifier is worse than the other classifiers, forcing it to use the CB4x4 HFP set improves the alternating cascade classifier when it comes to detection

| Type        | Model      | HFPset | Set2   | DR   | FPR    |
|-------------|------------|--------|--------|------|--------|
| Single      | NCC-HaarNN | BlockN |        | 0.89 | 0.0036 |
| Single      | NCC-HaarNN | CB4x4  |        | 0.93 | 0.026  |
| FH-AdaBoost | NCC-HaarNN | CB4x4  | BlockN | 0.91 | 0.0055 |

Table 5.17: These results show that forcing the use of one set with a bad false-positive rate also leads to a feasible combined classifier. Note the reduction of the false-positive rate compared to the single CB4x4 classifier.

| Type       | Model      | HFPset | Set2   | DR   | FPR    |
|------------|------------|--------|--------|------|--------|
| Single     | NCC-HaarNN | Block9 |        | 0.91 | 0.0056 |
| Single     | NCC-HaarNN | Base   |        | 0.88 | 0.0017 |
| FH-Cascade | NCC-HaarNN | Base   | Block9 | 0.89 | 0.0017 |

Table 5.18: The forced hybrid cascade classifier improves the false-positive rate of both single classifiers as well as the detection rate of one classifier.

rate. However, the false-positive rate of the combined classifier does not improve the false-positive rate for both single classifiers, but it does provide a comparable value.

### Comparing Forced Hybrid Ensemble Classifiers

To verify that the effect of forcing alternation is not a result of the cascade classifier training process with its different sizes and negative samples, we also train some ensemble classifiers with a fixed number of base classifiers. We chose two classifiers with high diversity, and as a third result, two classifiers with low diversity.

Table 5.19 shows the result of ensembles, trained with the HFP sets 2Sym and Block6. Their correlation value is 0.208. The first two classifiers are trained using only one HFP set, whereas the others are trained using the combined sets or are forced to alternate the sets. The classifier that uses the 2Sym and Block6 HFP sets as a combined set, and the classifiers that are trained to force to alternate both HFP sets, show greater accuracy ( $\text{Acc}(w)$ ).

The ensembles in table 5.19 are trained up to 80 base classifiers.

Table 5.20 also uses ensembles with the low correlation value of 0.237. To show that the former results are no exception, we provide results using the same configuration except for Block6. There, we use BlockN and combine it with the

| Type        | Model   | HFPset | Set2   | DR   | FPR    | Acc(w) |
|-------------|---------|--------|--------|------|--------|--------|
| Single      | NCC-HFP | Block6 |        | 0.54 | 0.0543 | 0.74   |
| Single      | NCC-HFP | 2Sym   |        | 0.77 | 0.4517 | 0.66   |
| H-Unite     | NCC-HFP | Block6 | 2Sym   | 0.61 | 0.0549 | 0.78   |
| FH-AdaBoost | NCC-HFP | Block6 | 2Sym   | 0.66 | 0.0638 | 0.8    |
| FH-AdaBoost | NCC-HFP | 2Sym   | Block6 | 0.64 | 0.0693 | 0.79   |

Table 5.19: This table shows the potential benefit of using two different HFP sets. Comparing accuracy (Acc(w)), all of the combinations outperform the classifier trained with a single set. The correlation value of both classifiers is 0.208.

| Type        | Model   | HFPset | Set2   | DR   | FPR    | Acc(w) |
|-------------|---------|--------|--------|------|--------|--------|
| Single      | NCC-HFP | BlockN |        | 0.57 | 0.0667 | 0.75   |
| Single      | NCC-HFP | 2Sym   |        | 0.77 | 0.4517 | 0.66   |
| Unite       | NCC-HFP | 2Sym   | BlockN | 0.63 | 0.0722 | 0.78   |
| FH-AdaBoost | NCC-HFP | BlockN | 2Sym   | 0.62 | 0.0816 | 0.77   |
| FH-AdaBoost | NCC-HFP | 2Sym   | BlockN | 0.6  | 0.0903 | 0.75   |

Table 5.20: This table shows the results of two ensembles trained with the HFP sets BlockN and 2Sym as well as three ensembles trained with a combination of the two sets. The correlation value of both ensembles trained with a single HFP set is 0.237.

HFP set 2Sym.

With a correlation value of 0.629, the ensembles in table 5.21 do not differ as much in their results compared to the previous combinations. Moreover, only the combined training of both sets slightly increases the accuracy compared to the single training results. However, the forced hybrid ensembles achieve the best false-positive rate.

### 5.5.3 Results for the Hybrid Alternating Classifiers

#### Samples of Improvement

In this section, we examine the forced hybrid classifiers that alternate different classifier types and not just HFP sets as was the case in the previous section. Figure 5.15 shows the detection and false-positive rate of single and hybrid classifiers,

| Type        | Model   | HFPset | Set2   | DR   | FPR    | Acc(w) |
|-------------|---------|--------|--------|------|--------|--------|
| Single      | NCC-HFP | Block6 |        | 0.54 | 0.0543 | 0.74   |
| Single      | NCC-HFP | BlockN |        | 0.57 | 0.0667 | 0.75   |
| Unite       | NCC-HFP | Block6 | BlockN | 0.57 | 0.0567 | 0.76   |
| FH-AdaBoost | NCC-HFP | Block6 | BlockN | 0.54 | 0.0438 | 0.75   |
| FH-AdaBoost | NCC-HFP | BlockN | Block6 | 0.53 | 0.0523 | 0.74   |

Table 5.21: Both ensembles using the Block6 and BlockN HFP sets for training have a correlation value of 0.629. The hybrid-trained ensembles slightly improve upon the single-trained ensembles.

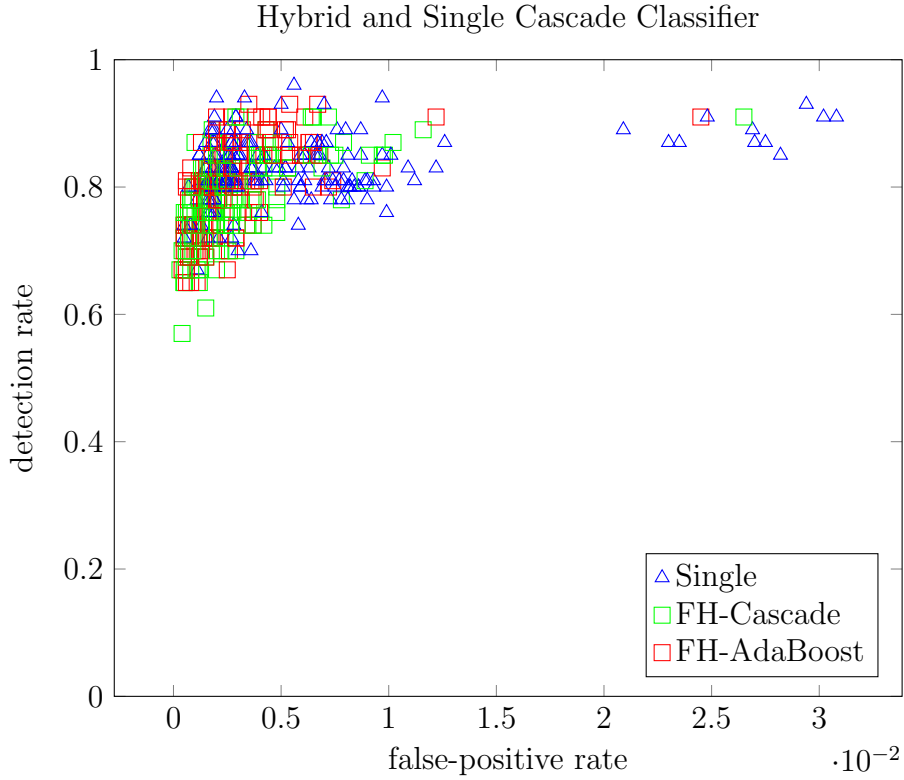


Figure 5.15: Comparing 550 cascade classifiers with their detection and false-positive rate. The blue triangles are the single classifier results, while the green squares represent the results of the hybrid classifiers that alternate the classifier type within the ensemble creation, and the red squares belong to the hybrid classifiers where the classifier type is altered after creating an ensemble node.

| Type        | Model      | HFPset | Model2     | HFPset2 | DR   | FPR    |
|-------------|------------|--------|------------|---------|------|--------|
| Single      | NCC-HaarNN | BlockN |            |         | 0.89 | 0.0038 |
| FH-Cascade  | NCC-HaarNN | BlockN | NCC-HFP    | Block9  | 0.88 | 0.0017 |
| Single      | NCC-HaarNN | BlockN |            |         | 0.86 | 0.0031 |
| FH-AdaBoost | NCC-HFP    | Block9 | NCC-HaarNN | BlockN  | 0.86 | 0.0023 |
| Single      | NCC-HFP    | Block9 |            |         | 0.85 | 0.0023 |
| FH-Cascade  | NCC-HFP    | Block9 | NCC-HaarNN | BlockN  | 0.85 | 0.0018 |
| Single      | NCC-HFP    | Block9 |            |         | 0.82 | 0.0021 |
| FH-AdaBoost | NCC-HaarNN | BlockN | NCC-HFP    | Block9  | 0.85 | 0.0013 |

Table 5.22: Direct comparison of single and hybrid classifiers. We chose two classifiers for the NCC-HaarNN, and BlockN set, and the NCC-HFP Block9 set, compared with four hybrid classifiers that use the same classifier models and sets. The single and hybrid classifiers are trained with asym-AdaBoost and 100 randomly chosen features during each iteration.

which gives us an idea of their performance distribution.

The classifiers in the tables 5.22 and 5.23 are trained using the same training parameters, but with different random sequences of the HFP sets. To compare the cascade classifiers, we take the best classifiers and not an average of results because if we use a classifier within a productive system, we would be using a dedicated classifier and not an average. Therefore, we took two single classifiers and their hybrid counterparts. The results are shown in the tables 5.22 and 5.23 are sorted by detection rate. First, the single classifier, followed by the hybrid classifier. In the first row, the single classifier has the better detection rate, but a poorer false-positive rate. Both of the middle rows show the same detection rate for both classifiers, but a better false-positive rate for the hybrid classifier.

| Type        | Model      | HFPset | Model2     | HFPset2 | DR   | FPR    |
|-------------|------------|--------|------------|---------|------|--------|
| Single      | NCC-HFP    | Block6 |            |         | 0.78 | 7.0E-4 |
| FH-AdaBoost | NCC-HFP    | BlockN | NCC-HFP    | Base    | 0.78 | 5.0E-4 |
| Single      | NCC-HFP    | Block6 |            |         | 0.86 | 0.0013 |
| FH-AdaBoost | NCC-HaarNN | CB4x4  | NCC-HFP    | Base    | 0.86 | 0.0012 |
| Single      | NCC-HFP    | Base   |            |         | 0.92 | 0.0019 |
| FH-AdaBoost | NCC-HFP    | Base   | NCC-HaarNN | TL6-2   | 0.92 | 0.0056 |

Table 5.23: We took some samples that show the tendency of the classifiers.

The final row shows a better performance for the hybrid classifier in terms of both detection and false-positive rate.

The average number of base classifiers is 462 per single classifier, 425 per hybrid *FH-Cascade*, and 336 per hybrid *FH-AdaBoost* classifier. The hybrid classifiers need fewer base classifiers to achieve their results.

### Comparing Best Classifiers Forced to Alternate

In the previous section, we considered samples of improvement. In the following figure 5.16, we show the distribution of the single and hybrid classifiers with a view to their detection and false-positive rates. To figure out how often the single and hybrid classifiers each achieve better results, we count the winner for every detection rate. To compare the single and hybrid classifiers, we sorted them by their detection rate and chose the three best classifiers, i.e. those with the lowest false-positive rate.

The single classifiers clearly have the higher detection rate, while the hybrid offers better results. However, there is no classifier better than all the others in terms of detection and false-positive rate.



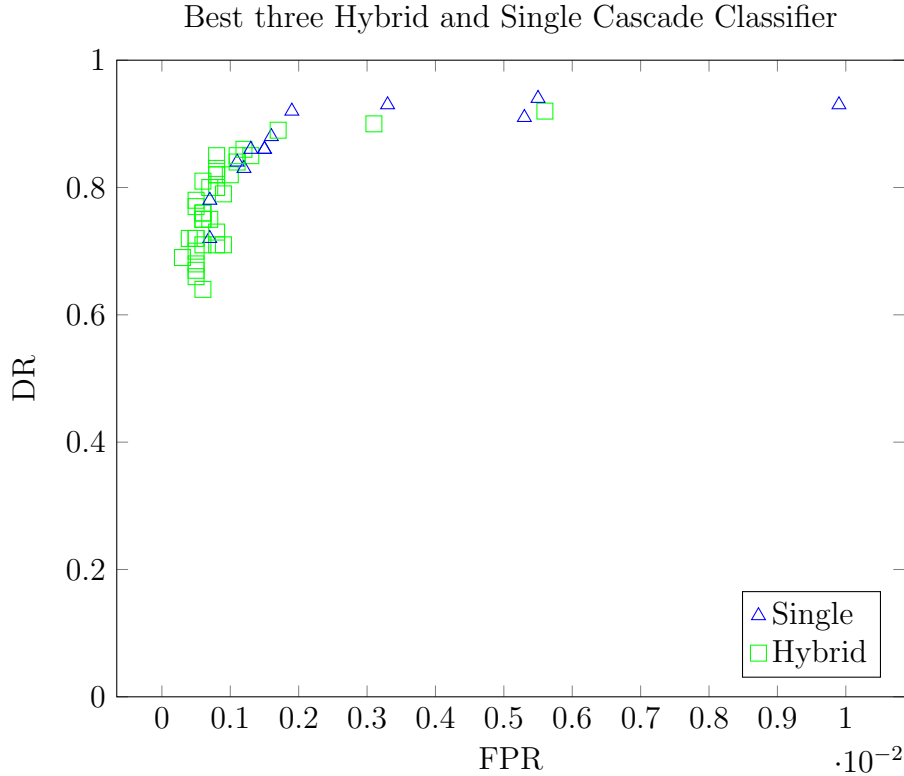


Figure 5.16: We compared 50 cascade classifiers with their detection and false-positive rate. They were chosen by taking the three best classifiers for every detection rate out of 550 classifiers. The only difference in parameters during training was whether they are single or hybrid classifiers and the HFP set.

#### 5.5.4 Discussion and Conclusion

While we finally do not use an average classifier, but the best one to suit our purpose when it comes to detection or false-positive rate, we consider the performance distribution of all classifiers.

Instead of creating an entire set where AdaBoost can freely select one classifier, we have used different sets and have forced AdaBoost to choose from a certain set. We have discovered two different methods to force the algorithm to combine classifier and HFP sets. One method forces alternation during ensemble creation, while the other forces the alternate addition of an ensemble to the cascade classifier.

In the first tables, we compare the single and the forced hybrid classifiers where only the HFP set was alternated. The clearest result would be if any of the hybrid classifiers were to increase the detection rate and decrease the false-positive rate. On the other hand, if the hybrid result is just an average of the single classifiers, there is no benefit to be gained from the forced hybrid architecture. However, the

forced hybrid classifier generally improves the detection or false-positive rate by keeping the other value mostly at a similar level.

However, we have to handle randomness. When visualizing all of the results, we mapped all of the created classifiers as a scatter plot (see figure 5.16). There, we can see the pattern that we confirm in the following tables where we have more single classifiers with a higher detection rate, but also a higher false-positive rate. While we have more hybrid classifiers with a comparatively lower detection rate, they also have a lower false-positive rate.

In the end, the hybrid and alternating architecture is a benefit because we can create better classifiers. Not every hybrid will be better than a single classifier; however, if we train classifiers, it is of more value to train hybrid classifiers, hence there will be more and better classifiers. Therefore, it is more suitable to train hybrid classifiers.

## **5.6 Summary**

In this chapter, we increased diversity by introducing more HFP sets and the forced hybrid architecture. Most sets achieve good results as the only set for training HFP or HaarNN classifiers. As a result, the HFP classifier model creates good classifiers for all sets, while there are some exceptions for the HaarNN classifier model. All classifiers trained with different sets are diverse, i.e. different from one another. Hence, we can increase diversity by separating and introducing more sets.

Further, we introduced the forced hybrid architecture, which forces the training algorithm to alternate the features or classifiers used. Instead of opting for the current best classifier for a single set, the best classifier for separate sets is chosen. The forced hybrid architecture increases the performance of the cascade and ensemble classifiers. In addition, if we combine a good set with a less good set, the final cascade classifier can improve the resulting classifier.

# Chapter 6

## Multi-Class Ability

### 6.1 Experiments Introduction

In this chapter, we examine whether our Haar-Feature-like patches, HFP and HaarNN classifier models can also be used as a multi-class solution. We consider it a prototype-like test. Thus, we do not train cascade classifiers and test them for whole images; instead we take ensemble classifiers and test them using images which already show cropped faces like those used in the training set. We use head-pose estimation to test the multi-class ability of our models because it is related to face detection and the needs of a social robot.

Hopfield Neural Networks can be used for learning multiple patterns as described in equation 2.22 in section 2.5. The best way to learn different patterns is to provide vectors that are orthogonal to each other (see Amit et al. [1]). There, Amit et al. [1] show that the number of different learned patterns is 0.13 multiplied by the number of the HNN's neurons ( $P = 0.13 * N_n$ ).

However, we disregard these constraints and use our HNN for small and fuzzy patterns. The number of neurons of our HNN corresponds to the number of rectangles of the HFP used. Therefore, we mostly have a small number of neurons, which leads, mathematically, to learnable patterns smaller than 1. However, we do not use the HNN as one strong classifier, but as a weak classifier within an ensemble.

#### Head-Pose Estimation

Head-pose estimation can be defined as the ability to recognize the orientation of a face gaze relative to the orientation of a camera. Therefore, head-pose estimation faces similar problems to other computer vision fields in maintaining invariance to

physical image changing factors such as camera distortion, projective geometry or multisource lighting.

In computer science, we can distinguish head-pose estimation at a coarse level and at a finer level. Coarse means only considering the left, frontal, and right gaze directions, disregarding the extent of the gaze point to the right or left side. At a finer level, we want to determine the (precise) angle of the gaze point. Psychological research by Langton et al. [32] defines that the final estimation of gaze direction is a combination of head pose and eyes. As a link to visual gaze estimation, head-pose estimation can be stated as a rough gaze estimation, especially if the eyes are not visible. Murphy-Chutorian and Trivedi [44] note that every gaze estimation that uses an eye tracker should have an underlying head-pose estimator. In human conversation, a head movement is often a gesture, which transports information as pointing to a target, for example. People also look at each other while talking to affirm awareness.

Some approaches solve head-pose estimation with a cascade classifier similar to face detection using a divide and conquer strategy. Wu et al. [77] create a parallel cascade where every cascade contains classifiers that are trained for only one view. Therefore, the whole cascade classifier detects precisely one view or background, while one cascade classifier classifies the input for every view in parallel. Every cascade classifier collects scores for its view, and the highest score determines the result.

A different approach is used by Li et al. [34]. They create a pyramid with three levels of classifiers. The first level is trained to detect every face in all poses. If the test window passes this level, the second level assigns the face to one of three directions: faces that look to the left (-90 to 30 degrees), front (-30 to 30 degrees) or right (30 to 90 degrees). If the test window also passes this level, the third level divides it into even finer angles.

Jones and Viola [30] proposed a detector in two steps. First, a classifier provides a forecast for the face view, followed by an expert cascade for the predicted pose which makes a final estimate. In 2005, Huang et al. [26] and Lin and Lui [37] proposed vector boosting respectively Bhattacharyya boosting (MBHBoost), which are based on similar ideas. Every trained weak classifier produces a vector of results for several categories. With this result vector, the classification task can be passed to different classifiers which then consider every category. Finally, by aggregating the vector outputs, the overall result is achieved.

Torralba et al. [62] proposed the idea of reducing complexity by sharing features and jointly training multiple classifiers. Tu [65] trained a probabilistic boosting

tree. For this, they trained an AdaBoost classifier for each tree node. Every node assigns a probability to split the data into two clusters. For a faster training, Wu et al. [78] first selected features for clustering. In their paper published in 2009, Murphy-Chutorian and Trivedi [44] discuss the pros and cons of different head-pose estimation approaches. They [44] note that using a pixel-based image as the source, there are a series of processes which have to be applied to describe the orientation of a face on a high level.

Similar to face detection, appearance template-based methods are also used for head-pose estimation. These methods calculate matching measurements from the input image to an existing template image with a known head direction. Beymer [2] proposed such a system that chooses the direction to the input value that is closest to the example template. Beymer [2] applies normalized cross-correlation to the template and the input images for a better comparison. The advantage of such methods is that they are easy to implement and do not need negative samples and do not involve searching for other dependencies like facial features. However, there are disadvantages, e.g. they mostly presume that the area of interest is already located and such methods are only capable of calculating discrete pose locations. Murphy and Trivedy [44] consider the most significant disadvantage to be the similarity assumption for different people with the same poses. They argue that the dissimilarity between different people is high, and it is difficult to arrive at a link between poses of different individuals. To overcome this disadvantage, the images can be preprocessed, for example with Laplacian-of-Gaussian filters [19].

## Datasets Used

To train and test the head-pose estimation ensembles, we use the database provided by Gourier et al. [20].

We use three images for training purposes; one for every pose by one person, as illustrated in figure 6.1. These images were cropped to only include the face area with random changes of width, height, and rotation. Then, we created some additional images by applying gamma correction which leads to 35 images per pose and 105 positive samples overall. We call this training set “Three View One Person” (3VOP). As negative images, we created an own set of samples from the internet, containing 30,000 randomly cropped images of landscapes, offices, and apartments. We divided these images into 9,000 images for testing and 21,000 images for training. As a second set of negative training images, we selected 500 of the 21,000. We converted all of the images to gray images and fixed them to a square of 24x24 pixels.



Figure 6.1: Samples of the training set. Three source images (left). Randomly created additional images with different width, height, rotation and gamma corrections (middle). Randomly cropped negative samples of landscape, office and apartment images taken from the internet.

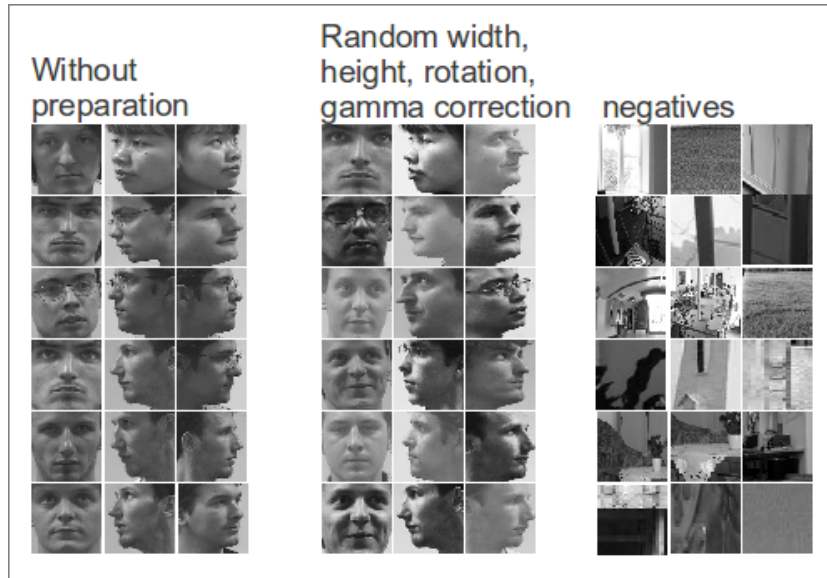


Figure 6.2: Samples of the test set. Cropped images without any changes (left). Randomly created additional images with different width, height, rotation and gamma corrections (middle). Randomly cropped negative samples of landscape, offices and apartment images taken from the internet.

We created the test set analogously. The tests were performed using two test sets. The basis for the sets were 208 images of 14 people including 64 frontal views, 79 left views, and 65 right views (figure 6.2). The first test set consists of these images transformed to a fixed width and height (test set 1). To simulate different lighting conditions, the second set contains additional images created by random rotation (50) and gamma correction (test set 2). Then, the second test set contains 12,480 positive (over three classes) and 9,000 negative samples.

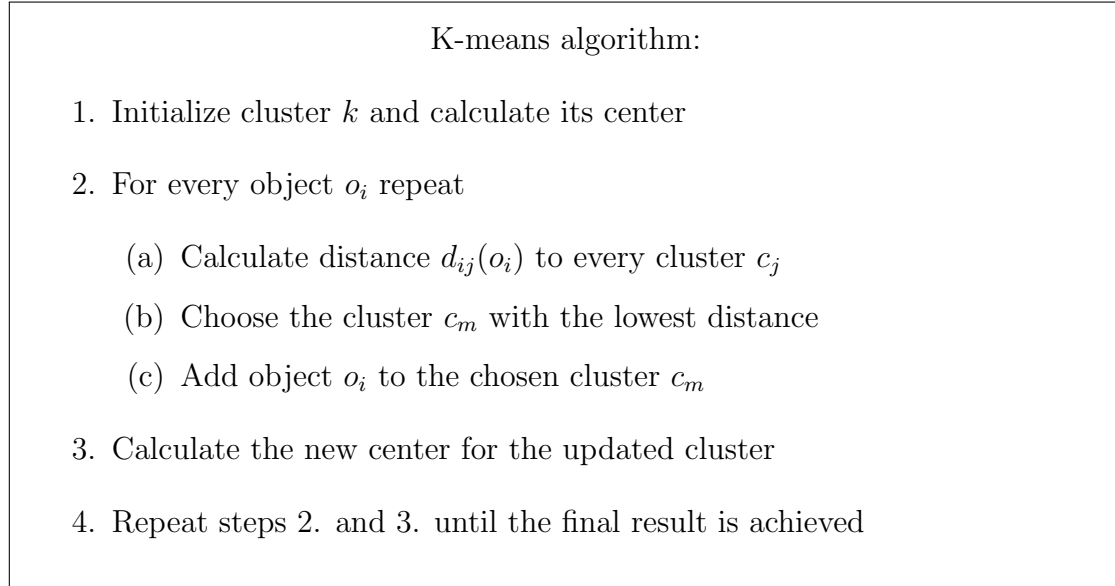


Figure 6.3: Pseudo-code K-means algorithm to group objects in a cluster by their similarity.

## 6.2 Methods

The HFP and HaarNN classifiers used so far distinguish between faces and background. To classify a head pose, we want to distinguish between the classes left, frontal, right, and background. As one approach, we use the Bayes-like method for multi-class recognition, which we introduced in section 3.2.3. The Bayes-like method uses the stable state of the HaarNN, but we also want to examine if we can use the HFP classifier as a multi-class solution. Therefore, we also introduce a K-means-based classifier which can use the HFP and the HaarNN classifier as a base classifier.

### K-Means

K-means is a clustering algorithm that aims to sort objects into groups which have the highest similarity. There are several clustering methods as surveyed by Ka-Chun in [76]. However, while we are not looking to optimize clustering, we use the basic method by Hartigan [22].

Hartigan's [22] K-means algorithm divides objects by minimizing the Euclidean distance between these objects as described in 6.3. Thereby, one object is assigned to a given cluster if the distance is less than a given threshold. Otherwise, if the distance is greater than this threshold for every cluster, a new cluster is created.

Here, mostly, the object to be grouped is a vector of numbers, and the center of the cluster is the average of all objects inside this cluster. The final result is achieved if the process does no longer change the assignments of the objects to the groups. However, there are also some weaker goals for finishing the process. We can, for instance, limit repeating the algorithm to a set number of iterations or use a threshold for objects that move between groups.

### K-Means as Base Classifier

We use both models, the HFP classifier and the HaarNN classifier, combined with K-means as base classifiers, and call them KM-HFP and KM-HaarNN. Basically, we measure the probability of the different classes by counting occurrences within the training set and assigning the result to a vector which belongs to a K-means group as depicted in figure 6.4. The vector that we use to calculate the group distance is the pattern of the HFP values directly and the stable state of the HaarNN (see figure 6.5).

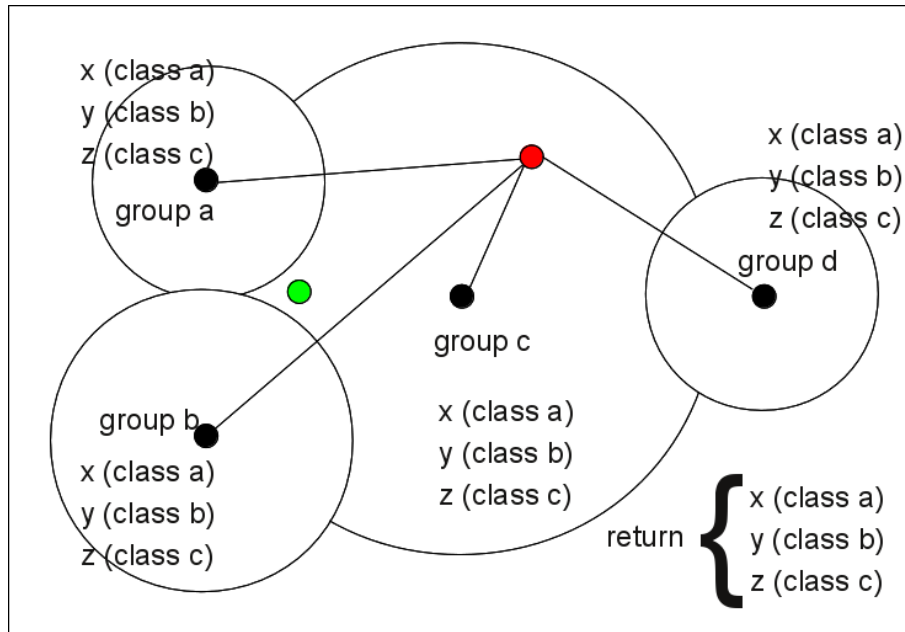


Figure 6.4: Instead of using a threshold to distinguish faces and background, head-pose estimation involves the KM-HFP and KM-HaarNN classifiers which use probabilities to create the four different outcomes: left, frontal, right, and background.

We train the KM-HFP and KM-HaarNN classifiers in two steps: First, we iterate over the training set and calculate the average of the HFP values, and learn



the HNN weights by Hebb-learning for the HFP classifier and the HaarNN classifier respectively, which is the same as learning the binary HFP and HaarNN classifiers. In a second iteration, we execute the K-means algorithm to establish the groups and count how often the given classes are allocated to the specified groups (as is the case with the Bayes-like method in section 3.2.3).

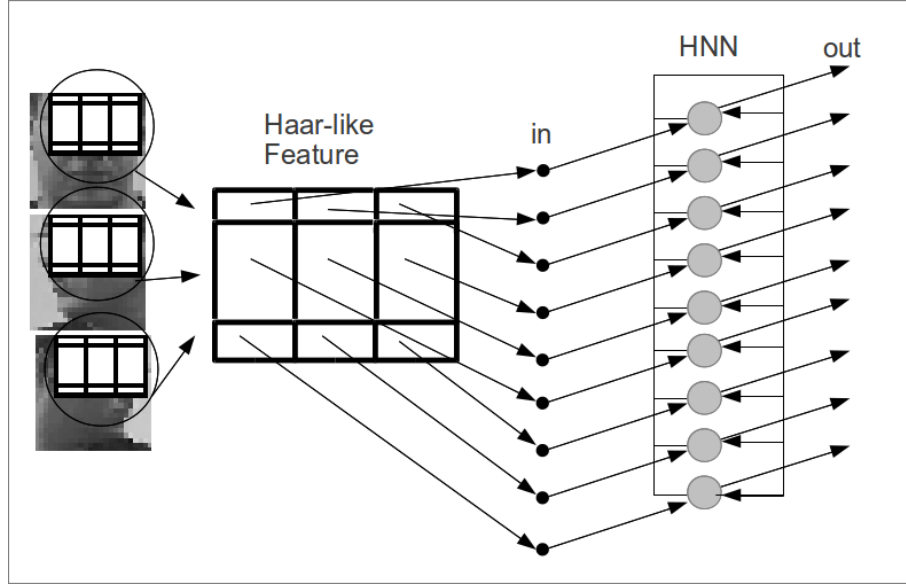


Figure 6.5: The execution process of the HaarNN for multi-class solutions is basically the same as for the binary solution. However, instead of one positive input, we use three different positive classes for training.

### Training the Ensemble

To train the ensembles, we also use AdaBoost, but with changes proposed by Zhu et al. [94]. Two concepts differ from the original algorithm. In contrast to binary classification, where we have one positive or negative result, multi-class classification returns a result vector. Originally, classification result is calculated as follows:

$$h(x) = \begin{cases} 1 & \sum_{t=1}^T \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^T \alpha_t \\ 0 & \text{otherwise} \end{cases} \quad (6.1)$$

There,  $\alpha_t$  is the weight of classifier  $t$  and  $h_t(x)$  its output for input image  $x$ . The first change is to calculate the sum of vectors, containing the probabilities of all classes (equation 6.2).

$$v(x) = \sum_{t=1}^T \alpha_t h_t(x) \quad (6.2)$$

Here,  $v(x)$  is the vector with the sum of all probabilities for all classes. The final result is the class with the highest value (equation 6.3).

$$h(x) = \max(v(x)) \quad (6.3)$$

The second change is to the training process. In the original work by Viola and Jones [71], the weight of a single weak classifier ( $\alpha$ ) is calculated as follows (equation 6.4):

$$\alpha = \log\left(\frac{1 - \epsilon}{\epsilon}\right) \quad (6.4)$$

We adapt this calculation according to Zhu et. al. [94]:

$$\alpha = \log\left(\frac{1 - \epsilon}{\epsilon}\right) + \log(n_c - 1) \quad (6.5)$$

Parameter  $n_c$  is the number of different classes excluding the negative class. Zhu et al. [94] re-weight the training set by increasing the weights of the misclassified samples using equation 6.6.

$$w_{i+1} = w_i \exp(\alpha) \quad (6.6)$$

We calculate the error of a base classifier by selecting the class with the highest value within the probability vector.

## 6.3 Findings

Tables 6.1, 6.2 and 6.3 show the results of the trained ensemble classifiers. The classifiers of the first two tables are HaarNN ensembles, while the ensembles of the third table are HFP ensembles. All of the ensembles are trained with up to 50 base classifiers, using 100 features for each training iteration. Further, they use the training set 3VOP, but only 500 negative samples. To test the ensembles, we use test set 1 which contains 208 positive and 9,000 negative samples, as described in section 6.1, and also use test set 2 which contains 12,480 positive samples. We tested several HFP sets and parameters.

Table 6.1 shows the results of the KM-HaarNN ensembles. The KM-HaarNN ensembles have at least a 75% detection rate and a false-positive rate of about 8% to 22%. In table 6.2 we show similar BP-HaarNN ensembles. They have detection rates ranging from 75% to 98% for test set 1. There, we also show the results of test set 2 which contains 12,480 positive samples. Both HaarNN models correctly classify about 80% to 95% of the negative samples along with most of the three different classes.

| Model     | HFPset | DR   | FPR   |
|-----------|--------|------|-------|
| KM-HaarNN | Base   | 0.81 | 0.146 |
| KM-HaarNN | BlockN | 0.94 | 0.127 |
| KM-HaarNN | BlockN | 0.92 | 0.086 |
| KM-HaarNN | Block9 | 0.93 | 0.185 |
| KM-HaarNN | Block9 | 0.78 | 0.106 |
| KM-HaarNN | 2Sym   | 0.88 | 0.225 |
| KM-HaarNN | 2Sym   | 0.75 | 0.152 |
| KM-HaarNN | TL6Sym | 0.87 | 0.105 |
| KM-HaarNN | Block6 | 0.81 | 0.148 |

Table 6.1: Results of the KM-HaarNN ensembles, trained with up to 50 base classifiers.

| Model     | HFPset | DR   | DR (set 2) | FPR   |
|-----------|--------|------|------------|-------|
| BP-HaarNN | BlockN | 0.98 | 0.93       | 0.107 |
| BP-HaarNN | BlockN | 0.9  | 0.84       | 0.067 |
| BP-HaarNN | Block9 | 0.91 | 0.83       | 0.114 |
| BP-HaarNN | Block9 | 0.84 | 0.78       | 0.071 |
| BP-HaarNN | 2Sym   | 0.78 | 0.71       | 0.125 |
| BP-HaarNN | CB4x4  | 0.75 | 0.64       | 0.104 |

Table 6.2: Results of the BP-HaarNN ensembles, trained with up to 50 base classifiers.

We show the results of the KM-HFP ensembles in table 6.3. The KM-HFP ensembles all have an almost 100% detection rate but incorrectly classify about half of the negatives. We repeated the training for KM-HFP ensembles using different parameters. The result was the same and similar to the results in table 6.3. We did not obtain results with an acceptable false-positive rate.

## 6.4 Discussion and Conclusion

The question of this chapter is whether our model is able to solve multi-class problems. Considering the results in tables 6.1 and 6.2, we conclude that our HaarNN

| Model  | HFPset | DR   | FPR   |
|--------|--------|------|-------|
| KM-HFP | BlockN | 0.99 | 0.458 |
| KM-HFP | Block9 | 0.99 | 0.506 |
| KM-HFP | 2Sym   | 1.0  | 0.529 |

Table 6.3: Results of the KM-HFP ensembles, trained with up to 50 base classifiers. The KM-HFM ensembles achieve high detection rates, but incorrectly classify about half of all negative samples.

model can be used as a multi-class approach. Although the results can be improved upon, the ensembles are trained in a straightforward way and tested without any special optimizations. The multi-class ensembles achieve comparable results to the binary ensembles from other sections, although they have to distinguish three classes on top of the negatives. As for the binary classification, we can use all different HFP sets.

The results of the KM-HFP model in table 6.3 are very different to results for the HaarNN model. Although we trained several KM-HFP ensemble classifiers using different parameters, none of the KM-HFP ensembles create a feasible false-positive rate. They clearly distinguish the three different classes. Compared to the KM-HFP ensembles, we can see the advantage of the HNN within the HaarNN ensemble. Learning the pattern and executing the HNN helps the K-means algorithm to distinguish between the various groups.

However, this is a proof of concept as to whether we can use our features and models as a multi-class approach. Based on our results, we argue that our HaarNN classifier model is able to solve multi-class problems. The HFP classifier model, however, did not show promising results.

# Chapter 7

## Thesis Summary and Conclusion

### 7.1 Thesis Summary

We presented a novel approach using simple features and classifiers for hybrid ensemble and cascade learning. The classifier models, features and hybrid architectures aimed to increase diversity to become a strong classifier. We built two classifier models, the HFP classifier and the HaarNN classifier, which use simple rectangle pixel sums as features. The HaarNN classifier works much in the same way as the HFP classifier, except that the HaarNN classifier uses a Hopfield Neural Network for pattern learning and as an input filter. We combined our classifiers and simple threshold classifiers to create a hybrid architecture. There, we combine the different classifiers into a single set of base classifiers, which form the source of the training algorithm. Although both classifier models are only distinguished by the additional HNN, they deliver different results and therefore increase diversity. Further, we increase diversity through the forced hybrid architecture which forces the learning algorithm to choose from different sets of base classifiers. Finally, we test our approach as a multi-class classifier, which works well using HaarNN ensembles.

#### Hybrid Approach

In chapter 3, we introduced our Haar-Feature-like patch and diversity approach. We described the features used along with the classifiers. In the results section, we showed that our methods could increase the accuracy of a (Viola/Jones) threshold cascade classifier within a hybrid architecture.

## Model Characteristics

In chapter 4, we examined the basic characteristics of both classifier models and their differences. While the HFP and HaarNN classifiers are very similar in terms of their methods, they choose different features during training, which causes different classification results. The HaarNN classifier in particular can increase diversity through its inner HNN parameters.

## Increasing Diversity

In chapter 5, we further increased diversity by separating features into different sets based on their geometric appearance and by creating new HFP sets. In general, all of the sets can be used to create proper cascade classifiers. However, every set is different from the next one. We use this diversity in the forced hybrid architecture where we change the ensemble and cascade creation algorithm by forcing it to alternate between different sets and base model types instead of choosing the best of a bigger set that contains everything.

## Multi-class Ability

In chapter 6, we extended our experiments to the field of head-pose estimation and examined whether we can use our classifier models for multi-class classification. We trained our models to distinguish between left, front, right face gaze direction, and background. There, we showed that the HaarNN classifier also works for multi-class problems.

## 7.2 Future Work

The focus of this work was to increase diversity using our models. There are other ways to create diversity which we have not used. We could introduce diversity within the training images so as to use different sets of training sample images, e.g. the bagging approach. Another option is to change the width and height of the training samples. Training images with a higher resolution draws different appearances. The same features are most likely to be different if they are applied to the same image but in a different dimension, which can further increase the diversity and, hopefully, the accuracy of the resulting ensemble classifier. We can use all these variations in training images by applying the forced hybrid architecture to further increase diversity.

Our features have the drawback that they are not very flexible. To overcome this drawback, we can train our classifiers with facial features like the mouth, nose and eyes, and then use a flexible template on top of them.

While we achieved promising results with almost every kind of set and with purely random features, it could be an interesting approach to use a genetic algorithm to shape the features by mutation instead of creating them randomly. Also, it could be seen as an opportunity to combine our none-trained hybrid solutions with learning. We have shown that we can create a benefit by merging two cascade classifiers, and we could train independent cascade classifiers and merge them in between as an additional creation step.

The aggregated image creates an image out of the learned Haar-Feature-like patches learned from faces. Indeed, the appearances of most of the aggregated images clearly look like faces. Instead of training every single classifier to use a value to determine a hit, we can use the aggregated image as an ensemble. Then, every HaarNN used as learnable filter draws part of this image and, finally, this image will be compared to the overall aggregated image. There, we would use the ability of the HNN. If the Haar-Feature-like patch of a frame to be classified is not similar to the learned HFP, the HNN converges to a different stable state, thus changing the aggregated image. Then, it is an ensemble, but a very different ensemble compared to those we have shown in this thesis because it is not a vote of the single classifiers, but a painting.

## 7.3 Conclusion

We recapitulate and focus on the three main questions in order to arrive at a conclusion. Could we create a strong classifier using our approach by focusing on increased diversity instead of more accurate and sophisticated features or classifiers?

- Could we increase diversity using our approach?
- Could we create a benefit out of the diversity?
  - Are all of the sets useful?
  - Are the additional HNN (HaarNN classifiers) useful?
- Could we create a strong classifier?

## **Diversity and its Benefit**

Our approach provides several opportunities for increased diversity. The different HFP sets increase diversity, as does the HaarNN classifier with a Hopfield Neural Net.

Both classifier models provide good results with almost all of the HFP sets. Some sets produce better results than others. Creating a raw ranking, we count the Base, Block6, and BlockN for the HaarNN classifier as well as Block9 and TL6 for the HFP classifier. Considering only single classifiers, we can conclude that we do not use the other sets. However, using the single classifiers in a hybrid approach, all the other sets become useful because they can increase the results of the single classifiers. Overall, the HFP classifiers mostly provide better results, and achieve them with fewer base classifiers. However, the HaarNN classifiers create higher diversity on their own and provide a benefit for the hybrid classifiers. Further, it is the HaarNN classifier that is also useful for multi-class problems.

## **Could we create a strong classifier?**

The leading question of this work is can we arrive at a strong classifier by developing features, classifiers, and architectures which focus on diversity? The results of our models show that we have developed feasible models which benefit diversity and also achieve feasible results. However, our models do not reach state of the art in terms of accuracy. Nevertheless, we think that with additional research, we could further improve upon our results.

## **Final Summary**

Finally, we built a novel classifier approach which creates a high diversity zoo of features and classifiers. It increases diversity and offers promising results, but requires further optimization to achieve state of the art results.



# Appendix A

## Appendix

### A.1 Glossary of Acronyms and Abbreviations

**HNN** Hopfield Neural Network

**HFP** Haar-Feature-like Patch

**ED-HFP** Classifier which uses an HFP and the Euclidean distance

**NCC-HFP** Classifier which uses an HFP and Normalized Cross-correlation

**KM-HFP** Classifier which uses an HFP and K-means

**ED-HaarNN** Classifier including HNN which uses an HFP and the Euclidean distance

**NCC-HaarNN** Classifier including HNN which uses an HFP and Normalized Cross-correlation

**BP-HaarNN** Classifier including HNN which uses an HFP and Bayes-like probability

**KM-HaarNN** Classifier including HNN which uses an HFP and K-means



**FH-AdaBoost** Forced Hybrid Architecture applied to the ensemble classifier creation

**FH-Cascade** Forced Hybrid Architecture applied to the cascade classifier creation



## **A.2 Classification Samples**

Here, we show some sample images of classification results of several different classifier models, features and architectures.

| Model  | HFPset  | DR  | FPR     | NF  | Sum-BC |
|--|---------|-----|---------|---|--------|
| NCC-HFP  | RandomN | 0.7 | 5.73E-5 | 100   | 607.0  |
|  |         |     |         |  |        |

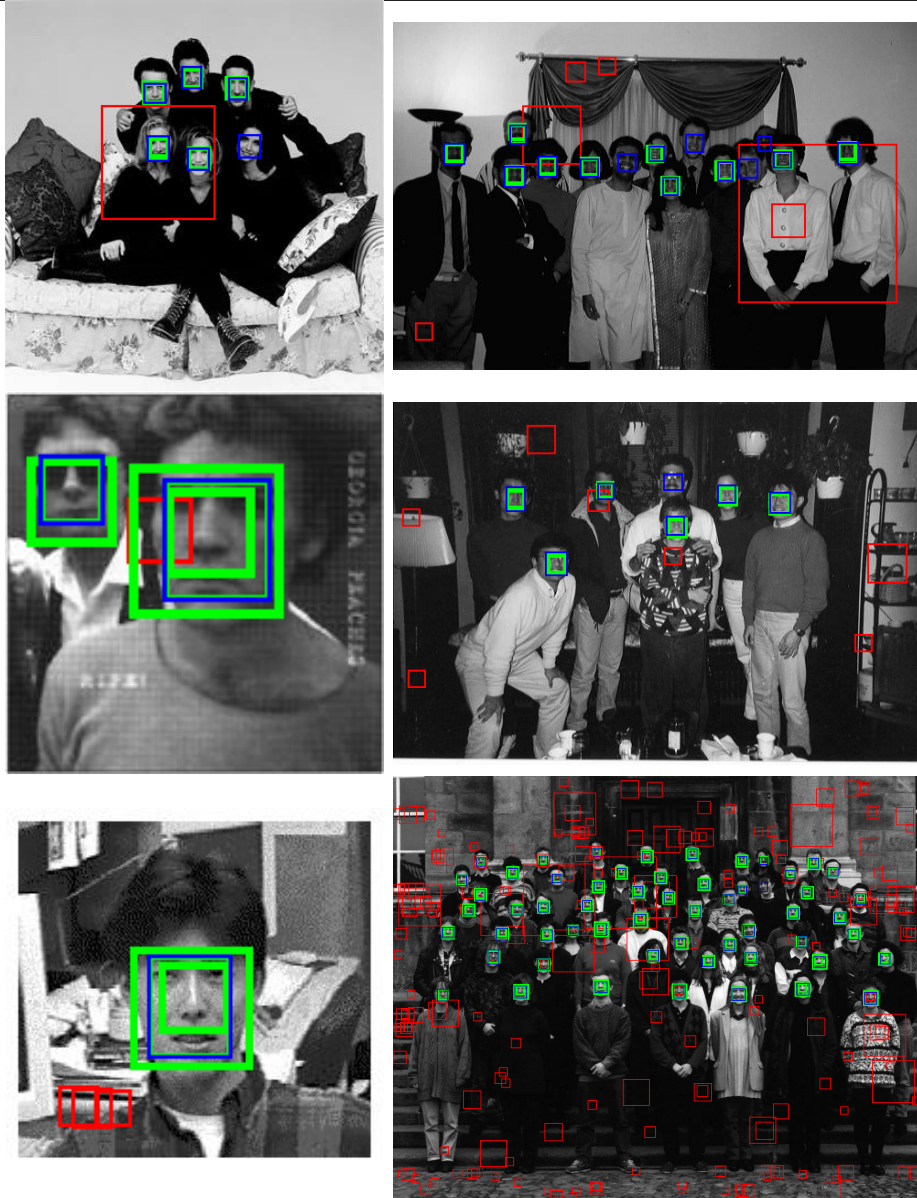
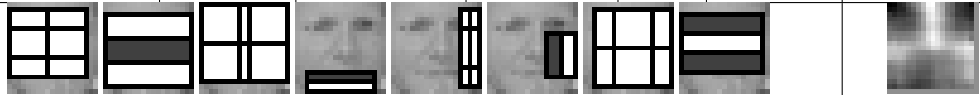


Table A.1: Single cascade classifier using as base classifier NCC-HFP, the HFP set RandomN, trained with 100 features per iteration.

| Model  | HFPset | Model2    | HFPset2 | DR   | FPR     | Sum-BC |
|--|--------|-----------|---------|------|---------|--------|
| NCC-HaarNN   | Block6 | Threshold | Base    | 0.62 | 2.31E-5 | 608.0  |
|  |        |           |         |      |         |        |

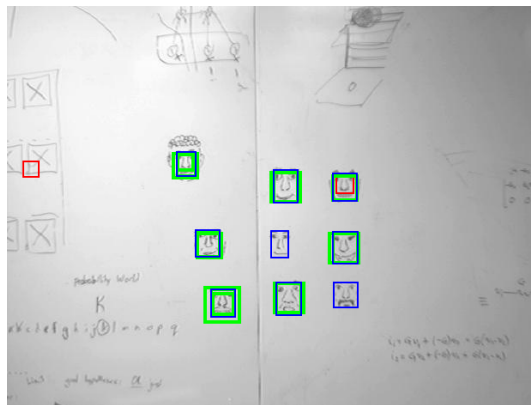
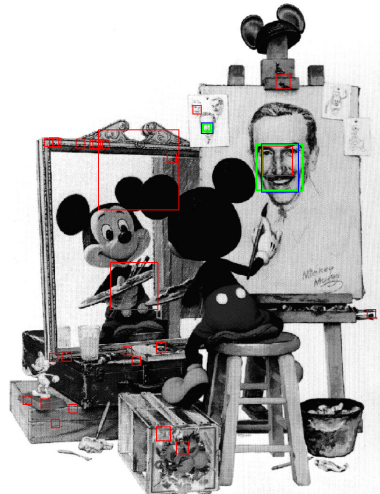
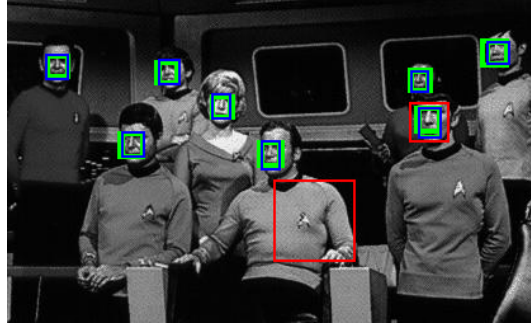
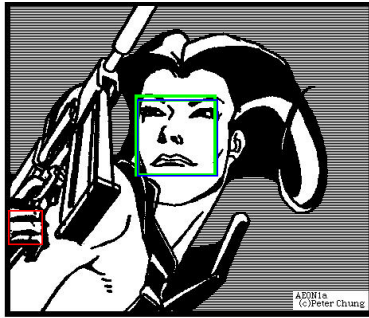
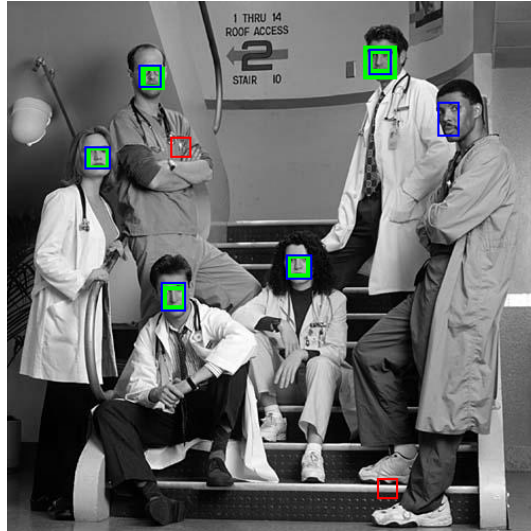
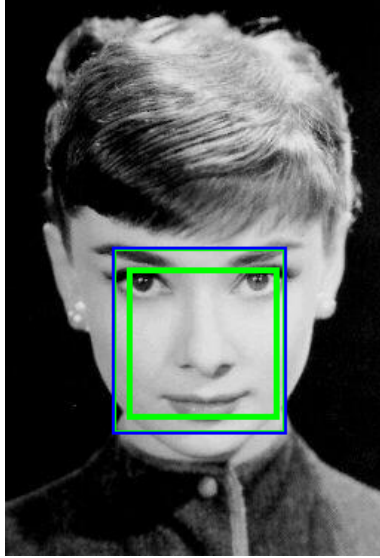









Table A.2: Forced Hybrid cascade classifier (FH-AdaBoost) using as base classifiers NCC-HaarNN and Threshold, the HFP sets Block6 and Base, trained with 100 features per iteration.

| Model   | HFPset  | Model2  | HFPset2   | DR  | FPR   | Sum-BC  |
|---|---|---|---|---|---|---|
| NCC-HFP   | Base  | Threshold   | Base  | 0.68  | 2.03E-5   | 404.0   |
|  |  |  |  |  |  |  |

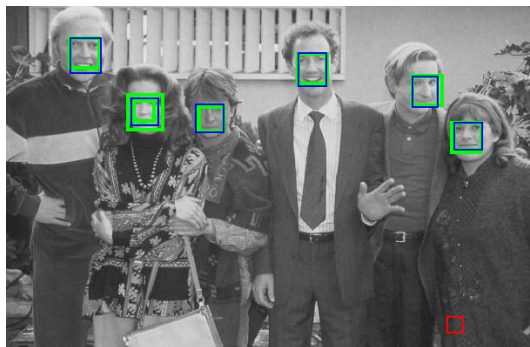
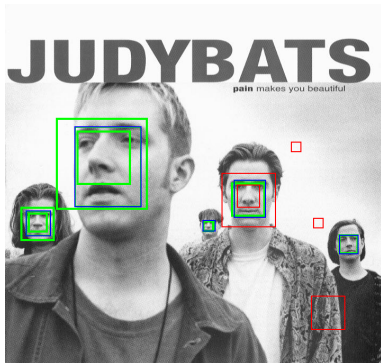
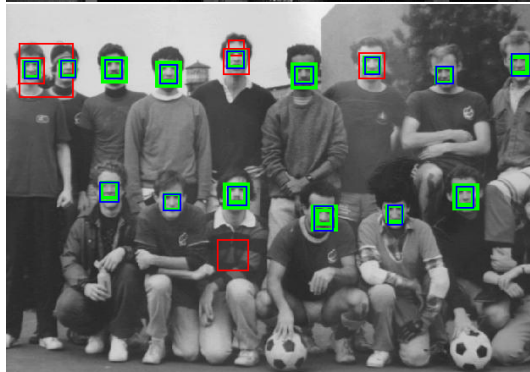
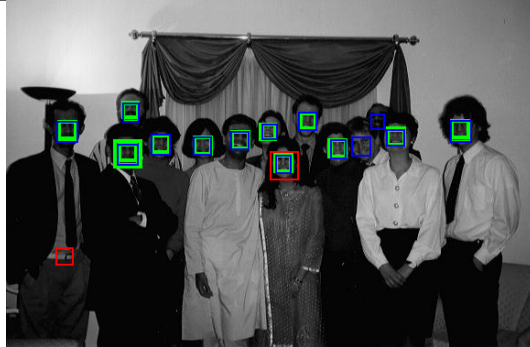
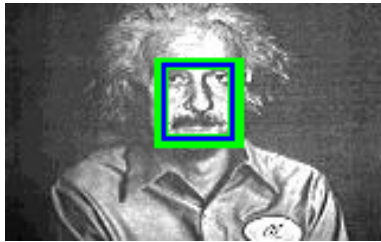
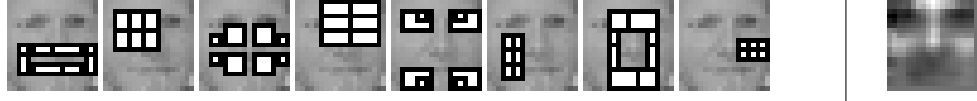


Table A.3: Hybrid cascade classifier (Unite) using as base classifiers NCC-HFP and Threshold, the HFP set Base, trained with 100 features per iteration.



| Model  | HFPset | Model2  | HFPset2 | DR   | FPR     | Sum-BC |
|--|--------|---------|---------|------|---------|--------|
| NCC-HaarNN   | 2Sym   | NCC-HFP | Block6  | 0.68 | 5.98E-5 | 663.0  |
|  |        |         |         |      |         |        |

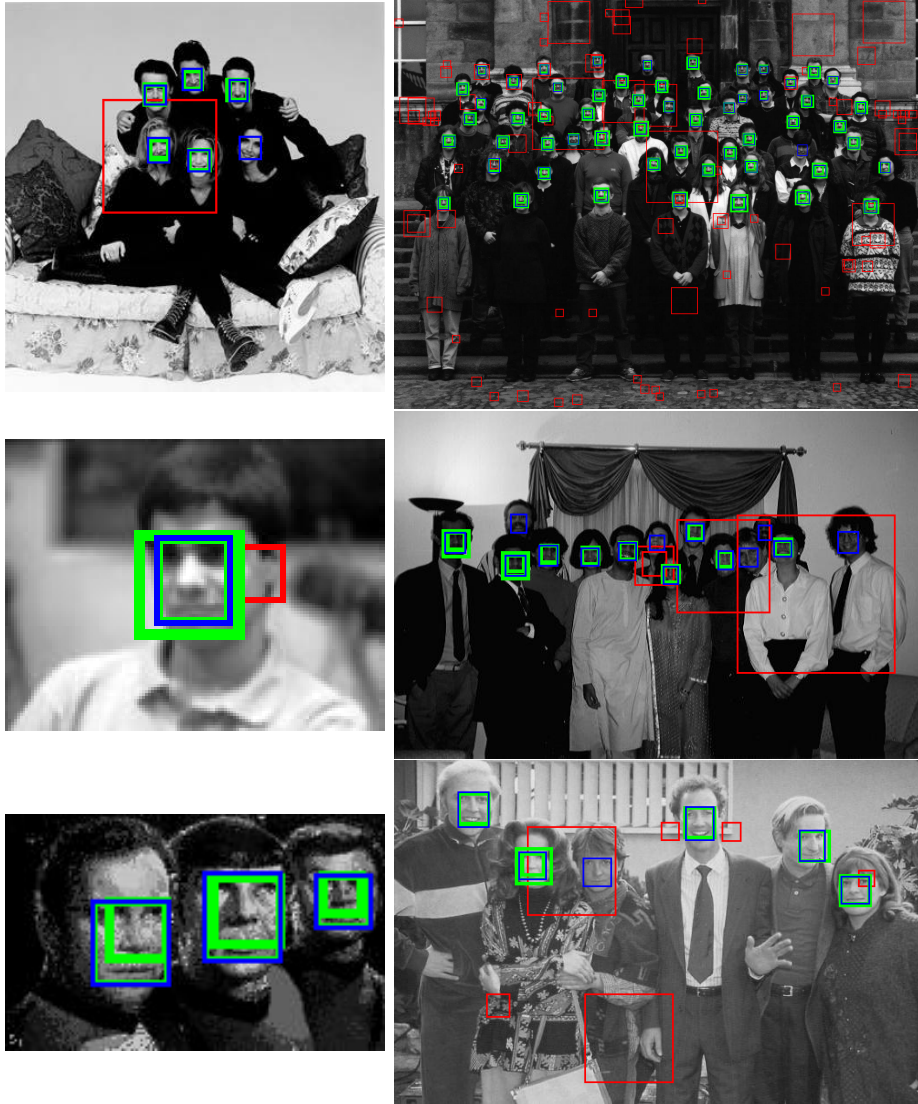









Table A.4: Forced Hybrid cascade classifier (FH-AdaBoost) using as base classifiers NCC-HaarNN and NCC-HFP, the HFP sets 2Sym and Block6, trained with 100 features per iteration.

| Model   | HFPset  | Model2  | HFPset2   | DR  | FPR   | Sum-BC   |
|---|---|---|---|---|---|--|
| NCC-HFP   | BlockN  | NCC-HFP   | Base  | 0.74  | 8.67E-5   | 373.0  |
|  |  |  |  |  |  |  |

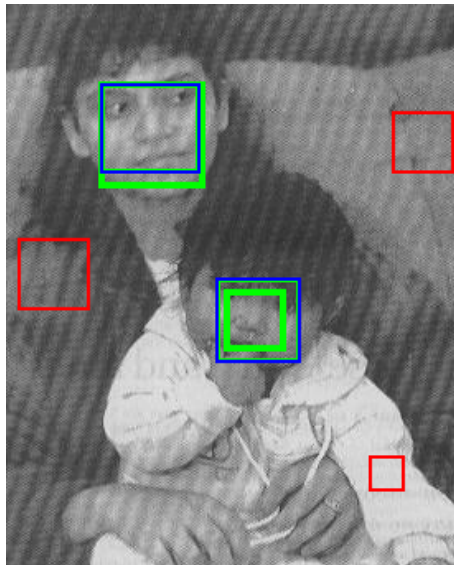
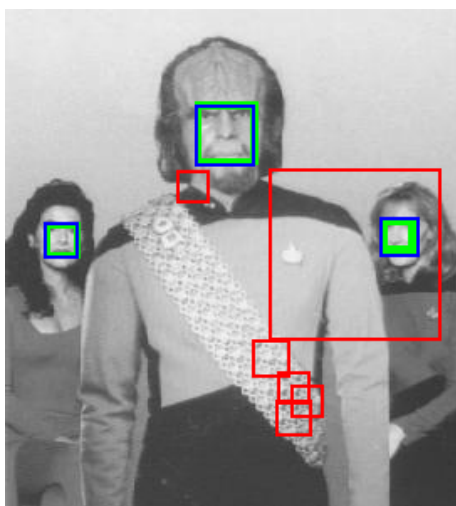
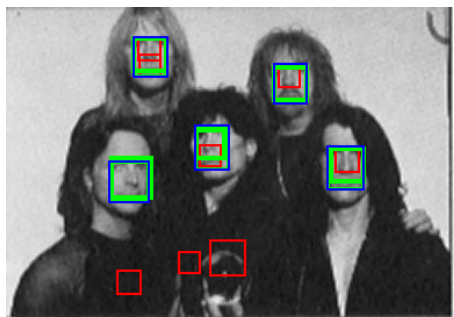
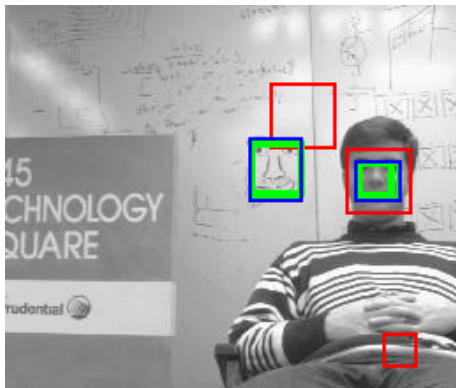
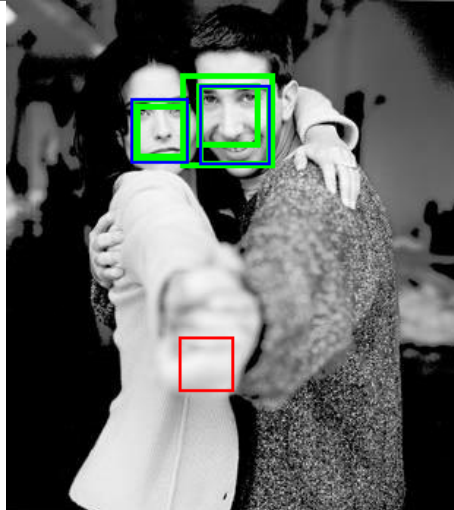
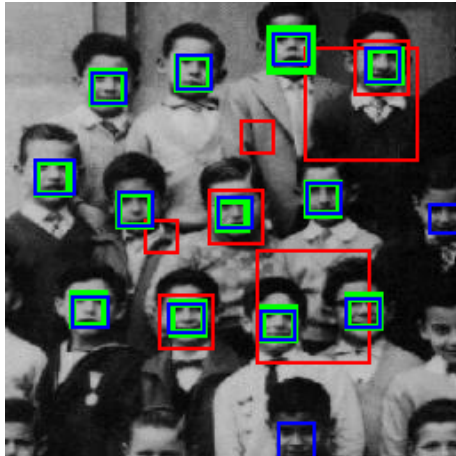


Table A.5: Forced Hybrid cascade classifier (FH-AdaBoost) using as base classifier NCC-HFP, the HFP sets BlockN and Base, trained with 100 features per iteration.



## A.3 Publications

The following publications formed part of this research:

- Nils Meins, Sven Magg, and Stefan Wermter. Neural Hopfield-ensemble for multi-class head pose detection. In *Neural Networks (IJCNN), The 2013 International Joint Conference on*, pages 1–8. IEEE, 2013.
- Nils Meins, Doreen Jirak, Cornelius Weber, and Stefan Wermter. Adaboost and Hopfield neural networks on different image representations for robust face detection. In *Hybrid Intelligent Systems (HIS), 12th International Conference on*, pages 531–536. IEEE, 2012.
- Nils Meins, Stefan Wermter, and Cornelius Weber. Hybrid ensembles using Hopfield neural networks and haar-like features for face detection. In *International Conference on Artificial Neural Networks*, pages 403–410. Springer, 2012.
- Johnson D. O., Cuijpers R. H., Juola J. F., Torta E., Simonov M., Frisiello A., Bazzani M., Yan W., Weber C., Wermter S., Meins N., Oberzaucher J., Panek P., Edelmayer G., Mayer P., and Beck C. Socially Assistive Robots: A Comprehensive Approach to Extending Independent Living. *International Journal of Social Robotics*, November, pages 1–17, 2013.
- Yan W., Torta E., van der Pol D., Meins N., Weber C., Cuijpers R. H., and Wermter S. *Robotic Vision: Technologies for Machine Learning and Vision Applications*, chapter Learning Robot Vision for Assisted Living, pages 257–280. IGI Global, 2012a. doi: 10.4018/978-1-4666-2672-0.ch015.

# Bibliography

- [1] Shun-Ichi Amari and Kenjiro Maginu. Statistical neurodynamics of associative memory. *Neural Networks*, 1(1):63–73, 1988.
- [2] David James Beymer. Face recognition under varying pose. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, pages 756–761. IEEE, 1994.
- [3] MIT Center For Biological and Computation Learning. Cbcl face database #1.
- [4] Lubomir Bourdev and Jonathan Brandt. Robust object detection via soft cascade. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 236–243. IEEE, 2005.
- [5] L Breiman. “bagging predictors” technical report. *UC Berkeley*, 1994.
- [6] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, August 1996.
- [7] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.
- [8] S Charles Brubaker, Jianxin Wu, Jie Sun, Matthew D Mullin, and James M Rehg. On the design of cascades of boosted ensembles for face detection. *International Journal of Computer Vision*, 77(1-3):65–86, 2008.
- [9] J Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, June 1986.
- [10] Franklin C. Crow. Summed-area tables for texture mapping. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '84, pages 207–212, New York, NY, USA, 1984. ACM.

- 
- [11] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893. IEEE, 2005.
  - [12] Thomas G Dietterich et al. Ensemble methods in machine learning. *Multiple classifier systems*, 1857:1–15, 2000.
  - [13] Bradley Efron and Robert J Tibshirani. *An introduction to the bootstrap*. CRC press, 1994.
  - [14] Yoav Freund. Boosting a weak learning algorithm by majority. In *COLT*, volume 90, pages 202–216, 1990.
  - [15] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory*, pages 23–37. Springer, 1995.
  - [16] Yoav Freund, Robert E Schapire, et al. Experiments with a new boosting algorithm. In *Icml*, volume 96, pages 148–156. Bari, Italy, 1996.
  - [17] Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics*, 28(2):337–407, 2000.
  - [18] Bernhard Froba and Andreas Ernst. Face detection with the modified census transform. In *Automatic Face and Gesture Recognition, 2004. Proceedings. Sixth IEEE International Conference on*, pages 91–96. IEEE, 2004.
  - [19] Rafael C Gonzalez and E Richard. Woods, digital image processing, third edition. ed: *Prentice Hall Press, ISBN 0-201-18075-8*, 2007.
  - [20] Nicolas Gourier, Daniela Hall, and James L Crowley. Estimating face orientation from robust detection of salient facial structures. In *FG Net Workshop on Visual Observation of Deictic Gestures*, volume 6, 2004.
  - [21] Feng Han, Ying Shan, Harpreet S Sawhney, and Rakesh Kumar. Discovering class specific composite features through discriminative sampling with swendsen-wang cut. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.
  - [22] J. A. Hartigan and M. A. Wong. A K-means clustering algorithm. *Applied Statistics*, 28:100–108, 1979.

- [23] B. Heisele, T. Poggio, and M. Pontil. Face detection in still gray images. A.I. memo 1687, Center for Biological and Computational Learning, MIT, Cambridge, MA, 2000.
- [24] Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE transactions on pattern analysis and machine intelligence*, 20(8):832–844, 1998.
- [25] J.J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. 1982.
- [26] Chang Huang, Haizhou Ai, Yuan Li, and Shihong Lao. Vector boosting for rotation invariant multi-view face detection. In *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, volume 1, pages 446–453. IEEE, 2005.
- [27] Chang Huang, Haizhou Ai, Yuan Li, and Shihong Lao. Learning sparse features in granular space for multi-view face detection. In *7th International Conference on Automatic Face and Gesture Recognition (FGR06)*, pages 401–406. IEEE, 2006.
- [28] Jun-Su Jang and Jong-Hwan Kim. Fast and robust face detection using evolutionary pruning. *IEEE Transactions on Evolutionary Computation*, 12(5):562–571, 2008.
- [29] Hongliang Jin, Qingshan Liu, Hanqing Lu, and Xiaofeng Tong. Face detection using improved lbp under bayesian framework. In *Image and Graphics (ICIG'04), Third International Conference on*, pages 306–309. IEEE, 2004.
- [30] Michael Jones and Paul Viola. Fast multi-view face detection. *Mitsubishi Electric Research Lab TR-20003-96*, 3:14, 2003.
- [31] Ludmila I. Kuncheva. *Combining Pattern Classifiers*. John Wiley and Sons, Inc., Hoboken, New Jersey, 2014.
- [32] Stephen RH Langton, Helen Honeyman, and Emma Tessler. The influence of head contour and nose angle on the perception of eye-gaze direction. *Perception & psychophysics*, 66(5):752–771, 2004.
- [33] Kobi Levi and Yair Weiss. Learning object detection from a small number of examples: the importance of good features. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages II–53. IEEE, 2004.

- 
- [34] Stan Z Li, Long Zhu, ZhenQiu Zhang, Andrew Blake, HongJiang Zhang, and Harry Shum. Statistical learning of multi-view face detection. In *European Conference on Computer Vision*, pages 67–81. Springer, 2002.
  - [35] Rainer Lienhart, Alexander Kuranov, and Vadim Pisarevsky. Empirical analysis of detection cascades of boosted classifiers for rapid object detection. In *Joint Pattern Recognition Symposium*, pages 297–304. Springer, 2003.
  - [36] Rainer Lienhart and Jochen Maydt. An extended set of haar-like features for rapid object detection. In *Image Processing. 2002. Proceedings. 2002 International Conference on*, volume 1, pages I–900. IEEE, 2002.
  - [37] Yen-Yu Lin and Tyng-Luh Liu. Robust face detection with multi-class boosting. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*, volume 1, pages 680–687. IEEE, 2005.
  - [38] Ce Liu and Hueng-Yeung Shum. Kullback-leibler boosting. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 1, pages I–587. IEEE, 2003.
  - [39] Hamed Masnadi-Shirazi and Nuno Vasconcelos. Asymmetric boosting. In *Proceedings of the 24th international conference on Machine learning*, pages 609–619. ACM, 2007.
  - [40] Hamed Masnadi-Shirazi and Nuno Vasconcelos. High detection-rate cascades for real-time object detection. In *2007 IEEE 11th International Conference on Computer Vision*, pages 1–6. IEEE, 2007.
  - [41] Llew Mason, Jonathan Baxter, Peter Bartlett, and Marcus Frean. Boosting algorithms as gradient descent in function space. NIPS, 1999.
  - [42] Jun Miao, Baocai Yin, Kongqiao Wang, Lansun Shen, and Xuecun Chen. A hierarchical multiscale and multiangle system for human face detection in a complex background using gravity-center template. *Pattern Recognition*, 32(7):1237–1248, 1999.
  - [43] Takeshi Mita, Toshimitsu Kaneko, and Osamu Hori. Joint haar-like features for face detection. In *Tenth IEEE International Conference on Computer Vision (ICCV’05) Volume 1*, volume 2, pages 1619–1626. IEEE, 2005.
  - [44] Erik Murphy-Chutorian and Mohan Manubhai Trivedi. Head pose estimation in computer vision: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 31(4):607–626, 2009.

- [45] Timo Ojala, Matti Pietikainen, and Topi Maenpaa. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on pattern analysis and machine intelligence*, 24(7):971–987, 2002.
- [46] Andreas Opelt, Axel Pinz, and Andrew Zisserman. A boundary-fragment-model for object detection. In *European conference on computer vision*, pages 575–588. Springer, 2006.
- [47] Constantine P Papageorgiou, Michael Oren, and Tomaso Poggio. A general framework for object detection. In *Computer vision, 1998. sixth international conference on*, pages 555–562. IEEE, 1998.
- [48] Pavel Pudil, Jana Novovičová, and Josef Kittler. Floating search methods in feature selection. *Pattern recognition letters*, 15(11):1119–1125, 1994.
- [49] H. A. Rowley, S. Baluja, and T. Kanade. Neural network-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1):23–38, 1998.
- [50] Payam Sabzmejdani and Greg Mori. Detecting pedestrians by learning shapelet features. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2007.
- [51] Toshiyuki Sakai, Makoto Nagao, and Shinya Fujibayashi. Line extraction and pattern detection in a photograph. *Pattern recognition*, 1(3):233–248, 1969.
- [52] Robert E Schapire. The strength of weak learnability. *Machine learning*, 5(2):197–227, 1990.
- [53] Robert E Schapire. The boosting approach to machine learning: An overview. In *Nonlinear estimation and classification*, pages 149–171. Springer, 2001.
- [54] Henry Schneiderman. Feature-centric evaluation for efficient cascaded object detection. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages II–29. IEEE, 2004.
- [55] Jamie Shotton, Andrew Blake, and Roberto Cipolla. Contour-based learning for object detection. In *Tenth IEEE International Conference on Computer Vision (ICCV’05) Volume 1*, volume 1, pages 503–510. IEEE, 2005.

- 
- [56] S.A. Sirohey. *Human Face Segmentation and Identification*. Technical report (University of Maryland at College Park. Center for Automation Research. Computer Vision Laboratory). University of Maryland, Center for Automation Research, Computer Vision Laboratory, 1993.
- [57] David B Skalak et al. The sources of increased accuracy for two proposed boosting algorithms. In *Proc. American Association for Artificial Intelligence, AAAI-96, Integrating Multiple Learned Models Workshop*, volume 1129, page 1133, 1996.
- [58] Peter HA Sneath, Robert R Sokal, et al. *Numerical taxonomy. The principles and practice of numerical classification*. 1973.
- [59] Jan Sochman and Jiri Matas. Waldboost-learning for time constrained sequential detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 150–156. IEEE, 2005.
- [60] K.-K. Sung. *Learning and Example Selection for Object and Pattern Recognition*. PhD thesis, MIT, Artificial Intelligence Laboratory and Center for Biological and Computational Learning, Cambridge, MA, 1996.
- [61] Robert H Swendsen and Jian-Sheng Wang. Nonuniversal critical dynamics in monte carlo simulations. *Physical review letters*, 58(2):86, 1987.
- [62] Antonio Torralba, Kevin P Murphy, and William T Freeman. Sharing features: efficient boosting procedures for multiclass object detection. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages II–762. IEEE, 2004.
- [63] Akitoshi Tsukamoto, Chil-Woo Lee, and Saburo Tsuji. Detection and tracking of human face with synthesized templates. In *Proc. First Asian Conf. Computer Vision*, pages 183–186, 1993.
- [64] Akitoshi Tsukamoto, Chil-Woo Lee, and Saburo Tsuji. Detection and pose estimation of human face with synthesized image models. In *Pattern Recognition, 1994. Vol. 1-Conference A: Computer Vision & Image Processing., Proceedings of the 12th IAPR International Conference on*, volume 1, pages 754–757. IEEE, 1994.

- [65] Zhuowen Tu. Probabilistic boosting-tree: Learning discriminative models for classification, recognition, and clustering. In *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, volume 2, pages 1589–1596. IEEE, 2005.
- [66] Kagan Tumer and Joydeep Ghosh. Theoretical foundations of linear and order statistics combiners for neural pattern classifiers. *IEEE Trans. Neural Networks*, 1995.
- [67] Oncel Tuzel, Fatih Porikli, and Peter Meer. Region covariance: A fast descriptor for detection and classification. In *European conference on computer vision*, pages 589–600. Springer, 2006.
- [68] M Viola, Michael J Jones, and Paul Viola. Fast multi-view face detection. In *Proc. of Computer Vision and Pattern Recognition*. Citeseer, 2003.
- [69] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511–I–518 vol.1, 2001.
- [70] Paul Viola and Michael Jones. Fast and robust classification using asymmetric adaboost and a detector cascade. *Advances in Neural Information Processing System*, 14, 2001.
- [71] Paul Viola and Michael Jones. Robust real-time object detection. *International Journal of Computer Vision*, 57(2):137–154, 2001.
- [72] Paul Viola and Michael J. Jones. Robust real-time face detection. *Int. J. Comput. Vision*, 57(2):137–154, May 2004.
- [73] Paul Viola, Michael J Jones, and Daniel Snow. Detecting pedestrians using patterns of motion and appearance. *International Journal of Computer Vision*, 63(2):153–161, 2005.
- [74] Peng Wang and Qiang Ji. Learning discriminant features for multi-view face and eye detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 373–379. IEEE, 2005.
- [75] Andrew R. Webb and Keith D. Copsey. *Statistical pattern recognition, Third Edition*. John Wiley & Sons, 2011.



- 
- [76] Ka-Chun Wong. A short survey on data clustering algorithms. In *2015 Second International Conference on Soft Computing and Machine Intelligence (ISCMi)*, pages 64–68. IEEE, 2015.
  - [77] Bo Wu, Haizhou Ai, Chang Huang, and Shihong Lao. Fast rotation invariant multi-view face detection based on real adaboost. In *Automatic Face and Gesture Recognition, 2004. Proceedings. Sixth IEEE International Conference on*, pages 79–84. IEEE, 2004.
  - [78] Bo Wu and Ram Nevatia. Cluster boosted tree classifier for multi-view, multi-pose object detection. In *2007 IEEE 11th International Conference on Computer Vision*, pages 1–8. IEEE, 2007.
  - [79] Jianxin Wu, S Charles Brubaker, Matthew D Mullin, and James M Rehg. Fast asymmetric learning for cascade face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(3):369–382, 2008.
  - [80] Jianxin Wu, James Matthew Rehg, and Matthew D Mullin. Learning a rare event detection cascade by direct feature selection. 2003.
  - [81] Rong Xiao, Huaiyi Zhu, He Sun, and Xiaoou Tang. Dynamic cascades for face detection. In *2007 IEEE 11th International Conference on Computer Vision*, pages 1–8. IEEE, 2007.
  - [82] Rong Xiao, Long Zhu, and Hong-Jiang Zhang. Boosting chain learning for object detection. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 709–715. IEEE, 2003.
  - [83] Lei Xu, Adam Krzyzak, and Ching Y Suen. Methods of combining multiple classifiers and their applications to handwriting recognition. *IEEE transactions on systems, man, and cybernetics*, 22(3):418–435, 1992.
  - [84] Shengye Yan, Shiguang Shan, Xilin Chen, and Wen Gao. Locally assembled binary (lab) feature with feature-centric cascade for fast and accurate face detection. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–7. IEEE, 2008.
  - [85] Guangzheng Yang and Thomas S Huang. Human face detection in a complex background. *Pattern Recognition*, 27(1):53 – 63, 1994.
  - [86] Ming-Hsuan Yang, David J Kriegman, and Narendra Ahuja. Detecting faces in images: A survey. *IEEE Transactions on pattern analysis and machine intelligence*, 24(1):34–58, 2002.

- [87] Shuo Yang, Ping Luo, Chen Change Loy, and Xiaoou Tang. Faceness-net: Face detection through deep facial part responses. *arXiv preprint arXiv:1701.08393*, 2017.
- [88] Junsong Yuan, Jiebo Luo, and Ying Wu. Mining compositional features for boosting. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.
- [89] G Udny Yule. On the association of attributes in statistics: with illustrations from the material of the childhood society, &c. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 194:257–319, 1900.
- [90] Cha Zhang and Zhengyou Zhang. A survey of recent advances in face detection, 2010.
- [91] Lun Zhang, Rufeng Chu, Shiming Xiang, Shengcai Liao, and Stan Z Li. Face detection based on multi-block lbp representation. In *International Conference on Biometrics*, pages 11–18. Springer, 2007.
- [92] Zhi-Hua Zhou. *Ensemble methods: foundations and algorithms*. CRC press, 2012.
- [93] Chenchen Zhu, Yutong Zheng, Khoa Luu, and Marios Savvides. Cms-rcnn: contextual multi-scale region-based cnn for unconstrained face detection. In *Deep Learning for Biometrics*, pages 57–79. Springer, 2017.
- [94] Ji Zhu, Saharon Rosset, Hui Zou, and Trevor Hastie. Multi-class adaboost. *Ann Arbor*, 1001(48109):1612, 2006.

# Declaration of Oath

## Eidesstattliche Versicherung

I hereby declare, on oath, that I have written the present dissertation by my own and have not used other than the acknowledged resources and aids.

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Dissertationsschrift selbst verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Hamburg, Date 7<sup>th</sup> Januar 2019  
City and Date  
Ort und Datum

Nils Meins  
Signature  
Unterschrift

