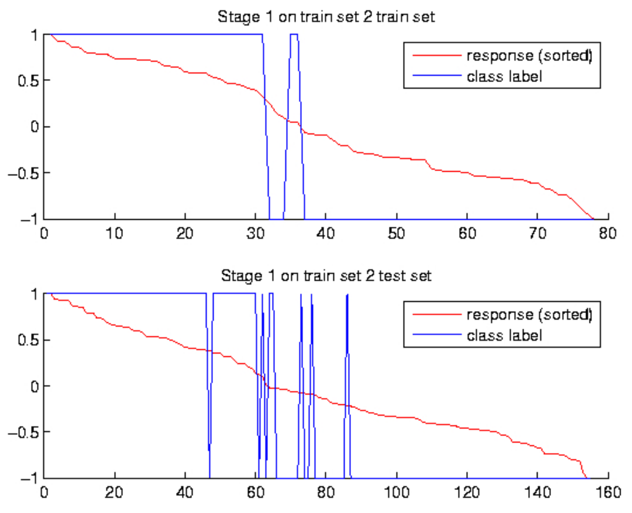
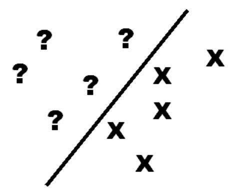
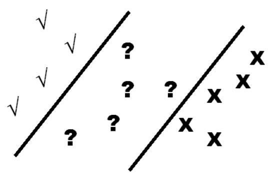


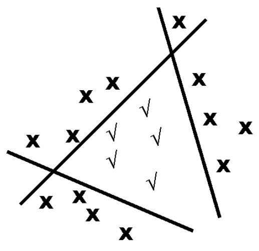
Figure 29: Strong Classifier Response. The red line is the response of the classifier sorted from highest to lowest, the top part is in the training set and the bottom is in the test set. The blue line is 1 when the image is class and -1 when it is non-class so the vertical lines are the errors. The errors are distributed around the classifiers medium response, and we can see that the classifier is a strong classifier as there are no errors in the region of highest and lowest responses.

The three-way cascade classifier works in stages similar to a standard 2-way cascade [5]. At each stage a strong classifier is applied to all samples that require further processing. Initially, all images require processing, the improvement is that if the classifier we use is a strong classifier, those samples that had a sufficiently high response are classified as class and will not be processed in additional stages, and those samples that had a very low response will be classified as non-class and will also not require any more processing. Only those images that are in the middle zone in terms of response strength will be further processed in the next stage.

In some cases the three way cascade scheme, not only enables cutting computational costs but also allows reaching lower error rates. We discussed in previous section that to improve the performance of a classifier the classifier needs to learn more and to process many features. Cascades allow adding more features online, while keeping reasonable computation cost on the average. The three-way cascade scheme may achieve better performance also by allowing more accurate partitions of the space, since it allows non-linear classification of the space; Figure 30 illustrates this geometrically and contrasts to the 2-way cascade method.


[A single 2-way cascade step]


[A single 3-way cascade step]


[A possible 2-way cascade partition]

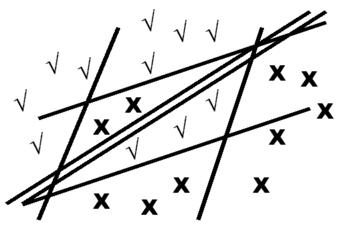

[A possible 3-way cascade partition]

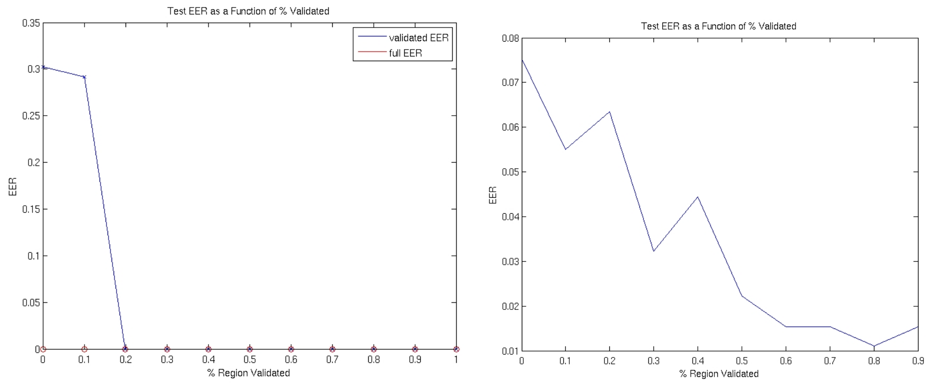
Figure 30: 2-way Cascade Induced Geometry vs. 3-way Cascade Induced Geometry. We compare here the different partitions of the space induced by the 2-way cascade method and 3-way cascade method.

5.1.1 Cascade Processing Region and Cost

Unlike ensemble classifiers, that combine separate classifiers into a single score, in the three-way cascade technique we do not combine the different classifiers using voting or any other method. Similar to cascade classifications, the score and thus the classification for different points in the space is determined by different classifiers. The three-way cascade process separates the space into different regions that determine which of the classifiers will classify each point.

One of the challenges of performing classification with three-way cascade is determining which samples should be further processed and which should not. For the current analysis we first consider a process of only two stages. Clearly, if the processing range is extremely large all samples will be processed by the first stage and then by the second stage, only to receive a final score of the second stage. So the first stage will be wasteful since it will not effect the final decision. On the other hand, if the validation range is extremely small, then all the samples will be classified by the first stage while the second stage will contribute nothing. Although the second stage was not applied and therefore exploited no resources, the performance of the first stage classifier did not improve.

Our process uses two thresholds T_{high} and T_{low}, at each stage, points that fall above T_{high} are immediately accepted and points that fall below T_{low} are rejected. We assume that the distribution of class vs. non-class is similar between the training set and the test set. We also assume that the distribution of errors is similar. This is reasonable if the training set is representative of the test set. One way to determine the thresholds is by setting them to a designated error rate at each stage in the training set; this however means there is no control over the percent of samples that will be processed by the second stage. If the main concern is the performance, we can set the thresholds according to the processing time we want to invest, we can test the train samples to get an idea of what the error rate will be. As we increase the region of processing or the computational cost the error rate decreases, Figure 31 shows this connection empirically for two types of validations that will be discussed below. In appendix 8.5 we describe a method to determine T_{high} and T_{low}. We now discuss the types of different classifiers that can be combined, specifically, combining orthogonal classifiers.

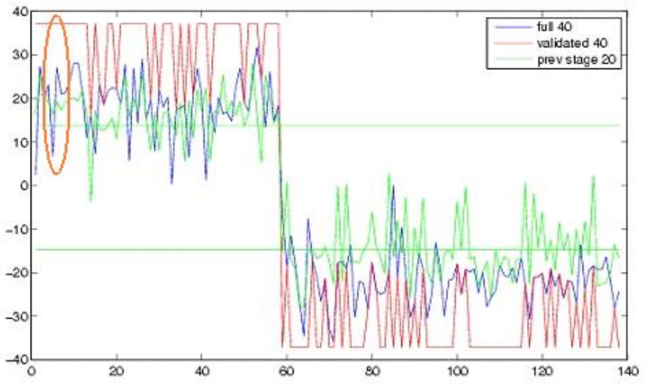


(a) (b)

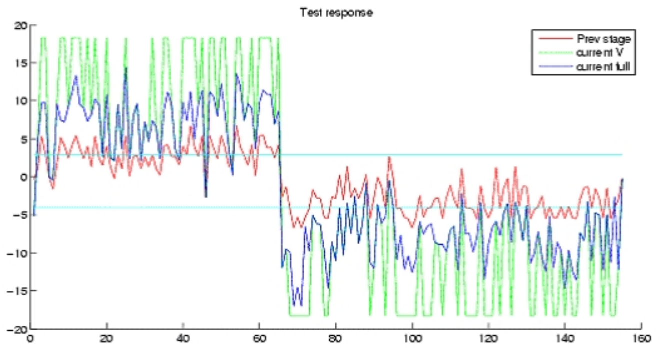
Figure 31: The EER as a Function of the processed Region. The EER when using 100 AdaBoost fragments applied in 5 stages - 20 at each stage. (a) Is on response of the simple AdaBoost classifier using a growing number of fragments. (b) Is using different classifiers at each stage, trained on different training sets in an online manner (see online cascades in Section 5.1.3). When the range is too small we get the errors caused by the weakest classifier. At around 0.5 we get a save in cost and the EER remains small similar to using the strongest classifier.

5.1.2 Orthogonal Cascade Processing

When using the cascades, the more the two classifiers are *independent*, the better off we are (assuming that the sample points that are not being processed using the second, more exhaustive stage, are classified correctly). If the two stages are co-dependant, which may happen when we simply add more features, the better classifier will err on similar sample points and the cascade can only be as good as the second classifier. On the other hand, if the second classifier is independent of the first and makes less errors, some of its errors may fall in the region not being further processed, and will be avoided, so we may even achieve an error rate that is lower than the error rate of the better classifier by validating, Figure 32 shows an example. Interestingly, when the different classifiers are trained on-line on different training sets, they are more independent, we discuss this further in the next section.



(a)



(b)

Figure 32: The effect of validation when the different stages are orthogonal or not. This shows the AdaBoost score of each image for the weaker stage, the stronger stage, and the combination of them using validation. The validation region is between the two horizontal lines. In (a) the stages were trained in the online approach and are independent. The added value of the validation is clear where the red signal is better than the blue signal, circled. In (b) which is batch AdaBoost, the weaker stage and the previous stage are in sync, and the green doesn't correct any errors of the blue signal.

5.1.3 On-Line cascades

In most on-line algorithms, there is a disadvantage to learning from only part of the samples. Yet, considering the effect of different stages being orthogonal we find that there is also an advantage to learning from clusters of samples discretely. Adding features to a classifier with a few features to improve performance creates a dependency between the two signals, and the cascade, as expected, cannot improve the performance of the best classification stage. However, if features are added while learning on a new training set, the signals are more likely to be orthogonal, and the cascade can even decrease the error rate of the better classification stage. Figure 32 demonstrates this idea. The full algorithm we used (combining AdaBoost classifiers) is provided in appendix 8.6 . At each new stage we use a new part of the training set and features found in the previous stage with new weights according to the current training set. This means that each additional stage of the cascade uses features extracted in previous stage, saving the time of detecting new features. Also, although the training set has changed, the class is still the same and it is sensible to assume the previous stage features are well descriptive of the class, Figure 33 shows a sample result.

5.1.4 Three-Way Cascade Results

As discussed previously, we ran tests for the pure 3-way cascade scheme and on the 3-way cascade when used in conjunction with online training; we show some results on a horse classification task. Our database is composed of 323 horses fairly cropped and scaled and 450 non-class images of various scenes; some example images can be seen in Figure 34.

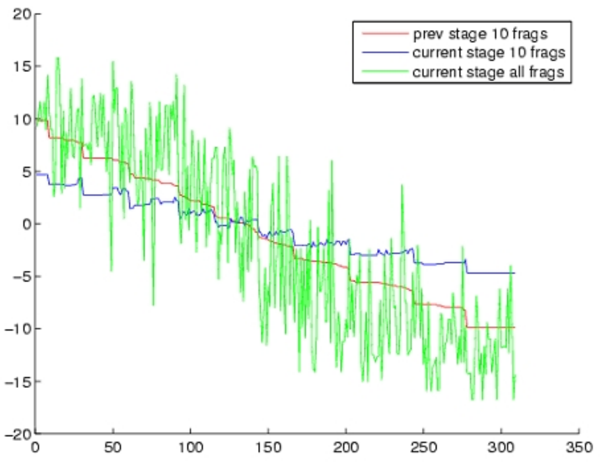
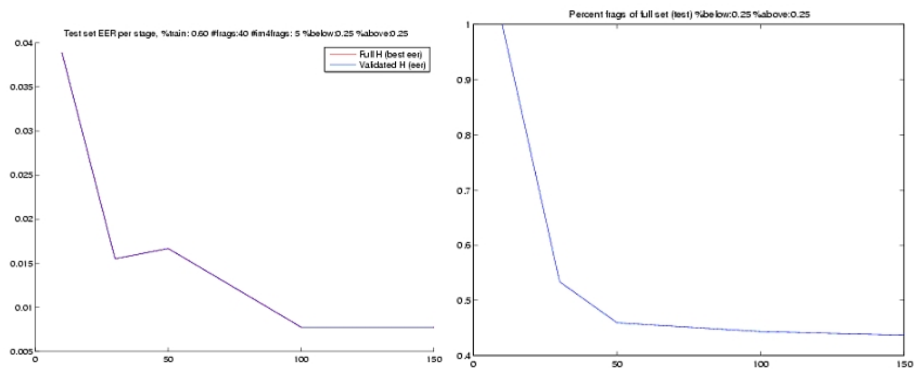


Figure 33: Online Validation comparing stages. In red, previous stage AdaBoost response with 10 fragments. In blue, using the same 10 fragments but with weights learned on the next stage training set achieves different but close response. In green, the response after adding 10 more fragments learned on the current training set. Note how the green and the red are unsynchronized.

Figure 34: Horses Data Set. The top row shows some horse images and the bottom row some samples of the non-class images.

When using the validation general scheme with a δ^{high}=0.25 and δ^{low}=0.25, the validation region is actually around 50% of the samples at each stage. We found the equal error rate wasn't affected relative to using the full data. Each strong classifier was an AdaBoost classifier with more features at each additional validation stage. We show the cost as the percent of fragments used in the cascade relative to the fragments used by the full classifier in Figure 35.



(a) (b)

Figure 35: Validation Cost. (a) shows the reduction in the EER on the testing set as we add more fragments for the normal scheme and for the cascade scheme, the red and blue plots (for the full set and for the three-way cascade are the same. (b) Shows the reduction in the average number of fragments used for classification in the cascade scheme.

In Figure 36 we show the usage of validation in two stages when the test set is 20% of the images, the initial stage is of AdaBoost with ten fragments, and the second stage that processes the results is with a hundred fragments. The validation improves the equal error rate from 0.046 to 0.011 which is a large improvement at these low rates, yet the average number of features detected is reduced by half. We show the improvement in terms of ROC as well.

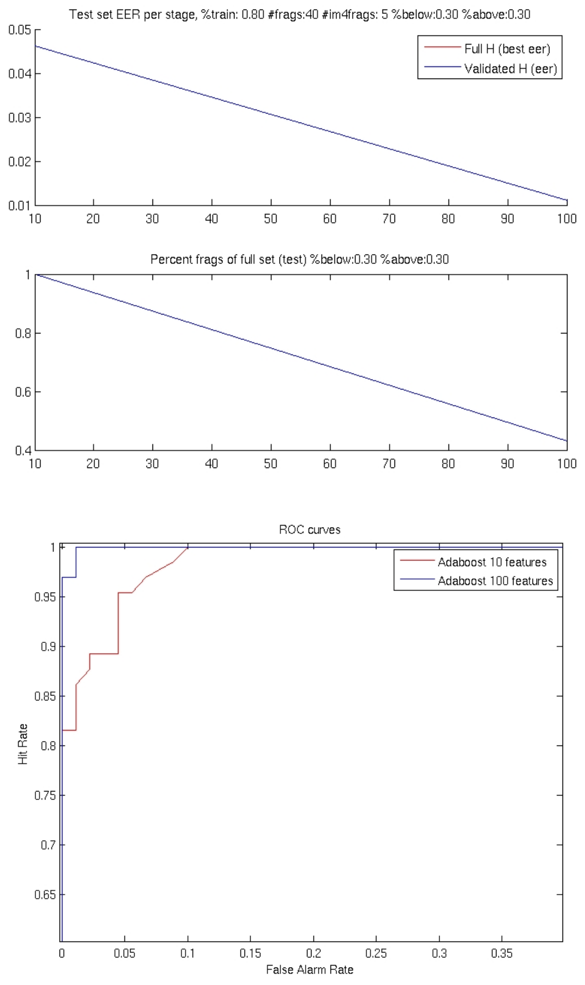


Figure 36: A Cascade of two stages. The top graph shows the reduction in the equal error rate, the results using the cascade are the same as when using the strong classifier on all the images (the lines merge). The center shows the average reduction in features used relative to the full strong classifier as features are added. The bottom shows the improvement in terms of ROC.

We also combined the 3-way cascade in an online setting as described in appendix 8.6 . We show the results averaged over 5 different trials where the training set was selected at random for each trial in Figure 37.

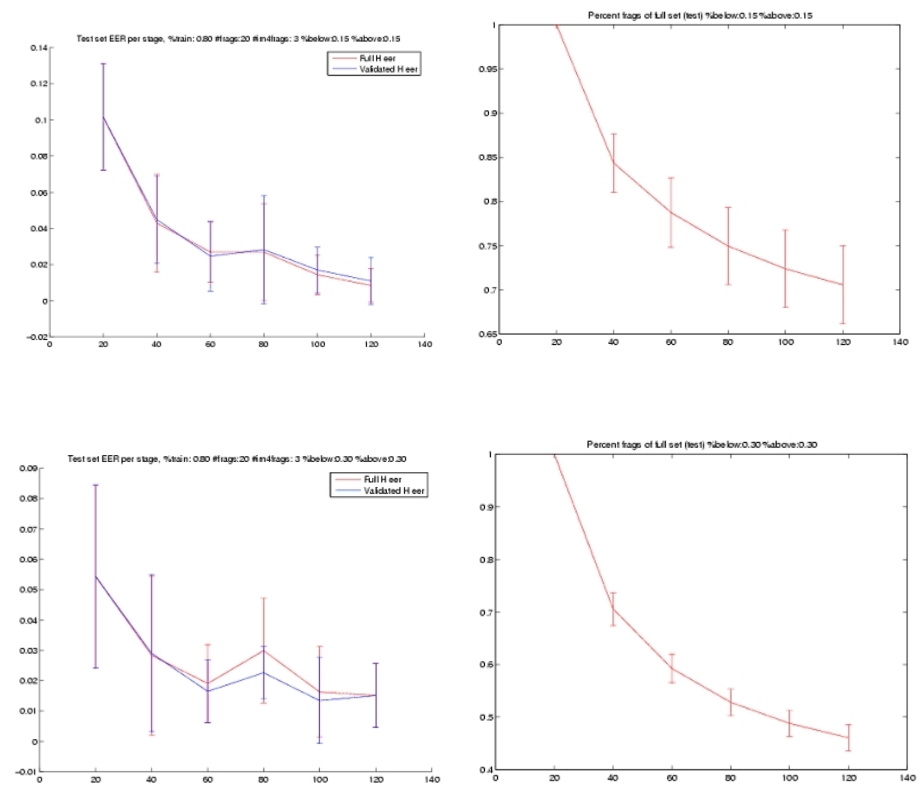


Figure 37: Online Three-Way Cascade. This shows the average change in EER for the full classifier (in red) and for the cascade (validation) classifier (in blue) at each stage, and on the right the save in computations described as the percent of fragments used relative to the full algorithm. The top row has a validation range of 0.7 and the bottom row has a validation range of 0.4.

5.2 Configuration Cascade

In this section we describe an idea of cascades in classification based on specific configurations or combinations of features. We start by describing the theoretical idea. We then continue to specify an algorithm and finally we show results.

5.2.1 Theory

Linear classifiers generate a single score for an image based on a collection of features. Treating the features as independent and basing the score simply on whether they appear or not is reasonable considering Ocam's Razor principle. However, it is highly unlikely that the real data is linearly separable. When such separation fails, some methods (such as kernel SVM), try to restore separability by mapping the problem to a space of much higher dimension. We suggest a cascade that leaves the data in a low dimension and handles specific combinations, creating 'islands' of rejections or 'acceptance' at certain points in the low dimensional space. Unlike the standard cascade methods, the new scheme identifies the examples which require an additional processing stage not on the basis of the score obtained in the first stage, but by identifying directly specific feature configurations that lead to an incorrect decision. Not only does this allow for a low cost non linear classifier, but also the scheme has a relatively simple implementation that is based on a possible biological architecture, which will be described below. Figure 38 displays schematically the idea of the exceptions, or islands in the feature space.

For example, when learning the concept of a car, two wheels are part of the useful features to detect, but the two wheels alone are not a sufficient combination since they may appear in a motorbike or in pair of rollerblades, etc. Therefore, the system, after encountering such a mistake, may learn to inhibit the combination of the two wheel fragments as sufficient for final decision (even if the obtained score is high above threshold), since by themselves, although they create a high response, they cause many errors.

A potential weakness of the idea is the question of over-fitting, or the creation of an unmanageable number of special cases. When creating an island, what is the probability that the same configuration will be found in additional images? The more features we use, the more configurations that are possible, and the probability of encountering a specific configuration again will go down. However, we show empirically that the number of different combinations that are the source of repeated mistakes is bounded; therefore it is reasonable to expect that there will not be many new problematic combinations after we have learned enough samples.

From a biological view, handling configurations in this manner appears sensible. If the normal classifier is implemented as a neural network and each fragment is a neuron in the network, then instead of having a single neuron that integrates the fragment neurons using weights we add connections between fragments that caused errors when firing simultaneously and a second stage that may inhibit or excite the configuration, according to the type of error. There is evidence of neurons in IT cortex responding to visual features

[30]. From what we know about the cortex connections, it is clear that the connections are dense, and the above architecture is one way to use these dense connections. The suggested network implicitly uses some type of feedback loops which still need to be verified. We show the neural architecture schematically in Figure 39.

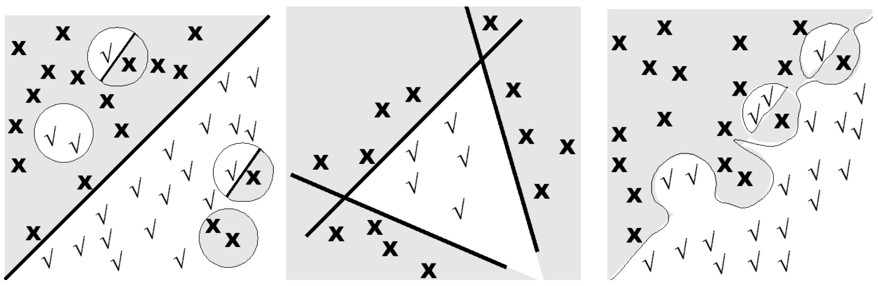


Figure 38: Combination Nonlinear Cascade. The gray region is the rejection region and in addition to the linear separation we form 'islands' around false positives, that may reduce the false positives in the test set, seen on the left. As long as the 'islands' are sparse, we can assume that they will not generate many new false negatives. Since our assumption is that in many cases the combination itself is the cause of the error, we should get some similar errors that generalize to the test set. In the center, we show the geometry formed by the know cascade method, which is very different. On the left we show a non linear seperator that can be formed using the islands, and that if we allow linear seperators within the islands then the posibilities are more than just a simple nonlinear seperator.

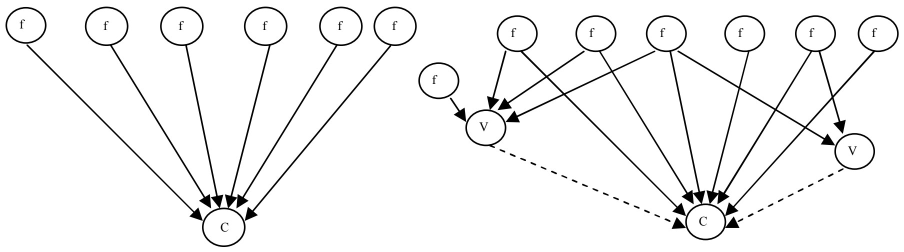


Figure 39: Combinatory Cascade. On the left we show the normal scheme with 6 fragments, and on the right, how two specific combinations are added as inhibition. The cascade may or may not use additional fragments.

5.2.2 Algorithm

We begin by describing the algorithm, and provide a summary in appendix 8.7 . Once we have a classification scheme using a set of k fragments, we select a threshold θ for the basic classification that will allow for a high false positive rate and a low miss rate, depending on the ROC of the basic classification. Using the selected θ we test for false positives in the training set. Each false positive combination is added to the Combination Database (CDB) as a faulty combination and samples that have similar combinations to the combination in the CDB are collected. A strategy to avoid future errors is learned, either separately for each CDB item or more generally for multiple combinations together, for more efficient learning. Initially there are several possible 'faulty' combinations, listed below, together with the corresponding strategies.

1. If all samples collected are of non-class, reject in the future new samples that have the specific 'faulty' combination.
2. If there are both class and non-class having the same feature combination, learn a new linear separator between them with additional class features. We also use the other training images, to allow generalization.
3. If there are class and non-class, look for 'inhibiting' fragment, or fragments in the non-class samples but not in the class samples.

The main difficulty is to use a strategy that will not over-fit the data and will generalize well to new examples. For combinations that had also some class samples, we found a separating set of fragments using a specially weighted AdaBoost that was trained on the combination samples and also on the rest of the training set samples, to allow learning more general features. The initial weights for the learning stage distributed half of the weight on the combination samples and the rest on all other samples, thus learning something that is concentrated on the samples with the faulty combination but also separates well the class and non-class.

5.2.3 Results

We tested the algorithm on a dataset of 316 cows and 450 non-cow images, and on a database of 1000 faces and 2400 non faces. The base classifier we used has 7 fragments, since our set of images is relatively small. We first show in Figure 40 and Figure 41 that the number of faulty combinations that cause errors is bounded and increases slowly with the number of samples. We show this both for the false positives and for the false negatives. Since the combinations are bounded, at some point the test set will not generate significant new errors due to new combinations. We show in Figure 42 that using the configuration rather than the score to determine which images will be processed by the cascade's second-level reduces the percent of images processed; hence the configuration method is also more efficient. The interesting result is that the 'island' processing of combinations in the test set, indeed reduces the minimum total error as seen in Figure 43. Figure 44 shows some specific images corrected by the combination. In Figure 45 we show the reduction in total error for the erroneous configurations after the second stage on a database of faces. Since we expect the cascade to deal with configurations it already encountered, this reduction is the crucial reduction to our scheme. As more configurations are learned, this reduction will improve; an observation that is not true for the range cascade. To do a complete experiment for this would require a very large and exhaustive set of images, see the future work section.

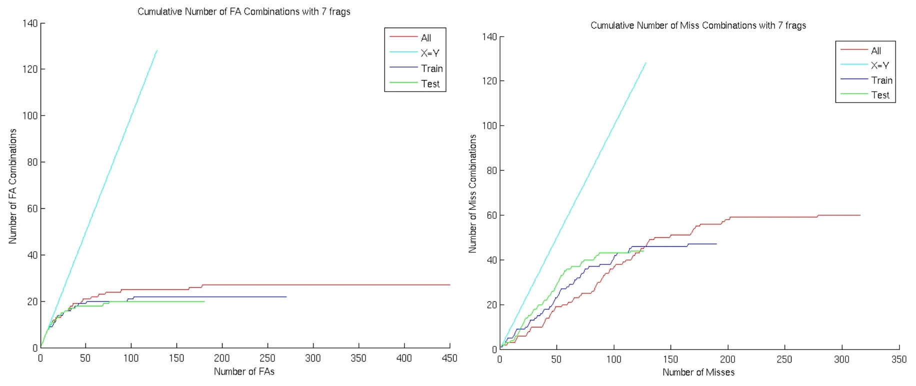


Figure 40: The Combination Distribution. The combinations that cause false alarms are bounded, and for 7 fragments, where the possible combinations are 2^7=128, the number of combinations that are found is bounded by less than 30. The cyan line shows the x=y for comparison, we show in red the configuration count for all non-class images, and the configuration count separate for the train and test set in blue and green respectively. The right side shows the same data for misses. The data here is an upper bound since we include all possible images, not only those that are likely to cause errors.

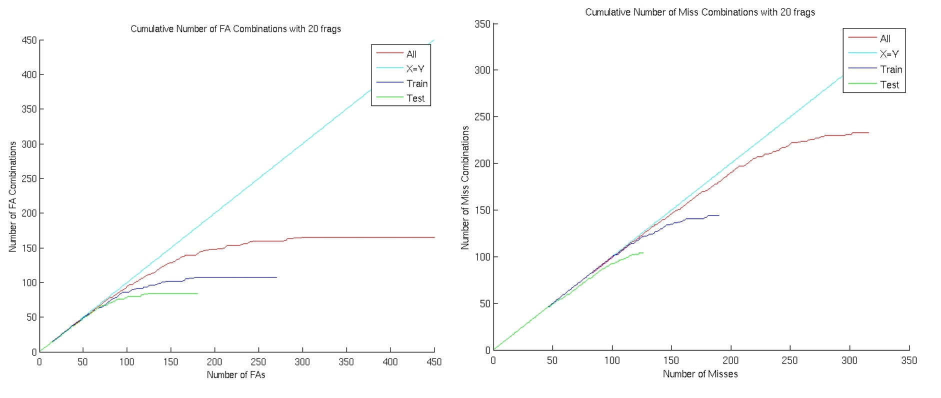


Figure 41: Combination Distribution for 20 fragments. This is the same data as the previous figure but for 20 fragments. The possible combinations are 2^{20}= 1048576, but actually up to 150 configurations are found in the false alarms, 1.4% of all configurations, similarly for the misses. The data here is an upper bound since we include all possible images, not only those that are likely to cause errors.

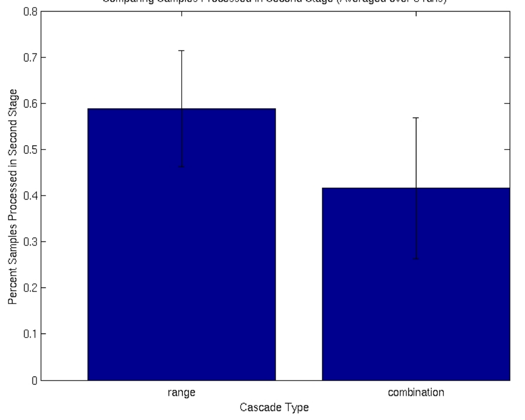


Figure 42: Percent of Images Processed by the Cascade. The percent of images processed by the second stage when the cascade depends on the range of the score (left bar) compared with when the cascade depends on the erroneous combinations (right bar), averaged over five runs.

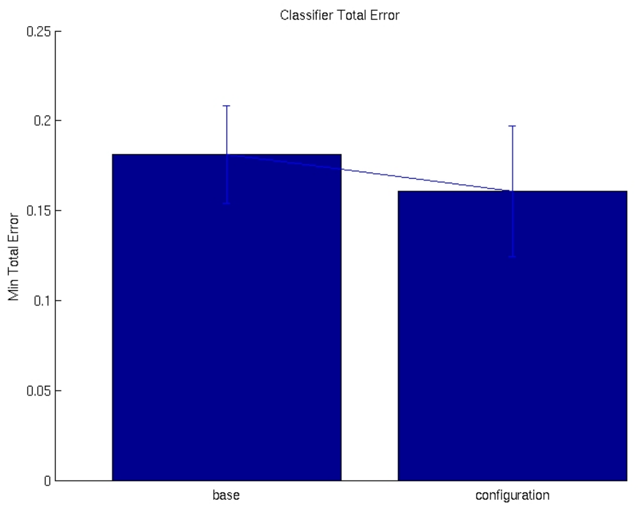


Figure 43: Classifier Total Error Reduction using the Configuration Cascade. The total error reduction when adding the configuration cascade, averaged over five runs.

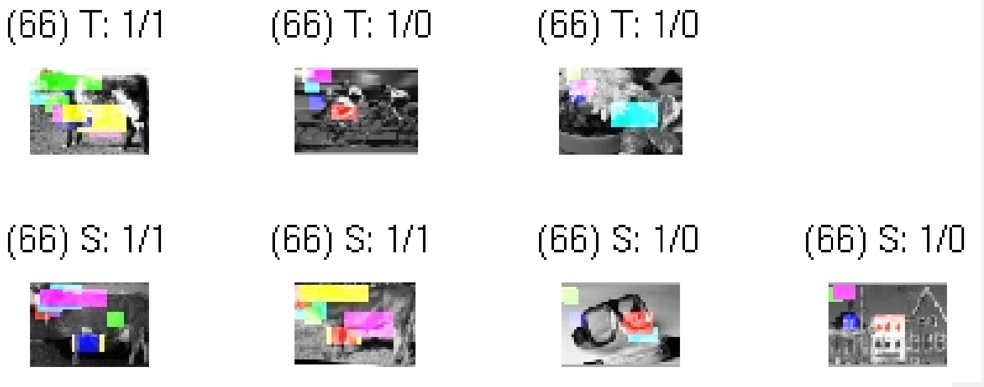


Figure 44: Specific Combinations Corrected by the Cascade. We see here combination with index 66, with 3 images in the training set (top row) and and 4 images in the test set (bottom row). The fragments marked are the original combination fragments and the additional fragments of the second level classifier that are found more in the cow images than in the non cows and allow better separation. The X/Y is the decision with the basic fragments, and the decision after the combination processing. We can see that the false positives were corrected.

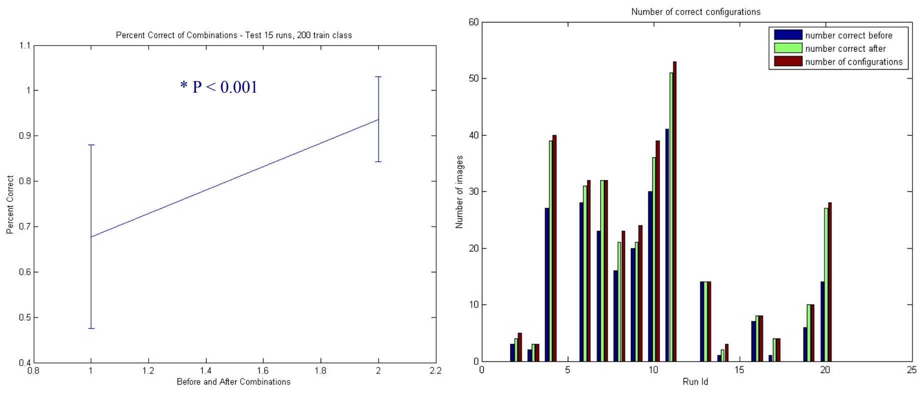


Figure 45: Reduction of the Configuration Error. Results of the configuration scheme on a database of faces (1000 faces, 2400 non faces, with a training set of 20%).The percent of total error found in the processed configurations, and the percent of errors after applying the second stage on the configurations, averaged over 20 runs. The different runs are shown on the right and are statistically significant, $p < 0.001$.

6 Conclusions

We have shown some inherent differences between state of the art classification algorithms and human categorization. The error rates made by current classifiers have improved significantly over the last years, but overall performance is still lower than human performance. In addition, we have shown that the errors made by current classifiers, especially in false alarm examples, are often considered highly unreasonable by human judgment. Incremental improvements to performance become more difficult at higher performance level, and therefore closing the gap with human performance may require significant improvement in current classification schemes.

We examined two general directions for future improvements in current classification scheme. One general direction assumes that current methods extract only a part of the available information for classification from the training set, and attempts to identify the main possible sources of additional information. Along this direction, we examined a number of plausible sources for possible improvements, including different similarity measures, increasing the number of features, adding more location information and geometric constraints, and using so-called perspective, satellites, and anti-fragments. Some of these methods contributed to improvements in performance, but the improvements appear modest and unlikely to be sufficient by themselves to reach human-level performance.

The second general direction assumes that a major limitation comes from the size of the training set. A limited data set, of say, a hundred to several hundred examples may be inherently insufficient to capture all the necessary variations needed to learn a truly high-performance classifier. Our experiments suggested that classification performance indeed increase monotonically and without clear saturation as the training set increases in size. We also examined whether to reach this improvement, the feature set used for classification needs to be increased. In theory, perhaps a limited set of K features is sufficient, and as the training set increases in size, the set of features is updated, but not increased. The results suggest that the set of features used by our scheme needs to be increased to capture the additional information in the increased training set.

These results have two implications to the classification scheme. First, they raise the advantage of constructing classifiers in an on-line manner, in which the classifier is continuously improved by new examples. Second, they raise the problem of increasing computational load as the number of features used for classification increases.

To deal with these problems we developed and tested two classification schemes. Both are multi-stage, or ‘cascade’ methods in which all images are analyzed in the first stage, and, depending on the results, only some of them are analyzed further by subsequent stages. The first of these methods was a so-called 3-way cascade, which is an extension of previously used cascade methods. In this method, a first-level classifier is first applied to the new image to be classified. If the response is high or low enough, a decision is made. Otherwise, a second-level classifier, which uses more features, is applied to the image to allow a more confident response. Ultimately this cascade achieves better performance while detecting few features on the average compared with just using the second-level classifier.

The second multi-stage scheme we developed is the so-called configuration cascade. In this approach, the decision to process an image in additional stages is based not only on the score obtained in the first stage, as in most cascade schemes, but on the particular feature configuration discovered in the image. In this scheme, erroneous configurations of the first-level classifier are processed by a second-level classifier which uses more features and achieves better performance. We showed that the number of possible erroneous configurations is bounded and therefore the configurations will repeat eventually and also that the error on the erroneous configurations is reduced after additional processing. We have also shown that using the configurations to determine which images need additional processing will require further processing on few images, fewer than in the 3-way cascade method described previously. We believe that using the configuration cascade in an on-line setting is straightforward, since new erroneous configurations can be added on-line, and that this method will enable achieving better overall performance.

7 Future Work

In this section we discuss some possibilities for extending our work, relating to the improvements we showed and the use of cascades. We also make some suggestions regarding the better understanding of the human visual system.

7.1 Improving Classification

Our schemes depend on the use of a large number of training examples, and continuously improving the classifier by learning problematic configurations. It will be of interest to use large-scale experiments to better evaluate some of our results. For example, it will be of interest to get a more quantitative relation between the number of training images, number of features used, and the obtained error rate. An interesting experiment would be to use a very large number of samples for learning, and try to apply a configuration cascade of the improvement methods we suggested to try and drive the total error below one percent for example.

7.2 Improving the Anti Fragments concept

The anti fragment concept has not been exhausted, its main weakness is that it did not generalize well to new examples. Since the basic idea of inhibition is appealing, perhaps one possibility to overcome this is to change the feature space into something that is more robust, for example, perhaps edge features or some curve model may generalize better. It is also natural to combine the use of anti-fragments with our use of combination cascade: an anti-fragment may be associated with a particular faulty configuration, to help distinguish between class and non-class examples which all share the same feature configuration.

7.3 Extending the Cascade

We discussed two types of cascades and there are more interesting tests to both. The 3-way cascade, although similar to the usual cascade has some advantages. It would be interesting to compare it to similar and known methods.

Since the 3-way cascade can also be formulated as a 3-way decision tree, it would be interesting to combine classifiers using decision tree theory and to compare the thresholds selected for the second-level classifier using our approach and the decision tree approach.

The configuration cascades may also have interesting trails to pursuit. It would also be interesting to test the configuration cascade with a testing set that has non-class images that belong to a different class than the class learned, and see whether the configuration method can be used to overcome the non-class images as well as to cluster them into an unknown new class category correctly.

Another extension of the configuration cascade may be to apply it recursively. When a perfect linear separator is found for a certain configuration, we stop. However, when no perfect linear separator is found using the set of K additional features, we can apply the configuration cascade on the K features used in the second level recursively.

Another idea, is to use a different type of second-level classifier in the cascade using the idea of segmentation and testing whether an image that has a low confidence classification score can be segmented satisfactorily. This can also be combined with the configuration cascade more specifically, since a configuration of fragments induces some initial segmentation and the difficult configurations will be the ones segmented.

7.4 Human Vision Experiments

It will be of interest to use cognitive and psychophysics experiments to test some of our ideas, and to learn more about the human classification processes. Progress made in machine classfication in general, and some of the directions proposed in this work in particualr, could be used as a basis for asking specific questions about the human classification system.

One general experiment we suggest is comparing categorization errors of automatic algorithms and of humans on several types of categorization (such as planes, dogs, cows, faces, etc.). It would be very interesting to analyze the misses and false alarms made by machines and humans on the same dataset. By analyzing machine mistakes we learned a lot, and a study that will collect this type of data from human subjects may help shed light about human classification processes. It will also be of interest to collect classification errors early in life, to examine the learning process in early development during the initial formation of categories and concepts.

Another interesting direction would be to test the errors in categorization made by humans with known visual deficits. For example an experiment of face classification by humans with Prosopagnosia, a syndrome in which people have trouble recognizing human faces, or by humans with Capgras Syndrome, a rare disorder in which a person holds a delusional belief that an acquaintance, usually a close family member, has been replaced by an identical looking imposter, might provide different types of errors than normal humans enabling to gather more insight about the underlying processes.

Another interesting path is to design experiments that try to gather evidence for validation in the human visual system or in human classification. This may be approached, for example, by comparing classification errors made from a brief exposure followed by a mask (which will allow only early classification stages to be applied), with errors made under extended viewing times.

8 Appendix

8.1 Max-Min Fragment Selection and Bayesian Classification

Formally, let f_i be some fragment, S_{f_i} the vector of maximum similarity with each sample, C the vector of labels for each sample. The binary feature is computed as in equation 1

$$f_i(q_i) = \begin{cases} 1 & if \quad S_{f_i} > q_i \\ 0 & otherwise \end{cases} \qquad (1)$$

The threshold θ is selected to maximize the mutual information computed in equation 2

$$I(f_i(q_i);C) = \sum_{\substack{f_i=\{0,1\} \\ C=\{0,1\}}} p(f_i,C)\log\left(\frac{p(f_i,C)}{p(f_i)p(C)}\right) \qquad (2)$$

Then the fragments are selected to maximize the mutual information with the class but minimize the dependency between the features. If K is the set of all fragments and F is the set of selected fragments then

$$f{=}argmax_{f_i\in K} I(F,f_i;C)\text{-}I(F,C) \qquad (3)$$

To minimize 3 is computationally hard and therefore the pair wise approximation is used as in equation 4:

$$f = \arg\max_{f_i\ K} \min_{f_j\ F} I(f_i,f_j;C) - I(f_j,C) \qquad (4)$$

The selection process ends when the best fragment adds no additional information or at some predetermined maximal feature number.

To combine the fragments selected into a single score a generative naïve Bayes model is used under the assumption that the features are independent of each other. If the selected features are $F{=}f_1,f_2,...,f_k$, the final score is computed as in equation 5

$$score = \frac{P(C=1 \mid f_1,f_2,...,f_k)}{P(C=0 \mid f_1,f_2,...,f_k)} \qquad (5)$$

that under the assumption of independence becomes

$$score = \frac{\prod_{i=1}^{k} P(C=1 \mid f_i)}{\prod_{i=1}^{k} P(C=0 \mid f_i)} \qquad (6)$$

And in log form the simplest form the score becomes:

$$score = \frac{\sum_{i=1}^{k} \log P(C=1 \mid f_i)}{\sum_{i=1}^{k} \log P(C=0 \mid f_i)} \qquad (7)$$

8.2 AdaBoost

AdaBoost linearly combines weak hypothesis by weighting the training set and concentrating on the errors at each stage. At each iteration round $t = 1,...T$ the algorithm maintains a distribution or weights over the training set denoted as $D_t(i)$. Initially all weights are equal but on each round the weights of incorrectly classified examples increase causing the next weak learner to focus on the hard examples. Below we present the full algorithm by Freund and Schapire [13]:

Given: $(x_1,y_1),...,(x_m,y_m)$ where $x_i \in X, y_i \in Y = \{-1,+1\}$

Initialize $D_1(i)=1/m$

For $t=1,...,T$:

1. Train weak learner using the distribution D_t
2. Get weak hypothesis $h_t:X\rightarrow\{-1,+1\}$ with error

$$\varepsilon_t = \Pr_{i\sim D_t}[h_t(x_i) \neq y_i] \qquad (8)$$

3. Choose $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$
4. Update:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & h_t(x_i) = y_i \\ e^{\alpha_t} & h_t(x_i) \neq y_i \end{cases} = \frac{D_t(i)\exp(-\alpha y_i h_t(x_i))}{Z_t} \qquad (9)$$

Where Z_t is a normalization factor

Output the final hypothesis: $H(x)=sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$ (10)

8.3 Anti-Fragments

Anti fragments are fragments that are similar to a class patch and dissimilar from non-class patches that are similar to the class patch, trey are a way to learn more from the errors. The algorithm we used to find anti-fragments is detailed below:

1. Select normal fragments by some method (max-min here) and generate a classifier (Naive Bayes here)

2. Select a threshold θ that will generate twice as many false positives then false negatives in the training set. Let *FA* be the set of false positive images.

3. For every image $i \in FA$, let F be the set of fragments detected in i:

    (a) Sort the set of F according to w the naive Bayes weights

    (b) For every $f_i \in F$ according to the sorted order, let a_i be the patch detected as similar to f_i, do:

        i. Let af_i as: $af_i = \begin{cases} 1 & f_i(\theta_f) = 1 \wedge a_i(\theta_a) = 0 \\ 0 & otherwise \end{cases}$

        ii. Find θ_a that maximizes $I(af_i;C)$

        iii. If $I(af_i;C) > I(f_i;C) + \varepsilon$ then $f_i = f_i \cap \neg a_i$

        iv. If f_i changed, check if image i is still misclassified, if yes continue to next a_i if no continue to find anti fragments for next f_i

8.4 Sattelite Fragments as a Second Stage

Here we describe in detail the algorithm for finding satellite fragments to use for validation.

In the initial learning stage, the samples are classified according to some small *k* number of 'anchor' fragments. Using these anchor fragments a geometrical model is computed as in [16] and patches are extracted from approximately the correct locations in the images and tested as satellites for verification. Since many of the non-class images contain no anchor fragments (in contrast to the original similar classes setting), the non-class set of patches for learning could be very sparse, to avoid this, patches from random locations are extracted from the non-class training set, since theoretically there shouldn't be any specific pattern at the candidate satellite location in the non-class, figure X shows a sample of these patches.

The satellite is a semantic fragment as described in [17] that allows a very high detection rate in the class and a low detection rate in the non-class random patches. To avoid over fitting we split the training set into a test set and a validation set (figureX). Each satellite has an alpha score, which enables selecting those satellites that will be found with high probability in the class and with low probability in the non-class. The satellites are created to optimize this value and the semantic brothers are added using a greedy search detailed below.

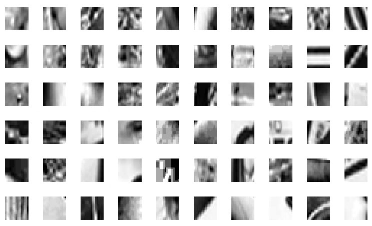


Figure 19: **Random patches.** A sample of random patches selected from the non-class images

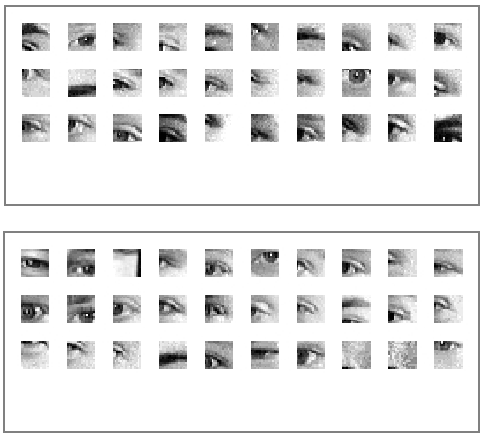


Figure 20: **Similar appearances.** We select the similar appearances of a specific location using the model geometry, and split this set into a training set (top) and a validation set (bottom).

When creating a satellite candidate, the threshold for each appearance from the training set is the minimum correlation over all the validation set patches. To allow variability in the class and keep the threshold tight so that it will allow differentiating between the class and non-class more than one appearance is added to the satellite. To find how close a patch in the validation set is to the satellite we find its closest match in the semantic fragments that compose the satellite and use the correlation with this closest appearance as the correlation of the validation patch, We set the satellite fragment's threshold θ_S to the minimum of all correlations with the validation set to allow detection of all validation patches. Then we check what fraction of non-class patches, α_S, this threshold detects. More formally, for a set of selected similar appearances *S* and the set of validation appearances *F'*, the threshold θ_S is set to:

$$\theta_S = \min_{f' \in F'} \max_{f \in S} Corr(f', f) \qquad (12)$$

For the set *P* of random patches α_S is defined to be the fraction of detected random patches:

$$a_S = \frac{|\{p : \max_{f\,S} Corr(f, p) > q_S\}|}{|P|} \qquad (13)$$

A good satellite fragment for validation is a fragment that using the same threshold that allowed detecting the satellite in 100% of the validation set is hardly ever detected in the set of random patches. Therefore the satellite fragments for validation are those with low α values. Below we summarize the selection process of the best satellite fragments for validation.

Input: A training set *C* of class images and a training set *NC* of non-class images.

Output: A set of *F* of anchor fragments, a model *M*, and a set *S* of satellite fragments sorted by their α values.

Algorithm:

1. Find a set *F* and a model *M* according to [16]
2. Create a list *L* of possible locations for satellite fragments on the model image.
3. Create a set *P* of random patches of the tested size by randomly cutting them from the set *NC*. The size of *P* should be at least 5|*C*|.

4. Select the next location $l \in L$ and a size to test (we used 16 by 16 pixels in our experiments)

5. Create a set of similar appearances for l using the model M and split it into two sets randomly. A testing set F and a validation set F'.

6. Initialize the current satellite $s=\varnothing$

7. For every $f \in F$

    (a) Let $s'=s \cup f$

    (b) compute $q_{s'} = \min_{f' F'} \max_{f s'} Corr(f', f)$

    (c) compute $a_{s'} = \dfrac{|\{p : \max_{f s'} Corr(f, p) > q_{s'}\}|}{|P|}$

    (d) Update f_{best} as f with the smallest $\alpha_{s'}$

    If the smallest $\alpha_{s'}$ is smaller than α_s, let $s=s \cup f_{best}$ remove f from F and return to step 7 to try and add more fragments to the current satellite. If the smallest $\alpha_{s'}$ is larger or equal to α_s and α_s>0.5 add s to the set of satellites S, either way, continue to step 4 to check the next satellite candidate.

When all locations have been tested, sort the set S from smallest α value to the largest

The classification phase is a two stage process. At the end of the learning there are k basic fragments and a geometric model for them, and also a set of $|S|$ satellite fragments for validation sorted from best to worst using their α values. When receiving a query image the first stage uses the basic fragments also referred to as anchor fragments to compute the likelihood score for the query image. If the likelihood score is below some threshold T_1 for the first stage we reject the image, if not we can continue to compute the model of the image using the learned model and the anchor fragments. We now validate the image using the satellite fragments and the assumed location of the satellites according to the model. We set the threshold T_1 for the first stage by requiring that there will be no miss images or a small tolerable percent of miss images in the training set. A similar approach is used in [5] when classifying with a boosted cascade.

There are two possible approaches to combine the validation satellites into a single score:

- Hard Decision: In this approach the number of *s* satellites from the ordered set of satellites *S* is determined in advance and require that each one of them will be found in the query image. If any one of them is not found, the image is rejected.
- Soft Decision: In this case we check for the detection of each of *s* satellite fragments, and combine this data using a learned probabilistic model, for example Naive *Bayes*.

We used the soft decision in the results we show.

The size of *S* in both approaches is directly connected to the confidence level we require of the classifier. The more confident we want to be when accepting an image the larger *S* should be. *S* also effects the computation time it takes to classify an image. The user may determine the required error rate relative to the required query time.

8.5 Three-Way Cascade, Determining the Processing Region

In this section we detail a method that can be used to determine the thresholds for further processing in a three-way cascade. We begin by describing the two stage case, which can be extended to more than two stages.

Let C_1 be the first stage classifier with an error rate of ε_1, a cost of S_1 and response R_1, Let C_2 be the second stage classifier with an error rate $\varepsilon_2<\varepsilon_1$, a cost $S_2\geq S_1$ and a response R_2. Also, let *p* be the probability of the sample being class in a binary classification. We require that the first stage classifier is a strong classifier in the sense that it has some δ percent of its extreme response range with no errors. We can also state this as $\delta^{low}+\delta^{high}=\delta$ where δ^{low} is the percent of the low classifier's responses that does not err and δ^{high} is the percent of the high classifier's response that does not err; thus we do not require the classifier to have symmetric errors, although in most classifiers this can be adjusted by changing the final decision threshold. To determine the thresholds we compute SR_1, a vector with the train response R_1 of C_1 sorted in descending order. Set

$T^{high} = SR_1[\delta^{high} \times p]$ and $T^{low} = SR_1[|SR_1| - \delta^{low} \times (1-p)]$. Below we describes the algorithm more generally.

Another view of three-way cascades is that we use the response of a classifier not only as a yes/no response that is based on some optimal threshold, rather, we view it as a confidence vote and we find a threshold for how confident the classifier needs to be to respond immediately.

Another trivial way to consider validation would be to combine different classifiers in a classification tree, also known as decision trees [23]. The analogue to our description is a 3 way tree, which receives the output of a 'strong' classifier and has three nodes: Yes, and No, where the classification ends, or Need Validation, where the classification precedes using another, orthogonal or simply better classifier. Figure 46 shows this schematically. There are decision tree algorithms that can learn the thresholds to optimize the final classification using these types of trees, and we do no elaborate on this idea, only point to the connection of a three-way tree to three way cascade.

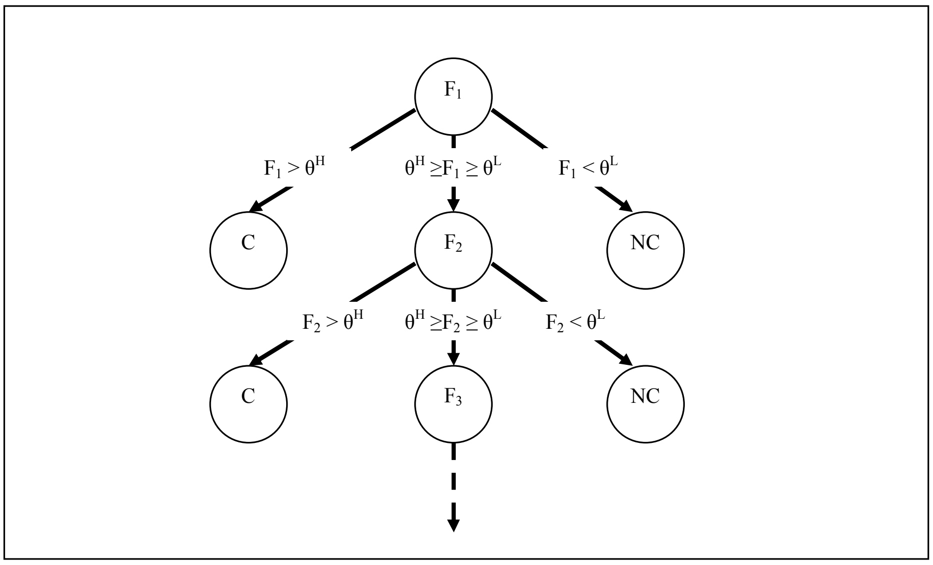


Figure **46: Three-way cascade as a decision tree.**

Determining the thresholds in general:

For a set of strong classifiers $\{C_1,...,C_m\}$ with error rates: [epsilon]$_1$>...>[epsilon]$_m$ with verified δ zones: $\{(\delta_1^{high},\delta_1^{low}),...,(\delta_m^{high},\delta_m^{low})\}$:

1. Let SR_i be the response of C_i sorted descending, and p the class prior then

T_1^{high}=∞ and T_1^{low}=-∞

For i>1 let: $T_i^{high}=SR_{i\text{-}1}(\delta_i^{high}\times p)$ and

$T_i^{low}=SR_{i\text{-}1}(|SR_{s\text{-}1}|\text{-}\delta_i^{low}\times(1\text{-}p))$

2. The validation classifier HH is defined recursively, i the index of the validation stage:

- If j=1 then $HH=C_1(x_i)$
- otherwise:

$$HH_j=\begin{cases} Max(C_j) & ifHH_{j-1}>T_j^{high} \\ Min(C_j) & ifHH_{j-1}<T_j^{low} \\ C_j(x_i) & otherwise \end{cases} \quad (14)$$

8.6 Online Three-Way Cascade using AdaBoost

Here we detail the algorithm we used to train online classifiers using AdaBoost and combine them in a three-way cascade. Possible variations may be to use bagging to sample different training sets for more iterations, stopping when there is no improvement.

The algorithm:

Split the training set (class and non-class samples) T into S mutually exclusive training sets of size reasonable for training the strong classifier of your choice (in this case AdaBoost).

Choose k, the number of features to add at each phase.

For each stage s=1,...,S:

1. On the training set T_s we train an AdaBoost classifier H_s similar to [13]:

- The pool of fragments available from the train samples are viewed as weak hypothesis h for the purpose of AdaBoost and at each iteration the selected fragment is the one with the minimum [epsilon]$_t$. The threshold θ was selected to maximize the mutual information with the class, but it can also be selected to optimize [epsilon]$_t$ which gives slightly better results.
- Force initial selection of features $\{f_1,...,f_{(s-1)k}\}$ and then choose additional k features
- The alpha weights are calculated according to the current training set

$$H_s = \sum_{j=1}^{sk} \alpha_{sj} f_j$$

2. Let $R_{s\text{-}1}$ be the final responses of the previous stage, $-\sum_{j=1}^{(s-1)k} a_{(s-1)j} \le R_{s-1} \le \sum_{j=1}^{(s-1)k} a_{(s-1)j}$, p the class prior, find the current stage validation range by sorting $R_{s\text{-}1}$ to $SR_{s\text{-}1}$ and setting $T_s^{high} = SR_{s-1}(d_{high}\,\text{‘}\,p)$ and $T_s^{low} = SR_{s-1}(|SR_{s-1}| - d_{low}\,\text{‘}\,(1-p))$

   (We used $\delta_{high}=\delta_{low}=0.25$)

3. The final score HH for all samples is computed recursively as follows: if $s \le 1$ then $HH_s = H_s$ For all $s>1$

$$HH_s = \begin{cases} Max(H_s) & if \quad HH_{s-1} > T_s^{high} \\ Min(H_s) & if \quad HH_{s-1} < T_s^{low} \\ H_s(x_i) & \quad otherwise \end{cases} \qquad (15)$$

8.7 Configuration Cascade

In this section we describe our algorithm for combinatory cascade classification. Initially the algorithm is meant as a configuration (or combination) cascade.

Algorithm:

Choose k, the number of features for the base classifier, C_1.

Choose v, the number of features to use for the second-level of the cascade.

For all train samples $I_1,...,I_t$ classify using the basic k fragments, select a threshold θ_1 on the response of C_1 to achieve a low miss rate and some false alarm rate, the equal error rate is one possibility.

Generate a combination index for each image: for every image we have a binary vector of size k with the detected fragments, so the decimal value of this vector can be used as the image combination index CI_j.

For each train image that was misclassified, add a row to the Combination Database (CDB), and add the image to the row's image set. Add all other train images with the same j to the combination image set.

For each CI_j in the CDB, train an AdaBoost A_j classifier with v steps on the full training set, but initialize the distribution with half of the weight on the images with the CI_j. This creates a linear separator for each of the configurations.

Another option we used for efficiency in some of our tests is to train a classifier with v fragments on the full train set and use this classifier as the second-level of the cascade for all the combinations. This has the draw back that the different combination 'islands' may not have a single linear separator.

In the classification phase: for each image I apply the C_1 classifier, get j the CI of I and check if it exists in the CDB. If it is, apply A_j to validate it.

Note: if the dataset of a CI contains only Non-class images, A_j is not trained and is a trivial stage that rejects all images.

9 Acknowledgment

I would like to thank my advisor Shimon for his many contributions and his patience as I learned the many lessons I had to learn through this work. I would also like to thank Boris Epshtein for his help and the datasets he provided me with and also my group members

Dan Michael-Levi, Leonid Karlinsky, Shai Litvak and Sharon Gilae-Dotan for their advise, discussions and helpful material. I also thank Ita Lifshitz who helped me find my way in the beginning and provided me with usefull datasets, and my many friends who were there for me among them Tali Sadka, Naama Swerdelin, Michal Golan Masiach, Lena Gorelick, Shira Kritchman, Ronen Gradwohl and many more. Finally I would like to thank my family for their amazing support through out my work.

10 References

[1] A. Adler, M. Schuckers, *Comparison of Human versus Automatic Face Recognition Performance*, Submitted May 2006,

[2] S. Ullman, E. Sali, M. Vidal-Naquet, *A Fragment-Based Approach to Object Representation and Classification*, IWVF 2001: 85-102.

[3] M. Vidal-Naquet, S. Ullman, *Object Recognition with Informative Features and Linear Classification*, International Conference on Computer Vision (ICCV) 2003: 281-288

[4] S. Ullman, M. Vidal-Naquet, E. Sali, *Visual Features Of Intermediate Complexity And Their Use In Classification* , Nature NeuroScience 2002: Vol. 5, No. 7, pp: 682 - 687.

[5] P. Viola, M. Jones, *Rapid Object Detection Using a Boosted Cascade of Simple Features*, IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR) 2001: 511-518.

[6] P. Viola, M. Jones, *Fast and Robust Classification using Asymetric AdaBoost and a Detector Cascade*, Advances in Neural Information Processing System 14, MIT Press, Cambridge, MA, 2002

[7] D. G. Lowe, *Object recognition from local scale-invariant features*, International Conference on Computer Vision 1999: 1150-1157.

[8] Face Detection and Face Recognition Betaface live demo, maintained by Oleksandr Kazakov at http://www.betaface.com.

[9] C. Shmid, R. Mohr, *Local grayvalue invariants for image retrieval*, IEEE PAMI 1997: 530-534.

[10] R. Fergus, P. Perona, A. Zisserman, *Object Class Recognition by Unsupervised Scale-Invariant Learning*, IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR) 2003: 264-271.

[11] F. Fleuret, *Fast Binary Feature Selection with Conditional Mutual Information*, Journal of Machine Learning Research 5 2004: 1531-1555.

[12] N. C. Oza, S. Russel, *Online Bagging and Boosting*, IEEE International Conference on Systems, Man and Cybernatics 2005: 2340-2345.

[13] Y. Freund, R. E. Schapire, *A Decision-Theoretic Generalization of on-line learning and an application to boosting*, Journal of Computer and System Sciences 1997: 55(1) 119-139.

[14] D. Levi, S. Ullman, *Learning to Classify by Ongoing Feature Selection*, Proceeding of Canadian Conference on Computer and Robot Vision 2006

[15] S. Das, *Filters, Wrappers and a Boosting-Based Hybrid for Feature Selection* , In Proceedings of the Eighteenth international Conference on Machine Learning 2001

[16] B. Epshtein, S. Ullman, *Satellite Features for the Classification of Visually Similar Classes*, CVPR 2006

[17] B.Epshtein, S. Ullman, *Identifying Semantically Equivalent Object Fragments*, IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR) 2005: Vol. 1, 2-9.

[18] B.Epshtein, S. Ullman, *Feature Hierarchies for Object Classification*, IEEE International Conference on Computer Vision (IICCV) 2005: 220-227.

[19] V. Kyrki, J. Kamarainen, *Simple Gabor feature space for invariant object recognition*, Pattern Recognition Letters 2004: 311-318.

[20] M. Hamouz, J. Kittler, *Feature-Based Affine-Invariant Localization of Faces* IEEE Transactions on Pattern Analysis and Machine Intelligence 2005: Vol 27, No. 9

[21] Check at home what paper you used

[22] T. Joachims, *Making large-Scale SVM Learning Practical. Advances in Kernel Methods - Support Vector Learning*, B. Schelkopf and C. Burges and A. Smola (ed.), MIT-Press, 1999

[23] R.O. Duda, P.E. Hart, D.G. Stork, Pattern Classification 2000, 2nd ed. John Wiley & Sons, Inc., New York

[24] R. E. Schapire, Y. Freund, P. Bartlett, W. S. Lee *Boosting the Margin: A New Explanation for the Effectiveness of Voting Methods*, Machine Learning: Proceedings of the Fourteenth International Conference 1997.

[25] Y. Freund, R. E, Schapire, *A Short Introduction to Boosting*, Journal of Japanese Society for Artificial Intelligence 1999, 14(5): 771-780.

[26] M. Fabre-Thorpe, G. Richard, S. J. Thorpe, *Rapid Categorization of Natural Images by Rhesus Monkeys*, Neuroreport 1998: 303-308.

[27] A. Treisman , G. Gelade, *A Feature Integration Theory of Attention*, Cognitive Psychology 12, 1980: 97-136.

[28] J. M. Wolfe, K. R. Kluender, D. M. Levi, L. M. Bartoshuk, R. S. Herz, R. L. Klatzky, & S. J. Lederman Sensation & Perception 2006, Sunderland, MA: Sinauer Associates, Inc.

[29] M. Ahissar, S. Hochstein, *The Reverse Hierarchy Theory of Visual Perceptual Learning*, Trends in Cognitive Neurosciences 2004: 457-464.

[30] K. Tsunoda, Y. Yamane, M. Nishizaki, M. Tanifuji, *Complex Objects are Represented in Macaque Inferotemporal Cortex by the Combination of Feature Columns*, Nature Neuroscience 2001: 832-838.