Neural Hopfield-ensemble for multi-class head pose detection

Nils Meins[1], Sven Magg[1], Stefan Wermter[1]

***Abstract*— Multi-class object detection is perhaps the most important task for many computer vision systems and mobile robots. In this work we will show that Hopfield Neural Network (HNN) ensembles can successfully detect and classify objects from several classes by taking advantage of head-pose estimation. The single HNNs are using pixel sums of Haar-like features as input, resulting in HNNs with a small number of neurons. An advantage of using these in ensembles is their compact form. Although it was shown that such HNNs can only memorise few patterns, by utilising a naive-Bayes mechanism we were able to exploit the multi-class ability of single HNNs within an ensemble. In this work we report successful head pose classification, which presents a 4-class problem (3 poses + negatives). Results show that successful classification can be achieved with small training sets and ensembles, making this approach an interesting choice for online learning and robotics.**

I. Introduction

Since the world surrounding us contains multiple objects at any given moment, reliable multi-class object detection is an important feature of any system that has to act in such environment. To efficiently use computer vision systems for object detection on robotic systems they need to be fast and of preferably low computational complexity. Previous work has already shown that the Adaboost Algorithm combined with Hopfield Neural Networks (HNNs) as weak classifiers can be used efficiently for classification tasks [4] and that the accuracy can be improved by using different image representations simultaneously[11]. In this work we will show that an extended approach can be efficiently used for multi-class object detection by taking advantage of head pose estimation. HNNs are used as weak classifiers in the Adaboost ensembles and we now directly exploit the multi-class ability of those HNNs. Hopfield [1] has shown that the number of patterns an HNN can memorize is $0.14\ N$, where N is the number of neurons. Our HNNs are using dedicated single rectangle pixel sums of Haar-like feature structures as input. Therefore our HNNs had a size of between 4 and 16 neurons and should not be able to memorize more than one pattern on average. We will however show that by using the HNN combined with a naive-Bayes mechanism as a weak classifier within an ensemble, we can use them for more patterns. In the presented experiments, we were able to learn three different positive classes together with negatives using HNNs that contain only nine neurons each.

[1]Knowledge Technology Group, Department of Informatics, University of Hamburg, Vogt-Kölln-Str. 30, D-22527, Hamburg, Germany. {meins,magg,wermter}@informatik.uni-hamburg.de

This work has been supported by the KSERA project funded by the European Commission under FP7 for Research and Technological Development, grant agreement n 2010- 248085, and project RobotDoC under 235065 ROBOT-DOC from FP7, Marie Curie Action ITN.

This paper is structured as follows: After describing some related research in section II, we will describe our used training and test set (section III). In section IV we will give a short introduction to Hopfield Neural Networks and to Adaboost in section V. Afterwards we describe how we use HNNs and Haar-like features together (section VI), how the learning of the entire HNN is accomplished (section VII) and how the learning for the multi-class Adaboost is performed in this work (section VIII). Finally we will show the results of our experiments (section IX), discuss these (section X) and finish this work with the conclusion and outlook to future work (section XI).

II. Related Work

Various methods for multi-class object detection have been proposed in the literature. Freund and Schapire [3] used a set of plausible labels instead of returning a single-class indicator for multi-class object detection. Gehler and Nowozin [12] vary different visual feature combinations for multi-class object detection and analyse different methods for learning and combining these features, by using kernel methods and boosting.

A different approach was presented by Torralba et al. [5] who used feature-sharing and JointBoost. Feature-sharing uses features that are relevant for different objects, e.g. different characters like D, B and R share some features, in this case the vertical line of the left side. Torralba et al. used these common features to collect the plausible types and to finally classify the correct object. Salakhutdinov et al. [9] also used feature-sharing combined with a hierarchical classification model based on a Bayesian framework for distinguishing a huge amount of different object classes. We also use the same Haar-like feature structure for different objects, which has a common aspect with feature-sharing but is different in its result. The common aspect is that, because one feature is used for different objects and is using just parts of the object, sometimes the Haar-feature has the same values for different objects. Therefore it is similar to and a natural compact form of feature-sharing. But although the HNN indeed gets the same feature type and structure as input, the produced output vector will still differ for different objects.

Jun and Gosh in their work [13] introduced the multi-class Adaboost.BHC algorithm. Adaboost.BHC creates a class hierarchy, which is learned according to the closeness of the underlying binary classifier, whereby Adaboost is applied to these binary classifiers. Some training strategies – like OAO, OAA and PAQ – were introduced for a multi-class ensemble consisting of binary weak classifiers by Ou et al. [6]. These strategies propose the partitioning of the classes for training.

OAO for example trains three single binary classifiers for a three-class problem. The first distinguishes between class one and class two, the second between class two and class three and the last between class three and class one. Considering these different training strategies, Ou et al. [6] analysed combinations of binary neural networks and single multi-class neural networks.

Also considering the different training strategies for multi-class combined binary classifiers, Kim et al. [7] emphasized the reduction of training time by using a weak classifier with multi-class ability for creating an ensemble. They used a Self-Organizing Map (SOM) and a Centroid Neural Network as weak classifier and introduced a multi-class classifier-based Adaboost Algorithm (Adaboost.MC) [7].

In comparison, Zhu et al. [8] analysed changes to the Adaboost algorithm itself to handle multi-class problems. They called this algorithm SAMME (Stagewise Additive Modeling using a Multi-class Exponential loss function). Following their analysis and comparing with some own solutions, we have adapted their changes for calculating $alpha$ and updating the weights of the training-set samples.

III. Training and Test Set

Fig. 1. Samples of the training set. Only three source images were used (left), afterwards we created additional images with different randomly changed width, height, rotation and gamma corrections, finally reaching 35 images per pose (middle). On the right are samples of randomly cropped negatives of landscape, offices and flat images taken from the internet.

For the experiments in this study we have used grey scale images for training and test set taken from the head pose database used by Gourier et. al. [10][1]. The basis for the training set was comprised of 3 images from just one person in three different poses as illustrated in figure 1. These images were cropped to include only the face area and random changes to width, height and rotation (3°) of the cropped images were introduced. Afterwards we created some additional images by applying gamma corrections to the current set to arrive at 35 images per pose, i.e. 105 positive samples. Negatives were taken from an own database containing 30000 randomly cropped images of landscapes, offices and flats taken from the internet. We divided these images into two sets, 9000 images for testing and 21000 images for training. We also added cropped parts from the positive set to get example backgrounds without the faces. These images all have different widths and heights and 500 of these were used together with the 105 positive samples (35 per pose) to create the training set as shown in figure 1. For training and testing, the images were converted to grey scale images and scaled to a fixed square of 24 pixels for most experiments.

[1]http://www-prima.inrialpes.fr/perso/Gourier/Faces/HPDatabase.html

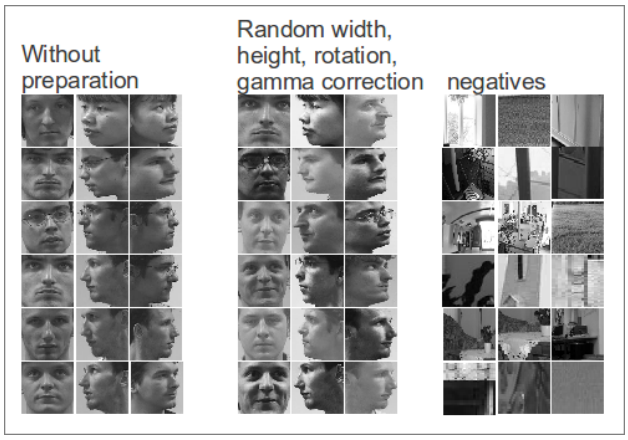

Fig. 2. Samples of the test set. 208 images from 14 people in three poses cropped without any changes (left). To simulate noise and different lightning conditions, we created in addition to these 208 head poses additional images with different randomly changed width, height, rotation and gamma corrections (middle). On the right are samples of randomly cropped negatives of landscape, offices and flat images taken from the internet.

In the experiments we used two test sets. A basis for both were 208 images of 14 people including 64 frontal views, 79 left views, and 65 right views (figure 2). The first test set consisted of exactly those images transformed to a fixed width and height (Test set 1). To include conditions not present in the training set, the second set contained additional images created using random rotation (5°), changes in width and height and gamma correction (figure 2) to simulate different lightning conditions (Test set 2). In the end, the test set contained 12480 positives (over 3 classes) and 9000 negatives.

IV. Hopfield Neural Network

The Hopfield Neural Network is a single layer, recurrent Neural Network where every neuron has a connection to every other neuron except itself. An HNN has the ability to reconstruct a learned pattern from noisy input, making it an ideal tool for reconstructing an image from a learned image pattern.

The nodes are perceptrons with a binary activation function (eq. 1). In contrast to the standard HNN, we have used a logistic activation function (eq. 2).

$$x_j = \begin{cases} 1 & s_j > \theta \\ -1 & \text{otherwise} \end{cases}, \; s_j \text{ see eq. 2} \tag{1}$$

$$x_j = \frac{1}{1+e^{-s_j}}, \text{ where } s_j = \sum_{i=1}^{N} w_{ij}\, x_i, \tag{2}$$

N in eq. 2 is the number of neurons.

Hopfield has shown in his work [1] that an HNN with a symmetric weight matrix will converge to a final stable state.

The training of the HNN is done by using the Hebb-learning rule (eq. 3) which has a high computational performance. This is important in the context of ensemble training because of the comparably long training times.

$$w_{ij} = \sum_{m=1}^{M} x_i^m \cdot x_j^m \text{ if } j \neq i,\ w_{ij} = 0 \text{ otherwise,} \qquad (3)$$

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

V. Adaboost

Adaboost is an ensemble algorithm that selects the classifiers for the ensemble (in this context mostly called "weak classifiers") and calculates a weight for these classifiers. Images also have a weight factor, which would be 1 if every image is considered equally important. The re-weighting of the images within the training set is an important mechanism of Adaboost. It focuses the selection of the weak classifiers during the creation process on different aspects of the considered objects (here faces).

The ensembles are created by iterating over the following steps:

1) The weak classifier which has the lowest error on the training set will be selected, taking into account the current weights of the training images.
2) The classifier is added to the ensemble and the weight is calculated according to its current error.
3) The training set will be re-weighted so that the weight of correctly classified samples will be decreased (this means, that misclassified samples are more important in the next iteration).

After a defined number of iterations T, the ensemble consists of T selected weak classifiers and the classification is henceforth performed by a vote over all members.

VI. HNN and Haar-like Feature Vector

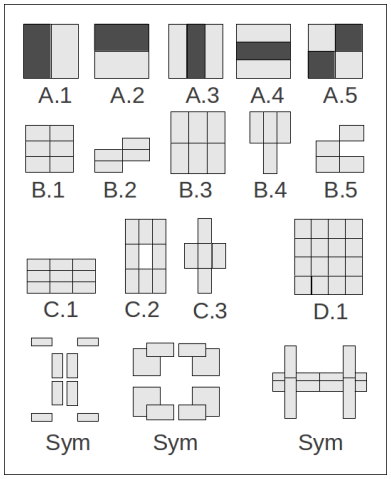

Fig. 3. Original and more complex Haar-like features. The feature value of the original features is the pixel sum difference of the grey and white rectangle. In this work, the input of the Haar-like feature to a HNN is the vector containing values of all individual rectangle pixel sums.

HNNs combined with Haar-like features (HaarNN) have already been used in our previous work [4]. There we have introduced different and more complex Haar-like feature structures as compared to Viola and Jones [2] and formed a vector using the pixel sums of the individual rectangles within the feature structure. In addition to the features used in [4] we have now also used the SYM Haar-like feature (see Fig. 3). These features were created by choosing two rectangles and flipping them first horizontally and then vertically. This was done without any constraints, i.e. also overlapping of different rectangles was allowed.

Through an image representation called "integral image", the pixel sum of an arbitrary rectangle can be calculated by accessing just four array references, instead of having to sum all pixels of the corresponding rectangle within the grey scale image (see [2]). Therefore, it is very fast to calculate all the pixel sums of even the described complex Haar-like features.

Every HNN used exactly one Haar-like feature as input and therefore the HNN has as many nodes as the Haar-like feature structure has rectangles. We are using the value of a pixel sum, normalised to the corresponding area and therefore have values between 0 and 255. The final input to the HNN is a vector containing the normalised pixel sums of all rectangle pixel sums of the Haar-like feature as described in figure 4.

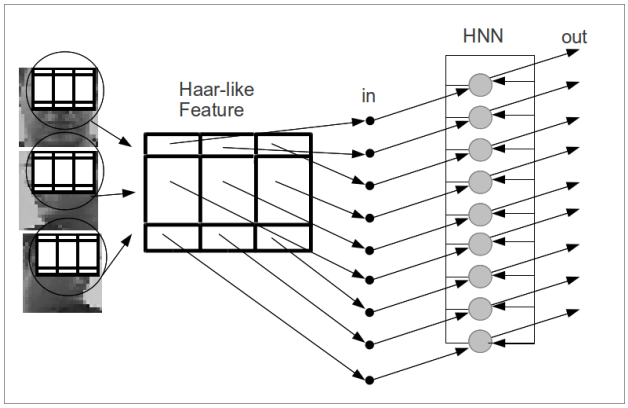

Fig. 4. The pixel sums of all rectangles within the Haar-like feature structure will be used as an input vector for the HNN.

It is important for the dynamics of an HNN to have a balance between positive and negative input values. In order to achieve that, an offset is calculated by which the input values are shifted before being fed into the HNN (i.e. in the extreme case, the values are shifted from $[0, 255]$ to $[-122, 123]$).

$$\phi_m = \frac{max(x_m) + min(x_m)}{2}, \quad \phi_a = ave(\phi_m) \qquad (4)$$

In equation 4, x_m is the input vector of image sample m. We adjust our current input vector x by using $p_i = x_i - \phi_a$ to get the final vector p, where p_i are the values of the final input vector and x_i the input vector and ϕ_a the average of the learned offsets over all image samples.

We can then adapt equation 2 for our purpose:

$$x'_j = \frac{2\beta}{1 + e^{-s_j}} - \beta \tag{5}$$

where $\beta = max(|p_i|)$ of the learned and adjusted vector p.

VII. Learning the HNN

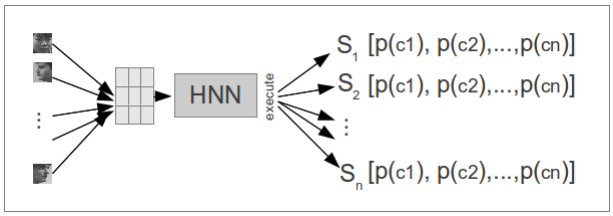

Fig. 5. After execution, the HNN will converge on one of the stable states and a set of probability values for all classes are calculated for each state (S_i).

As introduced in our previous work [4] we have also used Hebb-learning for training the HaarNNs. In our current work we have introduced two novel concepts: While we have used the Euclidean distance for distinguishing the positive and the negative classes before, in this work we now used a Bayes-like probability vector for every stable state as described in figure 5.

Learning was done in two iterations. Through the first unsupervised iteration, a HaarNN was trained using Hebb-learning with all positive samples according to equation 3, where w is the weight-matrix, x_i, x_j are input patterns and M is the number of learned patterns, e.g. in this case the number of training images.

For every sample, the Haar-like feature pixel sums were calculated and the corresponding vector was used as input for the Hebb-learning and thus for calculating the weight matrix of the HNN. The negative samples were not included in the Hebb-learning iteration. Through the Hebb-learning, only positive patterns of the different classes according to the training images were learned.

Please note that during its recurrent execution, every HNN with symmetric weights will converge on a stable state, i.e. the answers/outputs of single neurons will not change any more. These stable output vectors were used as "labels" for the different states.

In the second iteration we measured how often images from a given class were mapped to a specific stable state producing a vector of probabilities. These probabilities describe how likely it is that an image from a selected class will lead to this specific state. Through this step we account for the fact that the process is not a unique transformation from one input class to exactly one state. Inputs from different classes can result in the same stable state and two images from the same class can lead to different stable states. Nevertheless, HNNs combined with Haar-like features usually have a high preference for one class. In this second iteration, we also included the negative samples.

After both iterations, the result is a vector for each stable state of the HNN that comprises probability values for every class as described in figure 5. Classification can then be performed by first executing the HNN with the incoming input until it reached a stable state and then retrieving the vector for this stable state to get the probability values for each class. The HaarNN classifiers remember the probability vectors for the different states to have them readily available in the classification phase.

VIII. Learning the Ensemble

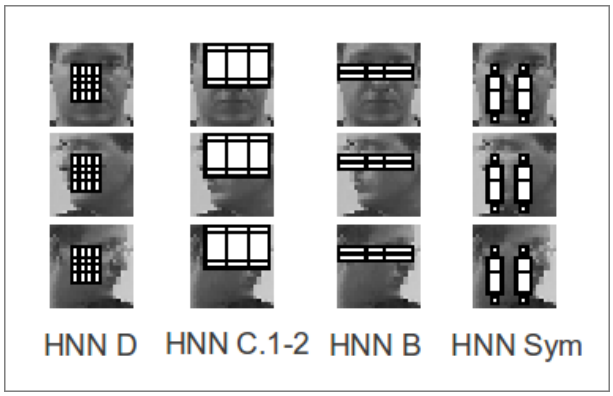

Fig. 6. The images shows the Haar-like features of type B, C, D and SYM that are chosen in the first iteration of the particular Adaboost training (see figure 3).

For learning the ensemble, we use the AdaBoost algorithm from Viola and Jones [2] with some changes taken from Zhu et. al. [8]. Adaboost generally chooses the HaarNN with the lowest error and decreases the weights of the correctly classified samples.

Two concepts are different from the original algorithm: The first change is within the classification process. In the original work the classification result is calculated as follows:

$$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

where α_t is the weight of classifier t and $h_t(x)$ its output for input image x. This can only be used for distinguishing between one class and background but we are now working with (multi-class) vectors. Therefore, we are calculating the sum of vectors, containing the probabilities for all classes:

$$v(x) = \sum_{t=1}^{T} \alpha_t h_t(x) \tag{7}$$

where $v(x)$ is the vector with the sum of all probabilities for all classes, which is similar to the solution from Freund and Schapire [3]. The final result is then the class with the highest value in this vector (winner takes all: Eq. 8).

$$h(x) = max(v(x)) \tag{8}$$

The second change is within the training process. In the original work of Viola and Jones[2], the weight of a single weak classifier (α) is calculated as follows (eq. 9):

$$\alpha = log(\frac{1 - \epsilon}{\epsilon}) \tag{9}$$

We adapted this calculation according to Zhu et. al.[8]:

$$\alpha = log(\frac{1-\epsilon}{\epsilon}) + log(n_c - 1) \quad (10)$$

where the parameter n_c is the number of different classes excluding the negative-class.

Based on Zhu et. al. [8] we re-weight the training set by increasing the weights of the misclassified samples by using equation 11.

$$w_{i+1} = w_i \; exp(\alpha) \quad (11)$$

As described, our HaarNN returns a vector of probability values for each class. For calculating the error of a HaarNN and selecting the best weak classifier in an iteration, we select the class with the highest value within the probability vector (similar to Eq. 8), i.e. we determine the winning class for a single HNN as we do it for the whole ensemble.

IX. Results

In multi-class classification we consider two types of false positives. A false positive can either be a negative sample, classified into one of the positive classes (type-1), or a positive sample, classified into the wrong positive class (type-2). We consider both of these types separately in the following graphs to analyse in how far the ensemble has the ability to classify different classes correctly and also to distinguish between trained classes and arbitrary negatives.

We have used four different feature sets for the following experiments. There is one ensemble trained for every Haar-like feature type (B, C, D and SYM) as described in figure 3. All ensembles contain 50 HaarNNs in the end. The cardinality of the feature sets and therefore the number of available HaarNNs for creating the ensemble differed; smaller sets naturally reduce the training time which is a desirable effect in online learning or mobile robotics.

Figures 7 and 8 show the results of the four different HaarNN-ensembles applied to the training set. Just a few (about 5) HaarNNs are needed in most ensembles to reach a detection-rate of 100% (figure 7) while the false-positive rate lies between 15% and 20% with 500 negative samples.

Figure 9 shows the detection-rate of the different HaarNN ensembles for “test set 1” containing 208 positives and 9000 negatives (see III). The x-axis shows the number of HaarNNs (weak classifiers) in the ensemble and the y-axis the corresponding detection-rate. The initial increase in performance in terms of detection-rate with increasing number of classifiers in the ensemble is clearly visible. The detection-rate stabilises from around 13 classifiers at about 90%, which shows that multi-class HaarNN-ensembles are an effective method to distinguish different classes. Combined with figure 10, which shows the reduction of type-1 false-positives with increasing ensemble size, it shows that already small ensembles are able to classify four classes (3x positive and 1x negative).

Similar results were obtained from experiments using “test set 2” (figure 11) containing 12480 positives and 9000 negatives (see III). The results for false-positives type-1 are comparable to the results from test-set 1. Combining the

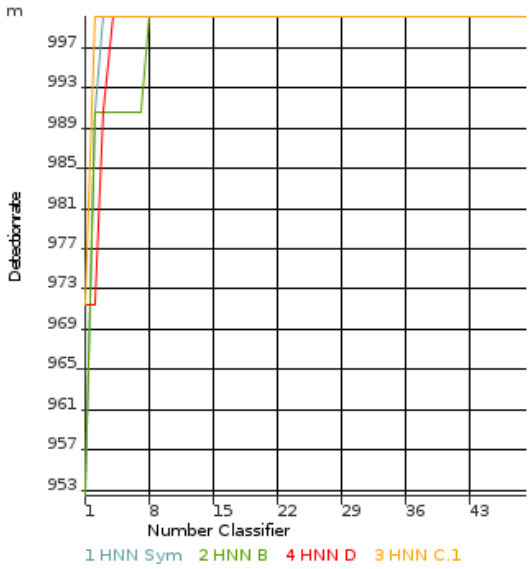

Fig. 7. Detection rate ($\times 10^3$) of HaarNN-ensembles, containing up to 50 HaarNNs, using Haar-like feature type B, C, D and SYM (see figure 3), when applied to the training set.

results on detection-rate with figure 12, where the error shows the combined effect of both types of false-positives (as does the detection-rate), it can be seen that the HaarNN SYM performs best.

Figure 13 shows the false-positive rate considering only those misclassified images where a positive class was classified as a different positive class, instead of the false-positives where a negative sample was classified as a positive class. By comparing this rate with the detection-rate for the same ensembles and test set, it can be seen that it is close to $1-d$ where d is the detection-rate. The conclusion is, that there are just very few positive samples that would be classified as background. This is an interesting result when considering a possible extension which uses a cascade structure as in [2]. There, the positive samples will be classified again by a subsequent, more complex ensemble, and the fact that very little positive samples are classified as negatives means that few errors are made early on in the cascade that can not be corrected later.

X. Discussion

For this work, we have performed a variety of different experiments, including training of ensembles with more than 50 members. Overall, the different types and parameters, most of the improvements in accuracy with increasing number of ensemble-members were achieved within the first 50 selected HaarNNs. This could be observed repeatedly and in order to reduce training times we henceforth limited ensembles to 50 HNNs. With more members, the accuracy can still be increased, but taking into account real-time constraints, stands in no relation to the necessary increase in ensemble size and therefore processing time.

The main advantage of the HaarNNs when used as weak classifiers is visible in Figures 7 and 9. HaarNN ensembles

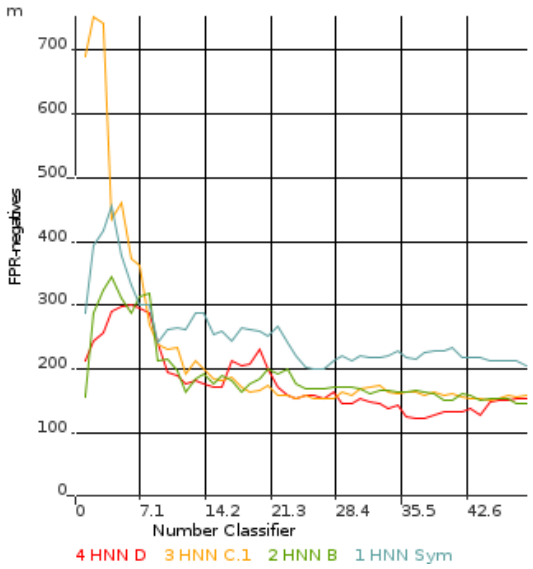

Fig. 8. False-positive rate ($\times 10^3$) of HaarNN-ensembles, containing up to 50 HaarNNs, using Haar-like feature type B, C, D and SYM (see figure 3), when applied to the training set. This chart contains only the false-positives of the negative samples (Type 1).

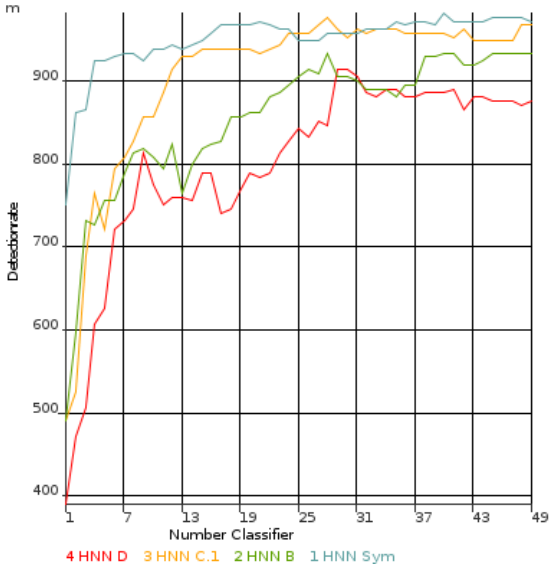

Fig. 9. Detection-rate ($\times 10^3$) of HaarNN-ensembles, containing up to 50 HaarNNs, using Haar-like feature type B, C, D and SYM (see figure 3), when applied to "Test-set 1" (see III).

Classifier	DR(%)	FP-negatives	Test-set	Set Cardinality
HNN D	87	163	Test-set 1	43200
HNN D	79	163	Test-set 2	43200
HNN C.1	96	183	Test-set 1	1890
HNN C.1	89	183	Test-set 2	1890
HNN B	93	178	Test-set 1	327600
HNN B	88	178	Test-set 2	327600
HNN SYM	97	218	Test-set 1	207936
HNN SYM	90	218	Test-set 2	207936
HNN C.1-6	94	190	Test-set 1	75625
HNN C.1-6	88	190	Test-set 2	75625
HNN C.1-7	94	201	Test-set 1	7344
HNN C.1-7	91	201	Test-set 2	7344

TABLE I

RESULTS OF THE ENSEMBLES CONTAINING 50 HAARNNS. THE ROW FP-NEGATIVES SHOWS THE FALSE-POSITIVE COUNT ONLY CONSIDERING THE NEGATIVE SAMPLES (TYPE-1). SET CARDINALITY DESCRIBES THE NUMBER OF CREATED WEAK CLASSIFIERS (HAARNNS) WHICH SPAN THE SEARCH SPACE FOR ADABOOST.

very quickly achieve a high detection rate already for small ensemble sizes, but at the expense of the false-positive counts (figure 8) compared to e.g. a threshold classifier which can also achieve zero false-positives on the training set. But we would like to emphasize that the amount of training data used in our work was quite small with just 105 positive and 500 negative samples. A small training set is an important constraint for online learning and in the area of mobile robotics. Here a trade-off between acceptable results and small training set is sought and the question on how much the training set can be reduced while retaining high detection rates is of interest to researchers in these fields.

As mentioned in the result section IX, the amount of features and therefore the HaarNNs available within the search space of Adaboost were different as described in table I. Since a higher amount of weak classifiers available to AdaBoost did not consistently improve the results (e.g. HNN B achieved lower results compared to HNN C.1 despite the huge difference in available classifiers), it seems that it is not the strongest factor affecting classification accuracy. The structure of the Haar-like features used also has a strong influence on the results. HNN B and HNN SYM are both ensembles created from the biggest underlying sets, but their results are strikingly different. While HNN SYM ensembles achieve the best results starting from small ensembles (see Fig. 12), HNN B can not achieve comparable results even with large ensembles in the end.

XI. CONCLUSION AND FUTURE WORK

In previous work, we have shown that ensembles with HNNs combined with Haar-like features as weak classifiers are an effective approach for classification. Exploiting the inherent multi-class capabilities of HNNs, we have extended our approach in the current work to a multi-class classification approach. The results for the HaarNN classifiers clearly show, that using them in an ensemble is an effective method for solving multi-class classification problems. We have shown that four classes – 3x positives and 1x negatives – can be distinguished with a relatively low error, i.e. negatives can be distinguished from the three positive classes. A slightly negative result was the type-1 false-positive rate, considering the large amount of detection windows that were to classify while searching for an object in a whole image. Too many background windows that are wrongly classified as a positive class would reduce the usability of the final subsequent processes that are using these false-positives. In a real setting, where objects have to be detected and classified within a larger image, it is more important to first distinguish

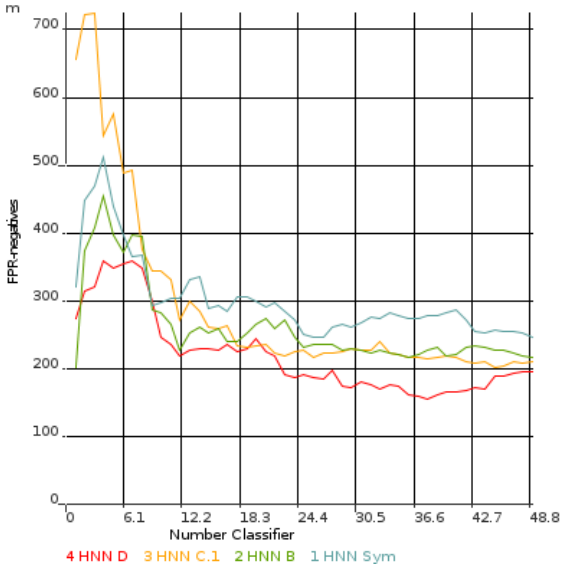

Fig. 10. False-positive rate ($\times 10^3$) of HaarNN-ensembles, containing up to 50 HaarNNs, using Haar-like feature type B, C, D and SYM (see figure 3), when applied to "Test-set 1" (see III) considering only the negative samples (9000) (see III.

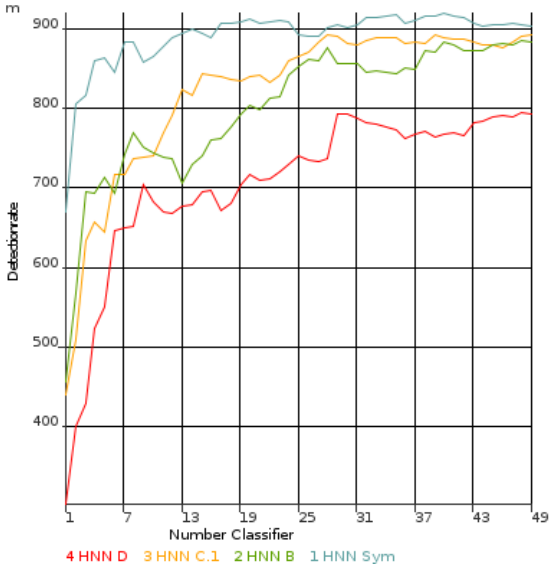

Fig. 11. Detectionrate ($\times 10^3$) of HaarNN-ensembles, containing up to 50 HaarNNs, using Haar-like feature type B, C, D and SYM (see figure 3), when applied to "Test-set 2" (see III).

between object and background windows (positive vs. negative). Therefore reducing type-1 false-positives is often more important than type-2. So one of our aims is to find ways to reduce the type-1 false-positives while keeping the good detection-rate by e.g using a cascade structure.

For the current work we have tested different Haar-like feature sets of different sizes. The sets were all chosen to be relatively small (even the large sets are considerably smaller than the set of available Haar-like features) and we have used only small input image sizes to decrease training times. Reducing the feature set sizes available to the AdaBoost mechanism did not lead to the expected decrease in accuracy and has proven to be a viable possibility to decrease training and processing times. The size of input images can be increased easily and no change in accuracy would be expected, since Haar-features can be increased accordingly and would just increase the number of the feature set used. Training with different image sizes would also increase the input diversity and therefore increase the accuracy of the combined ensembles.

There are several possibilities to extend and improve the presented method. So far we have shown that four classes can be distinguished and our main focus is on finding the maximum number of classes that can be separated by using these simple Haar-like feature structures. The number of classes could then be extended by using HaarNN ensembles combined with random forests with ensembles as tree nodes. This could increase the number of possible classes while keeping the ensembles compact.

In this work we calculated the winning class by using the winner-takes-all method with the maximum value of the probability vector for selecting the best HNN during the Adaboost iterations. This ignores the fact, that sometimes

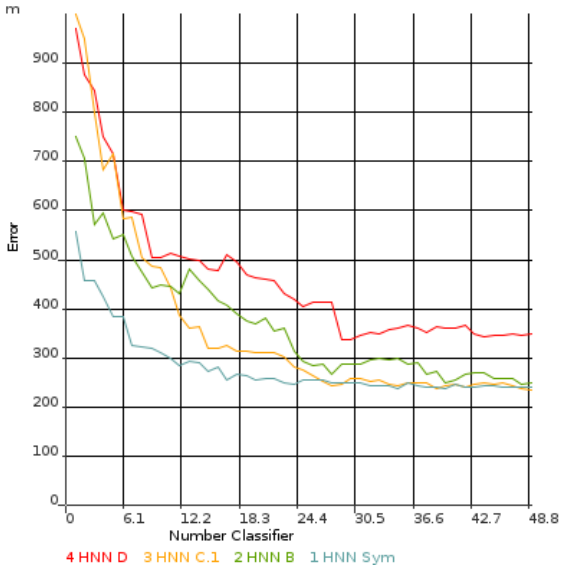

Fig. 12. Error ($\times 10^3$) of HaarNN-ensembles, containing up to 50 HaarNNs, using Haar-like feature type B, C, D and SYM (see figure 3), when applied to "Test-set 2" (see III).

probabilities for two classes are quite similar and just a small difference determines the winner. By considering the whole vector of probabilities (instead of the max value) we could select better weak classifiers for generalization in future work.

This could be aided by the use of neural networks instead of the Bayes probability vector. Calculating and storing the probability vectors for all HaarNNs is expensive both in processing time and storage. Using a neural network which takes the state of the individual HaarNN as input and learns to output the correct class could solve these problems. It

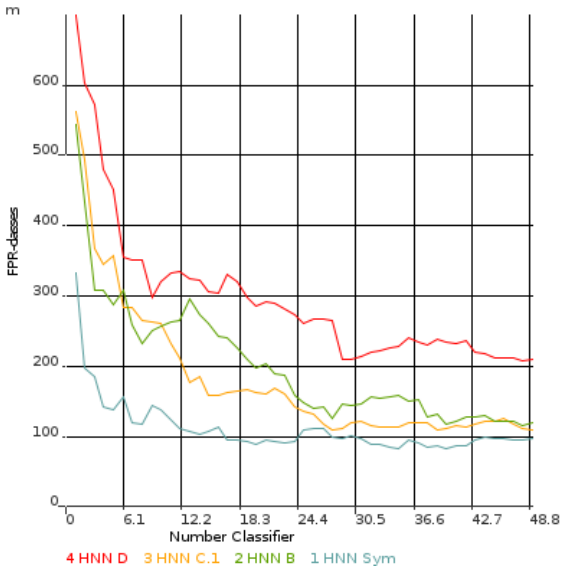

Fig. 13. False-positive rate ($\times 10^3$) of HaarNN-ensembles, containing up to 50 HaarNNs, using Haar-like feature type B, C, D and SYM (see figure 3), when applied to "Test-set 2" (see III) considering only the positive classes.

would also reduce the information that needs to be stored considerably and the HaarNNs would act as intelligent, adaptive filters for the subsequent neural networks.

Using HNN Cascades as proposed in [2] could also reduce the processing time by distinguishing positives from negatives (detection) with simple structures. Later on in the cascade, more suitable and complex structures can be used for classification and ultimately recognition. Using cascades in combination with HaarNNs is especially interesting due to the low count of false-negatives in the classification and the possible reduction in false-positives of type-1.

REFERENCES

[1] Hopfield, J.J., "Neural networks and physical systems with emergent collective computational abilities, " *Proceedings of the National Academy of Sciences*, Vol.79, pp.2554-2558 (1982)

[2] Viola, P. and Jones, M., "Robust Real-time Object Detection," *International Journal of Computer Vision (2001)*

[3] Freund, Y. and Schapire, R.E., "A desicion-theoretic generalization of on-line learning and an application to boosting," *Lecture Notes in Computer Science, Computational Learning Theory*, Vol.904, pp.23-37 Springer Berlin Heidelberg, 1995.

[4] Meins, N., Wermter, S., Weber, C., "Hybrid Ensembles Using Hopfield Neural Networks and Haar-Like Features for Face Detection," *International Conference on Artificial Neural Networks and Machine Learning (ICANN)* Springer Berlin Heidelberg, 2012.

[5] Torralba, A. and Murphy, K.P. and Freeman, W.T., "Sharing Visual Features for Multiclass and Multiview Object Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol.29, pp.854 -869 (2007)

[6] Ou, G. and Murphey Y.L. , "Multi-class pattern classification using neural networks," *Pattern Recognition, Vol.40, pp.4-18 (2007)*

[7] Kim, T.-., Park, D.-C., Woo, D.-M., Jeong, T., Min, S.-Y., "Multi-class Classifier-Based Adaboost Algorithm," *Intelligent Science and Intelligent Data Engineering*, Vol.7202, pp.122-127 (2012)

[8] Zhu, J., Rosset, S., Zou, H. Hastie, T., "Multi-class AdaBoost," *Tech. Rep., Department of Statistics, University of Michigan, Ann Arbor, MI 48109 (2006)*

[9] Salakhutdinov, R. and Torralba, A. and Tenenbaum, J., "Learning to share visual appearance for multiclass object detection," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2011)*

[10] Gourier, N., Hall, D., Crowley, J. L., "Estimating Face Orientation from Robust Detection of Salient Facial Features," *Proceedings of Pointing 2004, ICPR, International Workshop on Visual Observation of Deictic Gestures, Cambridge, UK*

[11] Meins, N. and Jirak, D. and Weber, C. and Wermter, S., "Adaboost and Hopfield Neural Networks on different image representations for robust face detection," *International Conference on Hybrid Intelligent Systems (HIS)* Springer Berlin Heidelberg, 2012.

[12] Gehler, P., Nowozin, S., "On feature combination for multiclass object classification," *IEEE 12th International Conference on Computer Vision (2009)*

[13] Jun, G. and Ghosh, J., "Multi-class Boosting with Class Hierarchies," *Lecture Notes in Computer Science, Multiple Classifier Systems*, Vol.5519, pp.32-41 Springer Berlin Heidelberg, 2009