A Comparison of Generic Machine Learning Algorithms for Image Classification

Raphaël Marée, Pierre Geurts, Giorgio Visimberga [†],
Justus Piater, Louis Wehenkel
Montefiore Institute, University of Liège
Liège, Belgium
http://www.montefiore.ulg.ac.be/~maree/

[†] On leave from Politecnico Di Bari
Bari, Italy

Abstract

In this paper, we evaluate 7 machine learning algorithms for image classification including our recent approach that combines building of ensembles of extremely randomized trees and extraction of sub-windows from the original images. For the approach to be generic, all these methods are applied directly on pixel values without any feature extraction. We compared them on four publicly available datasets corresponding to representative applications of image classification problems: handwritten digits (MNIST), faces (ORL), 3D objects (COIL-100), and textures (OUTEX). A comparison with studies from the computer vision literature shows that generic methods can come remarkably close to specialized methods. In particular, our sub-window algorithm is competitive with the state of the art, a remarkable result considering its generality and conceptual simplicity.

1 Introduction

Image classification is an important problem which appears in many application domains like quality control, biometry (face recognition), medicine, office automation (character recognition), geology (soil type recognition)...

This problem is particularly difficult for traditional machine learning algorithms mainly because of the high number of input variables that may describe images (i.e. pixels). Indeed, with a high number of variables, learning methods often suffer from a high variance (models are very unstable) which deteriorates their accuracy. Futhermore, computing times can also be detrimental in such extreme conditions. To handle this high dimensionality, image classication systems usually rely on a pre-processing step, specific to the particular problem and application domain, which aims at extracting a reduced set of interesting features from the initial huge number of pixels. The limitation of this approach is clear: When considering a new problem or application domain, it is necessary

to manually adapt the pre-processing step by taking into account the specific characteristics of the new application.

At the same time, recent advances in automatic learning have produced new methods which are able to handle more and more complex problems without requiring any a priori information about the application. These methods are increasingly competitive with methods specifically tailored for these domains. In this context, our aim in this paper is to compare several of these recent algorithms for the specific problem of image classification. Our hypothesis is that it will be possible to obtain with some of these approaches competitive results with specialized algorithms without requiring any laborious, manual pre-processing step.

In this goal, our approach was to choose several problems which we think are representative of the image classification domain and then to apply several learning algorithms on each of these problems, without any specific pre-processing, i.e. by directly using the pixel values. Among recent learning algorithms potentially able to handle this complex problem, we have chosen a panel of 7 algorithms, including one generic approach that we have recently proposed for image classification [17]. These algorithms will be compared essentially on two criteria: accuracy of the models and computational efficiency (of the learning and test phases). Although several of these algorithms have been already applied to image classification, usually these studies either do not compare several algorithms or are focused on only one particular application problem.

The paper is structured as follows. In Section 2, we briefly describe the machine learning algorithms chosen for our comparison. Of course, we spend more time on the description of our own proposal. The essential characteristics of the four datasets used for the comparison are summarized in Section 3. The experimentation protocols and the results of the experiments are discussed in Section 4. We end the paper with our conclusions and discusssions about future work directions.

2 Algorithms for generic image classification

The input of a generic learning algorithm for image classification is a training set of pre-classified images,

$$LS = \{(A^i, c^i), i = 1, \ldots, N\}$$

where A^i is a $W_x \times W_y$ matrix describing the image and $c^i \in \{1, \ldots, M\}$ is its classification (among M classes). The elements $a^i_{k,l}$ of A^i ($k = 1, \ldots, W_x$, $l = 1, \ldots, W_y$) describe image pixels at location (k, l) by means of an integer value in the case of grey level images or by means of 3 integer RGB values in the case of color images.

Then, to handle information from pixels without any pre-processing, the learning algorithm should be able to deal efficiently with a large amount of data, first in terms of the number of images and classes of images in the learning

set, but more importantly in terms of the number of values describing these images (i.e. the attributes). Assuming for example that $W_x = W_y = 128$, there are already $128 * 128 = 16384$ integer values describing images and this number is further multiplied by 3 if colors are taken into account.

We describe below seven classification algorithms that we think could work in such difficult conditions and that we have compared in our experiments. The common characteristics of these methods is that they are essentially non parametric and that they can efficiently handle very large input spaces.

2.1 Decision trees

Decision tree induction [3] is one of the most popular learning algorithms with nice characteristics of interpretability, efficiency, and flexibility. However, the accuracy of this algorithm is often not competitive with other learning methods due to its high variance [9]. In this study, we therefore do not expect decision trees to be satisfactory but we still evaluate it as it is the basis of other promising and recent algorithms.

2.2 Ensemble of decision trees

Ensemble methods are very popular in machine learning. These methods improve an existing learning algorithm by combining the predictions of several models obtained by perturbing either the learning set or the learning algorithm parameters. They are very effective in combination with decision trees that otherwise are often not competitive with other learning algorithms in terms of accuracy. Several ensemble methods have been applied to image classification problems, either in combination with traditional algorithms or with ad hoc computer vision system (e.g. in [7] and [12]). In this paper, we propose to compare four different ensemble methods based on decision trees. Two of these methods are the now famous bagging and boosting techniques. The two other methods, random forests and extra-trees, are two recent methods that essentially improve bagging in terms of accuracy but also in terms of computational efficiency. As the extra-trees method is our own proposal, we give below a more detailed description of it.

2.2.1 Bagging

Bagging [1] (for "boostrap aggregating") consists in drawing T boostrap learning samples from the original learning set (by random re-sampling without replacement) and then in producing from each of them a model using the classical decision tree algorithm. Then, a prediction is computed for a test instance by taking the majority class among the predictions given by the T trees for this instance.

2.2.2 Random forests

With Random Forests [2], each of the T trees is grown on a bootstrap sample of the original learning set like in bagging. But here, during tree construction, at each node of the tree, only a small number (k) of attributes randomly selected among the whole set of attributes is searched for the best test. In [2], it has been shown that random forests give better results than bagging and often yield results competitive with boosting (see below). It is also faster than these two algorithms since it requires to consider only a small subset of all attributes when developping one node of the tree.

2.2.3 Extremely randomized trees

Extremely randomized trees [9], [10] (extra-trees in short) are another ensemble method for decision trees that is extreme in terms of the randomization introduced when growing the trees of the ensemble. Indeed, an extremely randomized tree is grown by selecting at each node of the tree the parameters of the test fully at random. In the context of image classification, this yields the very simple recursive function shown in Table 1 to build an extra-tree. Several extra-trees are then built according to this algorithm and their predictions are aggregated just like in other ensemble methods. Experiments in [9] have shown that this method gives better results than bagging and is also competitive with boosting. Its main advantage with respect to other ensemble methods for decision trees is that it is also extremely fast. The complexity of the algorithm of Table 1 is independent of the number of attributes and, like other decision tree based algorithms, it is (empirically) linear with respect to the learning sample size.

2.2.4 Boosting

Boosting also builds an ensemble of decision trees but, contrary to previous ensemble methods, it produces the models sequentially and is a deterministic algorithm. It sequentially applies the learning algorithm to the original learning sample by increasing weights of misclassified instances. So, as the iteration proceeds, the models are forced to focus on the "difficult" instances. Several variants of this algorithm have been proposed in the literature. In our experiment, we will use the original algorithm, called AdaBoost.M1, described in [8] and we will apply it to decision trees.

2.3 Support Vector Machines

Support Vector Machine (SVM) is a machine learning algorithm originally motivated by advances in statistical learning theory [24]. It first applies a transformation of the initial input space into a new potentially very high dimensional transformed input space where classes are very likely to be linearly separable and then deriving a hyperplane to separate each pair of classes in this transformed input space. There exist several efficient implementations of this algo-

Build_extra_tree(input: a learning sample, LS):

- If LS contains images all of the same class, return a leaf with this class associated to it;
- Otherwise:
  1. Set $[a_{k,l} < a_{th}]$=Choose_a_random_split(LS);
  2. Split LS into LS_{left} and LS_{right} according to the test $[a_{k,l} < a_{th}]$ and build the subtrees $\mathcal{T}_{left}$ = build_extra_tree(LS_{left}) and $\mathcal{T}_{right}$ = build_extra_tree(LS_{left}) from these subsets;
  3. Create a node with the test $[a_{k,l} < a_{th}]$, attach $\mathcal{T}_{left}$ and $\mathcal{T}_{right}$ as successors of this node and return the resulting tree.

Choose_a_random_split(LS):

1. Select a pixel location (k, l) at random;
2. Select a threshold a_{th} at random according to a distribution $N(\mu_{k,l}, \sigma_{k,l})$, where $\mu_{k,l}$ and $\sigma_{k,l}$ are resp. the mean and standard deviation of the pixel values $a_{k,l}$ in LS;

Table 1: Extra-tree induction algorithm for image classification

rithm. In our experiment, we use the algorithm proposed in [5] with Gaussian and polynomial kernels. SVMs already gave very impressive results in terms of accuracy and computational efficiency in many complex domains including image classification problems (e.g. in [4], [11] or [23]).

2.4 Extra-trees with sub-window extraction

Even though ensemble methods with decision trees can handle a very large number of input variables efficiently (especially our extra-trees), the tree complexity (and, hence, the number of pixels that are combined along a path from the root node to a leaf in the tree) is limited by the size of the learning set. When the number of images is small compared to the total number of pixels, a tree cannot combine enough pixels to provide acceptable models. To solve this problem, we have adopted another generic approach that is popular in image classification (e.g. [13] and [6]). It artificially augments the number of images in the learning set by building models from sub-windows extracted from original images of the learning set.

Although it can be combined with any learning algorithm, we have combined this idea with extra-trees, essentially for computational efficiency reasons. In this variant, the construction of one extra-tree from the ensemble is carried out in two steps, given a window size $w_1 \times w_2$ and a number N_w:

- Extract N_w sub-windows at random from training set images (by first selecting an image at random from LS and then selecting a sub-window at a random

Table 2: Database summary

DBs	# images	# attributes	# classes
MNIST	70000	784 ($28 * 28 * 1$)	10
ORL	400	10304 ($92 * 112 * 1$)	40
COIL-100	7200	3072 ($32 * 32 * 3$)	100
OUTEX	864	49152 ($128 * 128 * 3$)	54

location in this image) and assign to each sub-window the classification of its parent image;

- Grow an extra-tree to classify these N_w sub-windows by using the $w_1.w_2$ pixel values that characterize them.

To make a prediction for an image with an ensemble of extra-trees grown from sub-windows, the following procedure is used:

- Extract all possible sub-windows of size $w_1 \times w_2$ from this image;
- Assign to the image the majority class among the classes assigned to the sub-windows by the ensemble of extra-trees.

3 Four image classification problems

To evaluate the machine learning algorithms for generic image classification and to allow replication of our experiments, we selected four publicly available datasets corresponding to common image classification problems: recognition of handwritten characters (here, digits), faces, objects, and textures. The main characteristics of the datasets are summarized in Table 2 and an overview of their images is given in Figure 1. We briefly describe each problem below.

3.1 MNIST, database of handwritten digits

The MNIST database[1] [16] consists of 70000 handwritten digits that have been size-normalized and centered in images of 28×28 pixels with 256 grey levels per pixel. The goal is to build a model that classifies digits. Different writing styles are characterized by thin or thick strokes, slanted characters, etc.

3.2 ORL, face database

The ORL database[2] from AT&T contains faces of 40 distinct persons with 10 images per person that differ in lighting, facial expressions (open/closed eyes, smiling/not smiling), facial details (glasses/no glasses) and contain minor variations in pose. The size of each image is 92×112 pixels, with 256 grey levels per pixel. The goal is to identify faces.

[1] http://yann.lecun.com/exdb/mnist/
[2] http://www.uk.research.att.com/facedatabase.html

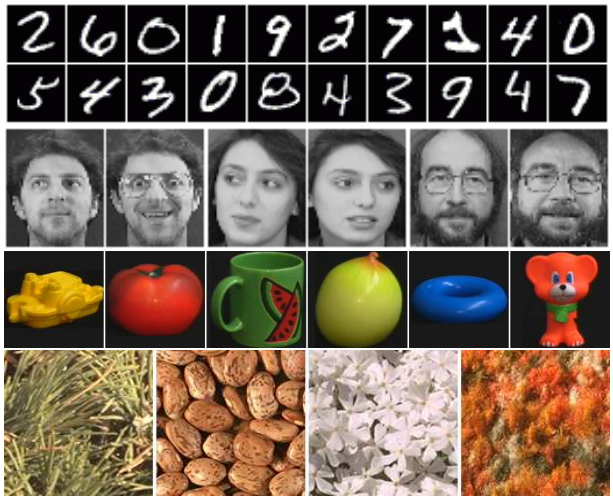

Figure 1: Overview of the four databases: MNIST, ORL, COIL-100, OUTEX

3.3 COIL-100, 3D object database

The Columbia University Object Image Library[3] COIL-100 is a dataset with colored images of 100 different objects (boxes, bottles, cups, miniature cars, etc.). Each object was placed on a motorized turntable and images were captured by a fixed camera at pose intervals of 5 degrees. This corresponds to 72 images per object. In COIL-100, each image has been normalized to 128×128 pixels and are in true color. For our experiments, we have resized the original images down to 32×32 pixels.

3.4 OUTEX, texture database

Outex[4] [21] provides a framework for the empirical evaluation of texture analysis algorithms. The Contrib_TC_0006 dataset we took from Outex has been derived from the VisTeX dataset. It contains 54 colored textures and 16 images of 128×128 pixels in true colors for each VisTex texture.

4 Experiments

In this section, the seven classification algorithms are compared on the four problems. Before describing the experimentation protocol and discussing the results, the next subsection discusses some implementation details and the way we have tuned the different parameters of the methods.

[3] http://www.cs.columbia.edu/CAVE/
[4] http://www.outex.oulu.fi/

4.1 Implementation and determination of the parameters

For all algorithms except for SVM, we have used our own software which is implemented in C. For SVM, we have used the LibSVM [5] package which is a C++ implementation of the algorithm presented in [5].

For each machine learning method, the values of several parameters need to be fixed. We discuss this tuning stage for each method below. For some algorithms, the values of the parameters are fixed on the basis of the resulting error on the test sample. We are aware that this would lead to slightly underestimated error rates, however, we believe that this will not be detrimental for comparison purposes, especially since the number of parameters in each method is quite small.

Classical decision trees are fully developed, i.e. without using any pruning method. The score measure used to evaluate tests during the induction is the score measure proposed in [25] which is a particuler normalization of the information gain. Otherwise our algorithm is similar to the CART method [3].

Ensemble methods all are influenced by the number of trees T which are aggregated. Usually, the more trees are aggregated, the better the accuracy. So, in our study, we have used for each problem and for each algorithm, a number of trees that appeared to be large enough to give stable error rates on the test samples. Usually, extra-trees that are more randomized requires more trees than the other variants (from 50 on MNIST to 500 on ORL and COIL-100). Extra-trees with sub-windows however are stabilized much sooner (10 trees are sufficient in all problems).

Random forests depends on an additional parameter k which is the number of attributes randomly selected at each test node. In our experiments, its value was fixed to the default value suggested by the author of the algorithm which is the square root of the total number of attributes. According to [2] this value usually gives error rates very close to the optimum.

Boosting does not depend on another parameter but it nevertheless requires that the learning algorithm does not give perfect models on the learning sample (so as to provide some misclassified instances). Hence, with this method, we used with decision trees the stop-splitting criterion described in [25]. It uses an hypothesis testing based on the G^2 statistic [14] to determine the significance of a test. In our experiments, we fixed the nondetection risk α to 0,005.

For SVM, we used LibSVM with default parameters. We tried their implementation of linear, polynomial (with degree 2 and 3) and radial basis kernel functions. Again, the best kernel was choosen on the test sample.

For extra-trees on sub-windows, additional parameters are the size of sub-windows $w1 \times w2$ and the number N_w of them extracted during the learning phase. Like for the number of trees in the ensemble, accuracy appears to be a monotonically increasing function of N_w. On the last three problems, N_w was fixed to 120000. On MNIST, as the initial learning sample size is already quite large, we further increase this number to 360000. Accuracy is on the

[5] http://www.csie.ntu.edu.tw/~cjlin/libsvm/

Table 3: Results on all problems

MNIST

Algorithm	Error rate
Classical Decision Tree	11.5%
Bagging ($T = 50$)	4.42%
Extra-Trees ($T = 100$)	3.17%
Random Forests ($T = 100$)	3.0%
Extra-Trees + Sub-Window	2.54%
Boosting ($T = 50$)	2.29%
SVMs (poly2)	1.95%
LeNet-4[16]	0.7%

ORL

Algorithm	Error rate
Classical Decision Tree	29.25% $\pm$ 6.89
Bagging ($T = 50$)	9.5% $\pm$ 5.7
Boosting ($T = 50$)	3.75% $\pm$ 2.79
Random Forests ($T = 200$)	1.25% $\pm$ 1.68
Extra-Trees ($T = 500$)	1.25% $\pm$ 1.68
SVMs (linear)	1.25% $\pm$ 1.25
Extra-Trees + Sub-Window	0.5% $\pm$ 1.0
-	-

COIL-100

Algorithm	Error rate
Classical Decision Tree	20.80%
Bagging ($T = 50$)	2.24%
Extra-Trees ($T = 500$)	1.96%
Random Forests ($T = 500$)	1.17%
Boosting ($T = 100$)	0.54%
SVMs (linear)	0.44%
Extra-Trees + Sub-Window	0.35%
Local Affine Frames [20]	0.1%

OUTEX

Algorithm	Error rate
Classical Decision Tree	89.35%
Bagging ($T = 50$)	73.15%
SVMs (linear)	71.99%
Boosting ($T = 50$)	69.44%
Random Forests ($T = 1000$)	66.90%
Extra-Trees ($T = 1000$)	65.05%
Extra-Trees + Sub-Window	2.78%
RGB Histograms [18]	0.2%

other hand very much influenced by the size of sub-windows. The optimal size is problem-dependent and has been tuned manually on each problem.

4.2 Protocols and results

The test protocols are discussed in this section on each problem. Results are summarized in Table 3. For each problem, the methods are sorted by decreasing error rates.

4.2.1 MNIST

In the literature, the first 60000 images are often used for learning and the remaining 10000 examples are used for validation. In [16], the results of many learning methods are reported, they range from 12% with a one-layer neural network to 0.7% with the authors' method "Boosted LeNet-4". Our results are obtained by strictly following this protocol. Error rates with generic methods vary from 11.5% with a classical decision tree to 1.95% with support vector machines. Using sub-windows, the error rate is 2.54% (with $T = 10$, $N_w = 360000$ and $w_1 = w_2 = 24$) which is less accurate than Boosting and SVM. In [4], an error rate of 1.1% was obtained with another implementation of SVM.

4.2.2 ORL

In the literature, various algorithms with pre-processing steps have been tested on this dataset, including hidden Markov models [19], convolutional neural networks [15], SVMs [11], and variants of nearest neighbors [22]. But the protocol for testing is different from one paper to another. Given the fact that this database is quite small and as there is no well-defined test protocol,

our experiments use 10-fold cross-validation to provide a fair assessment of the generic methods. It means that the learning set was randomly partitioned into 10 learning samples with 360 images (9 views per subject) while the remaining images (1 view per subject) used for tests samples. Following this procedure, we get an average error rate (10 runs) of 29.25%, with a classical decision tree, down to 0.5% with our sub-windows (with $T = 10$ and $w_1 = w_2 = 32$). Random Forests, Extra-Trees, and SVM give a slightly inferior result with an average error rate of 1.25%.

4.2.3 COIL-100

This problem was approached in the literature with different methods, some of them specific to 3D object recognition, that use different matching techniques of local or global features (color histograms, eigenwindows, locale affine frames [20], etc.). For the learning sample, we took 18 views for each of the 100 objects, starting with the pose at 0 and then going on with intervals of 20. The remaining views were devoted to the test sample. Methods in the computer vision literature provide error rates from 12.5% to 0.1%. Using this protocol, classical decision tree yields an error rate of 20.80% that drops down to 0.35% with the combination of extra-trees and sub-windows (with $T = 10$ and $w1 = w2 = 16$). Boosting and SVMs are close to our approach with respectively an 0.54% and 0.44% error rate.

4.2.4 OUTEX

This dataset has a small number of objects but a very large number of attributes. The OUTEX framework precisely defines the images to use in the learning and test sample (8 images for each texture in both ensembles). The paper [18] evaluates several feature extraction techniques and image transformation methods in combination with a traditional nearest neighbors algorithm. Their resulting error rates on this dataset vary from 9.5% to 0.2%. Using the same protocol, most of the popular learning methods are especially bad with an error rate varying from 89.35% (classical decision tree) to 66.90% (random forests). Extra-trees are also not satisfactory with an error rate of 65.05% (even with $T = 1000$). On the other hand, sub-windows reduce error rates down to 2.78% (with $T = 10$ and $w1 = w2 = 4$). These results can be explained by the nature of the problem. As textures are generally based on the repetition of small patterns, one can extract small sub-windows from the original images that are quite well classified by models since they contain sufficient information to classify the whole image. This statement is further confirmed by the fact that small sub-windows of size 4×4 give the best results on this problem. On the other hand, extra-trees alone and other learning algorithms are not able to find relevant information among the high number of pixels describing the textures because the characteristics patterns are not associated with specific image locations.

4.3 Discussion

As expected, for each problem, classical decision trees are not satisfactory. This is explained by the high variance of this method. However, ensemble methods are well-known in the literature for improving accuracy and this is confirmed by our experiments. Indeed, Bagging and in a more impressive way Boosting give much more accurate results for each problem. Random forests and Extra-Trees are competitive with Boosting. SVM is very close to boosting but maybe slightly better in average. Extra-trees with sub-windows is the best generic method in terms of accuracy on three of the four problems (ORL, COIL-100, OUTEX) but it is nevertheless beaten by boosting and SVM on MNIST. On all problems, the best results, always obtained either by SVM or extra-trees with sub-windows, are competitive with state-of-the-art techniques in computer vision. This is remarkable considering that these algorithms are very easy to use.

Another criterion for comparing algorithms for computer vision is their computational efficiency. To give an idea of the difference between the algorithms, Table 4 reports the computing times [6] for learning each model on the COIL-100 and MNIST problems. Extra-trees is undoubtedly the fastest method. On COIL-100, growing 500 extra-trees is even faster than growing one single decision tree. Not surprisingly, bagging and boosting that build T trees with the classical decision tree induction algorithm are the slowest methods. Random forests are much faster but still slower than the extra-trees (especially on COIL-100). SVMs are very fast on COIL-100 but slower on MNIST which consists of much more images. Our sub-windows method increases significantly the computing times of extra-trees but the resulting algorithm is still much faster than bagging and boosting on COIL-100. On MNIST, high computing times are attributed to the augmented learning sample ($N_w = 360000$).

The prediction times for all tree-based algorithms are negligible. With sub-windows however, the test of a new image requires the propagation of all sub-windows in the trees of the ensemble and it depends mostly on the size of the sub-windows. For example, it takes about 54s to predict the classes of the 5400 test objects on COIL-100 and about 12s to test the 10000 images on MNIST. Although our implementation is not optimal, we believe that these times are nevertheless reasonable. With SVM, prediction times depends on the number of support vectors. On both problems, it was the slowest method for testing with a prediction time of 3m19s for COIL-100 and 8m09s on MNIST.

5 Conclusions

We compared seven generic algorithms for image classification including our recent approach that combines building of ensembles of extremely randomized trees and extraction of sub-windows from the original images. Classical decision tree, Bagging, Boosting, Random Forests and SVM are the other meth-

[6] On a Pentium IV 2.53Ghz.

Table 4: Learning time on MNIST and COIL-100

MNIST

Algorithm	Time
Boosting ($T = 50$)	6h22m29s
Extra-Trees + Sub-Window	5h06m24s
Bagging ($T = 50$)	5h01m11s
SVMs (poly2)	28m28s
Random Forests ($T = 100$)	20m16s
Extra-Trees ($T = 100$)	11m39s
Classical Decision Tree	7m17s

COIL-100

Algorithm	Time
Boosting ($T = 100$)	5h25m01s
Bagging ($T = 50$)	1h53m25s
Random Forests ($T = 500$)	51m34s
Extra-Trees + Sub-Window	45m45s
Classical Decision Tree	3m08s
SVMs (linear)	1m02s
Extra-Trees ($T = 500$)	9s

ods we evaluated on four different image classification problems for which test protocols were rigorously specified. The accuracy of our generic sub-window technique is the best on three of the four problems and is comparable to state-of-the-art techniques but slightly inferior to the best known results. In fact, our experiments demonstrate that generic methods, in particular our sub-window algorithm, can come remarkably close to specialized methods. In many practical application contexts, a slight performance drop in exchange for reduced task-specific pre-processing and manual intervention may constitute a very desirable trade-off.

The future directions are two-fold. First, as our wrapping method of extraction and classification of sub-windows from images is generic, it is attractive to combine it with other learning algorithms (in particular Boosting and SVMs). Second, experiments should be carried out to compare the robustness of these generic approaches to rotation, scaling, occlusion, and noise. Although some algorithms are close in terms of accuracy, it is not sure that they would be all affected in the same way by these perturbations. In [17], we provided a preliminary study of the behavior of our sub-window approach in the presence of rotation, scaling, and occlusion on the COIL-100 problem. We observed good robustness to small transformations introduced in test images and also suggested some improvements that preserve the generic nature of the algorithm.

6 Acknowledgments

Raphaël Marée is supported by the "Région Wallonne" (Belgium). Pierre Geurts is a research associate of the F.N.R.S., Belgium.

References

[1] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[2] L. Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

[3] L. Breiman, J.H. Friedman, R.A. Olsen, and C.J. Stone. *Classification and Regression Trees*. Wadsworth International (California), 1984.

[4] C. J. C. Burges and B. Schölkopf. Improving the accuracy and speed of support vector machines. In M. C. Mozer, M. I. Jordan, and Thomas Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9, page 375. The MIT Press, 1997.

[5] Chih-Chung Chang and Chih-Jen Lin. Libsvm : a library for support vector machines. Technical report, Computer Science and Information Engineering, National Taiwan University, 2003.

[6] J. Dahmen, D. Keysers, and H. Ney. Combined classification of handwritten digits using the 'virtual test sample method'. In *Proc. Second International Workshop, MCS 2001 Cambridge, UK*, pages 99–108, July 2001.

[7] H. Drucker. Fast decision tree ensembles for optical character recognition. In *Proc. Fifth Annual Symposium on Document Analysis and Information Retrieval*, pages 137–147, 1996.

[8] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *Proc. Thirteenth International Conference on Machine Learning*, pages 148–156, 1996.

[9] P. Geurts. *Contributions to decision tree induction: bias/variance tradeoff and time series classification.* Phd. thesis, Department of Electrical Engineering and Computer Science, University of Liège, May 2002.

[10] P. Geurts. Extremely randomized trees. Technical report, Department of Electrical Engineering and Computer Science, University of Liège, 2003.

[11] G.-D. Guo, S. Li, and K. Chan. Face recognition by support vector machines. In *Proc. International Conference on Automatic Face and Gesture Recognition, 196-201.*, 2000.

[12] G.-D. Guo and H.-J. Zhang. Boosting for fast face recognition. In *Proc. IEEE ICCV Workshop on Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems*, pages 96–100, 2001.

[13] M.S. Hoque and M. C. Fairhurst. A moving window classifier for offline character recognition. In *Proc. of the Seventh International Workshop on Frontiers in Handwriting Recognition, Amsterdam*, pages 595–600, September 2000.

[14] T.O. Kvålseth. Entropy and correlation : Some comments. *IEEE Trans. on Systems, Man and Cybernetics*, SMC-17(3):517–519, 1987.

[15] S. Lawrence, C. Lee Giles, A. C. Tsoi, and A. D. Back. Face recognition: A convolutional neural network approach. *IEEE Transactions on Neural Networks*, 8(1):98–113, 1997.

[16] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. of the IEEE*, 86(11):2278–2324, 1998.

[17] R. Marée, Geurts P., Piater J., and Wehenkel L. A generic approach for image classification based on decision tree ensembles. Submitted.

[18] T. Menp, M. Pietikinen, and J. Viertola. Separating color and pattern information for color texture discrimination. In *Proc. 16th International Conference on Pattern Recognition*, 2002.

[19] A. Nefian and M. Hayes. Face recognition using an embedded HMM. In *Proc. IEEE Conference on Audio and Video-based Biometric Person Authentication*, pages 19–24, March 1999.

[20] S. Obrzalek and J. Matas. Object recognition using local affine frames on distinguished regions. In *Electronic Proceedings of the 13th British Machine Vision Conference, University of Cardiff*, 2002.

[21] T. Ojala, T. Menp, M. Pietikinen, J. Viertola, J. Kyllnen, and S. Huovinen. Outex - new framework for empirical evaluation of texture analysis algorithms. *Proc. 16th International Conference on Pattern Recognition, Quebec, Canada, 1:701-706*, 2002.

[22] R. Paredes and A. Perez-Cortes. Local representations and a direct voting scheme for face recognition. In *Pattern Recognition in Information Systems, Proc. 1st International Workshop on Pattern Recognition in Information Systems*, pages 71–79, July 2001.

[23] M. Pontil and A. Verri. Support vector machines for 3d object recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(6):637–646, 1998.

[24] V.N. Vapnik. *The nature of statistical learning theory.* Springer Verlag, 1995.

[25] L. Wehenkel. *Automatic learning techniques in power systems.* Kluwer Academic, Boston, 1998.