ECOLE POLYTECHNIQUE FEDERALE DE LAUSANNE
SCHOOL OF LIFE SCIENCES

Master's project in Bioengineering and Biotechnology

A SEGMENTATION-FREE IMAGE CLASSIFIER FOR BIOLOGICAL APPLICATIONS

Carried out in the Imaging Platform
at the Broad Institute of MIT and Harvard, Cambridge, MA, USA
Under the supervision of Anne E. Carpenter, Ph. D

Done by

Virginie Uhlmann

Under the direction of
Prof. John McKinney
In the laboratory of Microbiology and Microsystems
EPFL

External Expert Anne E. Carpenter, Ph. D

LAUSANNE, EPFL 2012

Acknowledgements

First I wish to express my sincere gratitude and indebtedness to my supervisor in Boston, Dr. Anne Carpenter, head of the Imaging Platform at the Broad Institute, for giving me the opportunity to work on such an interesting project. I would like to thank Dr. Carpenter for the confidence she expressed in me through allowing me to orient this research and design this project mostly on my own. Her advice and mentorship gave me the chance to live a unique and most enriching research experience.

I also would like to address special thanks to Shantanu Singh, Lee Kamentsky, Mark Bray, Carolina Wählby and David Logan for their highly valuable support and advice on various parts of the project. Not only did they gave me the change to learn a lot by sharing their knowledge on the theoretical aspects of this thesis, but their guidance in the design of analysis experiments, in the preparation of data, and in several steps of the code implementation also proved to be essential throughout this work. I owe much of the quality of my project to their help.

On the other side of the Atlantic, I am also indebted to Professor John McKinney from the Laboratory of Microbiology and Microsystems at EPFL for supervising this thesis both before and during the actual project. I mostly owe the opportunity to do my Master's thesis abroad to his help and counsel. I would also like to thank Professor Michael Unser from the Biomedical Imaging Group at EPFL for his "informal" supervision. His participation has been greatly appreciated.

I must acknowledge and thank as well every people at the Broad Institute's Imaging Platform. All contributed at some point to this project by giving me helpful advice and suggestions that participated to the quality of this thesis. Above all, their positive attitude and kindness made my experience abroad most enjoyable.

I would finally like to thank my family, friends, and all the other persons who supported me in various ways while I was doing this work. Their love, encouragements and patience were essential in making this Master's thesis an amazing experience.

La Rippe, 15th August 2012 V. U.

Abstract

Images are essential in biomedical research. Following the development of fast imaging techniques, large amounts of data are routinely acquired and automated image analysis has become unavoidable. This work focuses on a common image analysis problem, image classification. While automated classification using machine-learning usually relies on measurements ("features") extracted from an isolated region of interest in the image, I. Goldberg et al. proposed a classification algorithm, WND-CHARM, in which features are computed on the whole image thereby avoiding segmentation. WND-CHARM exhibited impressing results on a range of different data, but was only used in a few studies so far.

In this project we created a WND-CHARM-inspired algorithm where features extraction is carried out in the popular image analysis software CellProfiler. We kept the core idea of WND-CHARM but designed a more reliable and user-friendly method. In order to build our algorithm in an optimal way we analyzed the various “steps” of WND-CHARM and benchmarked our new method against WND-CHARM on a collection of datasets. We further explored its potential by applying it to real-life biological data. It was first tested on examples of high-throughput cell-based assay data and yielded excellent results. We then used our algorithm to classify tissue images and obtained satisfying classification accuracy. Finally we applied it to harder datasets to investigate its efficiency on a wide range of cases.

We proposed a new whole image-based classification algorithm having the same advantages as WND-CHARM: (i) the ability to capture various morphological aspects of the image, making it efficient on many types of data, and (ii) the absence of requirement for segmentation. Our classifier was shown to perform as well as its predecessor while being easier to use. By showing its efficiency on some application examples, we aimed to make whole-image based classification more accessible to the biomedical research community.

Contents

1 Introduction

1.1 General Information

Many different research projects in biology and medicine rely on information contained in images. From confocal microscopy to MRI or from time-lapse microscopy to computed tomography scan, an amazing number of imaging modalities are available today for biomedical research. With the advent of high-throughput methods in biomedical research, large amounts of image data can be available in a limited time. However, these large image datasets might not be analyzable "by hand" for time or financial reasons. Automated image analysis using Machine Learning has therefore become a requirement for many biomedical research labs in order to extract information out of the rapidly expanding amount of acquired data.

The focus of this Master's thesis is an image classification algorithm called WND-CHARM (Weighted Neighbor Distances using a Compound Hierarchy of Algorithms Representing Morphology). In this chapter, we first give an introduction to the general principles of Machine Learning, describe WND-CHARM, and finally explain how this relates to the actual project.

1.1.1 Classification

Classification (or pattern recognition) is a subproblem of Machine Learning that can be defined as the process through which a machine learns to identify an ensemble of features (called a "pattern") and discriminates inputs with different patterns, assigning a "class" to each input. Two steps are usually required to build a classifier: a training step followed by a validation step, after which the classifier can be used to classify unknown inputs (Fig. 1.1). A classifier can be trained using either "supervised" or "unsupervised" learning, the latter lying out of the scope of this project.

When the inputs are images, the pattern recognized by the classifier is usually composed of a collection of features extracted from the image. These features are numerical values describing the image and are obtained through different image processing algorithms. They range from

simple features e.g. the mean value or the variance of the image, to more complex ones e.g. the resultant image after applying a particular filter. The most common paradigm when working on biological images is to first segment the objects of interest in the image to isolate them from background, and then to extract features from these objects (Fig. 1.2). Unfortunately, accurate object segmentation is often non-trivial and is likely to be the bottleneck that constrains the quality of an analysis since an inaccurate segmentation yields inaccurate results. In addition, designing a segmentation algorithm might require significant expertise, and is often beyond the skills of a biologist. Tools like the open-source software CellProfiler ([9] and [31]) can help in this task, but the segmentation process remains a technically difficult step which is hard to generalize.

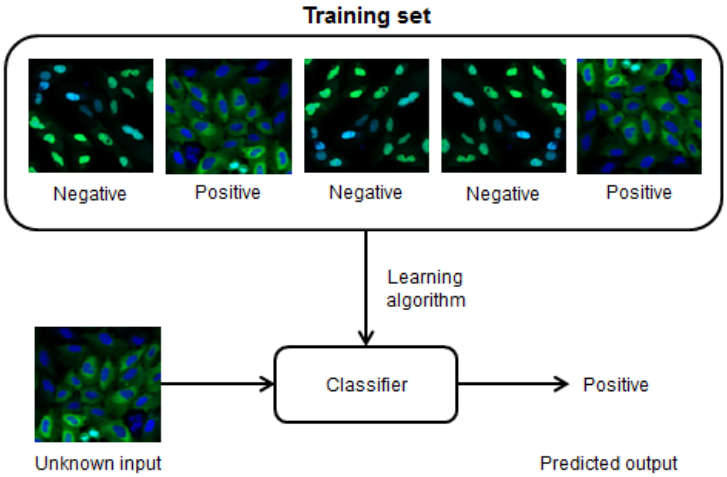


Figure 1.1: Overview of supervised classification. Images obtained from [2].

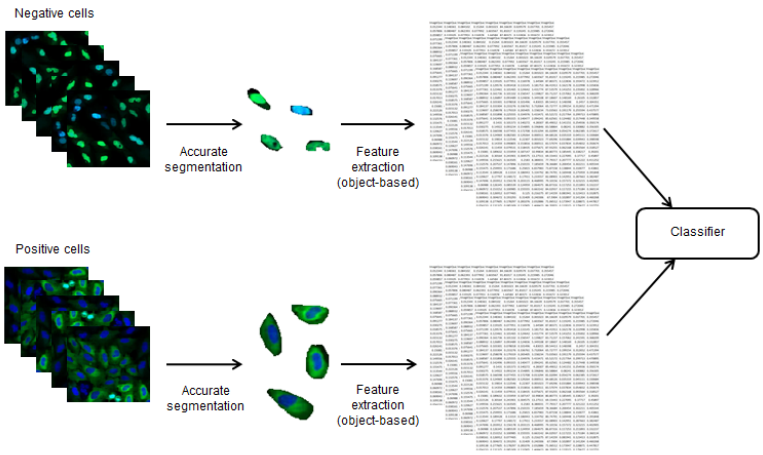


Figure 1.2: Object-based approach for classification. Images obtained from [2].

Training

When supervised learning is used to train a classifier, the training set is composed of inputs from known classes (i.e. "labels"). These inputs are fed into the classifier to serve as a basis for further classification of unknown data. A learning algorithm is then applied to allow the classifier extracting information out of the training set and learning from it. Many different learning algorithms exist, but we limit this introduction to the description of the one most closely related to the project, called k-Nearest Neighbors (k-NN). In k-NN the classifier assigns the class to the unknown input that is mostly represented among the k training set samples closest to it. The parameter k is a positive integer number making the classifier more (if k is small) or less (if k is big) sensitive to noise in the distribution of the classes. Fig. 1.3 illustrates k-NN classification for a simple case where only two features and two classes are considered: the unknown input (in black) would be assigned to class B for k=3.

Validation

In order to assess the efficiency of a classifier, a validation (also called testing) step is performed after the training. The validation consists of providing known inputs to the classifier, and then comparing the output of the classifier to the actual class of the input. The results consist of a score (percentage) for the efficiency of the classifier, and a confusion matrix (Fig. 1.4) summarizing the performance of the classifier on each class. The diagonal of the confusion matrix contains the number of items assigned to the correct class, while the off-diagonal items are the number of wrongly classified elements. The efficiency of the classifier is usually computed as a mean of the values in the diagonal.

When training a classifier, a big training set is usually preferred in order to allow the classifier to be more accurate, and a reasonable amount of data should be available for subsequent validation. However, in practice sparse annotated data is usually available for the training and validation steps. To address this problem, a method called k-fold cross-validation is commonly used. K-fold cross-validation is performed as follows: first the training set is randomly split in k subsets containing the same number of data (that is a fraction of $\frac{1}{k}$ of the entire dataset), then k-1 of these subsets are used for training and the remaining subset is left for testing. This procedure is repeated k times, using each different k subsets for validation.

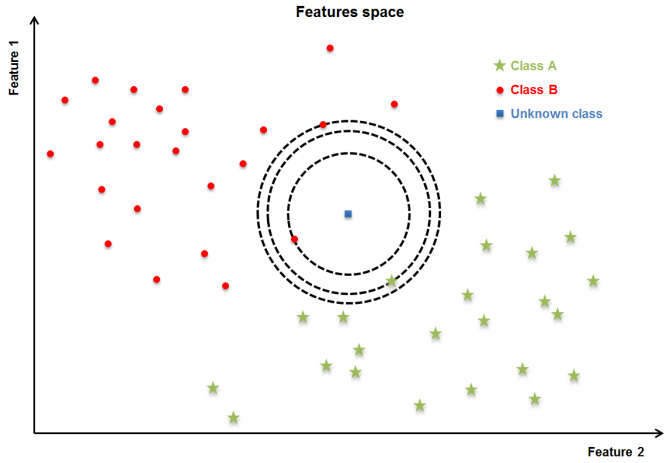


Figure 1.3: Illustration of k-nearest neighbors (k-NN) classification algorithm. Samples from two different classes (stars and rounds) are represented in a 2-dimensions features space. The unknown sample (square) is classified by looking at the most represented class among its k-nearest neighbors. k is here set to 3, and the three closest samples are therefore the elements reached by the three dashed circles.

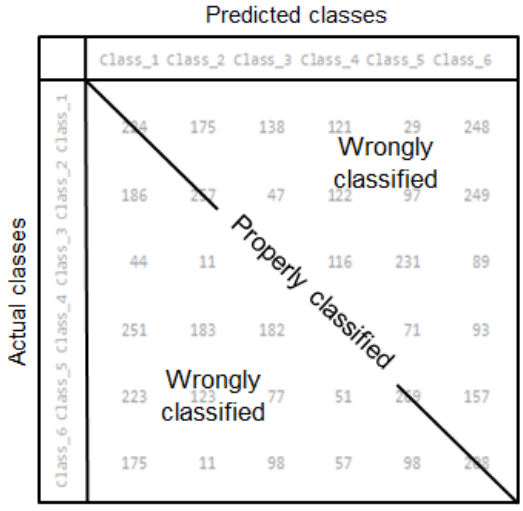


Figure 1.4: Description of a confusion matrix.

1.2 WND-CHARM

WND-CHARM (Weighted Neighbor Distances using a Compound Hierarchy or Algorithms Representing Morphology) is a classification algorithm proposed in 2008 by Ilya Goldberg et al. at the National Institute of Aging at the NIH ([38] and [46]). As opposed to an image-based classifier as described in Fig. 1.2, WND-CHARM computes features on the whole image hence no object segmentation step is required, as shown in Fig. 1.5. As segmentation might be a tedious and time-taking task depending on the images, an algorithm that bypass segmentation

by extracting features on a per image basis would allow for a substantial gain in time and efforts required to setup an image-based assay. Using image data instead of object data is, of course, a simple and straightforward idea. Difficulty stems from the choice of features to extract from the image. The segmentation step required to measure object properties ensures that all features to be computed capture actual information. However, when extracting measurements on the whole image, everything is captured and there is no way to focus on particular elements of the image. The key is then to design an algorithm able to capture enough useful properties of the image to overcome the lack of object-based information, yielding results that compare to the ones obtained with an algorithm using per-object features. We will refer throughout this work to the classification method as "WND", and to the feature vector as "CHARM".

1.2.1 Algorithm

The main steps of WND-CHARM are more or less the same as for any classical classifier. First features need to be extracted from the image to form a feature vector that will describe the image. WND-CHARM uses a feature vector composed of 1025 elements that fall into four main categories: high contrast features, polynomial decompositions, pixel statistics, and textures, all extracted from gray-scale images (i.e. if the input is an RGB image, the color information is not used). The High Contrast Features category contains information about the elements that compose the image, such as edges and shapes. The Textures category contains some well-known texture descriptors such as the Haralick ([21]) and the Tamura ([52]) texture features. The Pixel Statistics group is composed of information about pixel values distribution over the image, such as histograms with various number of bins and statistical moments. Finally the Polynomial Decompositions category is built by generating a polynomial that approximates pixels values up to a given error such as the Zernike or the Chebyshev polynomials, and whose coefficients describe the image. A complete list of the different features contained in the feature vector can be found in Table 1.1. A mathematical definition of each of these elements can be found in [38].

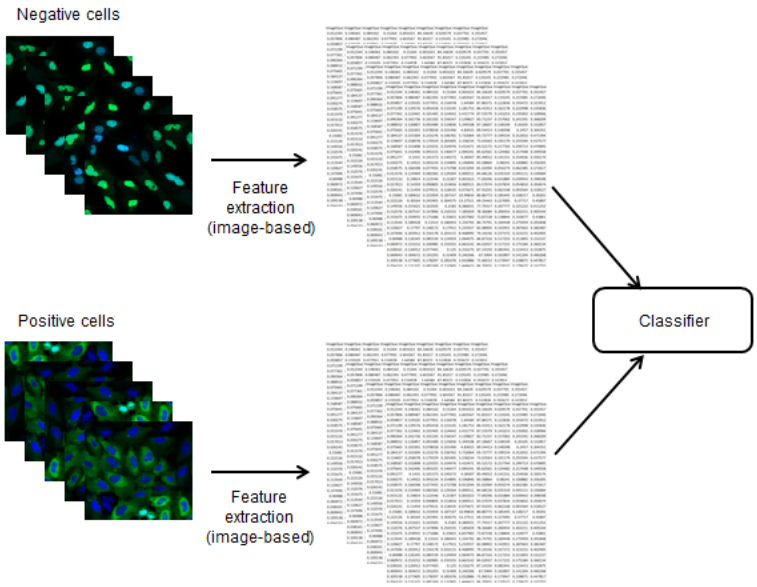


Figure 1.5: "WND-CHARM-like" whole-image based classification. Images source: [2].

High Contrast Features			Polynomial decompositions			Pixel statistics			Textures	
Edge Statistics (28)	Gabor Features (7)	Object Statistics (34)	Chebyshev Statistics (32)	Chebyshev-Fourier Statistics (32)	Zernike Polynomials † (72)	Moments * (4)	Multiscale Histogram * (24)	Radon Transform Statistics # (12)	Haralick Texture * (28)	Tamura Textures * (6)
• Mean • Median • Variance • 8-bins histogram • Number of edge pixels • Direction Homogeneity • Edge direction difference (4-bins histogram)	• Gabor features for f_0=[1…7]	• Euler number • Centroids (x and y) • Min • Max • Mean • Variance • Median • 10-bins histogram • Distance from object centroid to image centroid	• 32-bins histograms of the 400 coefficients of the Chebyshev Transform of the image	• Modulus of the complex coefficient of the Fourier Transform of the Chebyshev Transform of the image	• Modulus of the Zernike Features (i.e. the coefficients of the Zernike approximation of the image)	• Mean (1st) • Variance (2nd) • Skewness (3rd) • Kurtosis (4th)	• 3-bins histogram • 5-bins histogram • 7-bins histogram • 9-bins histogram	• 3-bins histogram at 0° • 3-bins histogram at 45° • 3-bins histogram at 90° • 3-bins histogram at 135°	• Statistics based on the co-occurrence matrix of the image	• Contrast • Coarseness • Directionality • 3-bins histogram of coarseness

† = features computed both on the original image and on its Fourier Transform
# = features computed on the original image, on its Fourier Transform, on its Chebyshev Transform, and on the Chebyshev Transform of its Fourier Transform
* = features computed on the original image, on its Wavelet Transform, on its Chebyshev Transform, on its Fourier Transform, on the Wavelet Transform of its Fourier Transform, and on the Chebyshev Transform of its Fourier Transform

Table 1.1: Elements of the CHARM feature vector.

Fig. 1.6 summarizes how the feature vector is constructed; these features are extracted from the original image as well as from transforms of the image and from compound transforms (transforms of transforms) of the image. The Fourier Transform, Wavelet Transform, Chebyshev Transform and composition of these transforms are used to give additional "higher level" descriptions of the image content.

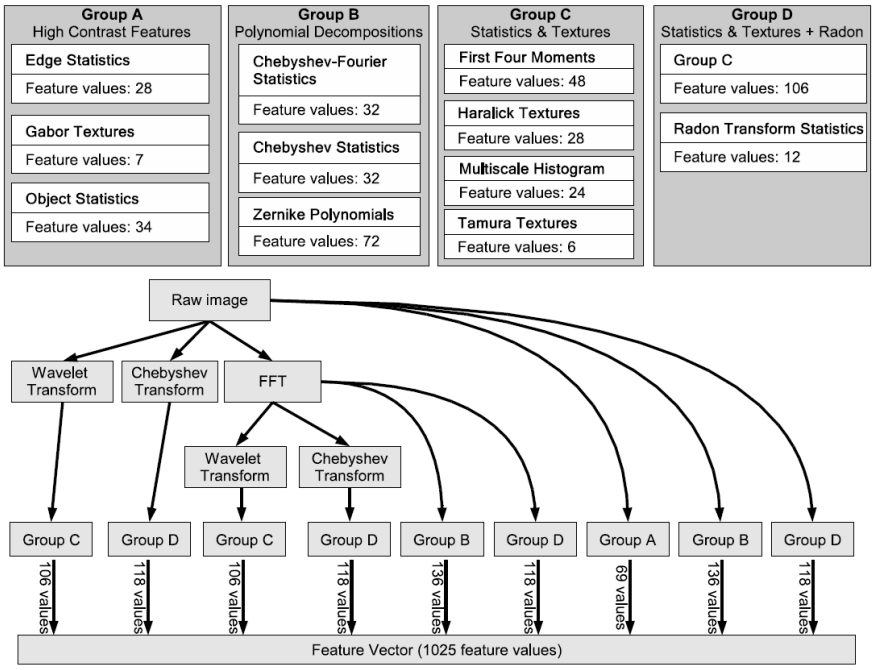


Figure 1.6: Construction of WND-CHARM feature vector. Source: [38].

The second step of the algorithm is a selection of the elements of the features vector to reduce the dimensionality of the feature space. Since the features vector is composed of so many different elements, we expect it to be partly composed of noisy or non-informative features that fail to give exploitable information on the image and that should therefore be discarded. A weight based on the Fischer Discriminant score ([3]) is computed for each feature as described in Eq. 1.1, where $\overline{T_f}$ is the mean of the values of feature f in the whole dataset, $\overline{T_{f,c}}$ the mean of the values of the feature f in class c only, $\sigma^2_{f,c}$ the variance of feature f among all samples of class c, and N the total number of classes.

$$W_f = \frac{\sum_{c=1}^{N}\left(\overline{T_f} - \overline{T_{f,c}}\right)^2}{\sum_{c=1}^{N}\sigma^2_{f,c}} \cdot \frac{N}{N-1} \tag{1.1}$$

Feature weight can thus be understood as the *ratio of variance of class means from the pooled mean to the mean of the within-class variance* [38]. The variances of the features among each class are computed after the feature values have been normalized between 0 and 1. In WND-CHARM's C++ code, values are normalized in the $[0,100]$ interval instead of $[0,1]$.

Following this weighting step, 35% of the features with lowest weights (i.e. the least informative ones) are discarded by setting their weight to zero. The remaining 65% of the features that best discriminate the image classes are used for the actual classification. The 35% threshold for weight selection is *selected empirically based on observations with several classification problems* [38]. In the actual implementation of the algorithm the default threshold value is set

to 15% [46].

For the classification step, a modified version of the k-nearest neighbor classifier that includes features weights is used. The similarity $s(x,c)$ from feature vector x to class c is described as Eq. 1.2, where T_c is the training set for class c, t a feature vector contained in T_c, $|T_c|$ the number of training samples of class c, $|x|$ the length of feature vector x, W_f the weight of feature f, x_f the value of image feature f in vector x, t_f the value of image feature f in vector t, and p is an exponent empirically set to -5 in [38]. The parameter p modulates the importance of elements in the training set as a function of their similarity to vector x; its value dampens the contribution of training samples that are very different from x. As the dissimilarity grows bigger, its contribution becomes smaller to the power p for increasingly negative values of p.

$$s(x,c) = \frac{\sum_{t \in T_c} \left[\sum_{f=1}^{|x|} W_f^2 \left(x_f - t_f \right)^2 \right]^p}{|T_c|} \tag{1.2}$$

The similarity between x and class c is the average of all weighted similarities elevated to the power p from the feature vector x to any feature vector belonging to class c. When an unknown feature vector x is presented to the classifier, the similarity to each class c is computed and the vector is assigned to the closest class, i.e. the class to which it has the maximal similarity. The difference between this Weighted Neighbor Distance (WND) and the traditional k-NN approach is that WND gives a weighted distance from a vector x to all elements in the training set and therefore each of these elements play a role in the actual classification and are weighted by their information content, while in k-NN only k elements in each class equally influence the classification result.

Finally, the classifier performance was validated using a different method than k-fold cross-validation: instead of randomly splitting the dataset into k parts, using one for validation while training with the remaining k-1, WND-CHARM's method was to use a random 25% of the data in each class for validation after training on the remaining 75% in each class. We will refer to this method as "save 25%" for the remainder of this work.

1.2.2 Performances & Pitfalls

WND-CHARM was tested on several datasets composed of biological, textural, and facial images and obtained surprisingly good results with classification accuracies as they outperform application-specific classifiers, for instance on biological datasets as shown in Table 1.2. The authors suggest that this impressive performance is due to the large feature vector used and the feature selection step. These results are encouraging given that they were obtained without any segmentation step on the original image WND-CHARM therefore seems to be a promising algorithm that could be used for a wide range of applications ranging from biomedical image analysis to face recognition. Unfortunately, the only available version of WND-CHARM is a C++ implementation that requires the installation of many libraries and that can only be run in command line. WND-CHARM has surprisingly only been cited in a very few papers even

Dataset	Benchmark algorithm	Benchmark No. of features used	Benchmark accuracy	Accuracy of WND-CHARM
Hela	(Murphy, 2004)	37	83	86
Pollen	(France et al., 1997)	8	79	96
CHO	(Boland, Markey' Murphy, 1998)	37	87	95

Table 1.2: Comparison of WND-CHARM accuracy versus classic classifiers. Source: [38].

through it is described as performing better than the traditional algorithms, probably partly due to its non-user friendly implementation. As command line programs can be difficult to handle for people without training in information technology, designing an easier-to-use implementation could allow the full potential of the algorithm to be better exploited.

Some criticism can be formulated against WND-CHARM as the implementation choices are not really supported in [38] and [46]. Concerning the feature extraction, the composition of the feature vector given little justification and analysis, in spite of the fact that some of its elements are quite difficult to interpret at the image level, and the thresholding of the features weights is empirical. The classification and validation steps can also be discussed: the WND method looks like a simple k-NN-like algorithm with empirical parameters, and the "save 25%" method is an uncommon and potentially biased way of validating the results.

1.3 Project

As this algorithm might hold a great potential that remains mostly unused due to the complexity of the existing implementation, this project's core aim was to implement a WND-CHARM-like algorithm in Python, and to integrate it into the widely used open-source software CellProfiler ([9], [31], and [27]) to make it more user-friendly and more easily accessible to researchers. This project was supported by Ilya Goldberg, the original author of WND-CHARM, who would like image-based classification algorithms to be more widely used. In order to make sure we fulfilled this objective our first goal was to obtain comparable results with our implementation as the ones described in [38]. We also aimed to make an analysis of the original WND-CHARM to better understand the way it operates, and to ultimately be able to propose a new WND-CHARM-inspired algorithm that performed equally well or better, and that keeps WND-CHARM's strengths while correcting its weaknesses therefore addresses most of the issues of the original implementation described above.

1.3.1 Implementation

Instead of translating the existing WND-CHARM's code from C++ to Python, our goal was to keep the global idea of Ilya Goldberg's algorithm to create a new WND-CHARM-like classification algorithm. The new version would also contain features extracted on a whole-image basis (the primary strength of WND-CHARM), but the identity of the features would not necessarily be the same as we thought that the aspects captured by features (textures, edges, etc.) were more important than each particular feature definition. The classification algorithm and

validation methods could also be modified according to the conclusions of our analysis on the original WND-CHARM.

1.3.2 Application

While developing our WND-CHARM-inspired implementation in CellProfiler we tried to reproduce the results from Ilya Goldberg's original paper, and we therefore first ran our implementation of the algorithm on the datasets used in [38] and [46]. Once we were confident in the fact our implementation was efficient, we presented some examples of applications in real-life cases. The additional datasets we used for the application part are thoroughly described in the Materials and Methods section of this thesis.

2 Materials and Methods

This Master's thesis was hosted by the Imaging Platform/Carpenter Lab. The Imaging Platform focuses on the development of methods to automate the analysis and extraction of the multitude of information contained in biological images, and is a world leader in this domain. Automation of image analysis became a major concern in biomedical research as high-throughput imaging techniques developed, yielding amounts of data that cannot be processed manually. Surprisingly only few laboratories around the world are fully devoted to developing tools for image analysis, and it is therefore a great opportunity to be able work in Anne Carpenter's Lab.

The researchers at the Imaging Platform are mostly Computer Science-oriented and concentrate their effort in three topics: software development, data mining and image assay development. The group benefits from the experience of each of its staff scientists, and from the unique dual training of the lab's Principal Investigator (Anne Carpenter, Ph.D) in both Cell Biology and Image Processing. Their most important work includes CellProfiler and CellProfiler Analyst, two open-source software packages dedicated to high-throughput image analysis ([9], [31], [27]).

2.1 Materials

Since our work was essentially computer-based, we will describe in this section the computational tools and resources we used to obtain the results to be presented.

2.1.1 CellProfiler and CellProfiler Analyst

CellProfiler (CP, [9], [31], and [27]) is a unique tool that allows scientists to perform advanced image analysis even in the absence of extensive computer vision skills. Image processing steps are organized in a user-friendly "pipeline", and the various operations on images can be performed by simply adding "modules" to the pipeline. The very simple interface (Fig. 2.1) makes the software easy to use, as the various parameters required by the different image

analysis algorithms are listed and described in a comprehensive manner. For all these reasons CellProfiler became a widely used tool in biological research and accumulated more than 382 citations in September 2011.

While CellProfiler allows extraction of a vast amount of data from the images, CellProfiler Analyst (CPA, [25]) encompasses the second part of the analysis: data mining. Using the same kind of simple interface as CellProfiler, CellProfiler Analyst gives a way to perform advanced statistical analysis and screenings on huge datasets. Apart from their well-designed interfaces these two software packages are also famous for being free and open-source, therefore potentially allowing every researcher with programming skills to design their own modules.

Aside from CellProfiler and CellProfiler Analyst the group also developed the Broad Bioimage Benchmark Collection (BBBC, [2]), a database of image sets that can be used to test and validate methods for image analysis.

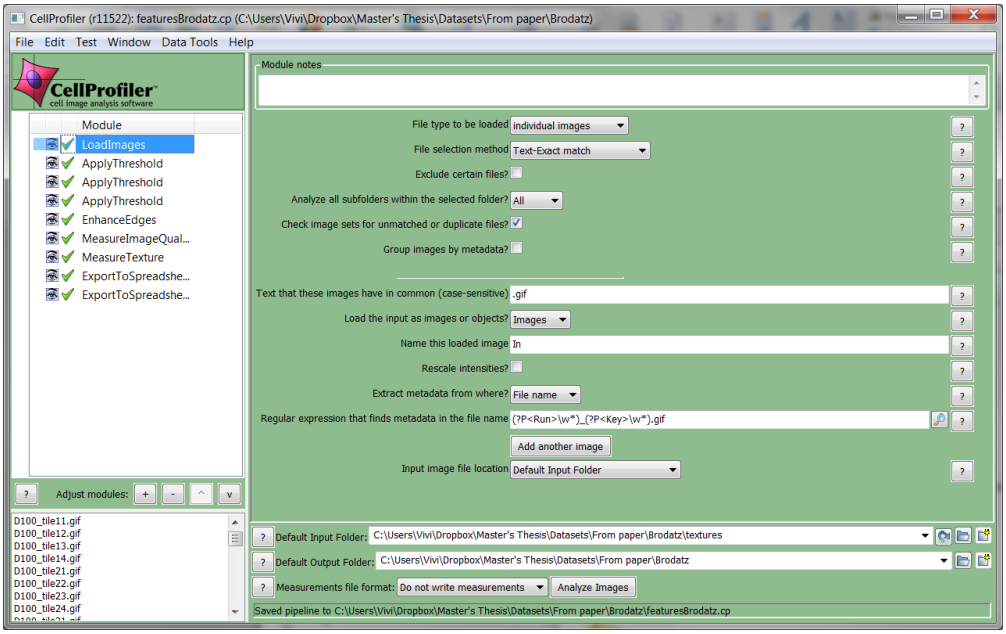


Figure 2.1: CellProfiler user interface.

As CellProfiler is composed of modules and pipelines the first question we asked ourselves was how to embed our WND-CHARM-like algorithm into CellProfiler. Two different options were considered for the features extraction step: either creating a module called WND-CHARM that would encapsulate the whole feature extraction process, or creating a pipeline composed of different modules for the extraction of the many measurements required to build the features vector. The main advantage of the first option is be that the user would be blind to the feature extraction process that could seem to be complicated and confusing. As explained before WND-CHARM uses a lot of different features obtained through several more or less complex image processing steps which might intimidate less experienced users. Hiding all this

process into a simple module to be used as a black box would therefore have the advantage of making the algorithm look simple to the user while it is actually not: this is the way the C++ implementation of WND-CHARM was designed. On the other hand, the second option offers greater modularity and thus flexibility: if the user thinks that for some reason there is no need to compute some features, if the user wants to add more features or change parameters, there would be the possibility to remove modules out of the pipeline, to introduce new ones or to modify existing modules settings. In this situation the parameters for each module would all be visible to the user, which would give the opportunity to easily fine tune and refine the analysis depending on the dataset. We finally opted for the second option as we thought the ability to add, remove or modify modules would be highly valuable and contribute to an improved user-friendliness. The second choice we had to make was how to organize the classification step. CellProfiler Analyst offers the possibility to interactively train a classifier, whereas CellProfiler is not initially designed to perform image classification. One option was be to design the classification step as a CellProfiler module. The overall WND-CHARM algorithm implemented in CellProfiler would therefore be composed of two steps: first for the training the user would run a pipeline composed of a series of modules for feature extraction followed by a module for training and validation that would output a "classifier file". The actual classification would then be performed in another pipeline composed of the feature extraction steps and of a classification step that requires as input the classifier file produced after training. The second option was be to leave the classification step out of CellProfiler, either including it in CellProfiler Analyst or leaving it as a standalone Python executable. This question is still debated, and no final decision has been made at the time we write this thesis. As an implementation choice was not settled we decided to keep the classification step as a separate Python script in the meantime such that it can easily be included in CP or CPA if needed.

2.1.2 Python

CellProfiler and CellProfiler Analyst are both written in Python, a high-level programming language that can be interpreted in many different operating systems. All the code created to implement our classifier was therefore written in Python to make it suitable for integration in CellProfiler. The overall process of image classification is currently split in two parts (feature extraction and training/validation). First a list of features is extracted from images using CellProfiler, which is then fed into a Python script for training and validation.

Features Extraction

We hypothesize that it is more the "per image" analysis rather than the actual features used that gives WND-CHARM its analysis power and we therefore assumed that any other whole-image-based feature vector composed of judiciously chosen elements should give good results. We also found very interesting the idea of having features "groups" as in WND-CHARM (High Contrast Features, Polynomial decompositions, Pixels statistics, etc.), so we decided to try to

create a vector that would retain this idea and be composed of the same "groups", but not necessarily of the same elements inside each group. Since many different modules exist in CellProfiler our first goal was to establish a list of the features we could obtain using the already existing modules, and then to compare this list with the list of features used by WND-CHARM. The feature groups missing from CellProfiler but used in WND-CHARM would give us leads for implementing new modules. Since we thought that the real interest of WND-CHARM does not rely on the exact nature of the features used but rather on the fact the features are computed on the whole image rather than on segmented objects, we started by implementing a "WND-CHARM-like" pipeline that would extract as many features on the whole image as possible using only existing CellProfiler modules, and analyzed these first results. If the results happened to be not satisfying, we would iteratively refine the pipeline by implementing new modules until results would be good enough. Another reason that helped us decide using this incremental implementation approach instead of directly implementing all the features from WND-CHARM is that we thought it could give us the chance to find a "sufficient" subset of features that yields good classification results. It could give us the opportunity to design a feature vector that would neither be too big (with too many features, some hardly ever actually used) nor too small (with too few features to allow for good classification results).

As already mentioned in the Introduction the CHARM vector is composed of features extracted from the original image, but also from transforms and compound transforms of the image. There is therefore a kind of "hierarchy" in the vector. Features are extracted from several "levels" (raw image, transforms and compound transforms) that are increasingly complex and abstract. If it is straightforward to connect features values to the actual image content when they are extracted on the raw image, it is however more subtle to understand what they mean when computed in a transformed domain, and it becomes even tougher to make conclusions when features are extracted from a compound-transformed image: higher features "levels" are generally harder to interpret. As an example if one sees that edge features have high values, one can easily look at the original image and observe that it is very "edgy". Similarly one could link high values at high frequencies in the Fourier transformed image to the presence of many details in the original image. However strong signals in the Fourier Transform of a Chebyshev Transform are much harder to connect to aspects of the original image. Aside from features groups, our other axis of exploration was therefore features levels: we implemented our CHARM-like feature vector by incrementally adding more representatives of features "groups", and in parallel by incrementally adding more features "levels". Our goal was to gather results for these different features vectors versions in order to understand how added complexity impacted on classification efficiency. We used the following nomenclature to describe our different features vectors: each version is labeled as vY.X. Y ranges from 1 to 3 and indicates features "levels" included in the vector as shown in Fig. 2.2: 1 is the lowest level containing features computed on the raw image only, 2 contains all elements from 1 and an added level of complexity with features extracted from transformed images, and 3 is the highest level that contains 1, 2 and features extracted from compound transforms of the image. The second index, X, takes the values of 1 or 2 and corresponds to different features sets: 1 uses

only the Fourier and Wavelet transforms in addition to the initial features set from CellProfiler (composed of edge statistics, Gabor textures and Haralick textures), while 2 works with the full set of transforms (Chebyshev, Fourier and Wavelet) and contains Chebyshev statistics, Tamura textures and more statistical moments as well as the features available in CellProfiler.

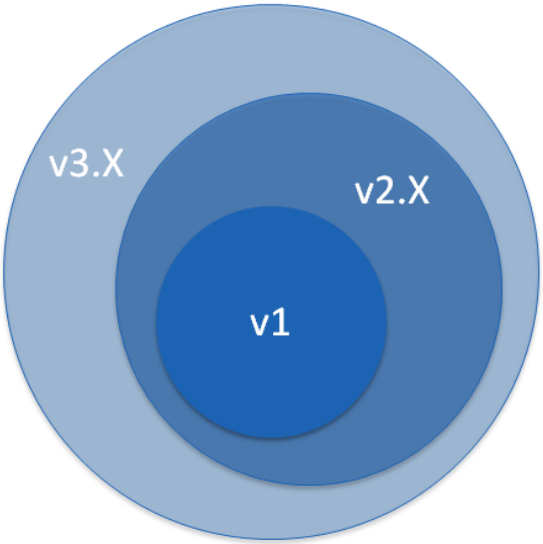


Figure 2.2: Hierarchical features levels.

The initial pipeline (called v0 in our nomenclature as it serves as a reference) that we designed in CellProfiler extracts a vector of 125 features from the image. These features come from the original image (1), the result of a Prewitt gradient filtering of the original image using the EnhanceEdges module (2), and the three results of applying global Otsu thresholding on the original image with 2-class thresholding, 3-class thresholding with pixels in the middle intensity class assigned to the foreground, and 3-class thresholding with pixels in the middle intensity class assigned to the background (3, 4, and 5) using the ApplyThreshold module. Nineteen global image statistics are extracted on a subset or all of the five images using CellProfiler's MeasureImageQuality module (Percent maximal and percent minimal on (1), maximum intensity on (1) and (2), mean intensity, standard intensity and total intensity on all five images). Finally 104 texture statistics plus 2 Gabor features are extracted from the original image using CellProfiler's MeasureTexture module (measures on two texture scales in all four directions, measures on 4 texture scales in all four directions, and Gabor features using four angles). Since in the WND-CHARM procedure only the 15% strongest features are taken into account for classification, only 19 features out of the 125 were used in this case versus 154 out of the 1025 contained in the CHARM vector. It is important to notice the difference in the absolute size of the subset of features used for classification between our implementation and the original one: we have in our case roughly 6 times fewer features available to discriminate between the different classes. v0 was our reference vector since it represents what we can obtain if we want to do whole-image classification using only modules already available in CellProfiler's current release.

The second versions also contained only lower-level features, i.e. features extracted on the raw image, and we therefore called them v1.1 and v1.2 following our nomenclature. v1.1 consists of the 125 features from v0 plus the result of four histogram binning of the image with various numbers of bins (3, 5, 7 and 9) from a new module called Histograms, yielding a total of 149 features. v1.2 is at the same features level as v1.1, but contains more new modules: a Moments module that computes the first four statistical moments of the image and a Tamura module that extracts three Tamura features (contrast, coarseness and directionality, described in the appendices) as well as a 3-bins histogram of coarseness for a total of 159 features. These two vectors are still very simple as only 22 and 24 features are left after thresholding.

The next versions v2.1 and v2.2 include another level of complexity: they contain all the features present in the v1.X versions plus features extracted from transforms of the image. v2.1 is composed of 421 features: the 149 features present in v1.1 plus the features from the MeasureTexture module, the histograms from the Histograms module, and most measurements from the MeasureImageQuality module (Maximum, minimum, percent maximum, percent minimum, variance and total intensity) computed on the Wavelet and Fourier transform of the image. The second version, v2.2, contains 661 features and is constructed in the same way as v2.1: it contains the 159 elements from v1.2 extracted on the original image, plus features (all elements from MeasureTexture, Histograms, Tamura and Moments, and selected measurements from MeasureImageQuality) computed on the Fourier, Wavelet, and this time also Chebyshev transform of the image. Some more features are also computed using this last transform: the new HistogramCheby module computes the "Chebyshev statistics" defined as a 32-bins histogram of the coefficients of the Chebyshev transform of the image. The Chebyshev statistics are also computed on the Fourier transform of the image, yielding the so-called "Chebyshev-Fourier statistics". When thresholded using the 15% limit, v2.1 therefore has 63 elements left for classification while v2.2 benefits from 99 features. With these two vectors we are getting closer to the scale at which WND-CHARM operates with its 125 features after thresholding.

Finally versions v3.1 and v3.2 contain all the elements present in v2.1 and v2.2 respectively as well as elements extracted from compound transforms. Everything extracted on the transforms from level 2 is computed as well on the Fourier transform of the available transformed images. v3.1 hence features the Fourier transform of the Wavelet transform, while v3.2 contains the Fourier transform of the Wavelet transform as well as the Fourier transform of the Chebyshev transform. As in level 2, v3.1 does not contain the Tamura, Moments and HistogramCheby modules while v3.2 does. The resulting v3.1 is composed of 557 elements and v3.2 contains 953 features. After thresholding, v3.1 is composed of 83 features and v3.2 of 143 elements. The last version, v3.2, is mostly similar to the CHARM vector in the way it is built even though it does not contain the exact same elements. Table 2.1 describes the composition of this CHARM-like vector we designed.

High Contrast Features			Polynomial decompositions		Pixel statistics		Textures	
Edge Statistics (4)	Gabor Features * (2)	Image Statistics (15)	Chebyshev Statistics (32)	Chebyshev-Fourier Statistics (32)	Moments * (4)	Multiscale Histogram * (24)	Haralick Textures * (104)	Tamura Textures * (6)
• Mean • Max • Variance • Number of edge pixels	• Gabor features computed at four angles	• Max * • Mean * • Percent Minimal * • Percent Maximal * • Variance * • Total Intensity * • Mean intensity after thresholding • Variance on thresholded image • Number of pixels above threshold	• 32-bins histograms of the 400 coefficients of the Chebyshev Transform of the image	• Modulus of the complex coefficient of the Fourier Transform of the Chebyshev Transform of the image	• Mean (1^{st}) • Variance (2^{nd}) • Skewness (3^{rd}) • Kurtosis (4^{th})	• 3-bins histogram • 5-bins histogram • 7-bins histogram • 9-bins histogram	• Statistics based on the co-occurrence matrix of the image	• Contrast • Coarseness • Directionality • 3-bins histogram of coarseness

* = features computed on the original image, on its Wavelet Transform, on its Chebyshev Transform, on its Fourier Transform, on the Wavelet Transform of its Fourier Transform, and on the Chebyshev Transform of its Fourier Transform

Table 2.1: Elements of the CHARM-like feature vector. Shaded regions indicate features only present in versions vY.2.

For the interested reader a description of the CellProfiler modules is available in CellProfiler's help interface, and a mathematical description of the new features and transforms mentioned above can be found in the appendices.

We designed our feature extraction pipelines to yield two outputs (as comma separated value text files, or csv): a "labels" file containing two columns (a unique identifier for each image and the class of the image), and a "data" file containing N+1 columns, where N is the number of features extracted from the image, plus one column containing the image identifier.

Classification

As explained before we implemented the classification part as an independent Python script since we did not decided yet whether it should be included in CellProfiler as a module or not. An instance of the classifier can be trained by calling a `train` function from our script and then validated using a `cross-validation` method. Our code takes as inputs the data file and associated labels files obtained after running the features extraction pipeline in CellProfiler. Several options are available for the analysis: confusion matrices summarizing the validation steps can be displayed and the number of times validation is repeated can be selected by the user. The user also have the possibility to chose which classification algorithm to use. We of course implemented WND as described in [38], but we found it important to add other options as we were not fully satisfied with this algorithm for the reasons explained in the Introduction.

In our code `validation` methods that automatically split the training set into a test set and

validation set are available such that the raw output files from CellProfiler can be directly used without the need to manually split them into a training set and a validation set. We implemented two different validation methods from which the user can choose: the standard k-fold cross-validation, and WND-CHARM's "save 25%" method which randomly chooses 75% of the images in each class to compose the training set and uses the rest as the validation set. The output after the desired number of training-validation runs is a text file containing a record of the input settings (input files and parameters), a list of the features with the 15% best weights actually used for classification (if WND is used) as well as the classification accuracy (mean of the diagonal elements of the confusion matrix, in percents) for each run. Finally the mean, median and standard deviation of classification efficiencies over all runs are also reported.

A classifier can be similarly trained and tested with Goldberg's C++ implementation of WND-CHARM using a `train` function and then tested using a `test` method. Unlike our version of the algorithm, the C++ WND-CHARM performs both feature extraction and classifier training together in a "black box", and the path to the folder containing the whole image dataset is given as an input to the program. The output is an HTML report containing classification accuracies, lists of top 15% features, and confusion matrices for each validation runs. More details on the original C++ implementation can be found in [46].

Our implementation also features the possibility to classify tiled images. Tiling was so far performed in MATLAB, but a module could be implemented in CellProfiler to give the user the opportunity to automatically tile large input images at the beginning of the features extraction pipeline. There are several reason why one might want to chop down images from a dataset into equally-sized tiles. First for instance when working with very crowded cell images tiling could be used as a naive segmentation step ([48]). One could also use tiling to increase the size of the training/validation set with the drawback that the features extracted from images created by chopping a large image into tiles have a high risk of being correlated. Finally, tiling could be used to simplify and speed up the analysis of very high resolution images as the execution time of some features extraction algorithms depend on image size. In this case the goal is to extract features on each image tile, but to output only one classification result for the big image composed of all tiles. In this situation the user can use the tiling option of our implementation that assigns a class to each image tile separately, finds out the most represented class among all tiles and outputs a classification result for the large original image.

2.1.3 Analysis Softwares

In order to perform our analysis of WND-CHARM we used several analysis softwares besides our Python script. We used Mathworks' well known software MATLAB ([34]) and the open-source integrated development environment RStudio ([43]) for the open-source programming language for statistical computing R ([41]). Since each software has its advantages and drawbacks we wanted to be able to use the most convenient solution for each experiment we made.

In order to be able to move from one program to another and make conclusions using data coming from these different platforms we needed to make sure outputs from these programs were all comparable. We therefore ran similar experiments using Python, MATLAB and R which results are presented in the appendices. We observed that all these methods yielded similar results when performing a similar task, therefore we used in confidence the most appropriate option between Python, MATLAB and R depending on the task we wanted to perform.

2.1.4 Computing Resources

Since accurate measures of classification efficiency are based on statistics we had to run a reasonably large number of tests on every dataset we used, using both WND-CHARM C++ version and ours, and therefore needed the appropriate computational power. Our Python implementation can be executed on various operating systems since Python interpreters are available for MacOS, Windows and Unix. We were able to launch the various tests that will be described in the Methods section below on the Broad's Linux computer cluster using Load Sharing Facility (LSF), a software that allows distribution of computing resources. The original C++ version was both available as a compiled executable (.exe) and as source code, and were hence able to recompile the source code and execute it on the Linux cluster as well. We also extensively used of the Windows Virtual Machine service provided by the Broad Institute for all the experiments that needed a user interface.

2.2 Methods

We describe in this section all the theory and setups related to the experiments we performed. All corresponding results are presented in the Results section.

Our main goal in this work was to propose a classification algorithm based on WND-CHARM that would keep its core idea, namely the facts that features are extracted on the whole image, and that the feature vector is composed of many different features "groups" and "levels", but differ in its weaker aspects. It would also be designed to be more "user-friendly" as being part of CellProfiler. We first carried out a complete analysis of the original WND-CHARM algorithm. Our lines of inquiry concerned every steps of WND-CHARM: the CHARM features vector, the WND classification method, and finally the "save 25%" validation, as described in Fig. 2.3. By investigating the different "weaknesses" we identified in WND-CHARM we wanted to be able to find arguments supporting our new algorithm that should be better understood and characterized.

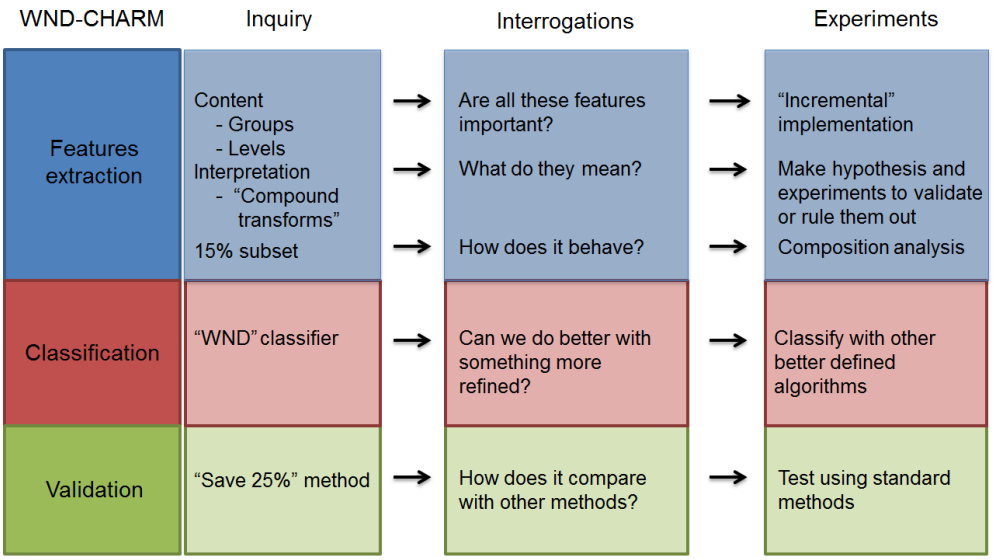


Figure 2.3: Plan of WND-CHARM analysis.

Our work was organized in three big parts. The first one was the WND-CHARM analysis that aimed to help us building a new "WND-CHARM-inspired" classification algorithm. Then followed an analysis of this new algorithm to make sure it was at least as powerful or better as its "parent", and to explore the possibility of making more improvements. Finally this new algorithm was used on real-case datasets to show some applications where the method we developed could be used.

As the WND-CHARM C++ code was made to be used as a black box we could not easily use it to perform experiments. For this reason we extracted the CHARM feature vector computed in the C++ implementation and re-implemented the WND classifier and the "save 25%" validation method in Python. In this way we were able to fully separate features extraction, classification and validation, so we could vary each parameter independently. For instance we could keep the classification method fixed and use different features vectors versions to be compared with CHARM, or conversely we could use the CHARM vector but vary the classification method. We first ran some calibration tests to make sure our re-implementation in Python was correct: we used the same experimental settings as in [38] and compared paper results, reproduced results using the C++ implementation, and results using the CHARM vector and our Python re-implementation of the classification and validation steps.

Before gathering any data with our implementation of WND, we also ran what we called "significance tests" on the reference data from [38] (described later in this chapter). While usual validation tests produce a measure of the classification efficiency of an algorithm on a given dataset, our significance tests compared the efficiency of a classifier to that of a distribution of classifiers trained on scrambled data (i.e. "random" classifiers). This allowed us setting a baseline for what "good" classification efficiency means: our goal is to be significantly better than the results obtained using the random classifier. Significance tests hence consist of

randomizing class labels in the training set to create a random classifier. Since class labels are assigned randomly, no common feature pattern should emerge from features extracted from the images in each class and the elements saved for validation should therefore be assigned to a random class at the validation step. These significance tests are a way to show that a good classifier's accuracy is obtained by really discriminating between classes using the elements in the features vector rather than by chance. When classifying random data on an N-class problem, the classifier is expected to have an average accuracy of $\frac{1}{N}$. However this is not always the case, for example if the dataset is biased by having different number of images per class. In this case the random classification probability can be modified. Our significance tests hence also allowed making sure our datasets were nicely set up in addition to providing us with a reference measure for classification efficiency.

By training a large number of these random classifiers we could experimentally obtain a reference distribution to compute an empirical p-value for the actual classification result. With 10'000 random classifiers, the empirical p-value was computed as follows:

$$\text{p-value} = \frac{\text{count}(\text{eff}(\text{rdm}) \geq \text{eff}(\text{true}))}{10000} \tag{2.1}$$

where eff(rdm) is the classification efficiency for a random classifier, eff(true) the classification efficiency of the true classifier, and count(eff(rdm)≥ eff(true)) the number of times the relationship eff(rdm)≥ eff(true) is satisfied among all the 10'000 elements in the distribution of random classification efficiencies. We ran two series of significance tests using our implementation: first using 10-fold cross-validation, and then using the "save 25%" method.

2.2.1 WND-CHARM Analysis

As described before WND-CHARM can be seen as a three-steps algorithm. First features are extracted to build the CHARM vector composed of 1025 elements extracted from the original image and from transforms of the image. Then these features are weighted using Fisher scores, ranked, and the features corresponding to the 15% best weights are used to train a classifier based on a k-NN-like algorithm using weights. Finally the performance of the algorithm is validated using the "save 25%" method where 25 random percent of the data in each class are saved to constitute the test set while the rest is used as a training set. We could formulate interrogations regarding any of these steps, so we decided to analyze each of them separately.

Features Extraction

Feature Vector Content The CHARM vector can be decomposed in many different features "groups", or in its three different features "levels" as explained in the Materials section. This overall construction appeared to us to be judicious and powerful as having these features groups allows the algorithm to perform well on a wide range of different data. A feature vector capturing a wider range of morphological aspects of the image will indeed make the

approach more general as it is able to detect which image properties differ between classes on a wide range of different data. In spite of this we did not wanted to simply re-implement the whole CHARM vector for two reasons. We wanted first to investigate how the presence of different groups and levels impacted on classification results. Then if we found out that all these groups and levels were actually useful, our hypothesis was that their presence was more important than their actual composition. In other words, our guess was that we could build a CHARM-like vector with the required groups and levels, but not necessarily the same elements present in the original CHARM.

Our first experiment was to split the CHARM vector into features groups. We ended up with eight different groups for all the elements of the CHARM vector described in [38]: ChebyHist (Chebyshev and Chebyshev-Fourier statistics - 128 features), Moments (288 features), Textures (Haralick, Gabor and Tamura texture features - 211 features), Edges (28 features), ImageFeatures (Object statistics - 34 features), Histograms (144 features), Radon (48 features), and Zernike (144 features). Then for each of our reference datasets we ran eight classification experiments using each time the CHARM vector minus one of the eight groups as a features vector. In this way we could see how the absence of each of these features groups impacted on classification accuracy. For this experiment we kept the other parameters fixed. We ran our experiments in MATLAB, used WND as classification method and 10-fold cross-validation for the validation method as "save 25%" was not available in this software. In order to obtain a robust estimate of classification efficiency, the accuracy should always be measured in several runs with different training and validation sets. For this reason all the results obtained using the 10-fold cross-validation method result from the average of 10 training and testing runs.

We then carried out the same experiment for features levels. This time we separated the CHARM vector in three groups corresponding to unique elements in each of the three features levels such that these three groups were non-overlapping: v1' corresponds to v1 and contains elements computed in the raw image only, v2' corresponds to v2-v1 and hence only keeps features extracted from the transforms (Fourier, Wavelet and Chebyshev), and as v3' amounts to v3-v2-v1 there remains only measurements computed on the compound transforms (Fourier-Wavelet and Fourier-Chebyshev). We gathered three classification results for each reference dataset using the CHARM vector minus one of the three levels, each time using a different one. The classification algorithm and validation method remain unchanged, and the results were again averaged over 10 training-testing rounds.

Finally after investigating the importance of each features groups and levels, we wanted to test our CHARM-like vectors extracted using CellProfiler and described earlier in the Materials section. These different versions were built incrementally by adding features levels and groups to make sure we were not building an unnecessarily big vector while good results could have been obtained with a smaller one. We ran 100 classification rounds using WND and the "save 25%" validation and averaged the results to get a robust estimate of classification accuracy for each of the reference datasets and each of our CHARM-like features vectors. We could hence compare the performance of our different versions versus the original CHARM.

Feature Vector Interpretation We mentioned before that while features extracted from the original image can be easily related to morphological properties of the image, it becomes quite hard to understand which properties are captured when dealing with compound transforms. It seems that these "higher order" features are actually helpful for classification, so we wanted to try to understand better which role they played and what they meant by formulating an hypothesis and designing experiments to test it. Our hypothesis to explain the role of the higher level features in the CHARM vector was the following: we supposed that the features extracted from compound transforms did not really capture morphological elements in the image, but simply increased the dimensionality of the feature space. Indeed in some cases the distribution of the different classes in features space can be interwoven and very difficult to separate, but when lifted to a higher dimension space the separation becomes much easier. In order to test this hypothesis, we used two methods that relies on this principle: the Linear and Radial Basis Functions (RBF) kernel Support Vector Machine ([54] and [55]) and the Random Fourier Features ([42]).

Support Vector Machines (or SVM) are a family of powerful and mathematically well-defined classification algorithms using statistical learning theory and employed for many different applications. Since what was of interest for us was not really the mathematical details behind SVM classification but rather the implications for the features space when working with SVM, we give here a brief description of the situation and let the reader refer for further information and mathematical demonstrations to a very good online tutorial ([8]), a book ([13]) and a website referencing most of the information regarding SVM ([28]).

In the classical Linear SVM algorithm a cost function involving Support Vectors is minimized in order to find the optimal plane that best linearly separates classes distributions. Finding this optimal plane amounts to maximizing the margin between classes, defined as the distance between the closest elements from each class and the separation plane. The Support Vectors design these critical elements lying closest to the discrimination plane between classes. In SVM, the classification rule is a function of the Support Vectors as these samples are the most difficult to classify. SVM is usually defined for 2-class problems, and can be generalized to N-class classification tasks by performing N-1 "one-against-all" classification steps. Linear SVM is thus "just another linear classifier". SVM theory however also propose a solution for non-linearly separable problems using "the kernel trick" ([1]). Sometimes when dealing with data that cannot be separated by a line, it might be useful to project them onto a higher dimensional space. By mapping the features vectors of a non-linearly separable problem into a non-linear space one might end up with a linearly-separable problem in the higher dimension space and be back to the linear problem as schematically described in Fig. 2.4.

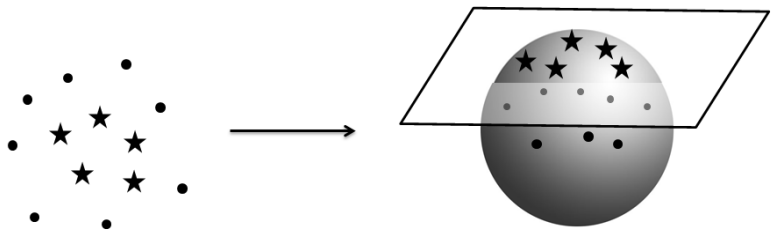


Figure 2.4: Example of simplification of classes separation in high dimensional space. The linear separation plane in the higher dimensional space corresponds to a non-linear discrimination function (a circle) in the original space.

The key element in Nonlinear SVM is therefore the mapping, a function Φ which takes as input a features vector and transform it in another vector containing many more dimensions. Classes can then be linearly separated in the new non-linear features space. The mapping function is in practice hardly used as it has a huge dimensionality, making it very inconvenient to compute. The trick is that by using some algebra it is possible to rewrite the SVM decision rule (or the optimization function) in a way that involves the dot product of Φ instead of Φ itself. A new function $K(\boldsymbol{x},\boldsymbol{y}) = \Phi(\boldsymbol{x})\cdot\Phi(\boldsymbol{y})$ called the kernel function can therefore be defined and used to compute the non-linear mapping implicitly. Two extremely widely used kernel function are the Radial Basis Functions kernel (RBF kernel) defined as $K(\boldsymbol{x},\boldsymbol{y}) = \exp -\gamma||\boldsymbol{x}-\boldsymbol{y}||^2$, and the Gaussian Radial Basis Functions kernel (Gaussian-RBF kernel) formulated as $K(\boldsymbol{x},\boldsymbol{y}) = \exp -\frac{||\boldsymbol{x}-\boldsymbol{y}||}{2\sigma^2}$ Both of these kernels correspond to a mapping in an infinite dimension space. As an example, the effect of the Gaussian-RBF can be explained with Fig. 2.5: this kernel acts by adding a Gaussian bump around each data point.

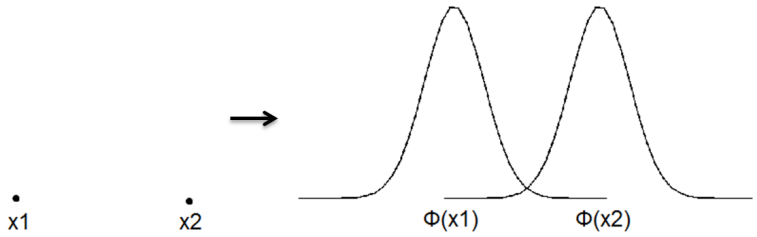


Figure 2.5: Effect of Gaussian-RBF kernel on data.

We hence decided to classify the lower level features only using RBF SVM and to compare this result to the whole CHARM vector classified using Linear SVM. If we obtained similar results, we could still hypothesize that the presence of the higher level features merely amounts to map the lower level features into a higher dimension space. We could also investigate if classifying the whole CHARM vector using Linear or RBF SVM yields similar results. If our hypothesis was true, then the CHARM vector should not benefit a lot from the RBF kernel since it already had a lot of dimensions.

Random Fourier Features is an algorithm proposed in 2007 by Rahimi and Recht ([42]) which

allows creating an explicit finite-dimension mapping approximating kernels corresponding to infinite-dimension mappings. First it requires a shift invariant kernel, i.e. a kernel function satisfying $K(\boldsymbol{x},\boldsymbol{y}) = K(\boldsymbol{x}-\boldsymbol{y})$. The Fourier Transform $\in \mathbb{R}^d$ of the kernel is computed, and an arbitrary D number of independent and identically distributed samples $w_1,...,w_D$ are drawn from it. D elements $b_1,...,b_D$ are also sampled from the uniform distribution $\in \mathbb{R}$ on $[0,2\pi]$. The randomized feature map $\boldsymbol{z}(\boldsymbol{x})$ is finally constructed using Random Fourier Bases $\cos(w'\boldsymbol{x}+b)$ with $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$, and is defined as Eq. 2.2 with the following relation to the kernel: $\boldsymbol{z}(\boldsymbol{x})'\boldsymbol{z}(\boldsymbol{y}) \approx K(\boldsymbol{x}-\boldsymbol{y})$.

$$\boldsymbol{z}(\boldsymbol{x}) \equiv \sqrt{\frac{2}{D}}[\cos(w_1'\boldsymbol{x}+b_1)...\cos(w_D'\boldsymbol{x}+b_D)]' \tag{2.2}$$

D is hence finite while the dimension of the mapping corresponding to $K(\boldsymbol{x}-\boldsymbol{y})$ might be infinite. If the Gaussian-RBF kernel is used we therefore simply have to sample the Gaussian distribution to get $w_1,...,w_D$ since the Fourier Transform of a Gaussian is a Gaussian with inverse variance. Using the Random Fourier Features followed by Linear SVM hence amounts to increase the dimensionality of the features space and then classifying using a linear classifier. If this gives us results that compare to Linear SVM on the CHARM vector, we could argue that the higher level features from the CHARM vector amount to these Random Fourier Features and are only adding dimensions.

With these tools in hands, we can summarize the experiments we did to test our hypothesis as follows. We used here the same nomenclature as before: v1' corresponds to the features of the CHARM vector computed on the original image only (first level).

1. RBF-SVM on v1' versus Linear-SVM on CHARM: if RBF-SVM on v1' allows us to obtain similar results as Linear-SVM on CHARM, our hypothesis cannot be rejected.
2. RBF-SVM on v1' versus RBF-SVM on CHARM: if RBF-SVM on CHARM and v1' gives us again somehow comparable results, our hypothesis cannot be rejected. In addition, we expect results obtained using RBF-SVM and Linear-SVM on CHARM to be comparable.
3. Linear-SVM and Random Fourier Features on v1' versus Linear-SVM on CHARM: if we can compete with Linear-SVM/CHARM results using Linear-SVM and Random Fourier Features on v1', then we could say that the compound transforms features are no better than Random Fourier Features. In other words, we could conclude that they amount to an approximation of the Gaussian Radial Basis Functions in finite space.

We used RStudio in order to perform this analysis. 10-fold cross-validation was used for validation as "save 25%" is not a standard method and was therefore not available in R. The results were averaged over 10 rounds of classification.

"Top 15%" Features In WND-CHARM, after features are extracted and weights are computed for each features, weights are ranked and a threshold is set such that all features with weights

below this threshold are ignored and only the best are used for classification. The threshold value was set to 15% in the publicly available version of the algorithm. Since we performed several runs of classification with different training and test sets in our experiments, the 15% features subset used for classification was not necessarily always the same. We wanted to investigate the nature of this 15% "top features" subset in two ways: its composition and its stability.

In order to analyze the top subset composition we extracted from the WND-CHARM HTML reports the lists of features used for classification on each of our reference datasets. We wanted to be able to check whereas subsets of features recur among the different datasets, again to make sure that there were no unique features subset that would be sufficient to provide good results. Since what interested us most was not which specific feature was highly ranked but rather which groups of features seemed to play an important role, we did not conduct an analysis of the occurrence of each particular feature among the top 15%, but rather observed the occurrence of each group in this subset. We expected that all the different groups would appear to be important as our reference datasets were composed of images of very different nature (face recognition, cell images, ...).

The second aspect we wanted to investigate was the stability of this top subset. As explained already in order to obtain a robust estimate of classification efficiency we usually run several training-validation rounds and then use the mean of the results as a measure of accuracy. We therefore wanted to observe whether the subset that played a role in classification was stable over these different runs, or if the top features varied a lot depending on the composition of the training set. This subsets had to be reasonably stable for a particular dataset if we wanted to be able to draw hypothesis on which features were important by looking at the top features subset of one classification experiment.

In order to assess the stability of our feature selection algorithm, we measured two correlation metrics based on [26]. First the Tanimoto distance measuring how similar is the content of features subsets s and s' from different classification rounds. This metrics is defined in Eq. 2.3 where $|s|$ stands for the number of elements (or cardinality) or subset s, and $|s \cap s'|$ is the number of elements in common (or intersection) between subsets s and s'.

$$S_S(s, s') = 1 - \frac{|s| + |s'| - 2|s \cap s'|}{|s| + |s'| - |s \cap s'|} \tag{2.3}$$

The numerator therefore measures how much elements are unique to each subset, while the denominator measures how much elements are present in total. Subtracting this fraction to one ensures that subsets that are very similar will get a high score (since the numerator will tend to zero) while subsets very dissimilar will get low scores. A score of 1 indicates that s and s' are identical, while a score of 0 indicates that no elements are shared between s and s'.

We also computed the Pearson's correlation coefficient measuring the similarity between

feature weights w_i in different classification rounds, defined as Eq. 2.4.

$$S_W(w, w') = \frac{\sum_i (w_i - \mu_w)(w'_i - \mu_{w'})}{\sum_i (w_i - \mu_w)^2 \sum_i (w'_i - \mu_{w'})^2} \tag{2.4}$$

Pearson's coefficient gives a measure of the correlation of the weights, and gives results in the interval $[-1, 1]$, where 1 (resp. -1) indicates perfect correlation (resp. perfect anticorrelation), and 0 indicates an absence of correlation.

Each of these two metrics gave us a NxN matrix for N runs of classification experiment. These matrices are symmetric with respect to the diagonal, so in order to analyze them it was sufficient to look only at the upper or the lower triangle. In the ideal case of a totally stable subset we would obtain a Tanimoto distances and a Pearson's coefficients matrices full or ones. We implemented these measurements in Python and computed them for a 100-runs classification experiment in each of our reference datasets.

Classification

The second step of WND-CHARM, the actual classification, is using the WND algorithm described in the Introduction, which is basically a modified version of the k-Nearest Neighbors algorithm where every points in the training set are considered and where features are weighted. Since this approach was quite simple we wanted to investigate how classification accuracy was impacted by changing the classification algorithm but keeping the CHARM vector. Another advantage that a more standard classification algorithm would have over WND is that it would be better defined and characterized. WND indeed contains empirical parameters and doesn't have nice mathematical properties as most usual classification algorithms do. We therefore tried to switch from WND to two classical classification methods: the traditional k-Nearest Neighbors, and the Linear Discriminant Analysis (LDA) coupled with Principal Components Analysis (PCA) as a pre-processing step.

k-Nearest Neighbors We already described k-Nearest Neighbors (k-NN, [17] and [11]) when mentioning WND in the introduction, and will here briefly recall its principle. k-NN is one of the simplest classification method and is often used as a first try when little is known about the data distribution. When an unknown sample is presented to the k-NN classifier, the distance from the new sample to each point in the training set is computed. The unknown sample is assigned to the class that is the most represented among its k nearest neighbors.

The parameter k in k-NN controls how the classifier is sensitive to noise. An example is shown in Fig. 2.6: the square data point to be classified seem to be more likely to belong to the stars distribution when we look at the whole dataset. However if k is set to be smaller of equal to two, the square will be classified as belonging to the rounds distribution because of the round outlier. If a bigger value of k is used, for instance k=3, the classifier is more robust and won't be influenced by noisy data points. For each of our reference datasets we therefore performed a

parameter exploration with a wide range of k values to be able to get the best possible results using k-NN. We ran these experiments in MATLAB using the CHARM vector and averaged the results of 10 rounds of 10-fold cross-validation.

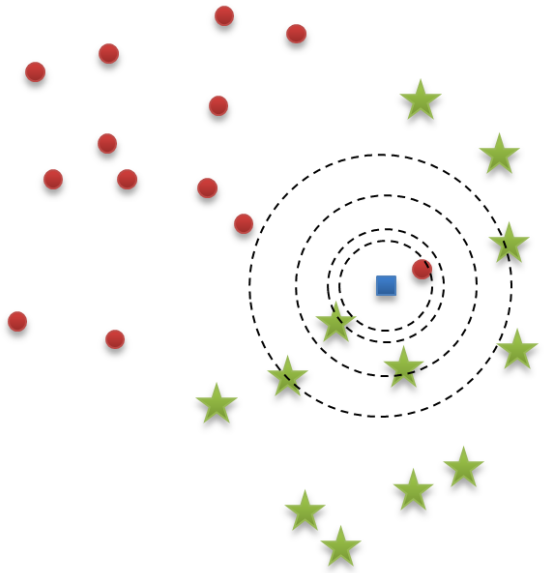


Figure 2.6: Effect of the k parameter in k-NN on the robustness of the classifier to noise in data.

Principal Components Analysis - Linear Discriminant Analysis The second classifier we tested is also a very standard method, but more refined than k-NN: Linear Discriminant Analysis, or LDA ([16]). LDA finds a lower-dimensional projection of the data such that the vaiance inside each class is small while the variance between classes is large, i.e. such that classes are compact and well-separated. The output of LDA applied to N classes are N-1 discriminant vectors perpendicular to the separation planes between classes. The classification rule is then a linear combination of these vectors. Unfortunately LDA alone doesn't work very well when the number of features is much larger that the amount of samples, that is when the dimensionality of the features space is too big for two reasons ([56]): first the estimate of the covariance matrix will most certainly be singular rendering the traditional discriminant rule inapplicable, and secondly the classification rule might become overly long and difficult to interpret as it is a combination of many different vectors. For these reasons a dimension reduction step is usually performed before applying LDA. In spite of this it is in general still useful to compute a larger set of features than what will actually be use after dimension reduction such that a wider variety of morphological aspects of the image can be captured, making the approach more general ([45]). The weighting and subsequent thresholding applied in WND-CHARM is an example of dimension-reduction method. A very commonly used and mathematically well characterized method for dimension reduction is Principal Components Analysis, or PCA ([39] and [22]). We therefore decided to use PCA as a pre-processing step and then LDA for classification. We will give a brief review of the principles behind PCA and LDA. We do not

aim here to fully re-demonstrate these two algorithms but rather to highlight the essential elements required to understand them.

PCA As mentioned earlier, PCA is a dimension reduction method and can therefore be used as a pre-processing step before doing LDA. For a data matrix X with M data (rows) and N features (columns), the goal of PCA is to reduce the features space to P dimensions where $P << N$ by re-expressing the data as a linear combination of the vectors of the X matrix (i.e. the features). The principal components (PC) are chosen in order to form an orthonormal basis. When doing PCA one makes two assumptions: linearity, and that the information content is proportional to the variance, or in other words that low-variance features correspond to noise. PCA therefore tries to maximize information over noise by keeping directions with largest variance and throwing away more constant ones. It looks for a rotation that would align with axis of maximal variance in the original data

The covariance matrix of data X is defined up to a constant as Eq. 2.5 where X^T indicates the transpose of matrix X.

$$\Sigma_X \equiv XX^T \tag{2.5}$$

Σ_X is a square and symmetric NxN matrix where the diagonal elements are the variance of each features and the off-diagonal values the covariance of each pairs of features in X. Since PCA looks for uncorrelated variables and maximal variance, it searches for a $\Sigma_{\tilde{X}}$ that would be zero everywhere except in diagonal. The algorithm can hence be summarized as follows: find A in $\tilde{X} = AX$, where X is MxN, A MxM, and $\tilde{X}$ MxN, such that $\Sigma_{\tilde{X}} = \tilde{X}\tilde{X}^T$ is diagonal. The rows of A will then be the PC of X. Using this and Eq. 2.5, we obtain Eq. 2.6.

$$\Sigma_{\tilde{X}} = \tilde{X}\tilde{X}^T = (AX)(AX)^T = AXX^TA^T = A(XX^T)A^T = A\Sigma_X A^T \tag{2.6}$$

By selecting A to be composed of eigenvector of Σ_X in rows, i.e. $A \equiv E^T$ where $\Sigma_X = EDE^T$ with E the eigenvectors matrix (in columns) and D the eigenvalues diagonal matrix, we obtain a diagonal $\Sigma_{\tilde{X}}$ as desired. Indeed since it can be demonstrated that $A^{-1} = A^T$, Eq. 2.7 is verified.

$$\Sigma_{\tilde{X}} = A\Sigma_X A^T = A(E^TDE)A^T = A(A^TDA)A^T = (AA^T)D(AA^T) = (AA^{-1})D(AA^{-1}) = D \tag{2.7}$$

Intuitively, the first PC is the normalized direction where the variance is maximal. The second PC is found by looking for the axis of second maximal variance, but it must be perpendicular to the first one since PCA vectors should form an orthonomal basis. The same process is repeated until N vectors are found. The PC then correspond to the eigenvectors of the covariance matrix of X, Σ_X, and the i-th diagonal value of $\Sigma_{\tilde{X}}$ corresponds to the variance of X along the i-th principal component.

In our case we use PCA to reduce the dimension of the features space while keeping enough variance. A threshold in percent is then set as the amount of variance in the original data to

be preserved. Since the diagonal elements λ_i of $\Sigma_{\tilde{X}}$ correspond to the amount of variance in the untransformed space, the constrain defined as "keep C percent of the variance in the original data" translates to "find the minimal p such that $\frac{\sum_{i=1}^{p}\lambda_i}{\sum_{i=1}^{N}\lambda_i} > C$" where p=1,...,N. The vectors associated to these p λ_i are the PC to use such that the desired amount of variance is conserved. PCA can also be defined using Singular Value Decomposition instead of eigenvalue decomposition, yielding a more general solution that can be used in the case where the covariance matrix is not diagonalizable. A SVD derivation of the algorithm can be found in [49].

LDA LDA makes the assumptions that class distributions follow normal laws of mean μ_i and covariance matrix Σ_i. It also assumes that all class covariance matrices have full rank and are equal (i.e. homoscedatic). LDA aims to maximize the ratio of interclass variance to intraclass variance, yielding to maximal separability. It is usually described as a two class problem and generalized to C classes by reapplying the 2-classes rule to each pair of classes. The intraclass variance Σ_i for each class i is computed as Eq. 2.8 where $\boldsymbol{x}_{i,j}$ is the j-th element and μ_i the mean of class i. The interclass variance Σ is defined as Eq. 2.9.

$$\Sigma_i = \sum_j (\boldsymbol{x}_{i,j} - \mu_i)(\boldsymbol{x}_{i,j} - \mu_i)^T, i = 1,2 \tag{2.8}$$

$$\Sigma = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T \tag{2.9}$$

The ratio of the variation between classes to the total variation inside classes is maximized as defined in Eq. 2.10 where $\boldsymbol{u}$ is a unitary vector designing the subspace onto which the data will be projected as $\boldsymbol{y} = \boldsymbol{u} \cdot \boldsymbol{x}$. It can be shown ([30]) that the optimal solution is defined as Eq. 2.11.

$$\max_{\boldsymbol{u}} \frac{\boldsymbol{u}^T \Sigma \boldsymbol{u}}{\boldsymbol{u}^T (\Sigma_1 + \Sigma_2)\boldsymbol{u}} \tag{2.10}$$

$$\boldsymbol{u} = (\Sigma_1 + \Sigma_2)^{-1}(\mu_2 - \mu_1) \tag{2.11}$$

The classification rule hence becomes Eq. 2.12 where C is a constant threshold and $\boldsymbol{x}$ an input of unknown class.

$$(\Sigma_1 + \Sigma_2)^{-1}(\mu_2 - \mu_1) \cdot \boldsymbol{x} < C \tag{2.12}$$

LDA therefore traces between each pair of classed (i, j) an hyperplane perpendicular to $\boldsymbol{u}$. If the vector $\boldsymbol{x}$ to be classified is on one side of the plane, it is classified as class i, otherwise as class j. The offset of the plane is defined by C.

All of our data were center and scaled along the columns in order to bring back all measurements to a common scale before doing PCA. As it was the case for parameter k in k-NN, we ran for each of our reference datasets parameter exploration experiments with a range of different C values, where C is the percentage of variance to be kept in PCA-transformed space. Usual values of C range from 95 to 99%. We ran these experiments in MATLAB, using the CHARM vector and averaging the results over 10 rounds of 10-fold cross-validation. We also implemented the use of PCA and LDA in Python using functions from the freely available scikit-learn Python library ([40]).

Validation

As explained before we implemented both the k-fold cross-validation and for the sake of comparison also WND-CHARM's "save 25%" method. We had a few concerns regarding this "save 25%" method. First we suspected it might give us over-optimistic results due to the fact the training set is built by taking the same proportion of samples from each class, which is quite an optimistic setup. In k-fold cross-validation the k splits of the data are built regardless of the classes, and it might therefore happen that a split does not contain all classes, or conversely that a split contains an entire class. The second aspect that did not convinced us was that the variance of the classification accuracy between classification runs was very large with "save 25%" compared to k-fold. Finally k-fold is a standard method widely used for validation and should therefore be preferred over "save 25%".

In order to investigate the differences between these two validation methods we ran a serie of experiments for each of the reference datasets in which we kept the CHARM vector and the WND classifier as in the original algorithm, but changed the validation method. We averaged the results over 100 runs of classification using "save 25%" for the validation, and 10 runs using 10-fold cross-validation. We chose to gather results on ten times more runs for "save 25%" than for 10-fold as the result obtained after one run of 10-fold cross-validation is already an average over 10 training-testing runs. We performed these experiments using our Python implementations of the two validation methods.

When using k-fold cross-validation the choice of the value of k is a very debated question. A value of 10 is most generally arbitrarily chosen, and in other cases it is suggested to use a value of k such that the number of samples per class is a multiple of k. Large values of k tend to have very long computation time and large variance, but give a less biased estimate of the actual accuracy as the size of the training set tends to be closer to the full dataset (since the splits are smaller). We compared the usual 10-fold with the "save 25%" method, but also wanted to separately investigate the effect of adapting k to the nature of the dataset. We tried to adapt the value of k to the number of images per class such that we would be confident in the fact enough representative of each class have the possibility to be in each split. Following suggestions from our colleagues we used "formula" 2.13 in order to find an optimal k for each reference dataset, where #Images per Class represents the number of images for a particular

class.

$$k \leq \frac{\min_{\text{classes}}(\#\text{Images per Class})}{5} \tag{2.13}$$

In this way the number of samples per class is at least 5 times bigger than k (5 was chosen arbitrarily) such that we do not have more splits that samples per class. This does not ensures us that we have an equal proportion of representatives from each class in each split, or that there is a representative at all from each class in each split. It is just a way to make sure we do not split the data in a way that it wouldn't even be possible to have samples from each class in each splits. We ran again two series of 10 runs of k-fold cross-validation for each of our reference datasets, once using the usual $k = 10$ and then using $k = k_{max}$.

2.2.2 A New "WND-CHARM-like" Algorithm

Using the results from the various steps of WND-CHARM we were able to create a new WND-CHARM-inspired algorithm that we called "PCA-LDA-CHARM-like". Our new algorithm is composed of the feature vector v3.2 described in the Material section which contains 953 features. Its construction is inspired from CHARM in the sense that it contains the same kind of features "groups", and it is built in a hierarchical way using elements computed on the original image, on transforms, and on transforms of transforms of the image. PCA is used for dimension reduction such that 98% of the variance present in the original data distribution is conserved. Classification is performed using Linear Discriminant Analysis, followed by 10-fold cross-validation for validation. A flowchart of our algorithm as compared to WND-CHARM is shown in Fig. 2.7.

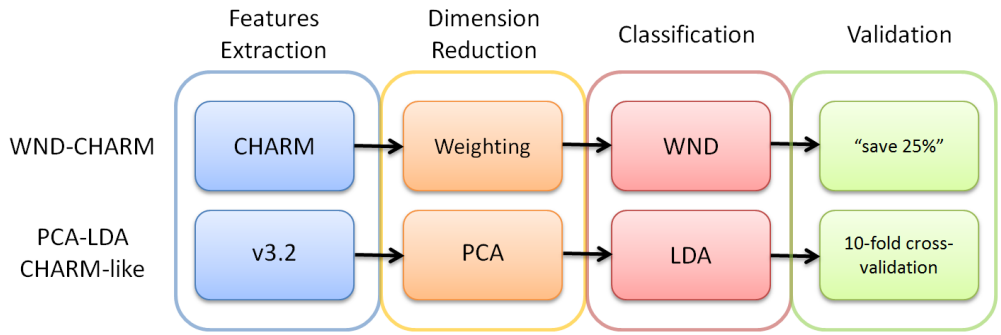


Figure 2.7: Comparison of the original WND-CHARM and our new PCA-LDA CHARM-like algorithm.

As a first step we needed to do some experiments to validate our new algorithm. First we ran twice on each reference dataset 100 rounds of classification using "save 25%" for validation and 10 rounds with 10-fold cross-validation using our Python code. The first time we used WND-CHARM, and the second time PCA-LDA-CHARM-like. We could in this way compare the results of our method versus WND-CHARM's for each validation method and make sure

the algorithm we propose was as powerful as the one it was inspired from.

After making sure our PCA-LDA-CHARM-like algorithm was valid we decided to perform some more experiments to identify how we could improve even more the performances we could get with it. We re-used the different features vectors we designed when testing WND-CHARM and evaluated the performance of each version using PCA-LDA. In this way we wanted to make sure the version we selected based on results using WND was the optimal solution for PCA-LDA as well. We then focused on trying different classification methods. We used R to gather results using our CHARM-like vector v3.2, three rounds of 10-fold cross-validation, and several different classifiers: k-NN, PCA-LDA, LDA and other dimension reduction methods, Penalized LDA, Random Forests and SVM. As k-NN, PCA-LDA and SVM were already explained previously in this chapter, we only give here a description of the two remaining classification methods.

LDA and other Dimension Reduction Methods As explained when describing LDA, this algorithm fails when working with a feature space of too many dimensions, and a dimension reduction step is therefore needed. We chose to use PCA, but other ways of reducing the number of dimensions exist. What we tested here consists of removing highly correlated features and features with a near-zero variance. In order to remove highly correlated features we first compute a correlation coefficient for each pair of features and obtain a NxN correlation coefficient matrix for a M samples of N features dataset. We then look at all pairs of features having a correlation coefficient higher than a threshold that we set to 0.9. For these pairs, we compute the mean correlation over all other features pairs, and the feature of the pair having the biggest mean correlation is excluded from the set. R contains a method that detects features with a variance close to zero that might cause trouble. A feature is considered as having a near-zero variance when it meets two conditions: first it only takes a small number of different values across the different samples (i.e. there are only a few unique values), and then the ratio of frequencies for the two most represented values is large. Two thresholds are set to determine when these conditions are met. Using these preprocessing steps we could make sure we would end up being able to use LDA.

Penalized LDA Penalized LDA or pLDA ([56]) is a new method published in 2011 and available as part of the `caret` package in R ([29]). It is basically a modification of LDA such that it includes an implicit dimension reduction method. The idea of pLDA is to penalize the discriminant vectors with a penalty function, and then to use only the highest $n < N-1$ penalized vectors, where n is arbitrarily chosen and N is the dimensionality of the feature space. The penalized function used in the R implementation is the $\mathscr{L}_1$ norm, such that features that vary a lot within each class get a higher penalty. A threshold can be set on the penalties such that the final decision rule only contains a subset of the features.

Random Forests Random Forests (RF, [6]) differs in its nature from the other algorithms we tested. It belongs to a family of methods called "ensemble classifiers" which consist of creating a collection of models to classify data with the best possible accuracy. When training RF, a set of data is first drawn from the original dataset, and a tree is constructed by randomly selecting a subset of features at each node and using the most powerful ones to make a decision. When the first tree is fully grown, another subset is drawn again (with replacement) from the initial set of data, and a new tree is built following the same method. This operation is repeated as many times as there are different classes in the training set to yield multiple decision trees, hence creating a "forest". For classification, a sample is simply presented to the forest and assigned to the class that is most represented in the outputs of all the trees. We wanted to try this methods for several reasons. The fact we are dealing with a potentially very large features vector often requires a dimension reduction step as it was the case for LDA, but RF has the advantage of containing an implicit feature selection procedure with the random sampling of the features at each node. For this reason the algorithm performs equally well depending on the size of the dataset. Unlike the weighting-thresholding process in WND-CHARM, the feature selection procedure in RF also has the advantage of keeping all the features so each of them can potentially play a role in classification. Last but not least, this method has the reputation of obtaining very accurate results. We used the `caret` package that uses the R implementation ([32]) or the algorithm.

2.2.3 Datasets

We gathered results on several different kind of datasets. Some of them were used for comparison purpose with the original WND-CHARM while others were chosen in order to show potential application fields of our method. We give a brief description of the nature and content of each of these datasets and explain why they were of interest.

Reference Datasets

What we called "reference datasets" are seven datasets of very different nature that were used to test WND-CHARM in [38]. We used these images for the whole of our analysis of WND-CHARM since they allowed us comparing our results with the ones presented in [38]. They contain various images ranging from biomedical to face recognition in order to show that the algorithm is efficient on a wide range of data. All these sets were composed of black and white tif images.

Faces Images The AT&T ([44]) and the Yale ([18]) datasets are two face-recognition image sets. In both cases the classes correspond to different human subjects, and in each class samples are images of the same person taken at different orientation (full-face or profile pictures), different illumination settings, with different facial expressions, and with different elements hiding the face (glasses, beard, etc.). The AT&T dataset is composed of 92x112 pixels

images separated in 40 classes with 10 images per class. The Yale image set contains 15 classes with 11 320x243 pixels tif images per class. This kind of dataset is typical for benchmarking face recognition algorithms.

Textures Images The Brodatz texture dataset ([7]) is a collection of pictures of various textures. There are 111 different textures in total which constitute the different classes. Each of the 111 big texture images are chopped down into 16 tiles of 160x160 pixels to yield a total number of 1776 images. The Brodatz dataset is widely used as a reference to test new algorithms as it presents a wide range of different textures that can be found in nature.

Biological Images Three biological datatsets are present in this group: the Chinese Hamster Ovaries (CHO) ([4]), the Pollen ([15]), and the HeLa dataset ([5]). The CHO is composed of fluorescence microscope images labeled with five components targeting different intracellular organelles (Golgi apparatus, DNA, lysosomes, nucleoli and tubulin) which constitute the five classes. Each of these classes are populated by different number of 512x382 or 512x512 pixels images for a total of 327 images. The HeLa dataset is built in a similar way: its 8 classes corresponds to antibodies or dyes staining different organelles and cellular components. The 689 images in the dataset are 382x283 tif fluorescence microscope images non-evenly split between the different classes. The ability to automatically identify components inside the cell can be of use for many different biological applications, for instance for localizing a fluorescently tagged protein, which can help discovering unknown proteins and genes functions. Finally the Pollen dataset is composed of 630 very small 25x25 pixels images acquired using phase contrast and evenly split in 7 classes corresponding to seven different kind on Pollen grains. This dataset is fairly easy to classify and usually requires only a very small amount of processing time due to the small size of its images.

Object Images The COIL-20 dataset [35] is composed of pictures of 20 different random objects ranging from toys to kitchenware. Each object was photographed 72 times yielding 128x128 images with different orientations or different illumination settings. The fact it is composed of objects of very different nature makes classification relatively easy. While the Brodatz database is widely used to measure the ability to make a distinction between classes based on textures, the COIL-20 is a classical set used to assess an algorithm's robustness to illumination and orientation changes.

The content of these seven datasets is summarized in Table 2.2.

Table 2.2: Description of each reference dataset's content.

Dataset	Number of classes	Total number of images
AT&T	40	400
Brodatz	111	1776
CHO	5	327
COIL-20	20	1440
HeLa	8	689
Pollen	7	630
Yale	15	165

IICBU Datasets

The IICBU ([47]) is a collection of biological images datasets put together by the NIH to serve as a benchmarking suite for new biological images analysis algorithm. This serie of datasets has also been used in [46]. Since it is composed of sets that cover a wide range of biological applications and since reference results using the original WND-CHARM were available we used these data to verify our algorithm. All these images were black and white tif files.

High-throughput Screens Two datasets in the IICBU serie are part of high-throughput screens: the RNAi and the Binucleate datsets. The Binucleate dataset is a two classes problem where the goal is to discriminate cells from the fly D. Melanogaster composed of two or only one nucleus. Each class is composed of 20 1280x1024 pixels fluorescence microscope images acquired at 60x where DAPI is used to label the DNA and hence highlights cell nucleus. No manual quality control has been made on these data since images were acquired automatically in a high-throughput setting. Polynucleated cells usually indicate a problem in the cell cycle and can therefore be the phenotype of interest when working with compounds suspected to affect cytokinesis. This kind of setup could for instance be found in cancer research experiments where researchers screen for drugs interfering with cell division. The second high-throughtput dataset, RNAi, is composed of fluorescence microscope images acquired automatically with a 60x objective. These images feature D. Melanogaster cells where several genes have been knocked-down using RNAi, and which have been stained with DAPI. The 10 classes correspond to ten genes targeted by selected RNAi, and each contain 20 1024x1024 pixels images. Even though the classificaton problem is similar as in Binucleate, the images from the RNAi dataset are much more difficult to classify as the differences between phenotypes are more subtle. Classifying cell phenotypes automatically from high-throughput data as it is the case in these two datasets is a recurring problem in biological image analysis.

Liver Images Liver Aging, Liver Gender CR and Liver Gender AL are different sets involving mouse liver brightfield microscopy images coming from a NIH-based project called the Atlas

of Gene Expression in Mouse Aging. All RGB data were obtained using a 40x objective and were subsequently transformed into grey-levels images in MATLAB. The 1388x1040 images represent sections of liver tissue from 30 different mice stained with eosin and hematoxylin with very low staining variability. The images in the two liver gender experiments Liver Gender AL and Liver Gender CR come from six months male and female mice on Caloric Restriction (CR) or Ad-Libitum (AD) diets. The 265 images in Liver Gender AL and the 303 images in Liver Gender CR are separated in both cases into two classes corresponding to the gender of the mouse. AD diet is known for inducing a larger variability in liver aspect in mice of the same gender, making the classification task more complicated. Finally the Liver Aging dataset is composed of 529 images coming from female mice on AL diet taken at 4 different time points (month 1, 6, 16 and 24). Since aging is a continuous process and greatly varies among individuals this classification task is expected to be harder than the Liver Gender series.

Lymphoma The Lymphoma dataset contains images of 3 types of malignant lymphoma, a type of cancer of the lymph nodes. The original color images were acquired using a brightfield microscope and were subsequently processed in MATLAB to make them grey-levels images. The sample tissues come from very different sources and therefore exhibit large variations in the way they are stained, which makes this dataset look like a difficult real-life problem. Each of the three classes are composed of 1388x1040 pixels images with a total of 374 images in the dataset. Automated classification of cancer types from tissue images is as area of growing interest as it would allow for faster and less expensive diagnosis and might therefore have a big impact in the medical world.

C. Elegans Images The Terminal Bulb dataset is composed of images of the C. Elegans nematode taken at different time points, where the goal is to determine the age of the worm based on the images. The dataset is composed of images of the pharynx terminal bulb, a part of the digestive tract of the worm. They were obtained using differential interference contrast (DIC) microscopy with a 20x objective. Here again the 970 300x300 pixels images are split into 10 classes corresponding to time points (days 0 to 12 with one time point every 2 days). The IICBU suite contains another dataset based on C. Elegans images composed of fluorescence microscopy images acquired at 20x of different muscles of the worm split in four groups corresponding to the day of acquisition (day 2, day 4, day 6 and day 8). Each class is composed of 1600x1200 pixels images, 237 images in total. We did not use this latter set because we were unable to obtain WND-CHARM results as the C++ version crashed on it. These dataset are expected to be difficult to classify for two reasons: first each class is composed of images of different worms and the process of aging greatly varies among individuals, and then classes form in these cases a continuum which makes the difference between them more subtle. In addition to that the Terminal Bulb data are DIC images, known to be more difficult to process than traditional microscopy data.

The CHO, HeLa and Pollen datasets already present in the Reference datasets are also part of

the IICBU suite. Table 2.3 summarizes the content of these ten image sets.

Table 2.3: Description of the content of each IICBU image set

Dataset	Number of classes	Total number of images
Binucleate	2	40
CHO	5	327
HeLa	8	689
Liver Aging	4	529
Liver Gender AL	2	265
Liver Gender CR	2	303
Lymphoma	3	374
Pollen	7	630
RNAi	10	200
Terminal Bulb	7	970

BBBC Datasets

The Broad Bioimage Benchmark Collection (BBBC) is a list of freely available biological datasets with ground truth that can be used to validate an algorithm in realistic conditions. We selected four datasets from the BBBC, all being fluorescence microscope images from high-throughput assays. In every cases signals from two channels were available: one from a dye labeling the DNA, and the second from GFP-tagged proteins. A difference between positive and negative phenotypes can more easily be found in the GFP channel as the DNA doesn't really changes between the two conditions. We first classified these datasets using the information from both channels, and then compared it to the results obtained using GFP channel only. If we found classification results using GFP channel alone to be satisfying, we would discard DNA channel and work only with the information from GFP channel as it would be useless to handle more data when they do not add any information for classification. In all these image sets the goal was to distinguish between "positive" and "negative" phenotypes. The sets can be separated in two groups corresponding to two types of experiments: translocation and transfluor assays. As most of these datasets were composed of only a small number of elements, which could be problematic for classification as the training and validation sets tended to be really small, we enriched these datasets by chopping down images in 2x2 equal-sized tiles. The number of data was therefore increased by four. We obtained results on the enriched datasets and compared them to the performance achieved on the original ones.

Human U2OS Cells Cytoplasm–Nucleus Translocation Both translocation assays involve a protein fused with GFP and transfected in human osteosarcoma cells (U2OS). In BBBC013 the protein of interest is located in cell cytoplasm and naturally moves inside the nucleus where it is actively exported back into the cytoplasm. Cells are treated with two different drugs

(Wortmannin and LY294002) such that the proteins responsible for the nuclear transport are inhibited and the protein of interest is therefore trapped into the nucleus. The drug-treated cells are considered as "positive". The original 640x640 pixels images came from a 96-well plate. The 8 Positive (respectively Negative) controls wells along with the highest (respectively lowest) doses of each drugs were selected to constitute the dataset, therefore ending up with 32 images. On the enriched datatset, these images were split in 4x4 tiles, yielding 128 320x320 images. The setup of BBBC014 is quite similar to BBBC013, with a different GFP-tagged protein of interest. The images in BBBC014 were acquired from a 96-well plate with two different cell lines (MCF7 and A549) using a 10x objective. We again used images from the controls and the first closest dose to represent the positive and negative phenotypes. This dataset is composed of 32 1360x1024 pixels images. The tiled version therefore contains 128 680x512 pixels images.

Human U2OS Cells Transfluor The two transfluor assays were performed using the same cell line (U2OS) as the translocation assays. Cells expressed a receptor and a GFP-tagged protein such that when stimulated by a particular compound the receptor triggered a cascade of events leading to the generation of protein vesicles inside the cell. In the BBBC015 dataset four fields were acquired in each wells of a 96-well plate, and we selected the wells containing the two highest and two lowest compound concentrations. Our dataset was initially composed of 48 1000x768 pixels images evenly split in positive and negative phenotypes. When enriched by tiling the original images, it yielded 192 500x384 pixels images. BBBC016 is the smallest dataset as only three fields from one row of a 384-well plate were acquired. By selecting the two highest and two lowest doses we obtained 9 positive and 9 negative 512x512 pixels images. When tiled to enrich the dataset, these data gave 72 256x256 pixels images.

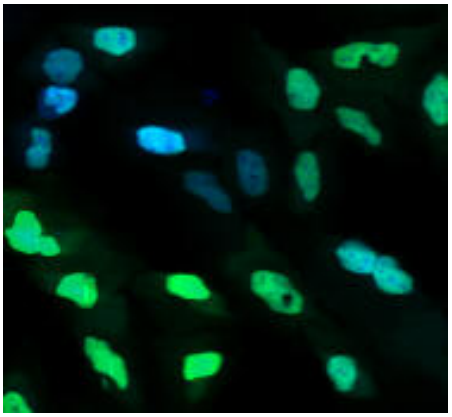
A summary of the content of each of the BBBC datasets we used can be found in Table 2.4, and examples of images for each class of the four datasets are presented in Fig. 2.8.

Table 2.4: Description of the content of the BBBC datasets

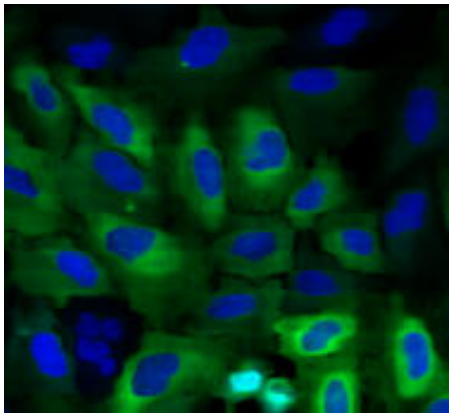
Dataset	Number of classes	Total number of images
BBBC013	2	32
BBBC014	2	32
BBBC015	2	48
BBBC016	2	18

For each image set we did not only gathered classification results, but also performed significance tests as we did earlier for WND-CHARM. These tests gave us a measure of random classification efficiency in the datasets, which could be used as a reference value to assess how good our actual classification efficiencies were.

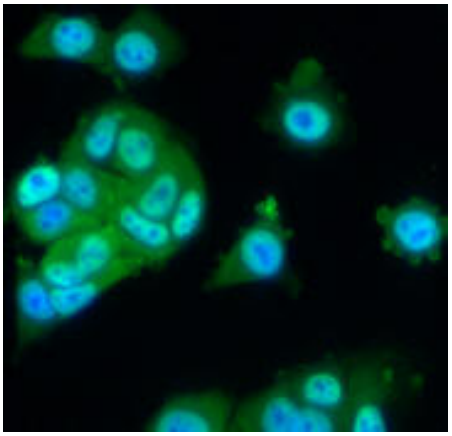
(a) BBBC013 - Positive



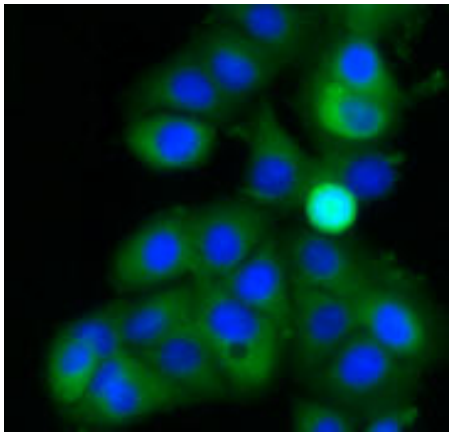
(b) BBBC013 - Negative



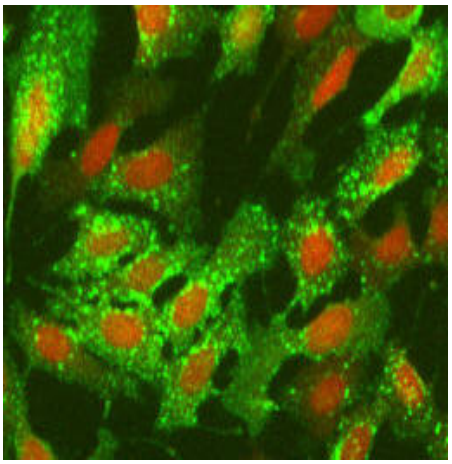
(c) BBBC014 - Positive



(d) BBBC014 - Negative



(e) BBBC015 - Positive



(f) BBBC015 - Negative

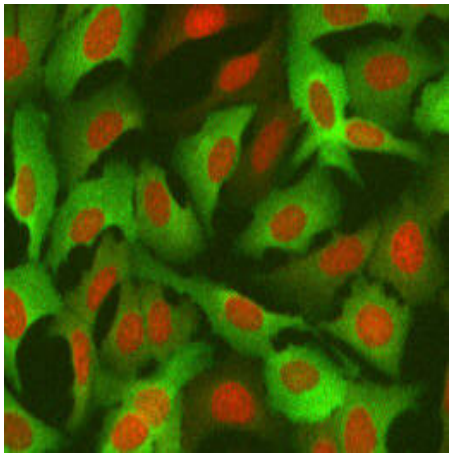
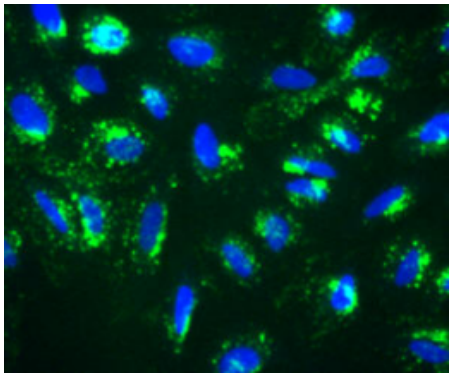


Figure 2.8: Example of images from each of the BBBC datasets. Source: [2].

(g) BBBC016 - Positive



(h) BBBC016 - Negative

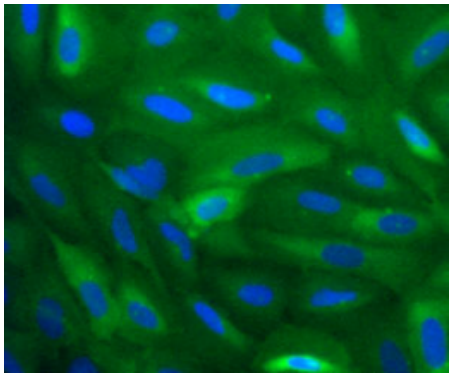


Figure 2.8: (Cont'd) Example of images from each of the BBBC datasets. Source: [2].

Since the BBBC datasets were good examples of the typical kind of data that would be processed using CellProfiler we used them to measure how much value the various modules for features extraction we implemented added to the software when doing whole image-based classification. We computed the Z' factor for every feature of our v0 and v3.2 vectors. As a reminder v0 is composed only of CHARM-like features already available in the current release of CellProfiler while v3.2 contains the addition of many more measurements obtained through our newly implemented moduled. The Z' factor ([58]) is defined by Eq. 2.14 where $\hat{\sigma_i}$ is the standard deviation and $\hat{\mu_i}$ the mean of samples for a particular feature in class i. It indicates how well two populations (in our case positive and negative phenotypes) are separated. It is therefore used as a measure of experiment quality: the value of the Z' factor tells how much discrimination power the measurements used to describe the two classes have. We preferred using this metrics for comparison over classification efficiency because as these datasets were fairly easy problems classification accuracies obtained using v0 and v3.2 did not exhibited significant differences. As the Z' factor is a more sensitive measure of the ability to separate classes we hoped it could give us a better estimate of the effect of the features we implemented.

$$Z' = 1 - \frac{3(\hat{\sigma_1} + \hat{\sigma_2})}{|\hat{\mu_1} - \hat{\mu_2}|} \tag{2.14}$$

The Z' factor can be easily understood graphically as shown in Fig. 2.9. This measurement obviously assumes that the data from the two classes follow normal distributions. The maximal possible value of a Z' factor is 1 and indicates the best possible setting (perfect separation). Usually Z' factors with values larger than 0.5 are considered to indicate an excellent setup while values larger than zero are acceptable ([33]).

We measured Z' factors for every features of the v0 and v3.2 vectors on the tiled BBBC datasets and reported the highest Z' values for each dataset as it technically corresponded to the best classes separation we could obtain with the features we measured on the images.

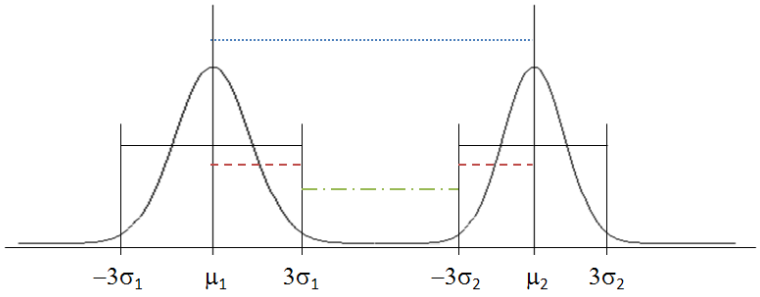


Figure 2.9: Graphical explanation of the Z' factor: 3σ corresponds to 99% of the data in each class. The numerator $3(\sigma_1+\sigma_2)$ corresponds to the two dashed red lines and the denominator $|\mu_1-\mu_2|$ to the pointed blue line. The Z' factor therefore indicates to which fraction of the blue line corresponds the separation band shown as a dashed and pointed green line.

Tissue Dataset

As we are working with whole-image-based classification, tissue data were an area of big interest. While we didn't think that a WND-CHARM-like algorithm could really beat segmentation-based methods on images of cell-based assays composed of a lot of background and subject to population effects, we however suspected that it could become very powerful in images where segmentation was not even possible for example in identifying organelles in tissue data. We therefore worked with images from the Human Protein Atlas (HPA, [53]), an open source database of tissue images. The HPA is a very large-scale high-throughput project involving high-resolution data obtained using tissue microarrays of 20 cancerous and 45 normal tissues from different organs. The goal of the HPA project is to study the spatial localization of every human protein using labeled antibodies in order to better understand their functions. More than 3000 different antibodies and close to 3'000'000 tissue images are freely available on the HPA website ([19]).

All data were bright-field microscope images acquired with a 20x objective and stained with two different compounds: one that targets the antibody against the protein to be localized and that exhibit a brownish color, and the second that non-specifically stains with a blue color the nucleus of all cells as well as extracellular compounds. These images are traditionally analyzed manually, and there is therefore a large need for automation. In collaboration with Carolina Wählby et al. at the Center for Image Analysis, Uppsala University, Sweden, we had the opportunity to work on a dataset built using images from the HPA in which the task was to identify different subcellular compartments. 1057 64x64 pixels images were manually selected from 61354 patches created by tiling 3000x3000 pixels images of 19 normal and 10 cancerous breast tissue from the HPA website. The original images were stained with labeled antibodies against two proteins of interest. These 1057 patches were manually annotated as belonging to one of three possible protein locations (cytoplasmic, nuclear and connective tissue) or as background. As a result, the 1057 images were non-evenly split into the four classes: 200 in the nuclei, 462 in the cytoplasm, 202 in the conjunctive tissue, and 193 patches containing background only. Samples from each class are presented in Fig. 2.10 to illustrate this dataset.

(a) Background

(b) Cytoplasm

(c) Nuclei

(d) Conjunctive Tissue

Figure 2.10: Example of elements of each classes of the Tissue datasets. Source: Carolina Wählby, personal communication.

Since the original images were RGB, a color deconvolution algorithm was applied using ImageJ ([36]) to separate the protein and nucleus channels. Since we did not know for sure which channel would provide the most information on the difference between classes, we performed classification experiments using each channel separately and all together. We compared these results in order to determine which images should be used to achieve good classification results. Using the best channels we obtained classification results and compared it to a distribution of random classifiers.

We wanted to work on more tissue data from the HPA, but the images available on the website were annotated in such a way that many preprocessing operations were required before obtaining a usable dataset. After tiling the small patches had to be manually annotated by a trained expert. Due to time constraints on this project we were therefore unable to create ground truth for a new dataset from the HPA.

HDAC Dataset

Our Histone Deacetylases (HDAC) dataset comes from an ongoing project at the Imaging Platform. This high-throughput study involves different laboratories at the Broad Institute and focuses on identifying HDAC inhibitors. The HDAC protein family is involved in many different core biologic processes and is linked to a variety of human diseases (neurological and cardiac diseases, cancer and stroke for instance) making HDAC inhibitors a target of

interest for drug research. HDAC can have several isoforms, but no isoform-specific cell-based assay has been proposed so far. It was therefore suggested to identify different HDAC by characterizing each isoform's phenotype using image analysis.

Our images were acquired from a 384-well plate using a fluorescence microscope with a 20x objective. Five fluorescence channels were recorded, each corresponding to a dye labeling cellular components. Six fields per well were acquired, but we used only one and discarded the five others to avoid correlation effects as we had enough data at our disposition using one field only. Images from the same well indeed have a higher risk of being correlated and biasing the classification results by underestimating the true classification error. siRNA were used to target nine different HDAC with three different siRNA construct per isoform, and we obtained data for construct against five HDAC: 3 constructs against HDAC 8, 2 against HDAC 7, 3 against HDAC 6, 3 against HDAC 3 and 3 against HDAC 1. Our goal with this dataset was therefore a 14-classes problem: we had to try to distinguish different phenotypes for each of the two or three constructs of the five HDAC isoforms. Since the difference in phenotype between HDAC isoforms is extremely subtle batch effect and plate-to-plate variations had to be minimized as much as possible. For this reason the isoforms were plated following a scrambled layout such that they did not form clusters that would favor global effects. Example of images from each isoform class are shown in Fig. 2.11 to let the reader appreciate how subtle the differences between classes are.

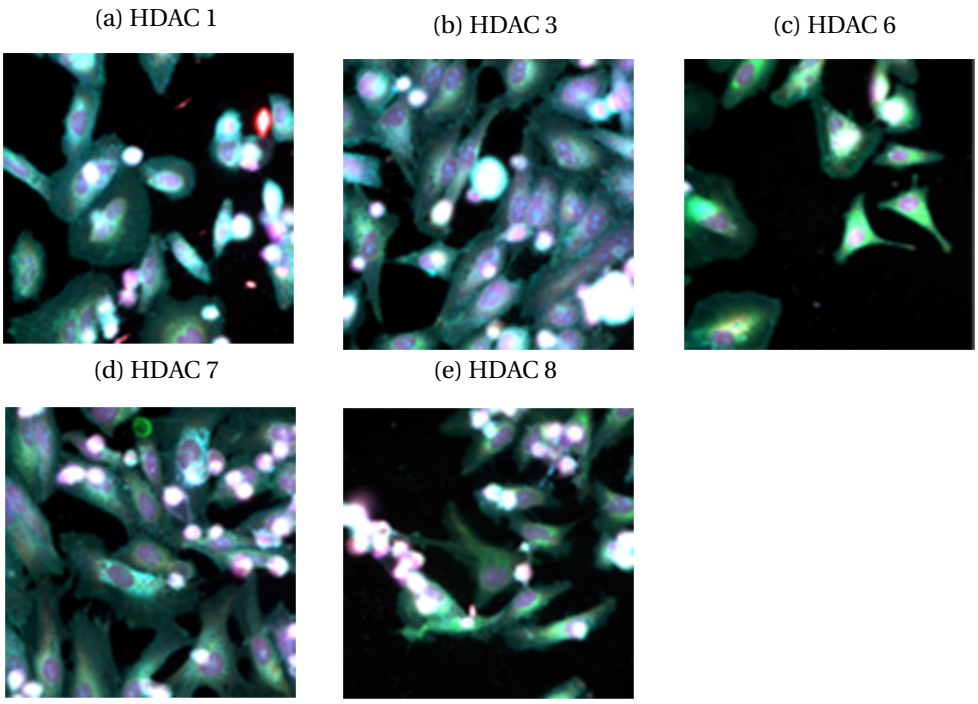


Figure 2.11: Example of samples from each HDAC isoform class. Constructs against the same HDAC are not illustrated. Source: Shantanu Singh, personal communication.

As we had no prior information on which channel captured differences between classes, we

initially performed classification experiments using all the five channels together. Following this, we carried out a per-channel analysis to see if we could identify channels that were more informative than the others. As we expected differences between constructs against the same HDAC to be very light, we also tried to group the different constructs together and classify based on isoforms only, which brought us back to a simpler 5-class problem. Finally we reproduced the analysis using all channels together and separately on a subset of the full HDAC dataset containing only two classes which were known to be more easily separable. Every time a classification efficiency measure was obtained, we compared it to a reference random efficiency.

This dataset presents many different difficulties: the classification task is in itself very challenging as phenotypical differences between isoforms are faint and images are highly subject to global bias (plate and crowd effects). Moreover this is an ongoing project which implies that issues are identified and new data are produced regularly. As we started with a very difficult problem, we designed this serie of experiments to have decreasing difficulty to try to identify the limits of our algorithm.

Neuronal Outgrowth Datasets

We gathered neuron images from an experiment performed within the framework of a collaboration with the Imaging Platform. Our images come from a 96-well plate with 60 wells of usable data. The aim of this experiment was to look at the influence of different chemicals on neuronal growth. Some chemicals inhibit while others enhance growth, resulting though their addition to a modification of the neurites. The plate contains a dose-response curve for a different chemicals on each row, with increasing concentration along the row. The six chemical used in the experiment are called Chiron99021, DOI, Quetiapine, Quetiapine Intermed, Serotonin and TDZD-8. We decided to consider each row as a separate dataset and tried to discriminate positive versus negative phenotype for each drug. We therefore selected images at zero concentration to represent the negative phenotype, and images at maximal concentration for the positive phenotype. In this experimental setup we therefore had 6 different datasets and hence performed 6 different classification experiments. Eight fields were acquired per well, and we conserved images from each field despite the risk of correlation because of the scarce amount of available data. Three fluorescence channels were acquired for each image. The first channel gathered signal from neuronal body, or nucleus, the second one imaged the dendrites, which are the shorter neurites, and the last channel contained signal from the axons, or longer neurites. Typical samples from each class of each dataset are displayed in Fig. 2.12.

As it was the case for the BBBC and Tissue datasets, we were not sure of which channel should be used to achieve best results. We hence carried out experiments using channels together and separately and compared the results. We then gathered classification results using the optimal channels combination, and assessed their quality by confronting them to the results of random classification of the data which served as a reference. A summary of the content of

the 6 datasets is presented in Table 2.5.

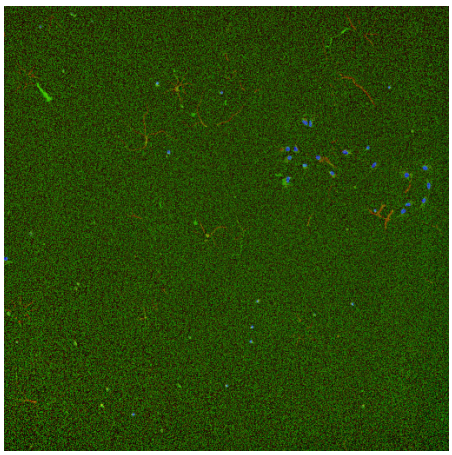
Table 2.5: Description of the content of the Neuronal Outgrowth datasets.

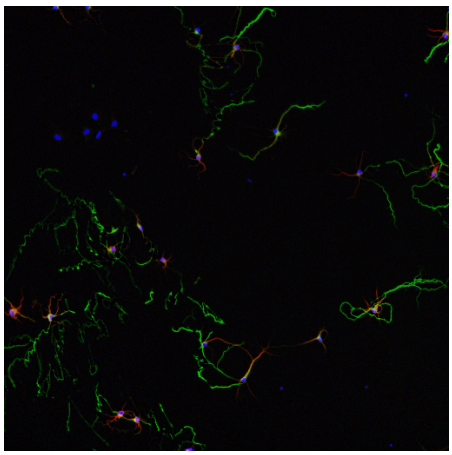
Dataset	Number of classes	Total number of images
Chiron99021	2	54
DOI	2	54
Quetiapine	2	53
Quetiapine Intermed	2	45
Serotonin	2	54
TDZD-8	2	54

These six two-class classification experiments remind of the BBBC as we are trying to identify positive and negative phenotypes, but the problem is here much harder for several reasons. The first difficulty comes from the nature of the available data: this plate contains a large amount of representative of the negative phenotype, but the best positive controls were known to be located in another plate whose images were not available. The experiment in itself is also more difficult than the BBBC. While we were looking at highly visible effects in the BBBC experiments, the action of the various chemicals on neuronal outgrowth is much lighter. These potentially very subtle effects therefore render the classification task much more complex. No quantitative results using cell-based classification were available for comparison on these data. However some metrics obtained after segmentation (axon length for instance) were known to obtain an acceptable Z' factor (David Logan, personal communication). This problem could therefore be characterized using per-object classification as hardly screenable, but not too subtle either.

(a) Chiron99021 - Positive

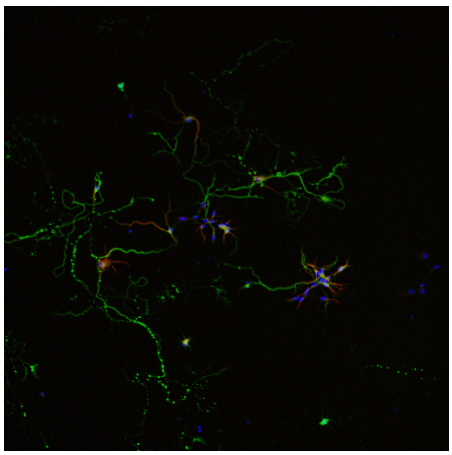
(b) Chiron99021 - Negative

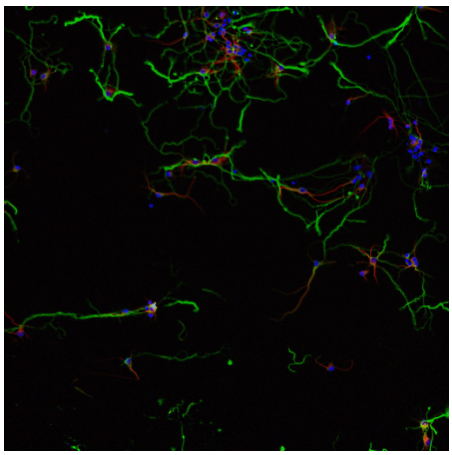




(c) DOI - Positive

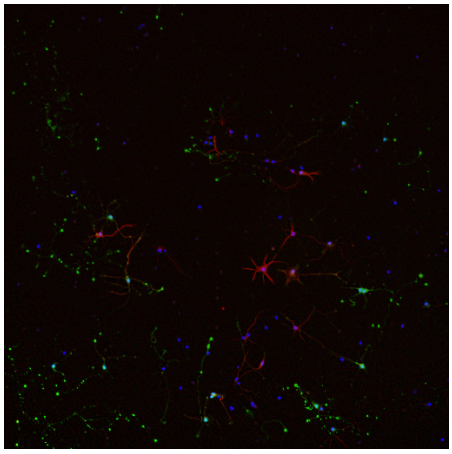
(d) DOI - Negative





(e) Quetiapine - Positive

(f) Quetiapine - Negative



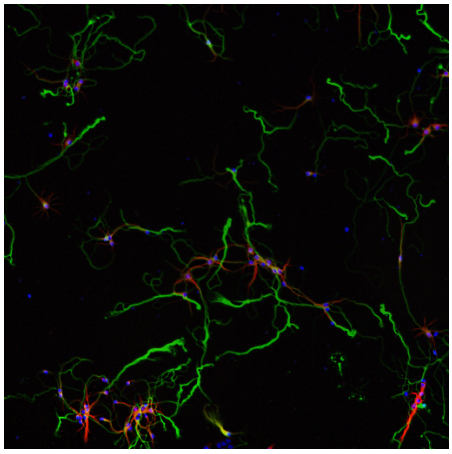
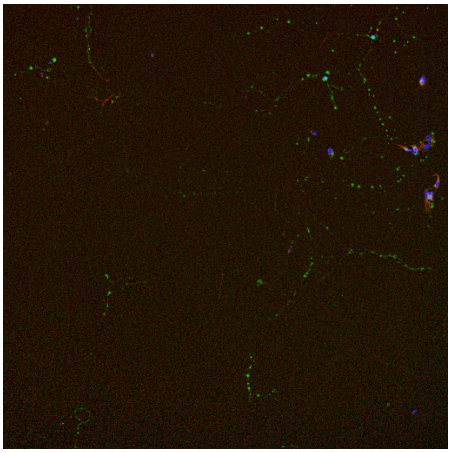
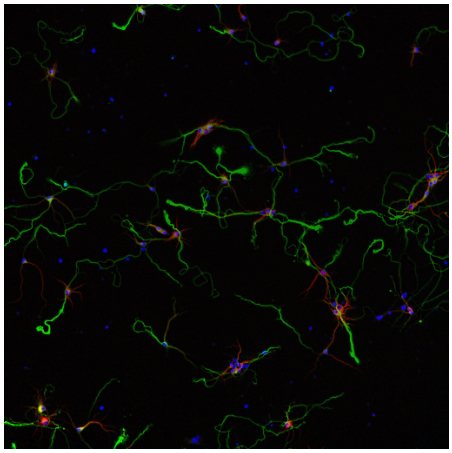


Figure 2.12: Example of images for each Neuronal Outgrowth datasets. Source: David Logan, personal communication.

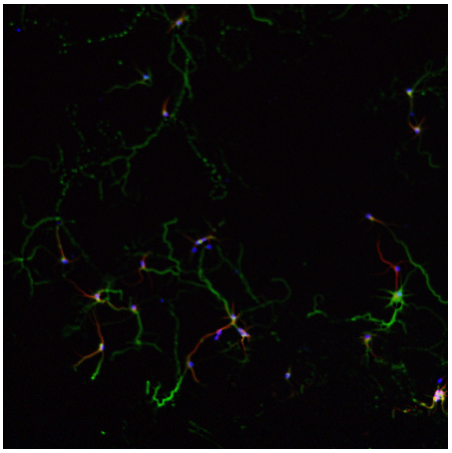
(g) Quetiapine Intermed - Positive



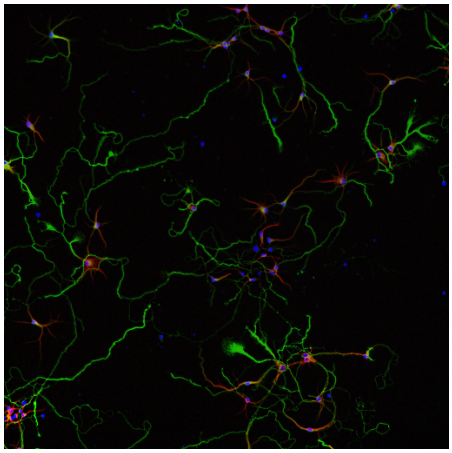
(h) Quetiapine Intermed - Negative



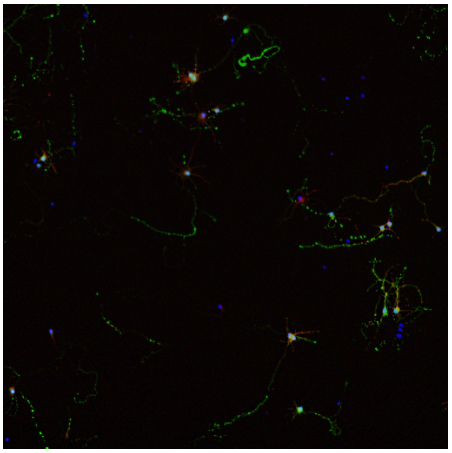
(i) Serotonin - Positive



(j) Serotonin - Negative



(k) TDZD-8 - Positive



(l) TDZD-8 - Negative

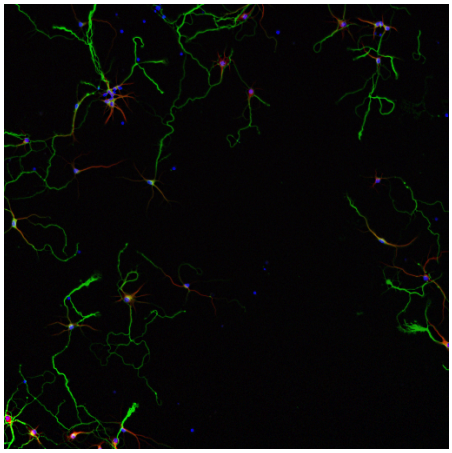


Figure 2.12: (Cont'd) Example of images for each Neuronal Outgrowth datasets. Source: David Logan, personal communication.

3 Results

We present in this chapter all results from the experiments described in the Methods section. We enforced ourselves to describe as comprehensively as possible the experimental settings and parameters we used for the sake of reproductibility. Since our results come from different sources (Python, MATLAB, R), several plot designs were used but we tried to be as constant as possible in the way we display our data. In general we used bar plots with error bars when we needed to compare different classification accuracies, and tables when we only wanted to list results from one source. In all bar plots, the mean value plus and minus one standard deviations were reported. Unless specified otherwise, when using bar plots we displayed misclassification rate (i.e. one minus the classification rate) since the variations of this metrics are more visible than the difference between classification rates. In this setting the algorithm with the lowest misclassification rate is the most efficient one. Value rage from 0 to 1, where a misclassification rate of 0 corresponds to a classification accuracy of 100%. When using tables we display classification accuracy, which is the ratio of the sum of the elements in the diagonal of the confusion matrix (i.e. properly classified items) to the total number of elements in the test set. This metric can be expressed as a fraction ranging from 0 to 1, or as a percentage ranging from 0 to 100%. For most results we also displayed the confusion matrix of one classification experiment in order to make sure the classification accuracy we observed was a good indicator of the efficiency of the algorithm. In this way we could detect situations where some classes were very well classified but others were extremely badly discriminated, yielding an good accuracy on average. Confusion matrices also helped us get a better understanding of where the difficulties of the dataset could be found: we could detect which groups of classes often got confused for one another.

3.1 WND-CHARM Analysis

The results we reproduced using our Python and the available C++ implementation of WND-CHARM are presented in Fig. 3.1 along with the results presented in [38]. We implemented our Python version of WND following the description of the algorithm in [38] and [46] and looking at crucial steps in the source code of the C++ version to make sure we captured eventual

discrepancies between theory and implementation. The accuracy for each reference dataset computed using the CHARM vector along with the Python or the C++ implementation of WND was obtained by averaging results over 100 runs of classification using the "save 25%" validation method. No error bars are displayed for the results extracted from [38] as this information was not available in the paper. We mostly aimed here to make sure we were able to reproduce paper data using the C++ implementation, and that our Python implementation was valid and also gave similar results. Fig. 3.1 displays misclassification rates, which means that a lower value is better and the optimal case is zero as it corresponds to the case where 100% of the data are properly classified.

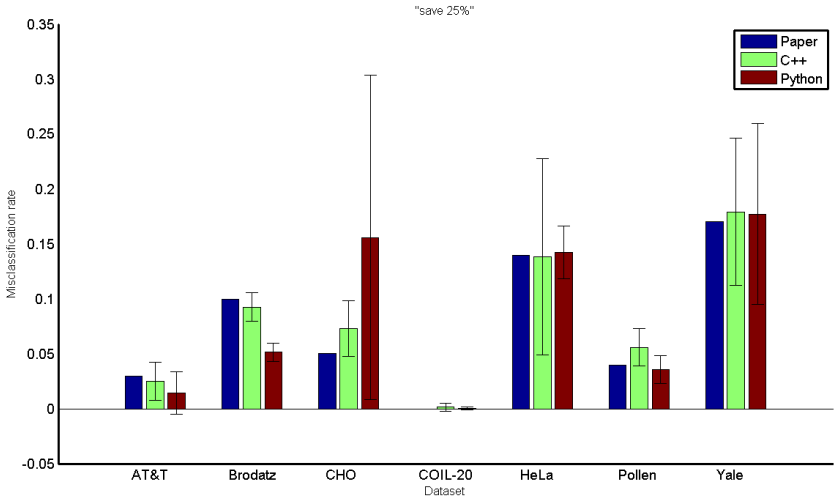


Figure 3.1: Comparison of misclassification rates across methods on the reference datasets using the "save 25%" validation method. Results range from 0 (0%) to 1 (100%).

We observe that the C++ results are comparable to the reference ones listed in [38], and that on most datasets there is no significant difference between our Python WND-CHARM and the C++ version. We see that the CHO dataset has a very large variance on our Python implementation while it is smaller when running the C++ version, and the opposite is true for the HeLa dataset. Our Python version performs significantly better on the Brodatz dataset, but it is important to keep in mind the scale of these results: the difference is in the order of 2-3% of classification accuracy.

As already explained in the Methods section we also implemented the WND classifier in MATLAB. Since the "save 25%" validation method was not available with this software and since we could be confident in the fact our Python WND was comparable to the C++ one following results on Fig. 3.1, we validated our MATLAB version of WND by comparing results with the Python version using the CHARM vector and 10-fold cross-validation. Misclassification rates averaged over 10 runs of classification using these two methods are shown on Fig. 3.2. When using 10-fold cross-validation we averaged results on a smaller number of rounds than when

using "save 25%" as the former method was in its nature more robust than the latter. This holds true as well for the other experiments of this chapter.

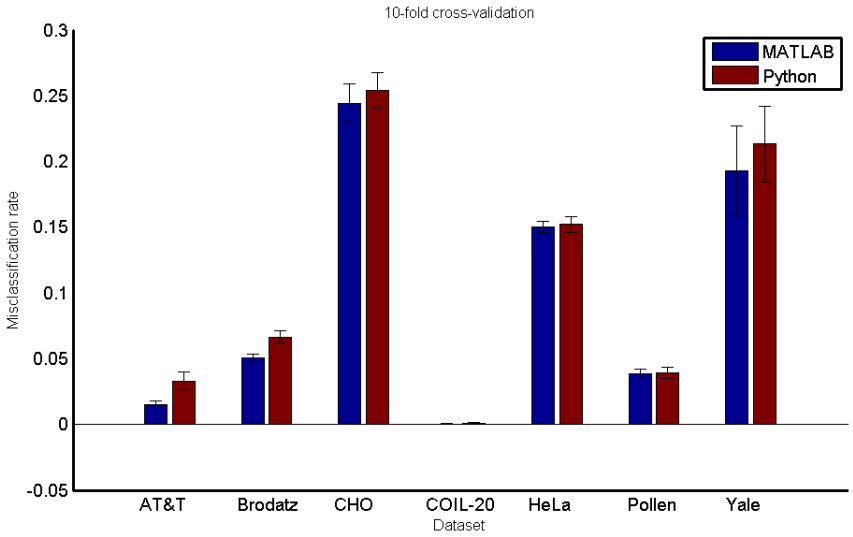


Figure 3.2: Comparison of misclassification rates across methods on the reference datasets using 10-fold cross-validation. Results range from 0 (0%) to 1 (100%).

We see that results are extremely similar on most datasets. The AT&T and Brodatz sets obtain significantly better results using the MATLAB implementation, but the difference in classification accuracy is of 1% in both cases. We therefore think that this discrepancy would disappear if results were averaged over more runs of 10-fold cross-validation. The variance of classification results is relatively conserved across the two methods. The accuracy we obtain with our re-implementations of WND-CHARM hence appear to be comparable to the reference results, using either "save 25%" or 10-fold cross-validation.

We also measured the significance of the results, as described in the Methods section. The mean and variance of the performances of 10'000 WND-classifiers using CHARM vectors extracted on randomized data and validated using either the "save 25%" method or 10-fold cross-validation are reported in Tables 3.1 and 3.2. As explained earlier, our goal was to show that the performances we obtained were significantly different than those of a random classifier. We recall that for a random classifier and an unbiased dataset we expect a mean performance of $\frac{1}{N}$ where N is the number of different classes since each element has a $\frac{1}{N}$ probability of being properly classified by chance.

We recall that a p-value of zero indicates that out of the 10'000 random classification rounds, no result was found to be equivalent or better that the actual classification results shown in Fig. 3.1 (for the "save 25%" method) and Fig. 3.2 (for the 10-fold cross-validation). We observe that random efficiencies correspond to the results we would expect for each dataset (i.e. $\frac{1}{N}$ where N is the number of classes), indicating that our datasets were well designed. We also notice

Table 3.1: Classification efficiency in percents on randomized datasets using WND-CHARM and the "save 25%" validation method.

Dataset	Mean	Standard Deviation	p-value
AT&T	2.50	3.05	0
Brodatz	0.89	0.19	0
CHO	21.25	20.31	0
COIL-20	5.07	1.36	0
HeLa	12.51	6.11	0
Pollen	14.3	7.78	0
Yale	6.69	13.82	0

Table 3.2: Classification efficiency in percents on randomized dataset using WND-CHARM and 10-fold cross-validation.

Dataset	Mean	Standard Deviation	p-value
AT&T	2.45	0.65	0
Brodatz	0.91	0.05	0
CHO	20.94	5.56	0
COIL-20	4.97	0.38	0
HeLa	12.50	1.66	0
Pollen	14.27	2.10	0
Yale	6.61	4.09	0

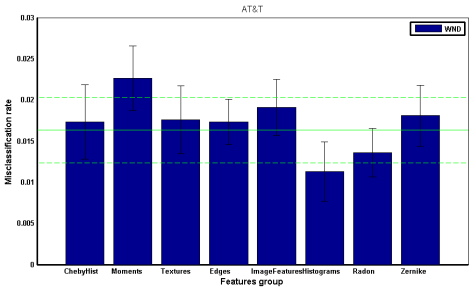
that there is no large difference between the mean results obtained with the two validation methods. As expected from the fact that we are working with random data the classification accuracy is more dependent on the nature of the dataset than on the validation method. We however observe that the "save 25%" method has a much larger variance in classification accuracy between runs than the 10-fold validation.

3.1.1 Features Extraction

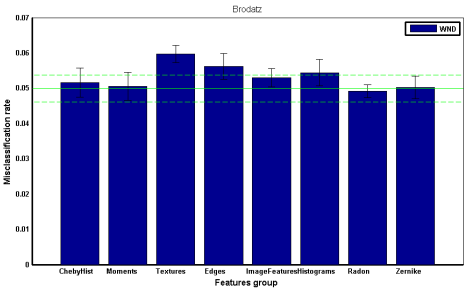
Feature Vector Content

Features Groups We split the CHARM vector computed using the C++ implementation of WND-CHARM in eight different features "groups" described in the Methods section, and we classified eight times the CHARM vector minus one of these groups using WND. The MATLAB results averaged over 10 rounds of 10-fold cross-validation are displayed in Fig. 3.3. The x-axis labels indicate which feature group was removed. For comparison reasons we plotted the mean (solid green line) and plus/minus one standard deviation (dashed green lines) of the reference accuracy obtained with MATLAB using 10-fold cross-validation, WND and the full CHARM vector along with the results on the reduced features sets.

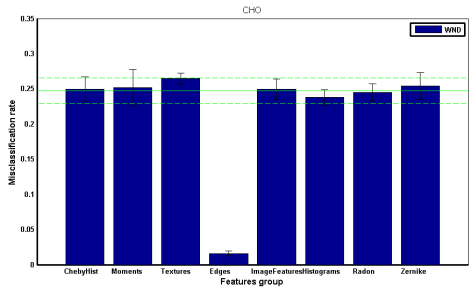
(a) AT&T



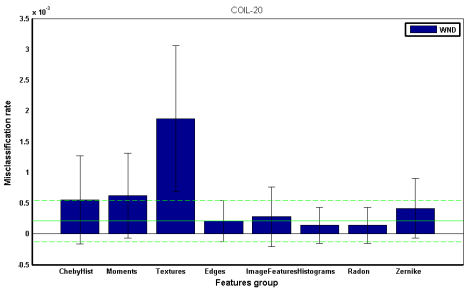
(b) Brodatz



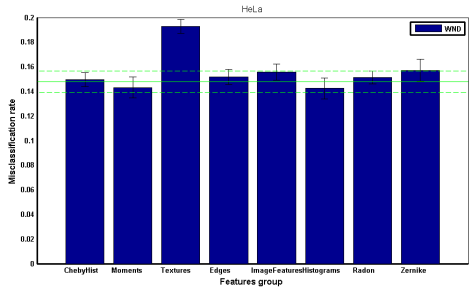
(c) CHO



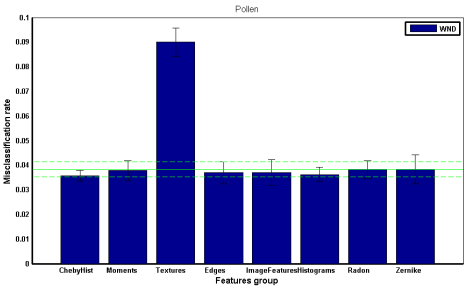
(d) COIL-20



(e) HeLa



(f) Pollen



(g) Yale

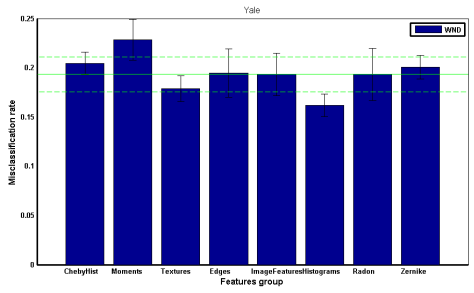


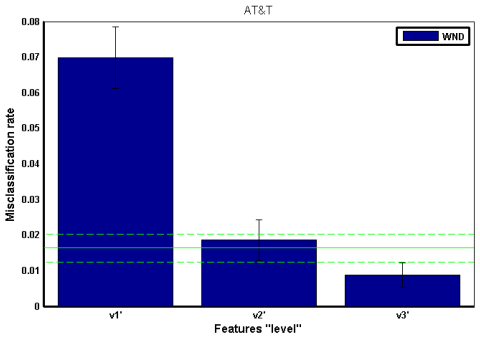
Figure 3.3: Misclassification rates per features "groups": the label under each bar corresponds to the group that has been removed from CHARM for classification. The solid (respectively dashed) line corresponds to the mean (respectively mean plus and minus one standard deviation) reference classification accuracy using the whole features set. Results range from 0 (0%) to 1 (100%).

We observe in 3.3a, 3.3b and 3.3g that no feature group seem to be significantly more important than the others as the individual removal of each of these groups doesn't significantly alter classification efficiency with respect to the reference value. Each of the different groups seem to contribute somehow to the overall results. In 3.3d, 3.3e and 3.3f the removal of the textures group affects classification result. This result is not surprising as the HeLa and Pollen datasets are composed of biological images where classes mostly differ based on textural properties. The effect of the textures features in the COIL-20 dataset is more surprising. Since this set is composed of images of objects one would expect edges and basic image features to play an important role along with textures. This doesn't seem to be the case as the elimination of either edges or basic image features doesn't affect classification accuracy. An interesting situation can be observed in 3.3c where the removal of edges features drastically improves classification results. One explanation for this behavior could be that edges features manage to get good weights during the weighting procedure of WND and therefore end up in the top 15% while they actually do not really help in discriminating between classes. By removing these features some slots of the 15% best are liberated for the features ranked right after edges, which appear to have a better discrimination power. The effect we observe could hence be explained by a failure of the weighting scheme that should yield high values for informative features only. In general we notice that there are no features groups that seem to be sufficient to explain the good results on all the datasets. Different groups appear to be important depending on the nature of the images as different morphological aspects allow for class separation. It is also interesting to note that the removal of certain features groups appear to improve classification accuracy in some cases as described for CHO. However here again there is no global tendency, and we cannot therefore flag a particular group as "useless".

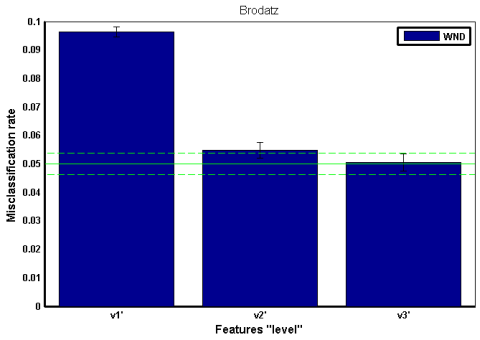
Features Levels After splitting the vector into groups we did the same experiment for features "levels". Our three features levels were designed to be non-overlapping, i.e. elements were unique to each group. Following the same settings as for the preceeding "per groups" experiment, we classified the CHARM vector minus each group using our MATLAB implementation of WND. The results shown in Fig. 3.4 were validated ten times using 10-fold cross-validation, and then averaged. We plotted again the reference mean and standard deviation in green as in Fig. 3.3.

In most of the datasets, 3.4a, 3.4b, 3.4d, 3.4e, and 3.4f, it can be seen that the first features level (i.e. the features extracted from the raw image) seem to contribute a lot to the good classification result we obtain with the CHARM vector. Indeed if this level if removed we observe a notable increase in the misclassification rate. This however doesn't imply that the two other levels are useless: we can see that the mean results when removing the second or the third are not completely equivalent. In 3.4d for instance one notice that the removal of the second level also impacts classification efficiency, though to a lesser extent. The Yale dataset shown in 3.4g seems to rely more on the third level features, even though this trend is not striking. In any case it appears that the first level features do not play here such a major role as in the previously mentioned cases.

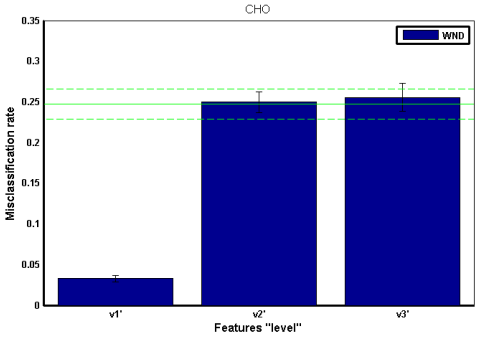
(a) AT&T



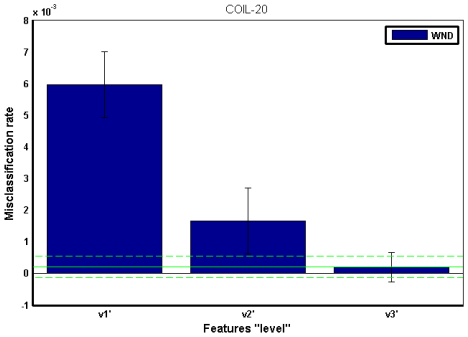
(b) Brodatz



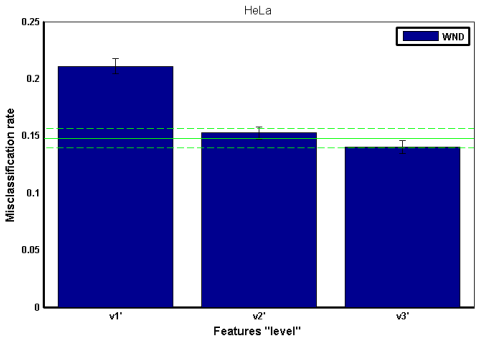
(c) CHO



(d) COIL-20



(e) HeLa



(f) Pollen

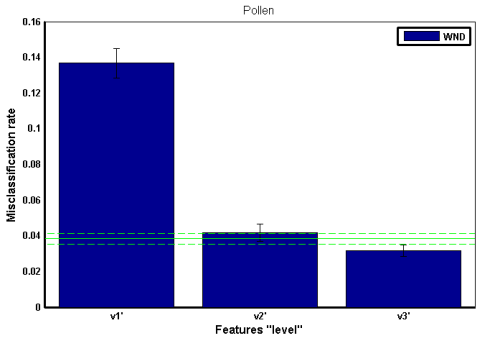


Figure 3.4: Misclassification rates per non-overlapping features "levels": the label under each bar corresponds to the level that has been removed from CHARM for classification. The solid (respectively dashed) line corresponds to the mean (respectively mean plus and minus one standard deviation) reference classification accuracy using the whole features set. Results range from 0 (0%) to 1 (100%).

(g) Yale

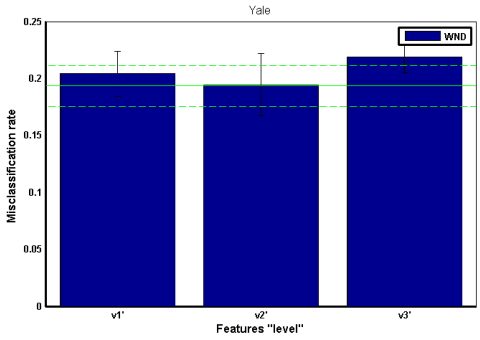
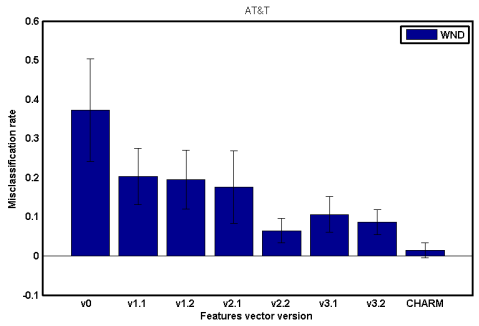


Figure 3.4: (Cont'd) Misclassification rates per non-overlapping features "levels": the label under each bar corresponds to the level that has been removed from CHARM for classification. The solid (respectively dashed) line corresponds to the mean (respectively mean plus and minus one standard deviation) reference classification accuracy using the whole features set. Results range from 0 (0%) to 1 (100%).

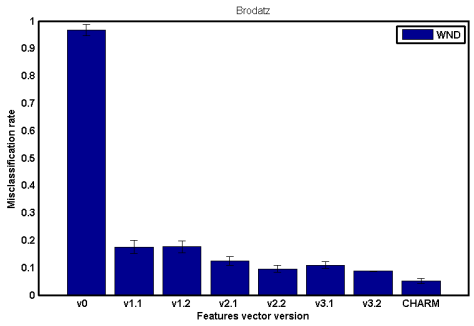
The exact opposite situation can be found in the CHO dataset in 3.4c: the removal of lower level features seem to benefit a lot classification efficiency. This result reminds us of the observation made in 3.3c and could be imputed to the same causes. Similar cases of features that apparently "penalize" classification can also be found in 3.4f and 3.4a although the trend is much more sublte and the difference is not that significant. We draw from our analysis of features levels the same kind of conclusions than for the analysis of features groups: even though the first level appears to play an important role in most datasets we cannot clearly discard the other features level as they seem to influence classification results as well, though to a lesser extent. Moreover in this case as in the previous "per groups" experiment a multi-groups and multi-level vector like CHARM seem to be an efficient solution to be general enough to perform well on all reference data.

Features Vectors Versions Assuming the power of the CHARM vector comes from its construction based on features groups and features levels as concluded from the observation of Fig. 3.3 and 3.4, we implemented several vectors containing not exactly the same elements as CHARM but built on the same scheme. These different versions were classified using WND in Python and validated with the "save 25%" method. Fig. 3.5 shows mean results over 100 classification runs obtained for each vector version on each reference dataset.

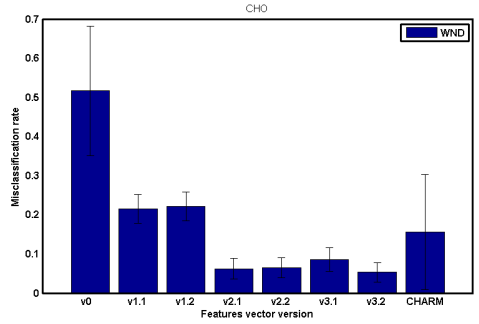
(a) AT&T



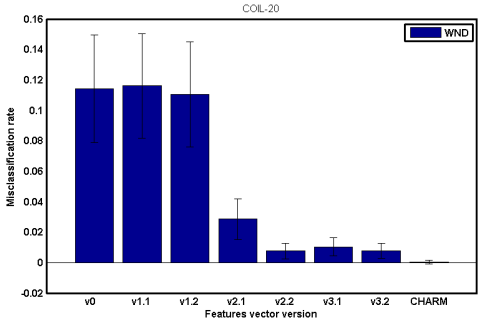
(b) Brodatz



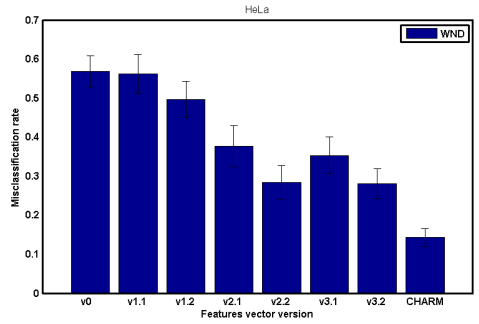
(c) CHO



(d) COIL-20



(e) HeLa



(f) Pollen

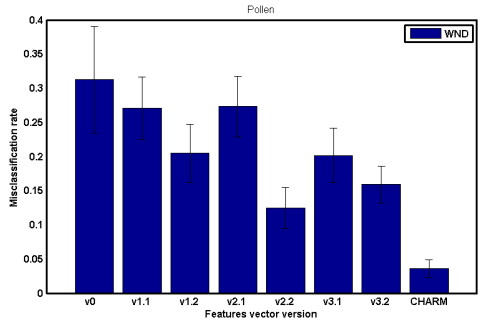


Figure 3.5: Misclassification results using different features vector version, WND classifier and the "save 25%" validation method. Results range from 0 (0%) to 1 (100%).

(g) Yale

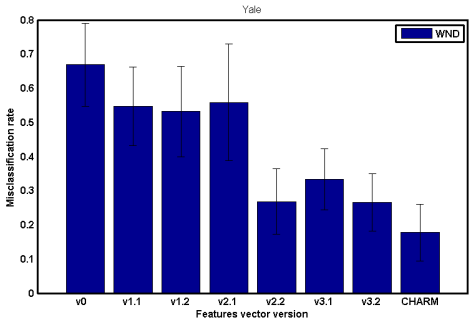


Figure 3.5: Misclassification results using different features vector version, WND classifier and the "save 25%" validation method. Results range from 0 (0%) to 1 (100%).

A nice trend of improvement can be observed as the features vectors versions we designed gain in complexity. On the v2 and v3 series, the larger vectors containing a bigger subset of features (v2.2 and v3.2) outperform the simpler versions on most of the datasets. In 3.5a, 3.5b, 3.5d, 3.5e, and 3.5g we observe that v2.2 and v3.2 get similar results. v2.2 even beats the more complex version in 3.5f. A hardly better mean result for v3.2 can be observed in 3.5c, but the difference with v2.2 is extremely faint. In general we observe that the results obtained using v3.2 are a little more constant as the variance is slightly smaller than the one obtained with v2.2. On most of the datasets our vectors do not achieve as good results as CHARM, but there is a net tendency towards approaching WND-CHARM's reference accuracy. As our v2.2 vector composed of transforms only seems to perform as well as the version containing all features levels (v3.2), we could drop the compound transforms. However considering the fact we do not really understand them, we think removing them directly would be delicate as we cannot predict how it would impact on the algorithm's ability to generalize on many different kind of image data.

Feature Vector Interpretation

We ran the three experiments described in the Methods section in order to investigate the nature of the hard-to-interpret higher level features. All results were gathered using RStudio [43] along with R 2.15 [41].

Our first test aimed at comparing the classification results obtained using RBF-SVM on v1' (first level features only) and Linear-SVM on the full CHARM vector. We used the `train` function from the `caret` R package [29] with `svmRadial` on v1' and `svmLinear` on CHARM. The features were centered and scaled to the interval $[0,1]$ using the `preProcess` function from R. We used the 10-fold cross-validation also implemented in the `train` function to validate our results. As usual, our results were averaged over 10 runs of 10-fold cross-validation. The SVM implementations `svmLinear` and `svmRadial` are extremely time consuming and

rapidly require large amount of resources as the number of different classes increases. For this reason we did not use here the AT&T and Brodatz as the execution time on these sets was not acceptable. Results were plotted using the `ggplot` R library and displayed in Fig. 3.6.

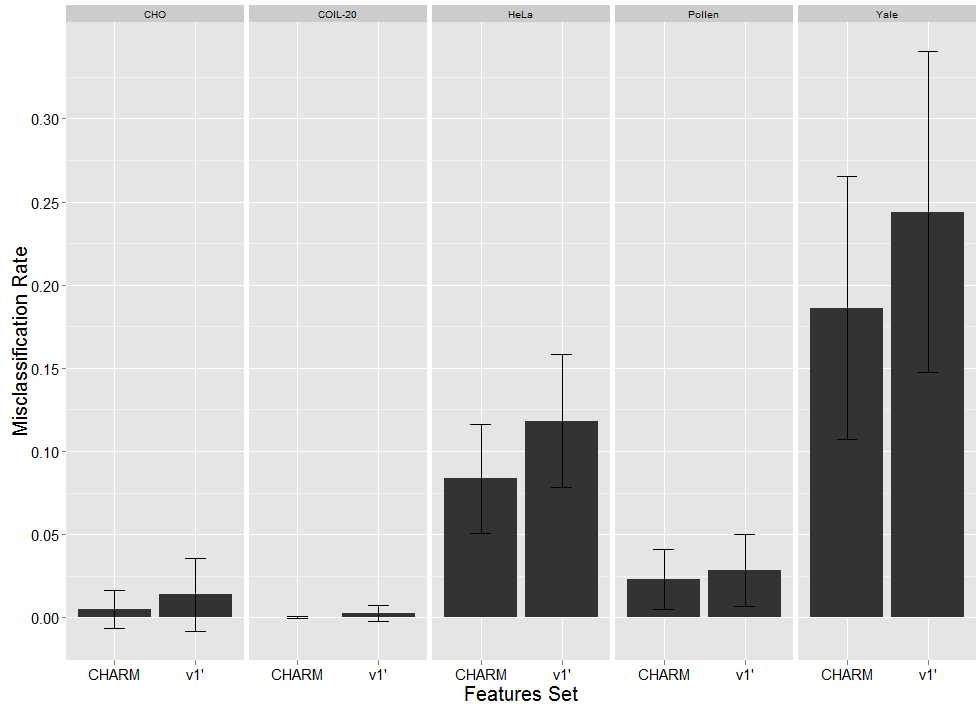


Figure 3.6: Misclassification rates using RBF-SVM on v1' versus Linear-SVM on CHARM. Results range from 0 (0%) to 1 (100%).

Apparently Linear SVM is a very powerful classifier when used with CHARM. We observe that the use of RBF-SVM on v1' does not compete with Linear SVM on CHARM. In all datasets the misclassification rate on v1' is larger than on CHARM. It therefore seems that the additional dimensions offered by the Radial Basis Functions kernel is not sufficient to imitate the effect of the higher level features present in the CHARM vector.

For the second experiment we used RBF-SVM on v1' and RBF-SVM on CHARM. The same R functions were used as before, namely the `train` function from `caret` with parameter `svmRadial` for the classifier and 10 runs of 10-fold cross-validation. Results are presented following the usual format in Fig. 3.7.

It is interesting to note that RBF-SVM on CHARM seems to penalize classification: the misclassification rate on the full vector is increased for all datasets except COIL-20 in Fig. 3.7 as compared to Fig. 3.6. A possible interpretation could be the following. As RBF-SVM is "just another classifier" with the advantage of being able to increase the dimensionality of the feature space, the low level vector v1' benefits a lot from this "lifting" in higher dimension space while the more complex vector only slightly benefits from this property of RBF-SVM since it already has many dimensions. In any case this situation brings misclassification rates

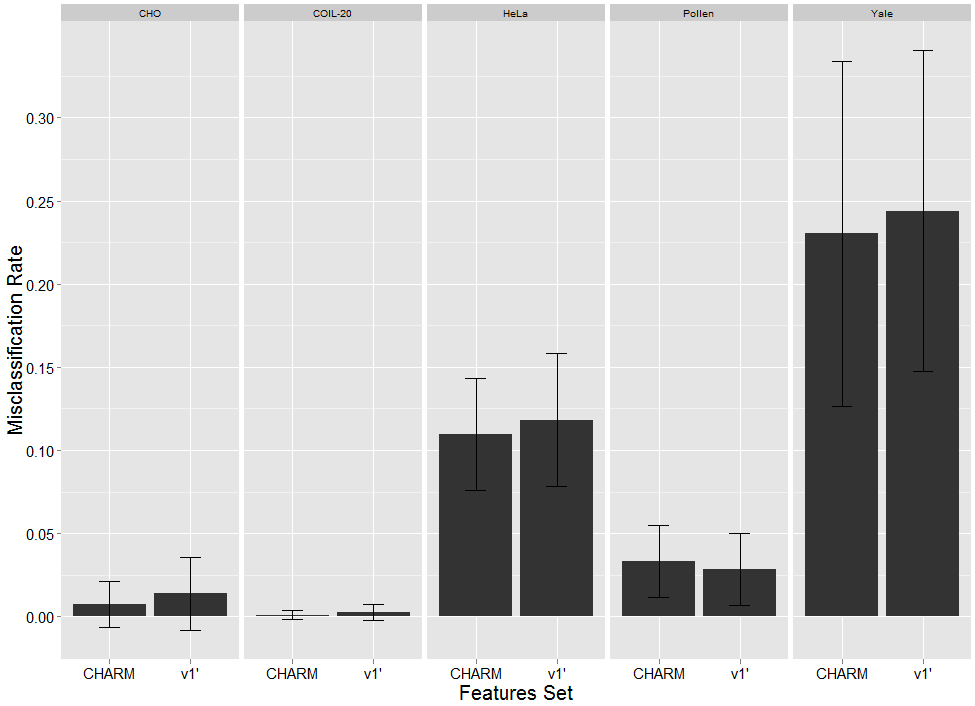


Figure 3.7: Misclassification rates using RBF-SVM on v1' versus RBF-SVM on CHARM. Results range from 0 (0%) to 1 (100%).

obtained with both features vectors to closer values. Our expectation that Linear SVM and RBF-SVM on CHARM would yield approximately equivalent results is contradicted, but the hypothesis that RBF-SVM on v1' and CHARM should give comparable results seems to be verified.

For our last analysis regarding higher level features we compared performances of Linear-SVM and Random Fourier Features on v1' versus Linear-SVM on CHARM. The same `svmLinear` as in the first experiment was used, as well as the same validation method. For the Random Fourier Features we used an ad-hoc `gen_rand_feats` R function implemented by Shantanu Singh at the Imaging Platform for this study. The R source code of the `gen_rand_feats` is extremely simple:

```
gen_rand_feats <- function (X,d=10) {
  X <- as.matrix(t(X))
  D <- dim(X)[1]
  W <- sqrt(2)*matrix(runif(d*D), d)
  V <-  W %*% X
  G <- t(rbind(cos(V),sin(V)))
}
```

Following [42], $d * D$ random samples are simply drawn from a uniform distribution using the

`runif` R method where d is the desired number of random features and D is the number of samples (lines) of X. Following the description of random features given in the Materials and Methods section, a new features space `G` with the addition of $2 * d$ random features is created. The factor 2 comes from the fact that the cosine and sine of the $W * X$ product are returned. `train` with `svmLinear` was then used on the new feature vector as in previous cases. Results using d=500 on the v1' vector are presented in Fig. 3.8.

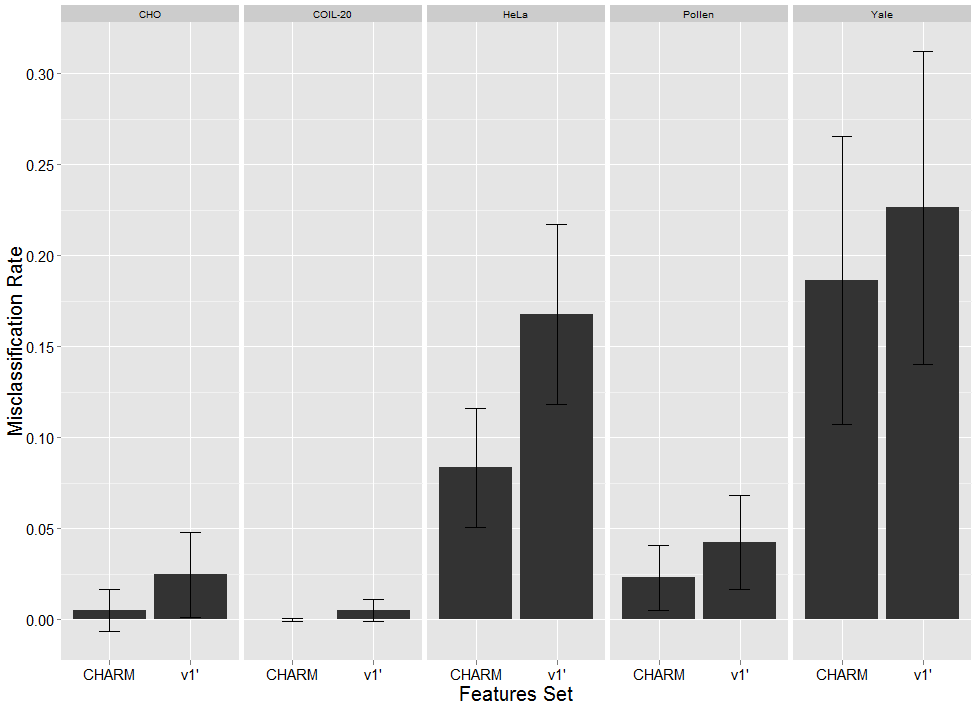


Figure 3.8: Misclassification rates using Random Fourier Features and Linear-SVM on v1' versus Linear-SVM on CHARM. Results range from 0 (0%) to 1 (100%).

With its 1000 random features, v1' contains a total of 1432 features, bringing it to a dimensionality of the same order than that of CHARM and its 1025 elements. We can however notice that this is still not sufficient: the misclassification rate obtained on v1' with the random features is still larger than what is observed on CHARM. Comparing v1' results on Fig. 3.7 and Fig. 3.8, we observe that results on v1' using RBF-SVM and Linear SVM with random features are similar. We can hence verify that the random features are performing the same kind of trick than the RBF kernel on the low level features vector. As it was the case in Fig. 3.6, our hypothesis stating that higher level features are no better than random features and are merely adding dimensions cannot be verified as results go against it. Apparently there is something more in the CHARM vector that we cannot mimic by generating random features to increase dimensionality of the features space.

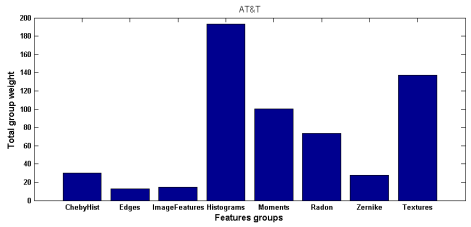
The hypothesis we proposed to explain the nature of the features computed on compound transforms couldn't be proved by these three experiments. While we still cannot easily un-

derstand how these complicated features relate to the original image, we at least showed that they are really capturing some aspects in the image. The higher level features might be hard to interpret, but we now know that they do more than solely adding dimensions to the features space. We also found out that Linear SVM apparently performs better than RBF-SVM when teamed up with CHARM,. This is an interesting result as it indicates us that it is possible to separate the classes in our reference datasets using a linear classifier. If class distributions were not linearly separable, we would probably have obtained better results when using RBF-SVM. This analysis eventually did not really helped us understanding the meaning of the compound transforms features, but allowed us appreciating the fact they indeed carry information from the image.

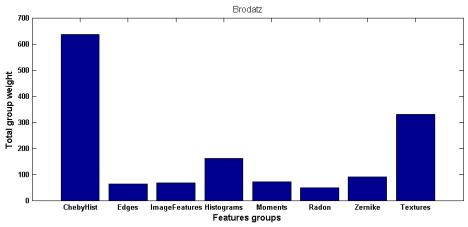
"Top 15%" Features

Dimension reduction of the features space is performed in WND-CHARM using weighting and subsequent thresholding: the 15% features corresponding to the highest weights are kept for classification and the others are thrown away. In order to gain insight into this step of the algorithm we analyzed the top 15% features subset in two ways: by looking at its composition and by investigating its stability over classification runs. We first extracted top features and corresponding weights from the HTML report outputted by the C++ implementation of WND-CHARM using ad-hoc Python scripts. Subset composition was then analyzed by gathering top features into groups according to the same scheme we used for our "per groups" classification experiments. In each groups features weights were summed to yield an overall "group weight". Results were then exported to MATLAB and plotted for each reference dataset as shown in Fig. 3.9. We aimed with this result to reach the same kind of conclusions as when we did "per groups" classification, namely that all feature groups appear to play a role among the different datasets.

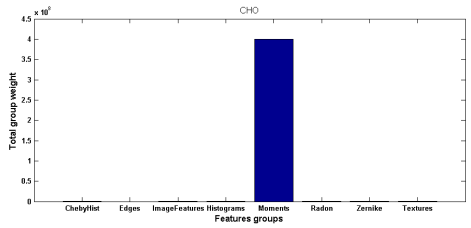
(a) AT&T



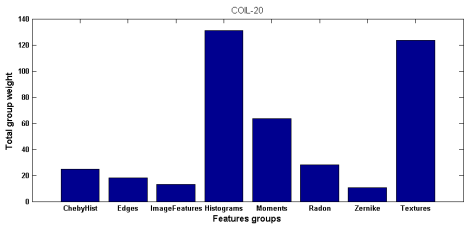
(b) Brodatz



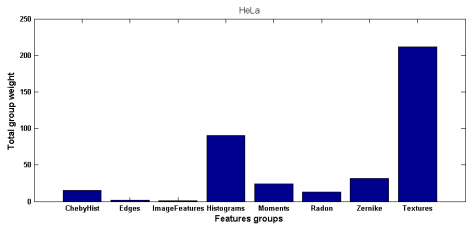
(c) CHO



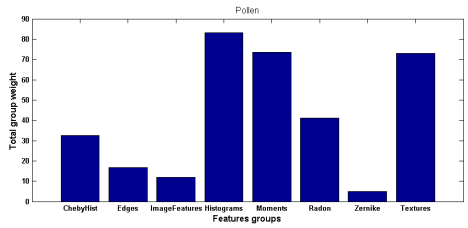
(d) COIL-20



(e) HeLa



(f) Pollen



(g) Yale

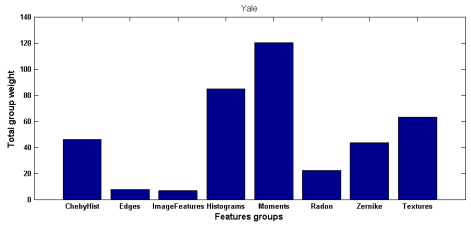


Figure 3.9: Features group composition of the top 15% subset selected after thresholding. High weights indicates feature groups identified as having a large discriminative power by WND-CHARM's weighting scheme.

Values indicated in the y-axis of Fig. 3.9 correspond to the sum of weights of all features inside each features groups. These values were not scaled, and we therefore look at the relative weights of features groups rather than at their actual values. It is interesting to look at these results while keeping in mind the ones we obtained in Fig. 3.3: we observe three different situations. In some cases the groups that appear to have highest weights seem to penalize classification, as we can see for ChebyHist in 3.9a (3.3a) and Histograms in 3.9g (3.3g). In these

two cases classification accuracy is improved when the groups having the strongest weight sums are removed. This could indicate that the weighting scheme based on Fischer scores used in WND-CHARM creates "false positives": some features do not actually have a good discrimination power but still get large weight values. As they are selected after thresholding they prevent "better" features to be used. When they are removed classification efficiency is thus improved. The second situation we observe is the expected one: feature groups accumulating the largest weights correspond to important elements for class discrimination. Their removal impacts classification results in a bad way as we can see for Textures in 3.9b (3.3b), 3.9e (3.3e) and 3.9f (3.3f), Moments in 3.9a (3.3a) and 3.9g (3.3g), and Moments and Textures in 3.9d (3.3d). These indicate cases where large weights properly correspond to strongly discriminating features. At last we can also notice some cases where features groups exhibiting the largest weight sums do not seem to be that important for classification as their absence do not affect the results shown in Fig. 3.3. ChebyHist in 3.9b (3.3b), Moments in 3.9c (3.3c), Histograms in 3.9d (3.3d) and 3.9e (3.3e), and Histograms and Moments in 3.9f (3.3f) are examples of such situation. A possible explanation could be that these features groups are highly represented in the 15% subset, but do not strongly contribute to classification as a few other features with much stronger weights are present. When adding up weights these features groups obtain a large sum only because of the fact they are numerous. Aside from these observations, the results on the CHO dataset displayed in 3.9c are extremely puzzling. The Moments features group obtains a weights score that strongly dominates all other groups, but it appears in 3.3c that the absence of this group does not influence classification results. Even stranger: we observe in 3.3c that the absence of the Edges features group improves classification accuracy even though it gets a near-zero weights sum in 3.9c. We do not have an explanation for this particular case, but since it is an isolated situation we do not consider it as a big source of concern. These experiments highlight some limitations of the features selection scheme used in WND-CHARM. There seems to be in some cases a discrepancy between the features selected by the weighting step and the crucial elements for classification.

In order to assess the stability of the feature selection algorithm we measured two metrics: the Tanimoto distance and the Pearson's correlation coefficient described in the Methods section. We wrote a Python script gathering these measurements from features names and weights we extracted from WND-CHARM's HTML report. The output is a tab separated matrix in a text file. As the Tanimoto distance and Pearson's correlation coefficient are measured for every pairs of subsets and since our data covered 100 runs of classification, the results are displayed as 100x100 pixels matrices. These matrices are shown as black and white images in 3.10 and 3.11 where black corresponds to the lowest value and white to the largest.

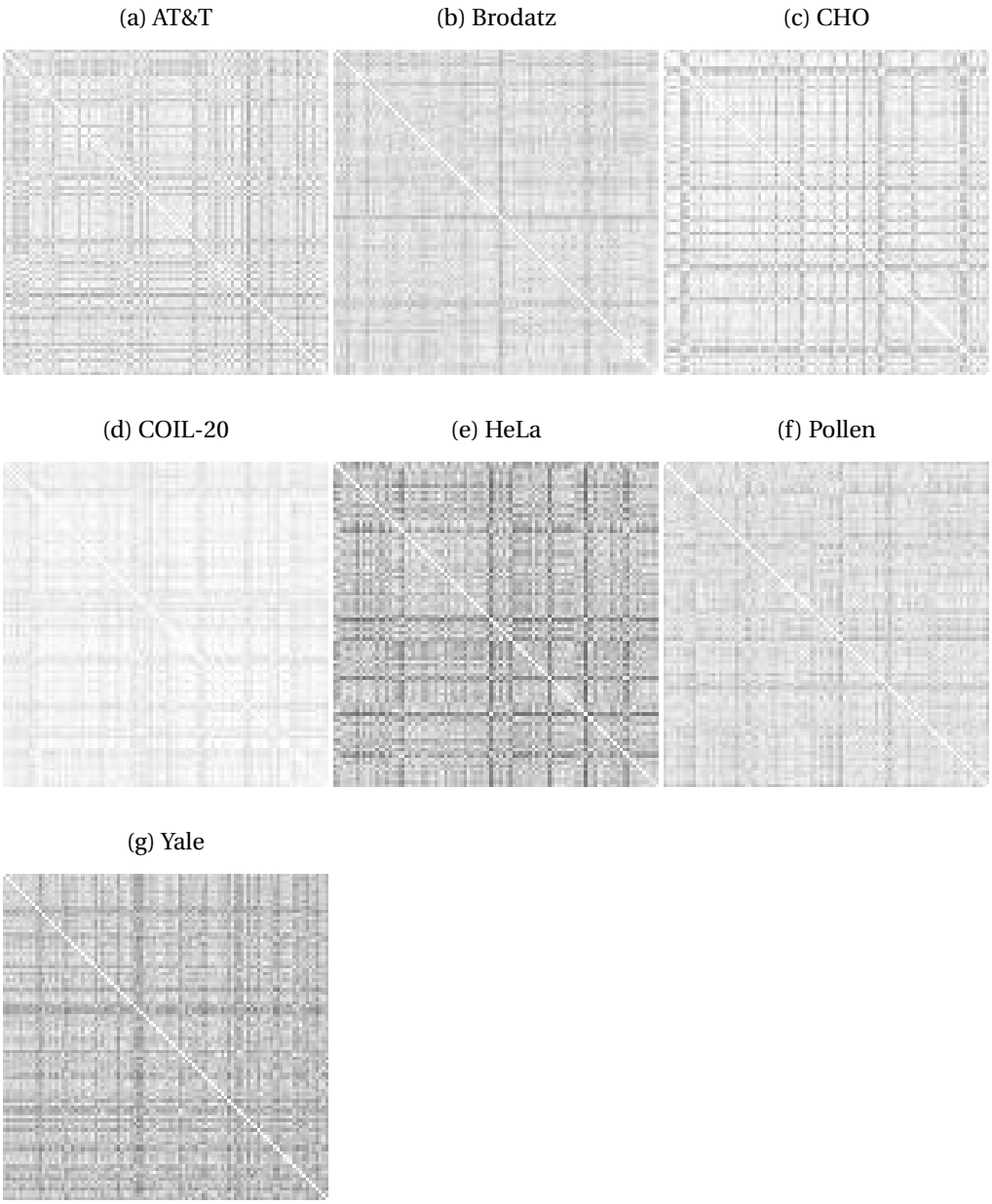
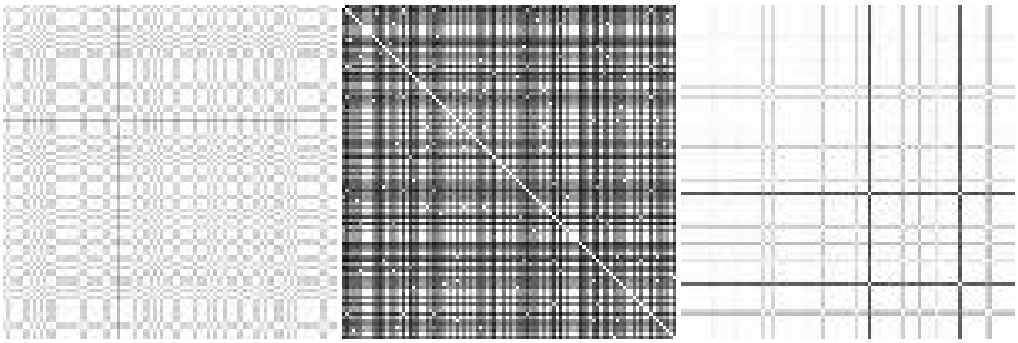


Figure 3.10: Tanimoto distance measuring the similarity of top features subsets content across 100 classification runs on the reference datasets.

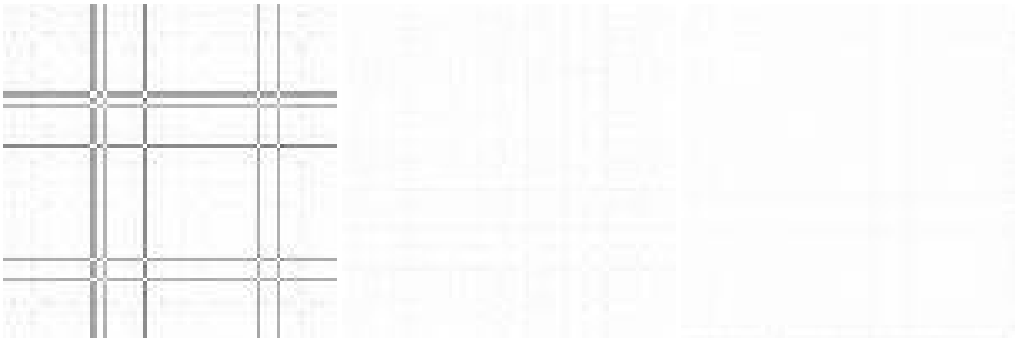
In the ideal case of a perfectly stable feature selection scheme each matrix will be all white as a value of one indicates perfectly similar subsets for the Tanimoto distance. We observe that for 3.10a, 3.10b, 3.10c, 3.10d, and 3.10f the overall tendency is grey, meaning that the subset of 15% best features used is rather unstable. The Tanimoto distance matrix of the COIL-20 dataset 3.10d is particularly white, which could probably be explained by the fact this dataset is fairly easy to classify and the features required for class discrimination are therefore "obvious".

Hela in 3.10e and Yale in 3.10g exhibit the most variations as seen by the larger amount of dark areas. This could be explained by the fact that they are difficult datasets, hence the classifier is maybe always "hesitating" between which features to consider. This could also indicate redundancy in the information given by the different features resulting in different possible "top subsets" of equal strength.

(a) AT&T (b) Brodatz (c) CHO



(d) COIL-20 (e) HeLa (f) Pollen



(g) Yale

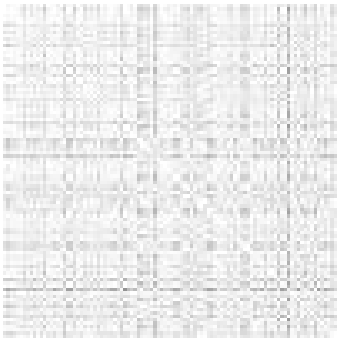


Figure 3.11: Pearson's correlation coefficient measuring the similarity between features weights across 100 classification runs on the reference datasets.

We recall that the Pearson's correlation coefficient measures the similarity of weights across classification runs. A value of 1 is synonym of perfect correlation, while -1 corresponds to perfect anticorrelation. When no correlation between weights is detected, Pearson's coefficient is equal to 0. In Fig. 3.11 black therefore corresponds to -1, grey to zero and white to 1. We observe several trends in these matrices. First 3.11e and 3.11f display an extreme degree of correlation between classification runs. The Pollen dataset appeared to have quite dissimilar subsets between rounds in 3.10f, and 3.11f therefore indicates us that even though the nature of the top features is changing the weights values in the subset seem to be conserved. The situation is very interesting for the HeLa dataset: we saw in 3.10e that the subset composition appeared to vary a lot between runs, but we observe in 3.11e that the weights are highly correlated. This observation gives more credit to our supposition that this dataset contains redundant features, i.e. descriptors with the same discriminative power. Depending on the nature of the training set, one feature set or another is used but the resulting weights appear to be similar as several features are equally powerful. The Brodatz dataset 3.11b presents a strong degree of anticorrelation between rounds, while we saw in 3.10b that the content of the subset was reasonably varying. This could mean that depending on the nature of the training set the features appearing on the top 15% vary slightly but have more or less discriminative power and hence get different weight values. The same conclusions could be drawn from 3.11a that exhibits absence of correlation between many rounds. In the case of 3.11g we already saw in 3.10g that the content of the subset was very unstable, hence a notable absence of correlation could be expected. An interesting situation can be found in 3.11c and 3.11d where a few runs appear to be highly anticorrelated or uncorrelated to the other ones, yielding isolated black and grey lines. As we observed that these two datasets have mildly stable subset contents we can imagine that the same situation as described for 3.11b is happening: the nature of the training set affects more the absolute (i.e. the weight value) than the relative (i.e. the fact feature A is more powerful than feature B) discriminative power of the features thereby changing weight values while keeping subset content more or less stable over runs.

As a general conclusion it seems that we should rely more on the top features content for our analysis than on the weight values as the latter displays greater variations over classification rounds than the former. While weight values vary differently between datasets as it can be seen on Fig. 3.11, the amount of variation in the identity of the features linked to highest weights seems to affect more uniformly the datasets as shown on Fig. 3.10. We could hence conclude that while a dimension reduction step is definitely appropriate, the weighting scheme proposed in WND-CHARM might not be the best solution.

3.1.2 Classification

We used MATLAB in order to try two standard classifiers on the CHARM vector and compare their efficiency with WND. We used the 10-fold cross-validation provided by the `crossval` method available in MATLAB and the `classify` and `classifyknn` functions for LDA and k-NN respectively. In order to perform PCA we called the MATLAB function `princomp`, and

we selected the desired amount of PC by simply computing the ratio of the cumulative sums of eigenvalues of the covariance matrix to the total sum. The `classifyWND` function we implemented performed WND classfication and was written in the same format as the two other "classify" functions for convenience's sake.

k-NN

First of all we performed a parameter exploration in order to find the optimal value of k for k-NN. We classified the CHARM vector using k-NN with 1 to 10 neighbors and plotted the results for every reference dataset in Fig. 3.12.

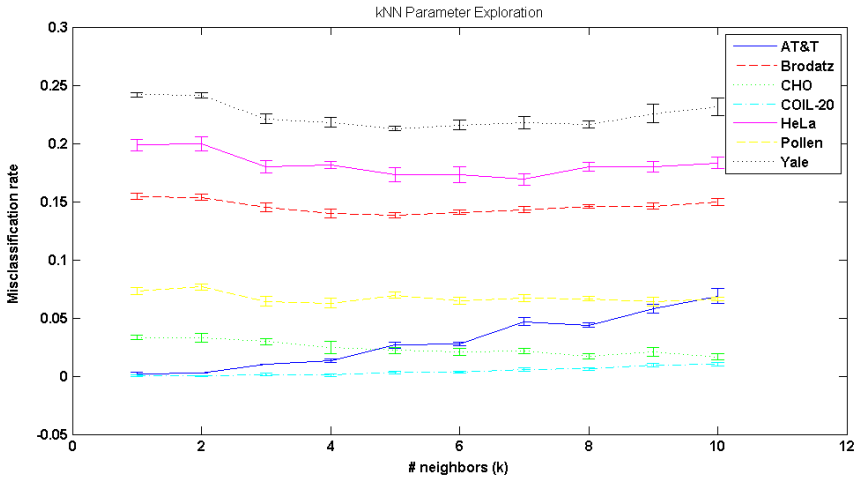


Figure 3.12: k-NN parameter exploration using MATLAB.

We observe that for every dataset except AT&T the influence of parameter k doesn't appear to be striking. With these results we obtained an optimal k value for each dataset, listed in Table 3.3

Table 3.3: Optimal k values for kNN.

Dataset	k
AT&T	1
Brodatz	5
CHO	9
COIL-20	1
HeLa	7
Pollen	4
Yale	5

We have to keep in mind that situations like AT&T and COIL-20 where only one neighbor is

considered are highly sensitive to noise in the data and therefore highly dependent on the nature of the training set.

PCA-LDA

When doing PCA as a pre-processing step for LDA the number of principal components we need depends on the amount of variance desired to be conserved in the PCA-transformed space. We hence ran an experiment similar to the k-NN parameter exploration discussed previously, this time with different amount of variance. We performed PCA-LDA on CHARM while keeping 95 to 99% (with a 1% increment) of the original variation. Results are shown in Fig. 3.13. The Yale dataset does not have result for %var=0.99. As explained in the Methods section LDA tends to fail when the feature space dimension is too large, hence requiring a dimension-reduction step before classification. As %var approaches 1 the number of principal component to be kept approaches the number of features in the original space and amounts to no dimension reduction. LDA hence crashes on the Yale dataset when %var=0.99 as the covariance matrix becomes singular.

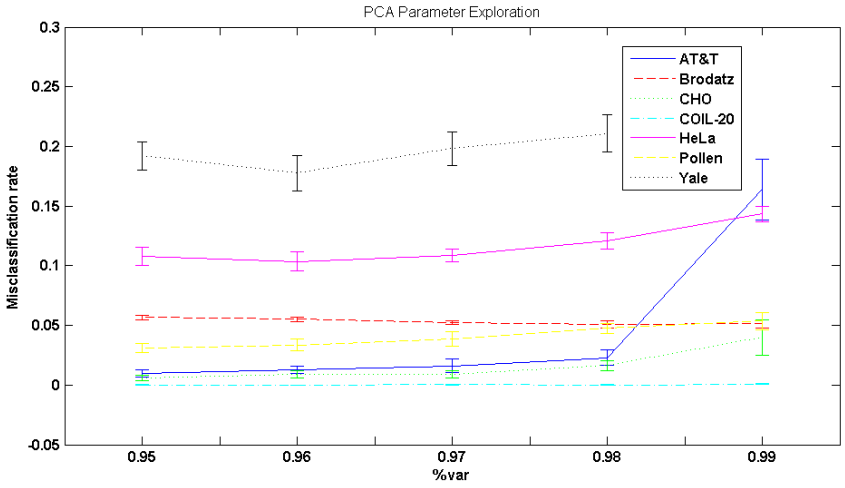


Figure 3.13: PCA parameter exploration using MATLAB.

We again see that except in the case of AT&T the influence of %var is light. Most variations are not significant, and it is therefore difficult to decide which value to choose. We listed the %var values that yielded the best classification results in Table 3.4.

We chose the value quite arbitrarily for COIL-20 as it can be seen in Fig. 3.13 that the difference of results is indistinguishable. 98% is a usual choice for PCA parameter, so we decided to go for this value. Most datasets seem to require only a small amount of the variance in the original data to be able to give good results.

Table 3.4: Optimal %var for PCA.

Dataset	%var [%]
AT&T	95
Brodatz	99
CHO	95
COIL-20	98
HeLa	96
Pollen	95
Yale	96

3.1.3 Comparison with WND

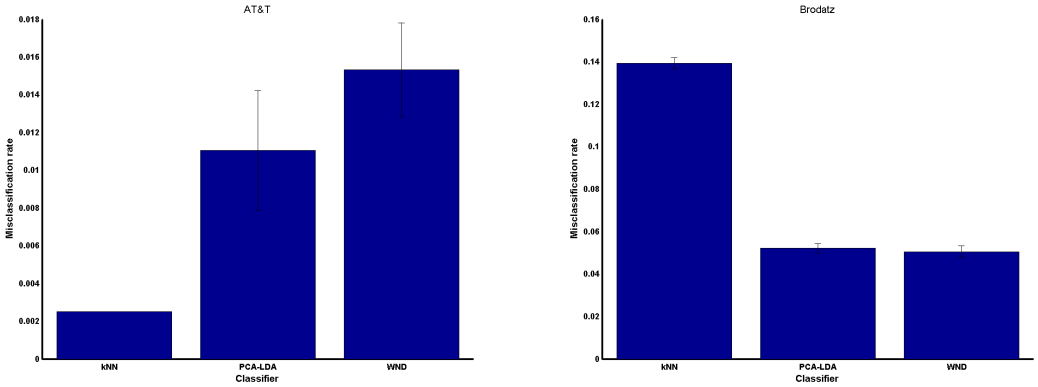
Taking into account the observations of k-NN and PCA parameters exploration we ran our classification experiments in MATLAB using WND, k-NN and PCA-LDA. Results presented in Fig. 3.14 were averaged over 10 classification runs. Values of k for k-NN were chosen for each dataset according to results presented in Table 3.3, and PC were selected in order to keep the optimal amount of the variance in the original data as listed in Table 3.4. In all cases features were centered and scaled before classifying.

Modifying the classification method from WND to k-NN and PCA-LDA yielded interesting results: in most cases we managed to obtain lower misclassification rates with either algorithms. We observe in 3.14a that k-NN wins over all other methods even though the difference is minimal (about one percent). However in 3.14b, 3.14e, 3.14f, 3.14g, and in 3.14c (although the effect is extremely small in this last case) the common efforts of PCA and LDA achieve better performance than k-NN, and in more than half of the cases better than WND. Results on the COIL-20 dataset (3.14d) are so good in any cases that it is hardly possible to make any conclusion, but as a general trend PCA-LDA seems to be a better options than k-NN in terms of results. Moreover caution should be taken with k-NN results especially in cases where only one neighbor was considered and no variance was observed (3.14a): good results could come only from the fact that the "important" neighbor was present on every run in the training set by chance.

In the light of these results we can conclude that LDA combined with PCA seems to be a promising approach at the results level. It is also interesting from a theoretical standpoint as this standard method is more widely accepted and less empirical than WND.

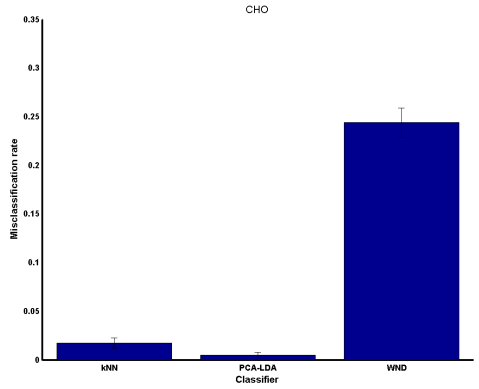
(a) AT&T

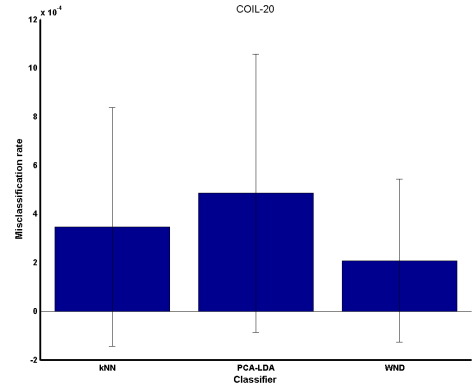
(b) Brodatz



(c) CHO

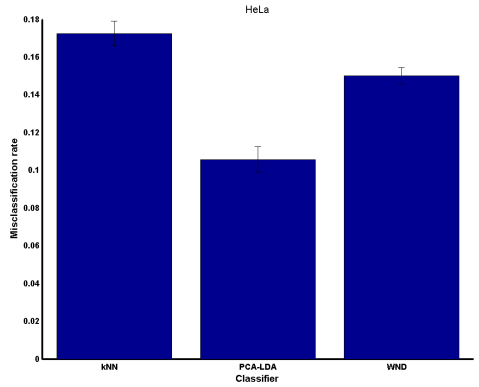
(d) COIL-20





(e) HeLa

(f) Pollen



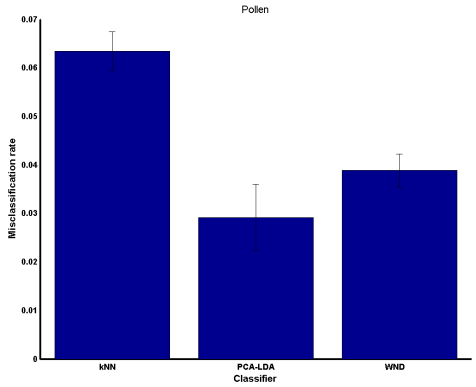


Figure 3.14: Misclassification results using the CHARM vector, different classification methods and 10-fold cross-validation in MATLAB for the reference datasets. Results range from 0 (0%) to 1 (100%).

(g) Yale

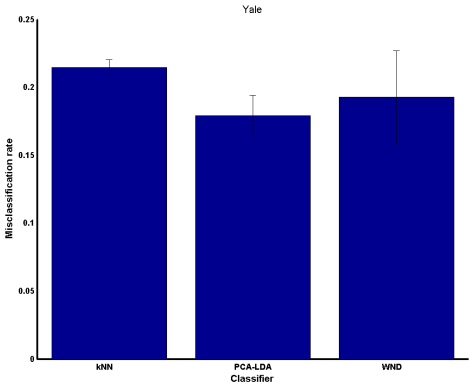


Figure 3.14: (Cont'd) Misclassification results using the CHARM vector, different classification methods and 10-fold cross-validation in MATLAB for the reference datasets. Results range from 0 (0%) to 1 (100%).

3.1.4 Validation

We implemented two different validation methods to test our implementation as we explained before in the Material and Methods section. The original C++ implementation of WND-CHARM uses "save 25%", but as we saw in 3.1 this method has a very large variance and is not a standard one. We therefore preferred k-fold cross-validation which is more reliable. In Fig. 3.15 we reported results using WND-CHARM and validating with either method using our Python version. Results were averaged over 100 classification runs for "save 25%" and 10 for 10-fold cross-validation as the result obtained by training and validating one classifier is actually already the average of 10 splits of the dataset in 10-fold while in the "save 25%" case only one split is made. We selected k=10 as it is a standard choice of parameter for cross-validation.

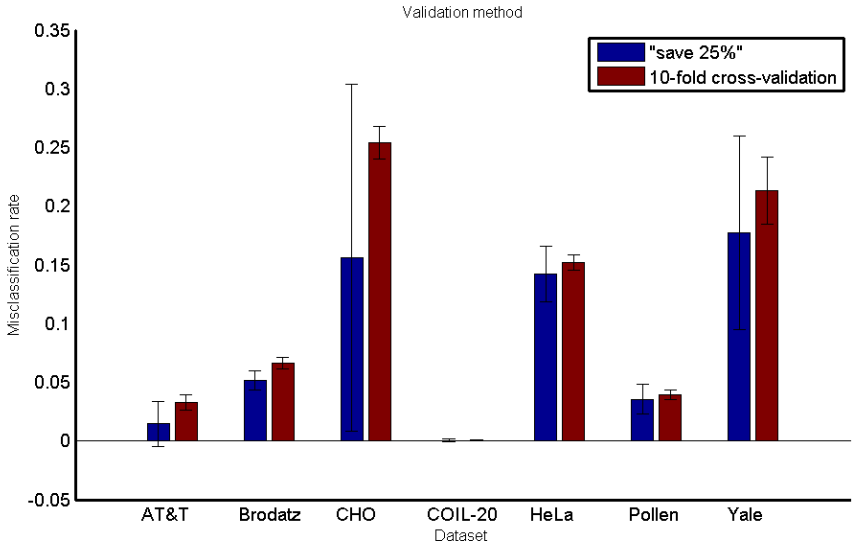


Figure 3.15: Comparison of misclassification rates when validating WND-CHARM with the "save 25%" method and 10-fold cross-validation. Results range from 0 (0%) to 1 (100%).

Let us recall the principle of each method: while k-fold cross-validation splits the dataset into k subsets and validates k times the classifier using k-1 subsets for the training and the last one for validation, "save 25%" only randomly selects 25% of the dataset in each class and uses it for validation while training on the remaining 75% per class. It therefore makes sense to observe in Fig. 3.15 that 10-fold cross-validation exhibits a much smaller variance than "save 25%". We also see that 10-fold tends to give slightly worse results than "save 25%". This was expected from the fact that 10-fold is a much more stringent method than "save 25%": on the 10-fold method no constraint is put on the way the 10 splits are created and it might be possible that all the elements of one class are present in only one split by chance, therefore affecting the overall performance when this split is kept for validation. This situation cannot happen in the "save 25%" method which is more "gentle" as we ensure that the classifier is trained and validated with the same number of elements from each of the classes. In spite of this, results using either methods are quite comparable.

While k=10 is usually used for k-fold cross-validation as a standard value, we thought it could be interesting to try to use a value depending on the way each dataset was built. We proposed a formula in the Methods section that would allow us to find an empirical k_{max} being a function of the number of images per class. We obtained the values presented in Table 3.5. We used again our Python implementation of WND-CHARM and k-fold cross-validation to compare the effect of 10-fold and k_{opt}-fold cross-validation. Fig. 3.16 displays a MATLAB plot of our results averaged over 10 runs of testing and training. Datasets with a k_{max} larger than 10 are not shown as the results obtained using k=10 were already appropriate.

Table 3.5: Optimal k values for our reference datasets.

Dataset	Number of classes	Minimal number of images per class	k_{max}
AT&T	40	10	2
Brodatz	111	16	3
CHO	5	33	6
COIL-20	20	72	14
HeLa	10	73	14
Pollen	7	90	18
Yale	15	11	2

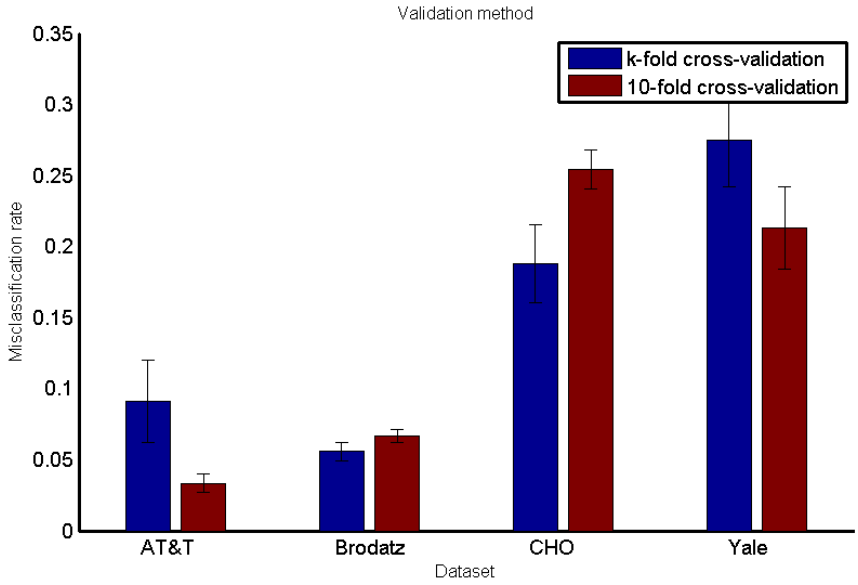


Figure 3.16: Comparison of misclassification rates when validating WND-CHARM using 10-fold and k-fold cross-validation, with k adapted to the nature of each dataset. Results range from 0 (0%) to 1 (100%).

We observe that adapting the value of k depending on the nature of the dataset as we proposed does not generally benefit classification results. Misclassification rate is larger in AT&T and Yale using $k = k_{max}$ instead of $k = 10$, and results do not seem to be affected in Brodatz. The CHO set is the only one to benefit from the adapted value of k, probably because the k we selected in this case was not too small (6, versus 2 in the case of AT&T and Yale). As already mentioned when describing k-fold cross-validation in the Methods section, the choice of k is very debated among the machine learning community. Taking into account the unconvincing results of Fig. 3.16 with different values of k, we dropped the idea of changing the validation parameter value and used the standard 10-fold instead. Adapting the value of k could still be a possibility. The optimal value should however be obtained using something different from the

formula we adopted to obtain values in Table 3.5 as it tends to give too small k values.

As a conclusion we observe that in spite of the slight differences in performance between the two validation methods the overall rank order of classification accuracies on the different datasets is mostly conserved. We conclude then that the design of the features vector and the choice of the classification algorithm have a greater impact on classification results than the method chosen for validation. We therefore safely decided to switch for the more standard and robust 10-fold cross-validation.

3.2 "PCA-LDA-CHARM-like"

The analysis of the different steps and components of WND-CHARM we performed until now helped us designing our new "PCA-LDA-CHARM-like" algorithm as follows:

Feature vector: v3.2, extracted using pre-existing and newly implemented modules from CellProfiler. This CHARM-like vector was built using features groups and "levels" as suggested by our analysis (Fig. 3.3 and 3.4). The same transforms as in CHARM were used to built the "higher levels". We kept this structure as our analysis of higher order features (Fig. 3.6, 3.7 and 3.8) suggested that even if we were not fully able to explain what they meant, their effect couldn't be mimicked using random features, hence the fact they capture actual information on the image couldn't be proved to be wrong.

Dimension reduction: PCA, as our analysis of the "top 15%" feature subset helped us notice that while the weighting scheme from WND-CHARM seems to be efficient in the sense that it yields good results, it also behaves in a quite anarchic way. Switching for the widely accepted PCA therefore seemed safer. We observed in Fig. 3.13 that on average 96% of the variance needed to be conserved to give the best results on the reference datasets. As these datasets were used in order to benchmark the algorithm and might be simpler than real-life cases. We decided to use instead the standard value of 98% for further experiments. This value indeed yielded good results on all datasets as well while keeping a reasonable amount of information present in the untransformed data, which might help for more difficult classification tasks. As in WND-CHARM, features were centered and scaled before performing dimension reduction.

Classification: LDA, as this reliable, widely used and well-defined method exhibited promising results in our classifier analysis shown in Fig. 3.14.

Validation: 10-fold cross-validation, following our observations from Fig. 3.15.

In this way we saved the core ideas of WND-CHARM, namely features extracted on the whole image and vector built from groups and levels.

3.2.1 Comparison

Before presenting the actual results we found important to discuss the relative computation time required by each method. First computation time is dependent on the number of elements (images) in the dataset, hence training a classifier using for example the COIL-20 or the Brodatz dataset would requires a long computation time, while it could be much faster on the small Yale dataset. Secondly, the size of the images greatly varies between the different datasets, and since most features extraction algorithms are dependent on image size the typical dimension of images in the dataset also affects execution time. We present in Tables 3.6 and 3.7 WND-CHARM and PCA-LDA-CHARM-like execution times on two extreme cases (Brodatz, the biggest dataset, and Yale the smallest one) and on a typical example (HeLa). We displayed the time needed for features extraction, for classification, and the total time. In the C++ version the process is separated into "training" and "testing", so the time reported in the features extraction column actually takes into account the time needed to train the classifier as well. In our algorithm the feature extraction (CellProfiler) and training/testing (Python script) steps are totally independent, therefore the classification column records the time required to train and test the classifier. All experiments were run via the Broad's LSF cluster.

Table 3.6: Execution time using WND-CHARM on the LSF cluster (in seconds).

Dataset	Features extraction	Classification	Total
Brodatz	34507.42	390.06	34897.48
HeLa	27264.35	69.75	27334.1
Yale	7949.96	3.56	7953.53

Table 3.7: Execution time using PCA-LDA-CHARM-like on the LSF cluster (in seconds).

Dataset	Features extraction	Classification	Total
Brodatz	74711.62	597.01	75308.63
HeLa	58267.01	123.19	58390.2
Yale	7755.00	17.28	7772.28

First, we notice that in both cases features extraction is obviously the most time-taking step. For small datasets like Yale our algorithm appears to be faster than WND-CHARM, though the difference is not especially remarkable. For larger datasets the execution time difference becomes much more striking: our implementation requires at least twice the time needed by WND-CHARM to perform a full classification experiment, essentially due to features extraction. We think that this observation can be better explained by the programming language used in each case rather than by the structure of each algorithm. Benchmarking tests [10] show that in order to perform a similar task Python can take ten to hundred times more time than C++. This slowness is due mostly to the fact C++ is a compiled language while Python is interpreted,

but other aspects of the way the language is conceived make Python more time-consuming. However Python has a big advantage over C++ at the code readability level. Moreover our goal here was to make our program user-friendly, and this is what motivated our choice for Python: we wanted to be able to make it part of CellProfiler. As classification experiments applied to biology do not have stringent time constrains we think the benefits brought by its availability in a widely used open-source software outweigh the relative slowness of our algorithm versus WND-CHARM

One of the essential aspects of our method had to be be the ability to recreate equally good or in best-case scenario better results than WND-CHARM. In order to make sure this condition was respected we performed several comparisons of results obtained with WND-CHARM and PCA-LDA-CHARM-like. In order to cover a wide range of possible data types we of course re-used our reference datasets from [38], but we also gathered results on the IICBU datasets presented in [47]. These two sources contain most of the published comprehensive results using WND-CHARM and therefore appeared to be a sufficient benchmarking set for our new algorithm. In order to make results comparable we brought the two methods to a common basis by using the same validation technique, once with each algorithm's method. Fig. 3.17 compares the two algorithms using the "save 25%" from WND-CHARM and Fig. 3.18 using 10-fold cross-validation from PCA-LDA-CHARM-like, in both cases on reference datasets. All results were obtained using our Python code and averaged over 100 ("save 25%") and 10 (10-fold) classification runs.

We observe that for both validation methods our algorithm appears to be a good competitor for WND-CHARM. On most datasets the results are comparable, and they even seem to be better on a few cases as it can be observed in Fig. 3.18. Compared to the results in Fig. 3.18, there is not indeed much significant differences between the performance of the two approaches in Fig. 3.17 because of the large variance of the "save 25%" method. The global trend is conserved across the different methods: PCA-LDA-CHARM-like tends to obtain better mean results than WND-CHARM on AT&T, CHO and Yale, and the opposite situation is observed for HeLa and Pollen. In our results on Brodatz, the approach obtaining the best mean classification accuracy depends on the classification method, and results on COIL-20 using either approach are so close it is hard to determine which method performs best.

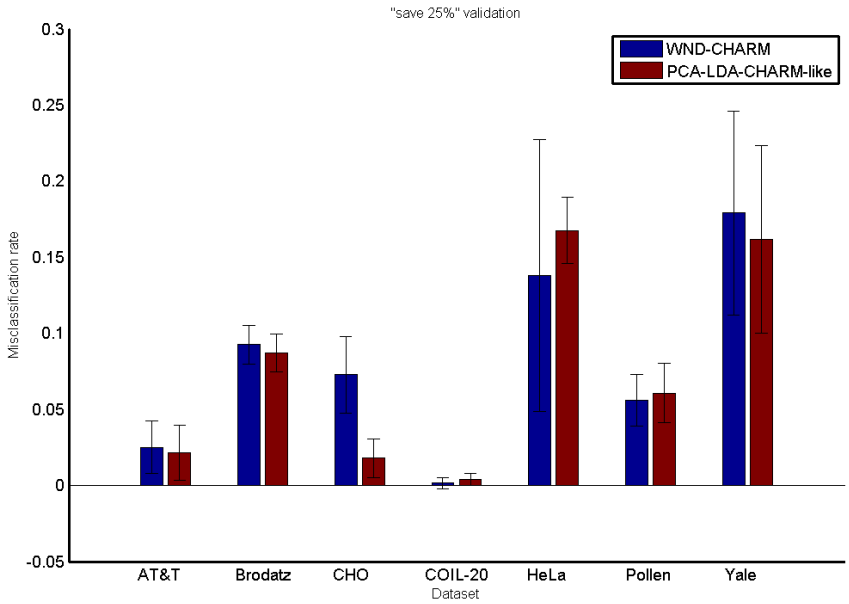


Figure 3.17: Comparison of misclassification rates obtained with WND-CHARM and our algorithm using the "save 25%" validation method. Results range from 0 (0%) to 1 (100%).

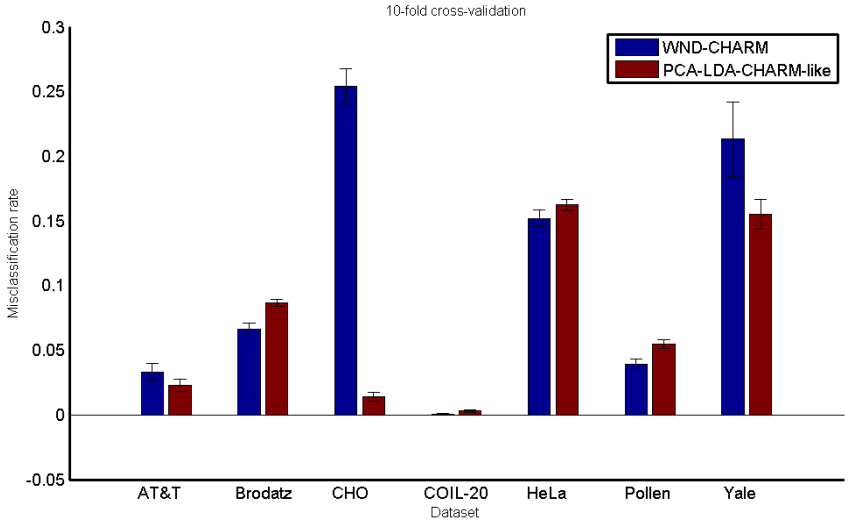
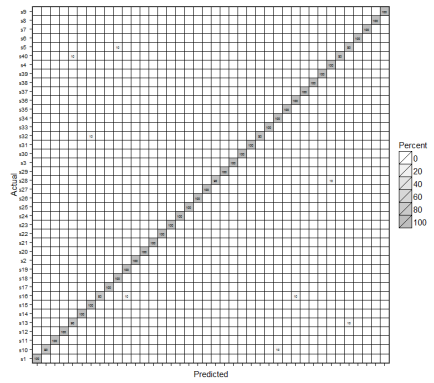


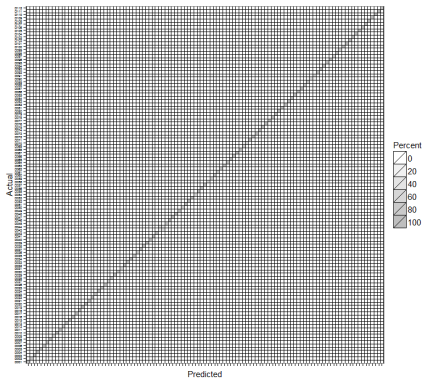
Figure 3.18: Comparison of misclassification rates obtained with WND-CHARM and our algorithm using 10-fold cross-validation. Results range from 0 (0%) to 1 (100%).

In order to make sure the average classification result we see for PCA-LDA-CHARM-like truly indicates the ability to separate accurately each different class, we displayed a typical confusion matrix obtained with 10-fold cross-validation for each dataset in 3.19. These results helped us making sure each class was reasonably well separated.

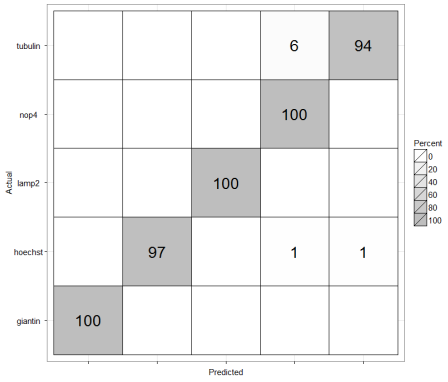
(a) AT&T



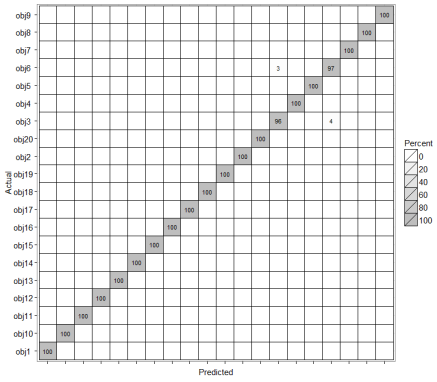
(b) Brodatz



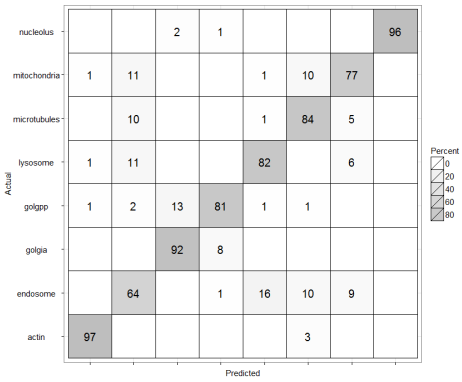
(c) CHO



(d) COIL-20



(e) HeLa



(f) Pollen

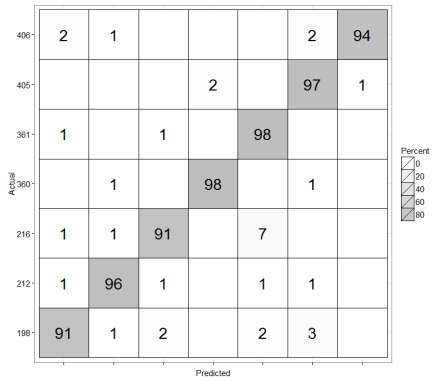


Figure 3.19: Typical confusion marices for the reference datasets using PCA-LDA-CHARM-like and 10-fold cross-validation.

(g) Yale

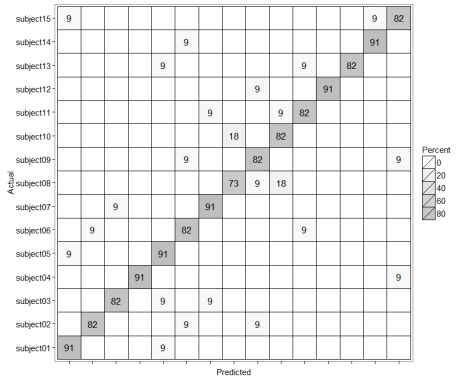


Figure 3.19: (Cont'd) Typical confusion marices for the reference datasets using PCA-LDA-CHARM-like and 10-fold cross-validation.

In the cases of 3.19a, 3.19b, 3.19d, 3.19c and 3.19f we observe little confusion as suspected from the good classification rates (98, 91, 100, 98 and 94% correct on average, respectively). More confusion can be seen on 3.19e, and 3.19g. These two sets seem to possess many elements that look like each other, presenting a harder classification task. We know for instance in the case of HeLa that images from the different Golgi classes golgpp and golgia are difficult to identify by eye, and that cell components like lysosomes and endosomes also are troublesome to classify manually ([47]). We can indeed observe confusion between these elements in 3.19e. In general we see with these matrices that our PCA-LDA-CHARM-like performs well. The difficulties it encounters seem to truly emerge from the nature of the data and not from a bad design of the algorithm.

We also compared our results on the IICBU sets ([47]). These 11 image sets were classified using WND-CHARM and PCA-LDA-CHARM-like and validated with "save 25%" in order to make comparison with WND-CHARM meaningful. Results averaged over 100 runs are presented in 3.20.

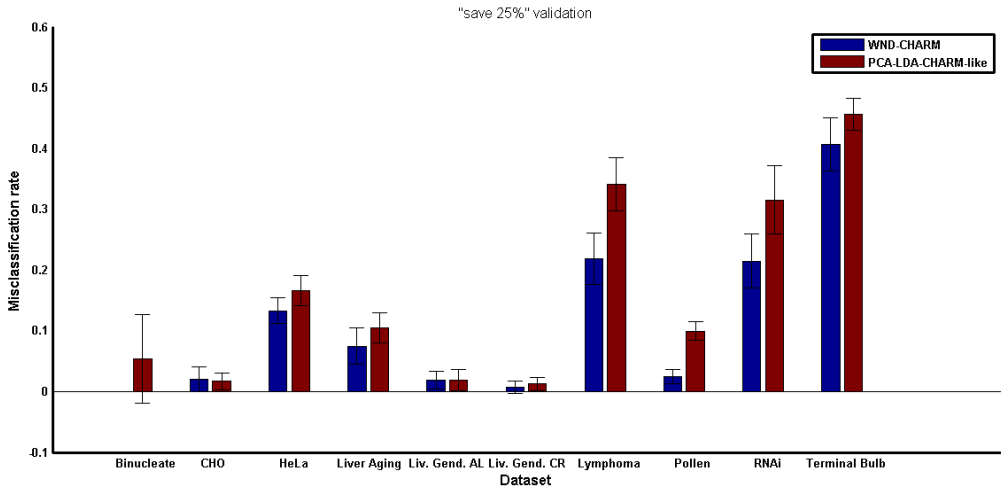
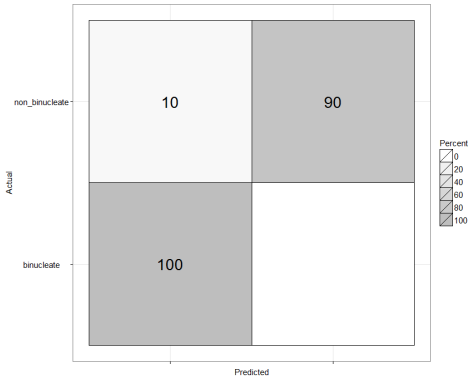


Figure 3.20: Comparison of misclassification rates using WND-CHARM and our algorithm with "save 25%" for validation on the IICBU datasets. Results range from 0 (0%) to 1 (100%).

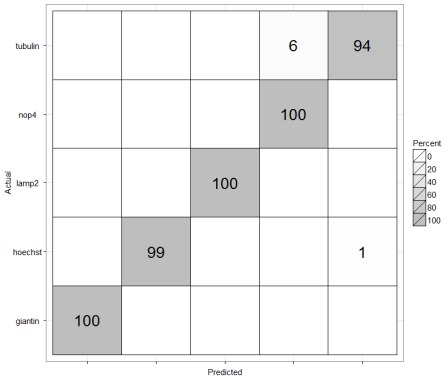
PCA-LDA-CHARM-like again obtains results that compete with WND-CHARM's on most datasets. On Lymphoma, RNAi and Pollen our algorithm is outperformed, but the different is on the order of 1%, Moreover we can see that the biggest discrepancy between results is found on the Pollen dataset, which was also part of our reference datasets and obtained comparable results with WND-CHARM in Fig. 3.17. This allows us to conclude that we are globally able to obtain satisfactory results on the IICBU datasets. Apart from the relatively bad performances on Terminal Bulb, Lymphoma and RNAi, our algorithm seems suitable for the analysis of varied types of biological image data.

Once we made sure PCA-LDA-CHARM-like was able to perform well enough on the IICBU suite, we switched back to its original 10-fold cross-validation and displayed typical confusion matrices for each of the IICBU set in Fig. 3.21.

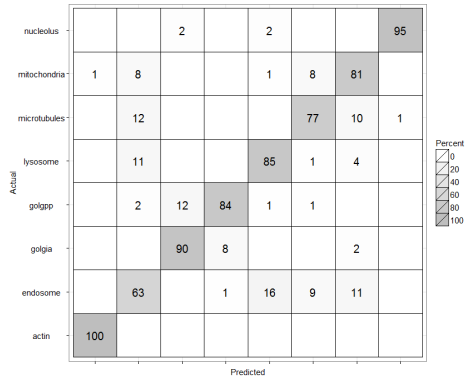
(a) Binucleate



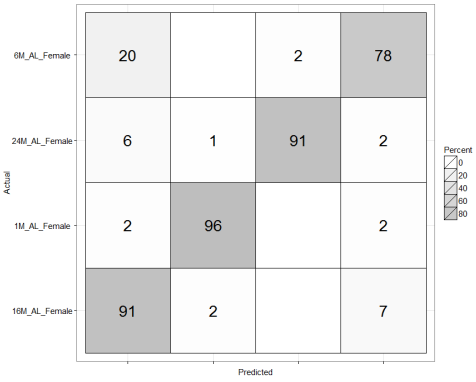
(b) CHO



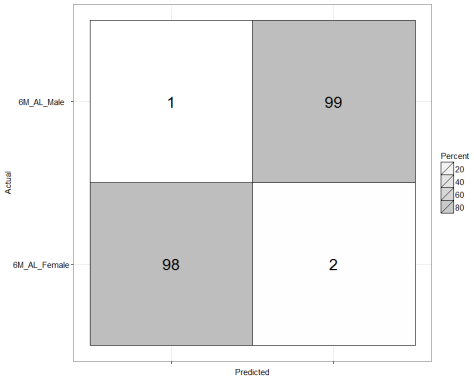
(c) HeLa



(d) Liver Aging



(e) Liver Gender AL



(f) Liver Gender CR

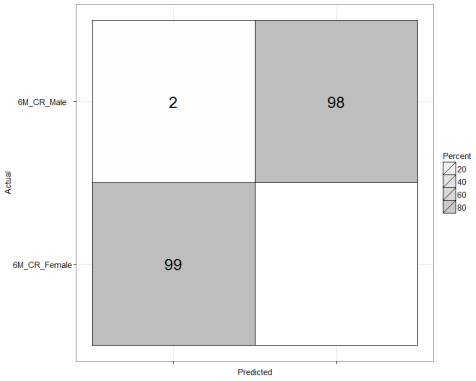
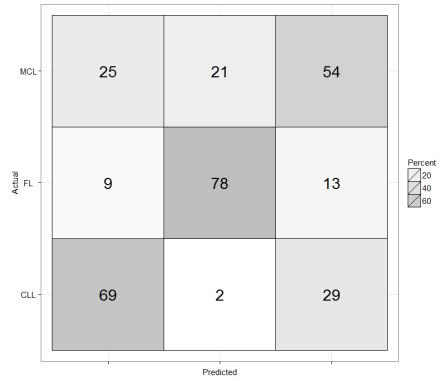
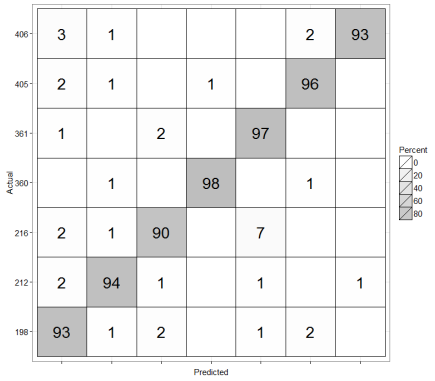


Figure 3.21: Typical confusion marices for the IICBU datasets using PCA-LDA-CHARM-like.

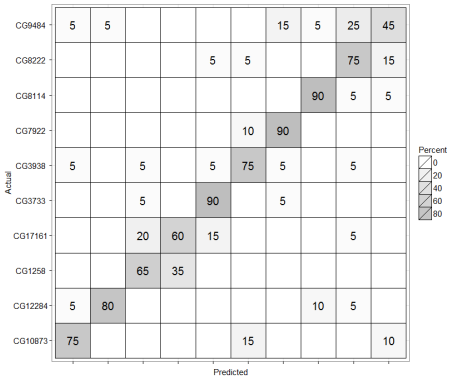
(g) Lymphoma



(h) Pollen



(i) RNAi



(j) Terminal Bulb

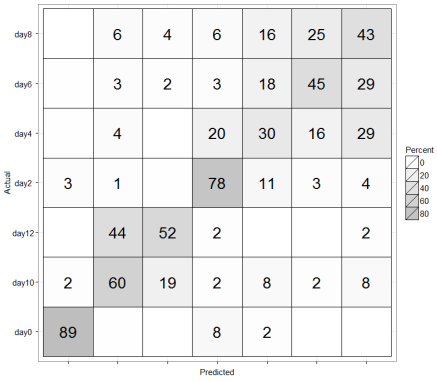


Figure 3.21: (Cont'd) Typical confusion marices for the IICBU datasets using PCA-LDA-CHARM-like.

As most of these sets are composed of less classes than the reference datasets the resulting confusion matrices are more readable. These images represent more realistic biological conditions and are therefore harder to classify than the reference datasets used earlier. We however see that the results are satisfying in most cases: 3.21b, 3.21d, 3.21e, 3.21f and 3.21h display only a slight amount of confusion and all classes are well separated. In the cases of 3.21c, 3.21a, 3.21i and 3.21j elements in the confusion matrix are less concentrated in the diagonal and we detect groups of classes that are badly separated, for instance day8, day6 and day4 in Terminal Bulb or CG9484 and CG8222 in RNAi. This is not surprising as these datasets are expected to be difficult problems. It is for example mentioned in [47] that while the RNAi image set presents a similar problem as Binucleate, it is expected to be a harder classification experiment. This is what we observe by comparing 3.21i and 3.21a. A dramatic situation is seen in 3.21g where the confusion matrix appears to be mostly scrambled. Here again we think this result comes from the intrinsic difficulty of the image data. As already discussed

when presenting it in the Materials and Methods section, the Lymphoma dataset is composed of images taken in very different conditions and therefore exhibits large in-class variations making classification difficult.

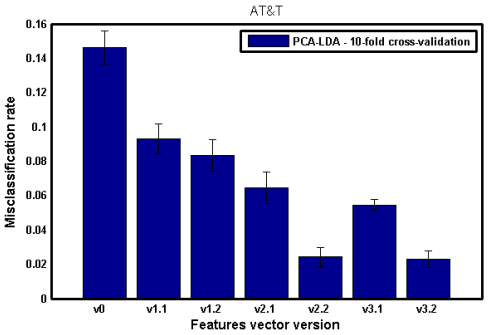
With these results we can affirm we managed to build a whole-image classification algorithm which performances compare to WND-CHARM. It can therefore be used without further comparison on real-life datasets.

3.2.2 Analysis

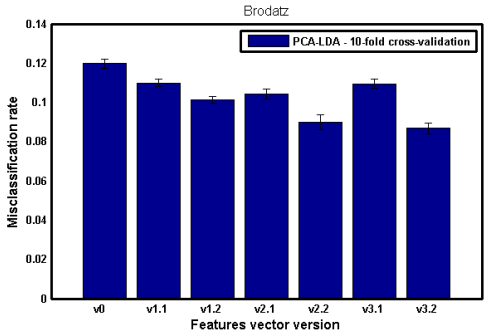
We felt satisfied by our method as described at the beginning of this section and therefore used it on some application examples, but we performed some experiments concerning the features vector and classification method of PCA-LDA-CHARM-like in order to determine what could be further improved to design in the future a better version of our algorithm.

We first went back to the different features vector versions we built to imitate the CHARM vector. As a reminder, we chose v3.2 for our algorithm as it appeared to perform well with WND, but results using v2.2 were equivalently good. In order to check if our choice was also an optimal solution for PCA-LDA, we classified all the different features vectors versions using PCA-LDA and validated them with 10-fold cross-validation in Python. Results averaged over 10 runs can be found in Fig. 3.22.

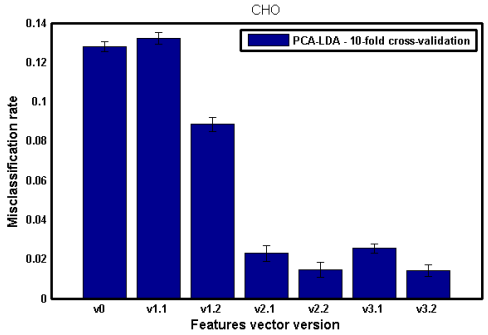
(a) AT&T



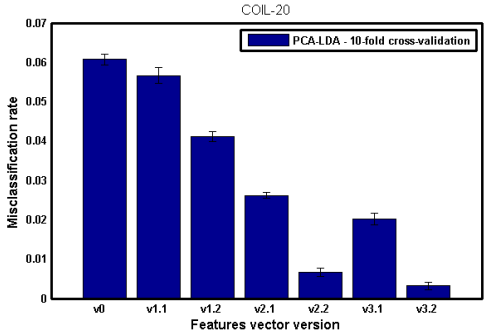
(b) Brodatz



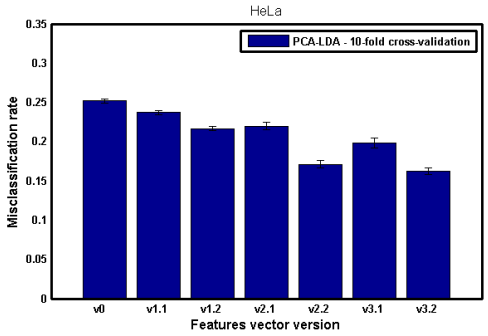
(c) CHO



(d) COIL-20



(e) HeLa



(f) Pollen

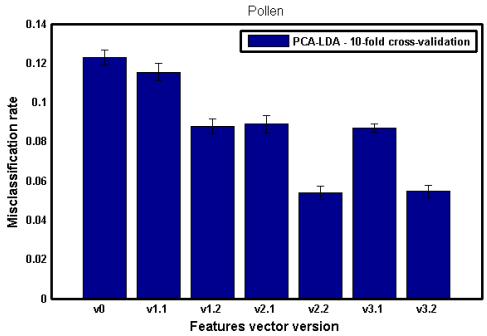


Figure 3.22: Misclassification rates using the different features vector version, PCA-LDA and 10-fold cross-validation. Results range from 0 (0%) to 1 (100%).

(g) Yale

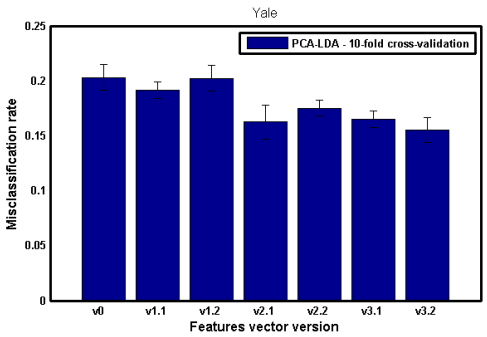


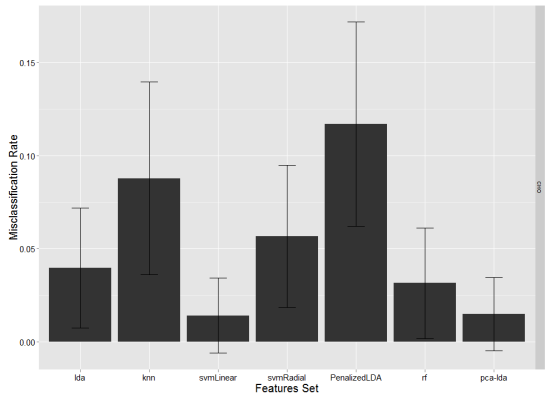
Figure 3.22: (Cont'd) Misclassification rates using the different features vector version, PCA-LDA and 10-fold cross-validation. Results range from 0 (0%) to 1 (100%)

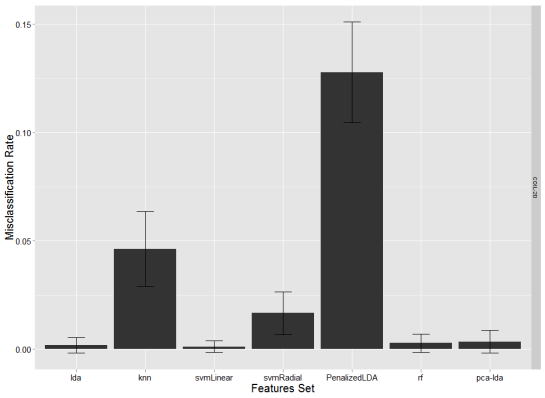
It can be seen that the v3.2 we chose indeed gives the best performance on all reference datasets. However just like it was the case with WND, v2.2 competes with v3.2 in every cases except 3.22d, but the difference there is in the order of less than a percent. We hence reach the same conclusion as earlier: a possible improvement could be to use v2.2 as a feature vector. This would benefit the algorithm in two ways: first the computation time needed for features extraction would be reduced as v2.2 is composed of less elements than v3.2, and second we wouldn't be using compound transforms. As we extensively mentioned, on one hand we do not really understand what higher level features capture at the image level, and we could hence feel safer by not using them. However on the other hand as we suspect these features are actually capturing some aspects of the image, removing them might restrict the power of the algorithm in certain situations. In any case the results we see in Fig. 3.22 are reassuring in the sense that v0 gives bad performances in all datasets, meaning that the features extraction modules we implemented in CellProfiler add a real value when it comes to whole-image-based classification.

We then wanted to look again at the classification method and try more algorithms combined with our v3.2 vector and 10-fold cross-validation. We used RStudio and the R `caret` package ([29]) already mentioned before to try classifying with k-NN, LDA and other dimension reduction methods than PCA, Penalized LDA, RBF-SVM, Linear-SVM, and Random Forests all of which were previously discussed in the Methods section.

(a) CHO

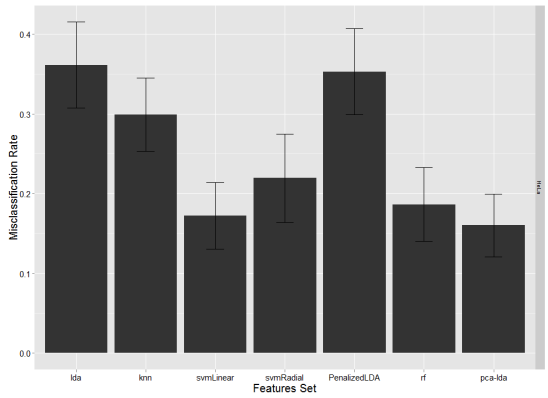
(b) COIL-20

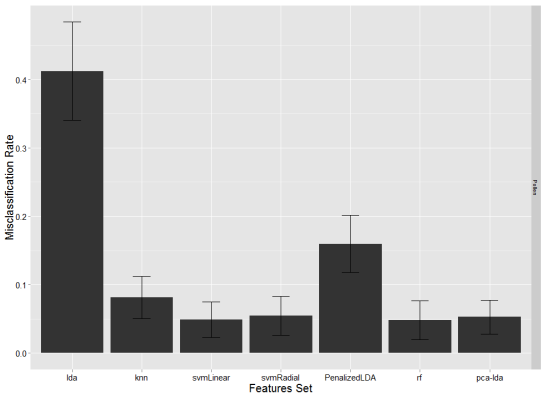




(c) HeLa

(d) Pollen





(e) Yale

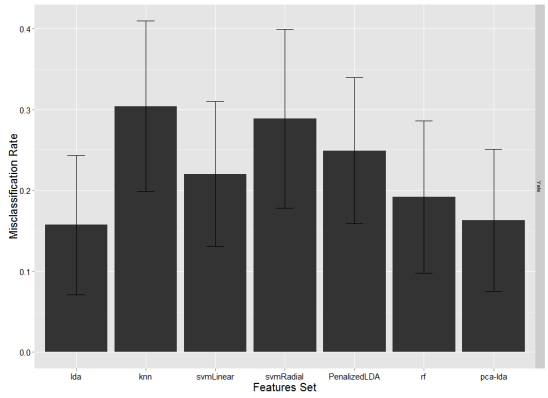


Figure 3.23: How good can we get: Misclassification rates using our features vector v3.2, different classifiers and 10-fold cross-validation on R. Results range from 0 (0%) to 1 (100%).

These algorithms were all available in the `train` function of the `caret` package: `knn`, `lda`, `PenalizedLDA`, `svmRadial`, `svmLinear`, and `rf`. As the R SVM implementation execution time explodes on problems with a large number of classes we excluded the Brodatz and AT&T datasets from these experiments. Features were centered and scaled as a pre-processing step in every cases, and we averaged results in Fig. 3.23 over 10 runs.

First of all we notice that COIL-20 (3.23b) is definitely too easy to classify to be the subject of an analysis as most methods perform similarly well. In the other datasets we observe that RBF-SVM, pLDA, o-LDA (LDA with other dimension reduction methods) and k-NN display a large variation of results. In some cases they obtain good results while in others their performances are catastrophic. k-NN, pLDA and o-LDA most often obtain significantly worse performances than the other algortihms. PCA therefore seems to be a good option for pre-processing before LDA as better results are obtained than when combining LDA with other features selection methods. While the bad results of k-NN are not extremely surprising from the fact that it is a very simple method, we did expected better performances for pLDA. RBF-SVM gives always either similar or worse results than Linear SVM on average, a trend we already observed when using the CHARM vector. This can probably be explained by the fact that our datasets are already linearly separable and therefore do not need to be mapped in a higher dimensional space. RF and Linear SVM give results that compete with our PCA-LDA in most cases. They however display a greater variability of results across datasets. PCA-LDA still seems to be the best option as it tends to be the most constant method: it is the one that appears most often in the top results. This motivates the choice of PCA-LDA as it looks to be the option with the largest generalization power.

This analysis of our algorithm leaves us with two potential leads for improvement: switching to a features vector version with less levels but same groups, i.e. using v2.2 instead of v3.2, and try using Support Vector Machines or Random Forests instead of PCA-LDA. One of the drawbacks of switching for SVM would be to render the algorithm highly sensitive to the number of classes, and execution time could become prohibitive for multiclass problems.

3.3 Applications

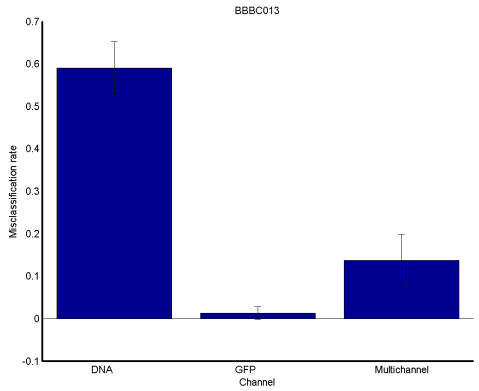
We will present in this section some examples of applications of our PCA-LDA-CHARM-like whole-image-based classifier. These examples aim to confront our algorithm to real-life scenarios and to show how it could be used in actual research.

3.3.1 BBBC Datasets

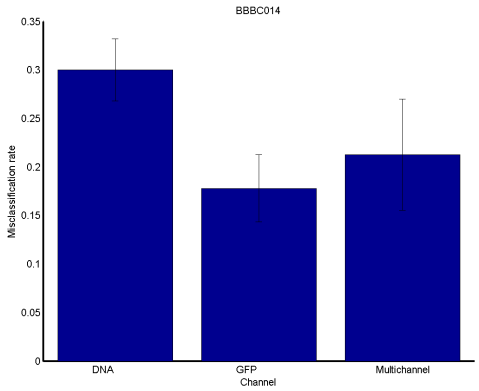
The BBBC datasets ([2]) are typical benchmarking sets for image analysis algorithms. The four sets we selected are two-classes problems where the goal is to discriminate between a "negative" and a "positive" phenotype. All sets were composed of images from two channels: one containing a GFP signal linked to the phenomenon to be observed, and the second

containing signal from a stain that nonselectively binds cell nucleus. We therefore expected DNA channel to be much less informative on class difference compared to the GFP channel. We classified each channel separately as well as the two channels together and displayed the results in Fig. 3.24. All results were averaged over 10 classification runs for robustness purpose.

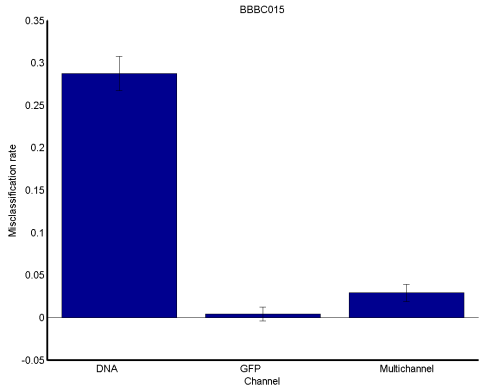
(a) BBBC013



(b) BBBC014



(c) BBBC015



(d) BBBC016

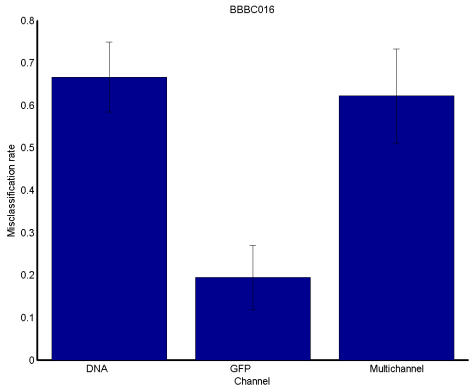


Figure 3.24: Mislassification rates using images from different channels of the BBBC datasets. Results range from 0 (0%) to 1 (100%).

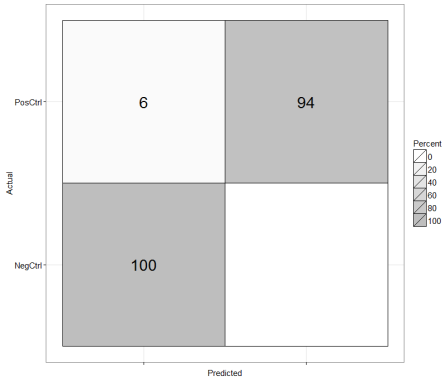
As expected classification using images from the DNA channel only yielded very bad results, while images from the GFP channel obtained a low misclassification rate, and hence a good classification accuracy. It is interesting to note that multichannel classification gives worse performances than GFP-only: the DNA channel images seem to confuse the classifier and make class separation harder. We will therefore use only data from the GFP channel for these BBBC experiments as it is observed to yield better results, and saves us computation time as less images need to be processed.

Classification accuracies obtained with PCA-LDA-CHARM-like for each of the BBBC dataset using GFP channel images only are presented in Table 3.8. These results are very satisfying as each case is a two-class problem with equal number of images per class, and we hence expect random classification accuracy to be 50%. The variance in classification efficiency is observed to be quite large for BBBC016. This can be explained by the fact that this dataset is very small, meaning that the classifier only has a small set of images to train on. Inside a class there can be slight variations, but the presence of many elements in the training set usually allows the classifier to detect enough common elements among class samples. When the training set is too small it can happen by chance that the images left for the test set differ from the one in the training set and form a "subset" inside the class. The classifier will then give bad results as there are not enough representatives of this "subset" to be present in the training set. This is probably what happened here as we were training and testing on 9 images per class only. In order to check the validity of our results we displayed typical confusion matrices for these datasets in Fig. 3.25.

Table 3.8: Classification efficiencies in percent on the BBBC datasets.

	Accuracy [%]	
Dataset	Mean	Standard Deviation
BBBC013	98.75	1.53
BBBC014	82.19	3.44
BBBC015	99.58	0.83
BBBC016	80.55	7.56

(a) BBBC013



(b) BBBC014

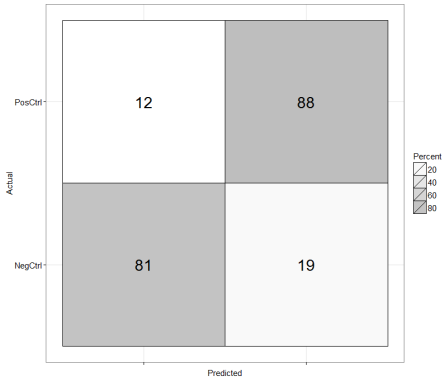
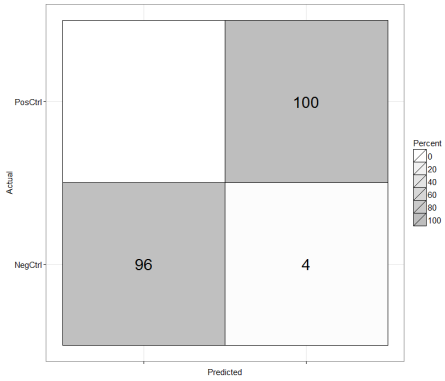


Figure 3.25: Typical confusion matrices for the BBBC datasets.

(c) BBBC015



(d) BBBC016

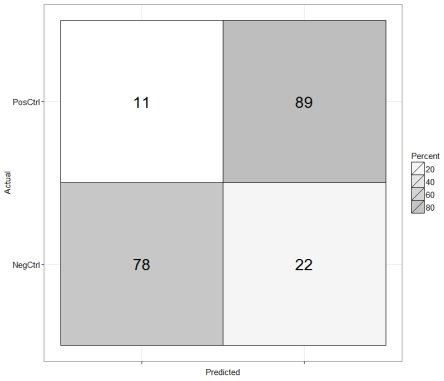


Figure 3.25: (Cont'd) Typical confusion matrices for the BBBC datasets.

These confusion matrices confirm the good results presented in Table 3.8. In order to validate our results we performed significance tests on the BBBC datasets. We shuffled class labels and ran 10'000 classification experiments on these randomized data. We obtained from this a measure of the random classification accuracy, and were therefore able to compute a p-value for the results presented in Table 3.8. In this way we could make sure that the performances of our classifier were better than what could be obtained randomly. Results are presented in Table 3.9. We observe that for BBBC013 to 015 no random classifier was found to obtain an equal or better classification accuracy that the actual one. BBBC016 on the contrary obtained a p-value of 0.02, which is not really surprising from the fact that this dataset was highly underpopulated. The small size of the set made it difficult for the classifier to identify patterns specific to each class, and it was hence more likely to classify randomly. It can however be noted that the significance threshold for a p-value is usually set either to 0.01 (1%) or 0.05 (5%). This means that a result is considered as significantly different from random if it is within either the 1% or the 5% extreme values of the distribution. If we select the stringent threshold of 1% we cannot reject the hypothesis that our classifier performs as random on BBBC016, but if we choose the more permissive 5% value our classifier would be found to be valid. The conclusion we could draw from this was that results on BBBC016 should be taken with care.

Table 3.9: Classification efficiency in percents on randomized datasets using PCA-LDA-CHARM-like.

Dataset	Mean	Standard Deviation	p-value
BBBC013	49.31	10.54	0.00
BBBC014	49.21	10.71	0.00
BBBC015	49.46	8.77	0.00
BBBC016	49.09	14.42	0.02

In order to make sure our results were not too much biased by the limited amount of training data we had at our disposition, we re-ran our classifier on "enriched" BBBC sets. We increased the number of images in each dataset by 4 by chopping down every image into 2x2 equally sized tiles. While this technique helped us addressing the lack of images, it added another bias: four images created out of a big image are exposed to an enhanced risk of being highly correlated. Indeed, if an image is subject to a particular global effect independent of the class difference we want to measure results would probably not be too much affected as long as only a few images are concerned. However classification efficiency could be affected to a greater extent when this effect is present in a high proportion of the dataset. By creating more images out of a single one we potentially duplicate these unwanted effects. Comparison of misclassification rates using the original sets and the enriched ones are presented in Fig. 3.26.

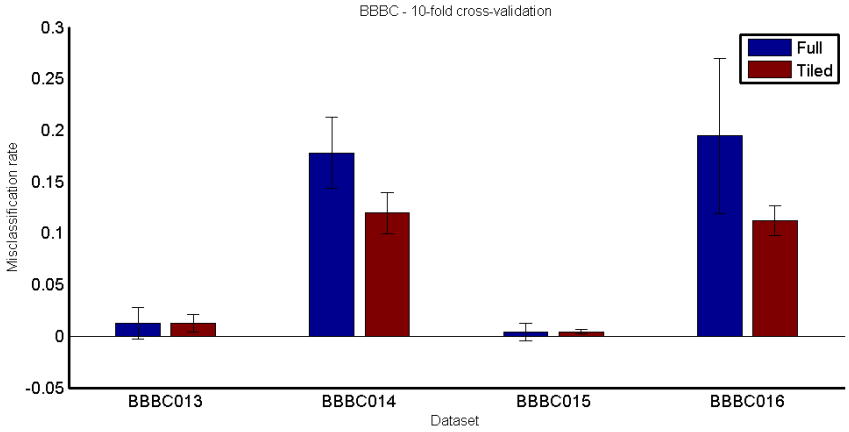


Figure 3.26: Comparison of results between tiled and full images of the BBBC datasets.

We observe that misclassification rates on the tiled sets are either lower or comparable to the original ones, which is a good news as it indicates that we do not suffer from unwanted correlation in our tiled images. It also seems that the tiling had the effect we wanted on the results: the variance of classification accuracy between runs is diminished indicating a more robust classifier. This is mostly observed on BBBC016 where the variance is greatly reduced, which is not surprising as it corresponded to the least populated dataset and hence the most prone to instability depending on the nature of the training set.

We performed on these enriched data the same significance tests we did before on the original datasets. Class labels were again shuffled, and the results of 10'000 classification runs on these randomized data were gathered. Mean random classification accuracies can be found in Table 3.10, as well as p-values for results on the tiled datasets presented in Fig. 3.26.

Table 3.10: Classification efficiency in percents on randomized enriched datasets using PCA-LDA-CHARM-like.

Dataset	Mean	Standard Deviation	p-value
BBBC013	49.70	5.49	0.0
BBBC014	49.80	5.51	0.0
BBBC015	49.82	4.56	0.0
BBBC016	49.67	7.23	0.0

As it was the case for the actual classification results, we see that the variance is reduced by two on the enriched data as compared to Table 3.9. It is also interesting to observe that the BBBC016 set benefits from the addition of images in the dataset: our classifier appeared this time to be clearly relying on features information and not simply classifying randomly. This result indicate us that the p-value we obtained on the original dataset reflected more the poor amount of training and testing data than a failure of our classification algorithm.

Since these datasets were the most relevant to CellProfiler, we computed the Z' factors for every features contained in the v0 and v3.2 vectors on the enriched BBBC datasets. The largest Z' value is indicated in Table 3.11 for each dataset. We only reported this metrics as we recall that it indicates the best classes separation that could be obtain using the features we extracted on the images.

Table 3.11: Highest Z' scores for the BBBC datasets using v0 and v3.2.

	Accuracy [%]	
Dataset	Z' (v0)	Z' (v3.2)
BBBC013	-0.120	0.065
BBBC014	-3.382	-1.815
BBBC015	0.174	0.174
BBBC016	-0.778	-0.551

We see that in the case of BBBC013 new features available in v3.2 benefited the Z' score, bringing its best value over zero. In BBBC014 and BBC016 the initial situation with v0 was quite bad and the added elements did not bring the highest Z' to a positive value, but the new measurements still helped as the best value got closer to zero. No improvement was observed for BBBC015, probably because it was a very easy case where class separation was obvious with basic features. These results were gratifying as it showed us that the features extraction algorithms we implemented for CellProfiler give an advantage when using a whole image-based classifier.

As a conclusion our PCA-LDA-CHARM-like algorithm seemed to give satisfying results on these typical example data of high-throughput screens. It is however important to keep in

mind that the BBBC datasets are assumed to be easy, hence the good performances of our method were not very surprising.

3.3.2 Tissue Dataset

Our tissue dataset created using images from the Human Protein Atlas ([53]) is a typical example of tissue image classification where the goal is to identify several cellular compartments, as well as background images where no cells are present. The original color images were deconvolved to isolate the two stains present in the tissue: a brown dye targeting a protein of interest and a blue dye nonselectively labeling some cellular components. Different locations could be identified by looking at both channels. Not only the blue stain probably exhibits different patterns depending on the cellular compartment, but the pattern of accumulation of proteins is also certainly affected by its location. We ran 10 runs of classification experiments using the signal from each dye both individually and together as shown in Fig. 3.27.

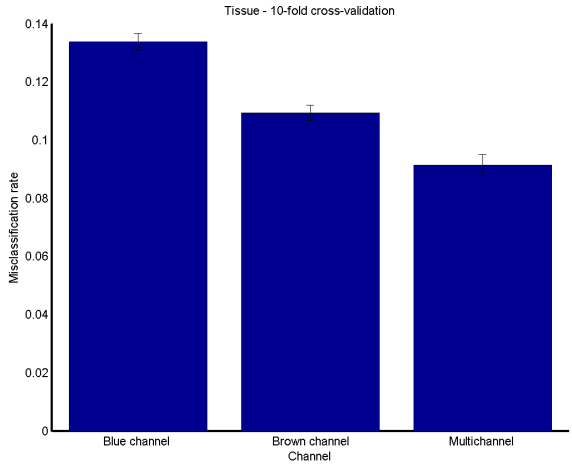


Figure 3.27: Misclassification rates using images from different channels of the Tissue dataset. Results range from 0 (0%) to 1 (100%).

As expected the presence of both channels together gives the best results. Unlike the BBBC situation observed earlier we see that the difference in discrimination power of images from either channel is not striking: images containing only information from the non-selective blue dye leave us with a larger misclassification rate than the ones including signal from the protein-linked brown stain, but the difference is in the order of 2 to 3% only. The averaged classification result on the Tissue dataset using PCA-LDA-CHARM-like using images from both channels can be found in Table 3.12.

Table 3.12: Classification efficiency in percent on the Tissue dataset.

	Accuracy [%]	
Dataset	Mean	Standard Deviation
Tissue	90.86	0.36

To better analyze this result we present a typical confusion matrix for this set in Fig. 3.28.

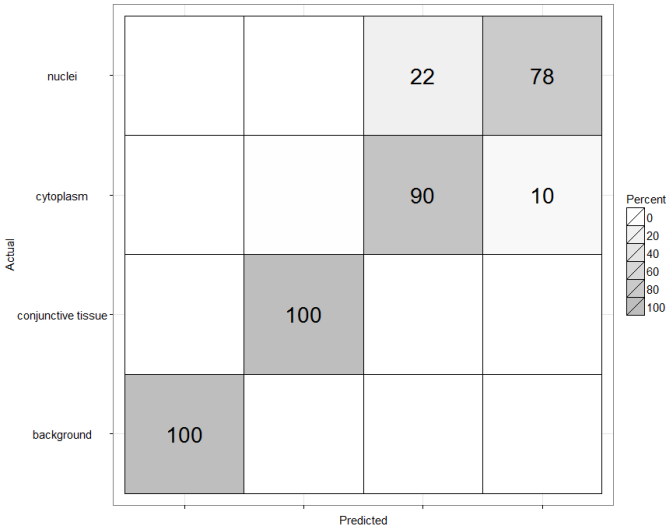


Figure 3.28: Typical confusion matrix for the Tissue dataset.

The background class can be easily distinguished from the other ones as expected by the fact it is highly recognizable as it contains no protein signal. The confusion matrix also allows us noticing that the cytoplasm and nuclei classes seem to be very hard to distinguish as they often get confused for one another by the classifier while the conjuctive tissue class is always properly identified. Some samples from each of these three classes are shown in Fig. 3.29.

(a) Cytoplasm (b) Nuclei (c) Conjunctive Tissue

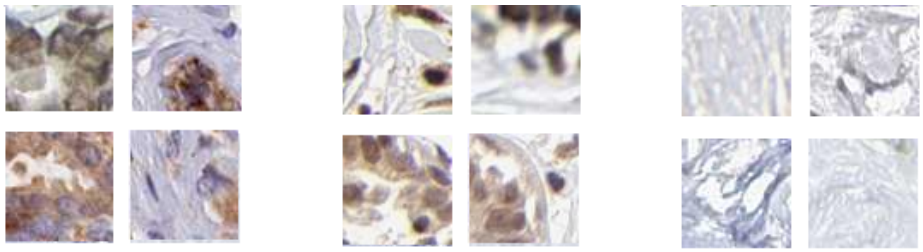


Figure 3.29: Examples of elements of the three cell compartments classified in the Tissue dataset. Background class is not shown.

We indeed see that images from the nuclei and cytoplasm classes are less different from one another than from the samples drawn from the conjunctive tissue class, therefore explaining the behavior observed in Fig. 3.28. The nuclei and cytoplasm classes contain signal from both stains while the conjunctive tissue and background ones exhibit information from only one or no stain at all. It is also noticeable when looking at the whole content of the two confused classes that they are much more heterogeneous than the conjunctive tissue and background class, thereby complicating the classification task.

We setup significance tests to validate our classification efficiency result presented in Table 3.12. 10'000 classification experiments were performed on the Tissue data with shuffled class labels. Using this random classification accuracy distribution, we computed a p-value for our result, given in Table 3.13.

Table 3.13: Classification efficiency in percents on the randomized Tissue dataset using PCA-LDA-CHARM-like.

Dataset	Mean	Standard Deviation	p-value
Tissue	32.07	2.20	0.0

We could apparently trust the decency of our result with tissue data as in 10'000 random classification runs no better or equal accuracy was obtained. The p-value of 0 allowed us concluding that PCA-LDA-CHARM-like is significantly more efficient than a random classifier on these data, and hence that the feature we extracted from images were actually useful in discriminating between classes.

These results are very promising. Tissue images analysis in the context of the HPA project is mostly performed manually so far, and the good classification accuracy we obtain with our algorithm on this subset of data indicates that automation of this process could be possible with a reasonable error rate. We think whole-image-based classification algorithms like our method dully display their strength in tissue images. In such cases segmentation is meaningless, and every part of the image contains information that can be captured by our feature vector as opposed to cell-based screens like BBBC where images are mostly composed of background and global effects could easily render our algorithm useless. The tissue data results presented in Table 3.12 show a realistic example of the power of our PCA-LDA-CHARM-like algorithm.

3.3.3 HDAC Dataset

The HDAC dataset consists of images of cells treated with different siRNA constructs against five HDAC protein isoforms. The resulting 14-class problem is a good example of a difficult high-throughput cell-based screen. This dataset came from an ongoing study carried out at the Imaging Platform. For this reason the results we present here are probably not definitive

as the imaging setup might evolve and images from new batches are likely to be produced. We constituted our dataset of only one image out of the six acquired for each well in order to avoid correlation effects since we had a sufficient amount of data. Five channels were recorded for each image, each one containing a signal from a tagged cellular component. As we had no prior information telling us which channel contains the looked-for phenoypical differences, we first analyzed images using all channels together. We used PCA-LDA-CHARM-like in 10 classification runs and averaged results to obtain the efficiency shown in Table 3.14.

Table 3.14: Classification efficiency in percent on the entire HDAC dataset.

	Accuracy [%]	
Dataset	Mean	Standard Deviation
HDAC	18.13	0.58

We notice that the classification result we obtained is far from being acceptable. The variance is very low, indicating that the mean accuracy of 18.63% seems to be a reliable measure of what we could expect from our classifier on these data. An example of confusion matrix for the HDAC set is presented in Fig. 3.30 in order to further analyze this mediocre result.

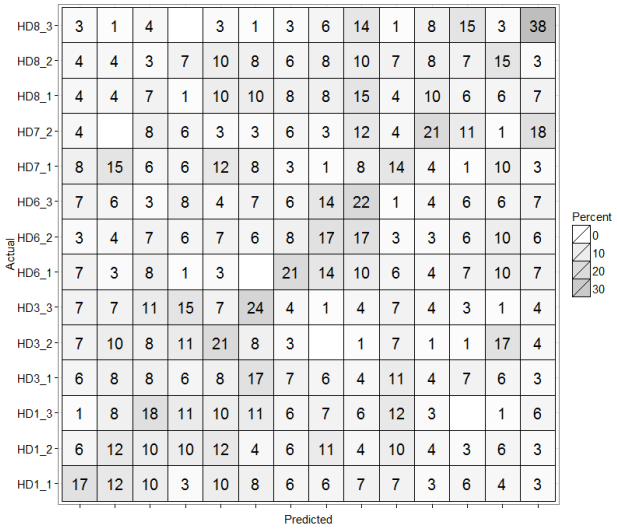


Figure 3.30: Typical confusion matrix for the HDAC dataset.

The confusion matrix looks quite catastrophic: nearly no trend could be observed in the diagonal. Apart from HDAC8_3, HDAC6_3 and HDAC3_3, the other classes look extremely confused. We noticed that there is no global tendency for confusion: we don't observe particular columns where confusion is concentrated. It is also interesting to notice that a lot of confusion appears within constructs against the same HDAC isoform. The difference between these classes is probably lighter than the dissimilarity between isoforms. A better classification efficiency

could hence probably be obtained by changing the experiment to a six-class classification problem where we try to separate the HDAC isoforms but ignore the different constructs for an identical isoform.

As the result we obtained in Table 3.14 was rather bad we thought necessary to perform significance tests to determine if it was at least better than what can be obtained by chance. We trained and tested 10'000 classifiers using the HDAC dataset with randomized class labels, and obtained the results displayed in Table 3.15.

Table 3.15: Classification efficiency in percents on the randomized dataset using PCA-LDA-CHARM-like.

Dataset	Mean	Standard Deviation	p-value
HDAC	7.05	0.88	0.0

Even though the mean classification accuracy displayed in Table 3.14 is quite poor, we observe that it is clearly significantly better than random. This result is encouraging as it informs us that while we cannot use our method to efficiently characterize profiles for the constructs against different HDAC, we are definitely able to significantly discriminate classes.

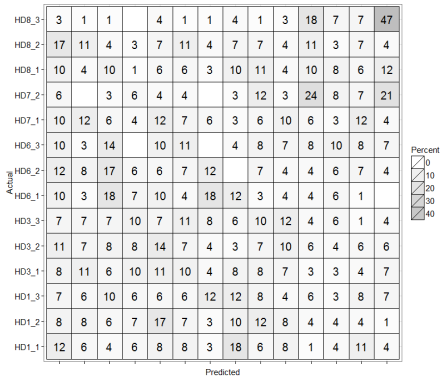
So far we handled images from the five fluorescent channels altogether. We performed classification experiment using only images from one channel at the time. Classification accuracies per channel are shown in Table 3.16. These results could help us determining if some channels contain more information to discriminate between classes than the others. This could be used to design experiments in such a way that only useful data are acquired.

Table 3.16: Classification efficiency in percents on the different channels using PCA-LDA-CHARM-like.

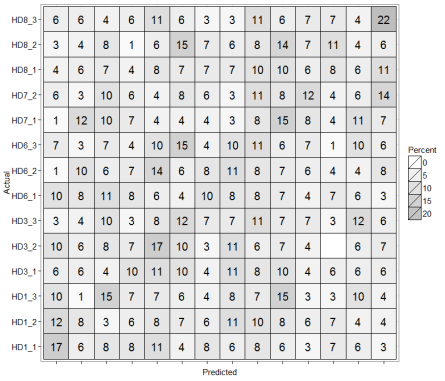
Dataset	Mean	Standard Deviation
Channel 1	13.60	0.41
Channel 2	12.28	0.49
Channel 3	10.35	0.49
Channel 4	13.02	0.56
Channel 5	10.40	0.33

We observe that results on each separate channel are worse than the multichannel result obtained in Table 3.14. Classification hence seems to benefit from the joint information of all channels together, as class separation is slightly easier when using information from every channel at once. No channel gave especially good results over the others. Confusion matrices for each channel can be found in Fig. 3.31.

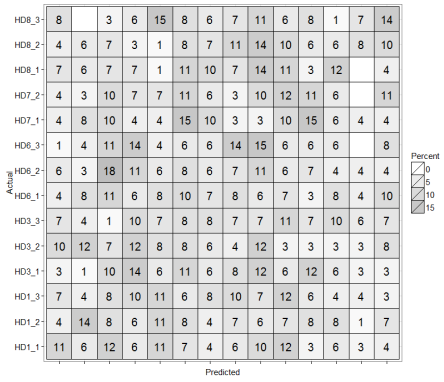
(a) Channel 1



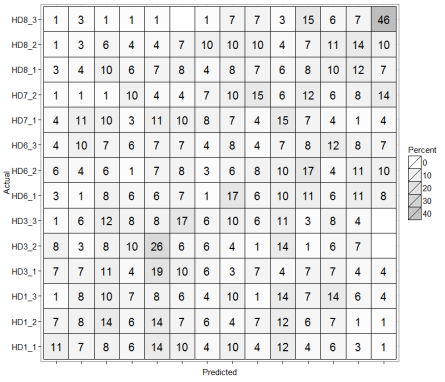
(b) Channel 2



(c) Channel 3



(d) Channel 4



(e) Channel 5

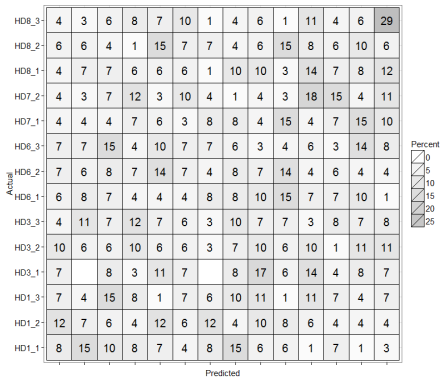


Figure 3.31: Typical confusion matrices per channel for the HDAC dataset.

All confusion matrices indeed look very much like Fig. 3.30: we observe strong confusion in every channel. Channel 2 and 3 are even confused to a more extreme level. It is interesting

to notice in 3.31a and 3.31d that the confusion seems to be less general at least for HDAC8_3 and HDAC7_2 in channel 1, and HDAC3_2 and HDAC8_3 in channel 4 which display a better classification rate than all other classes. These values were compared to the efficiencies of 10'000 random classifiers in Table 3.17 in order to test their significance over random results.

Table 3.17: Classification efficiency in percents on the different channels with randomized labels using PCA-LDA-CHARM-like.

Dataset	Mean	Standard Deviation	p-value
Channel 1	7.02	0.82	0.0
Channel 2	7.02	0.91	0.0
Channel 3	6.97	0.86	0.0
Channel 4	6.70	0.9	0.0
Channel 5	7.03	0.80	0.0

We observe the same situation as when we were using all channels together: classification efficiency on the separate channels is mediocre, but it is still nearly twofold better than what can be obtained by chance. We are therefore able to detect a significant effect of the different classes using channels separately.

As suggester earlier these poor results on the full dataset using either all channels together or separately could be strongly due to the fact different constructs against the same HDAC isoform are extremely difficult to distinguish. We gathered results on a simplified problem by ignoring the different constructs for same isoforms and classifying based on isoforms only. We were hence back to a five-class classification task where all constructs against one specific isoform were put together in one class. The classification result averaged on 10 runs using PCA-LDA-CHARM-like is presented on Table 3.18.

Table 3.18: Classification efficiency in percent on the entire HDAC dataset discriminating only between isoforms.

	Accuracy [%]	
Dataset	Mean	Standard Deviation
HDAC, isoforms only	35.00	0.66

While these results were again not very good, a notable improvement in classification efficiency is observed compared to Table 3.14. More development result from the observation of an example of confusion matrix for these data as shown in Fig. 3.32.

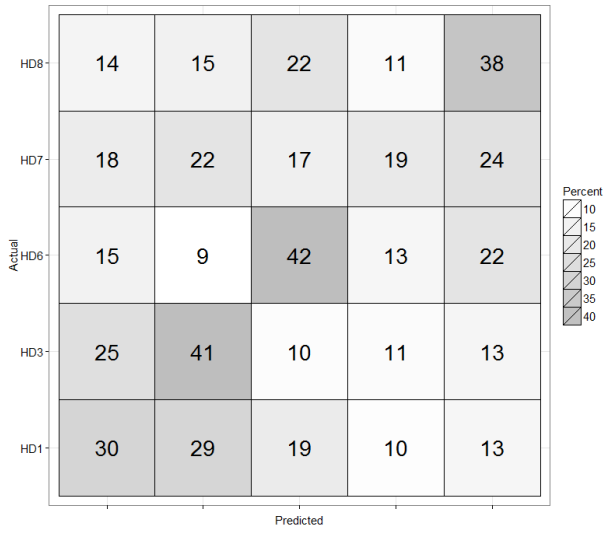


Figure 3.32: Typical confusion matrix for the HDAC dataset discriminating only between isoforms.

We see in particular that there is less global confusion for than in Fig. 3.30. HDAC6 and HDAC3 are better discriminated than in previous experiments. The same holds for HDAC8, although it already obtained quite good results for construct number 3 in Fig. 3.30. HDAC7 suffers from confusion with almost every class, and HDAC3 and HDAC1 seem to be often confused for one another. These results were nevertheless still insufficient as the best success rate we could get in this example was 42% properly classified, which is obviously not satisfying for a profiling experiment. Finally in Table 3.19 we again compared result from Table 3.18 to the efficiencies of 10'000 random classifiers in order to assess how better than random our performance was.

Table 3.19: Classification efficiency in percents on the randomized dataset using PCA-LDA-CHARM-like.

Dataset	Mean	Standard Deviation	p-value
HDAC, isoforms only	20.32	1.51	0.0

The estimated p-value of 0 indicates us that our approach clearly manages to classify HDAC isoforms using features information and does not assign class labels randomly. While our method might not be sufficient for identifying isoforms with an acceptable error rate (as observed with the result shown in Table 3.18), it is apparently able to identify differences between classes distributions.

As the HDAC dataset has primarily been analyzed using cell-based classification algorithms, we displayed for comparison a confusion matrix obtained using such kind of method in the context of the HDAC inhibitor study. Fig. 3.33 is an example of what could be obtained using

cell-based classification experiments specially designed for the HDAC dataset. Cells in images were segmented, and features were extracted from them.

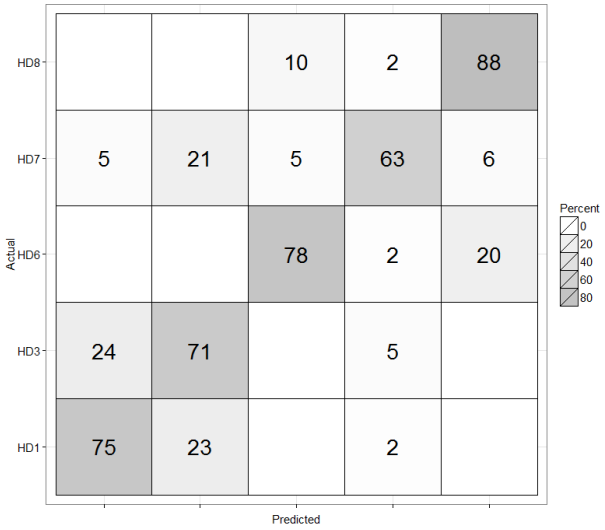
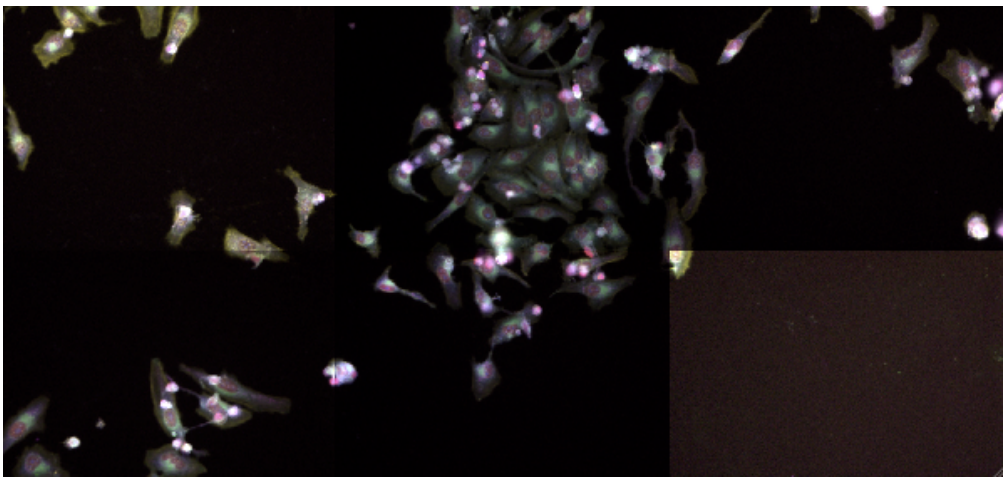


Figure 3.33: Typical confusion matrix for the HDAC dataset discriminating only between isoforms using cell-based classification. Source: Shantanu Singh, personal communication.

We observe than even though our results based on isoform classification (Table 3.18) are better than random as observed in Table 3.19, they are clearly not good enough and much poorer than the ones obtained using cell-based classification. We suspect our method suffers a lot in this dataset from "background issues". We displayed in Fig. 3.34 two patchworks made of 6 images from the HDAC dataset. What we want to point out with these images is that the number of cell present in the image greatly varies from patch to patch. As a result the bottom right patch in 3.34a or the two bottom right patches from 3.34b barely contain any cell or no cell at all while the others are more populated. As our algorithm captures every information in the image, it identifies patches containing only background as representatives of the class they were assigned to. Cell distribution therefore certainly affects our classification results as most of the features heterogeneity we capture probably comes from cell arrangement rather than from cell properties.

(a) Example 1



(b) Example 2

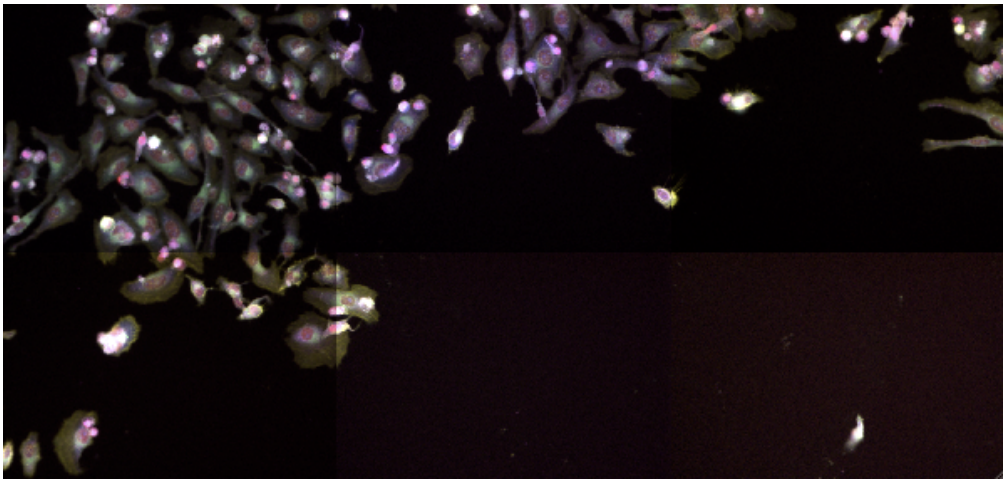


Figure 3.34: Illustration of the "background issue" encountered by PCA-LDA-CHARM-like on the HDAC dataset. Both examples are patchworks composed of 2x3 images from the HDAC dataset. Source: Shantanu Singh, personal communication.

Since our method was not convincing on the full dataset we focused on a subset of images composed of the first construct against HDAC 6 and HDAC 7 only. We selected these two isoforms as they appeared to be relatively well distinguishable in early results of the reference cell-based analysis carried out in this study. As plate effects were known to be important, we selected constructs HDAC6_1 and HDAC7_2 as constructs for these isoforms were located on the same plate. This setup brought us back to a two-class problem in which we thought we had better chances of success. We carried out the same experiment as with the full dataset: 10 runs of PCA-LDA-CHARM-like, summarized in Table 3.20.

Table 3.20: Classification efficiency in percent on HDAC 6 and 7 only.

	Accuracy [%]	
Dataset	Mean	Standard Deviation
HDAC 6-7	64.44	3.63

The mean result we obtained is obviously again not sufficient. Hardly more than one sample over two was properly classified. A representative confusion matrix for this problem is shown in Fig. 3.35 to help understanding this result.

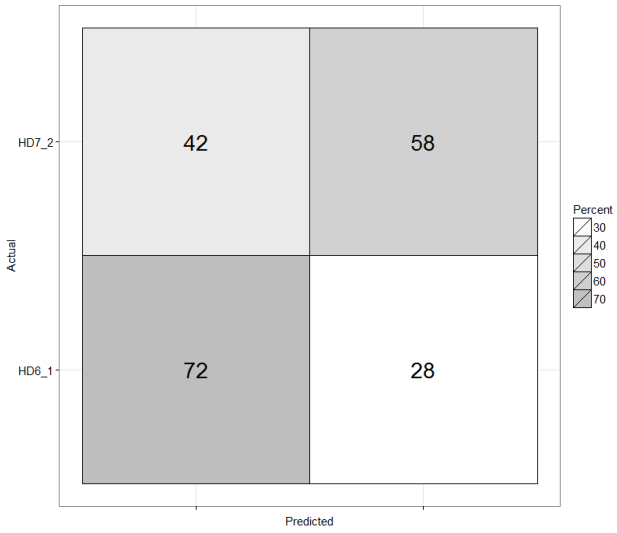


Figure 3.35: Typical confusion matrix for HDAC 7 and 8.

The prevalence of data in the diagonal is not very remarkable as the antidiagonal is highly populated as well. It seems that HDAC7 is more often confused for HDAC6 than the other way around. Our classifier globally seemed to have a hard time discriminating between these two classes. We again also performed significance tests with 10'000 classifiers to observe how our result differed from random. Results are presented in Table 3.21. We notice that even on this reduced dataset our method doesn't perform well enough. The p-value of 0.01 allows us to reject the hypothesis that our classifier is assigning class labels by chance only to a 5% confidence level. A per-channel analysis similar to the one performed on the whole dataset was also carried out using only HDAC 6 and 7. Classification results can be found in Table 3.22.

Table 3.21: Classification efficiency in percents on the randomized HDAC 6 and 7 using PCA-LDA-CHARM-like.

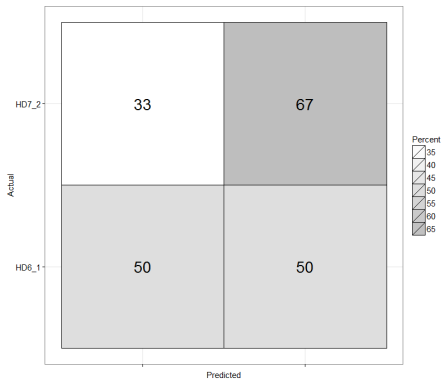
Dataset	Mean	Standard Deviation	p-value
HDAC 6-7	49.75	6.77	0.01

Table 3.22: Classification efficiency in percents on the different channels using PCA-LDA-CHARM-like on HDAC 6 and 7 only.

Dataset	Mean	Standard Deviation
Channel 1	59.44	3.45
Channel 2	50.42	3.05
Channel 3	50.55	4.90
Channel 4	46.53	4.08
Channel 5	48.75	4.91

As it was the case for the full dataset, the results obtained on each individual channel are poorer than the one we get when considering all channels together. Channels 1 however appears to perform slightly better than the others. We displayed one example of confusion matrix per channel in Fig. 3.37.

(a) Channel 1



(b) Channel 2

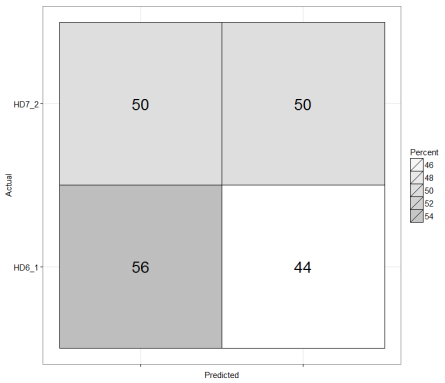
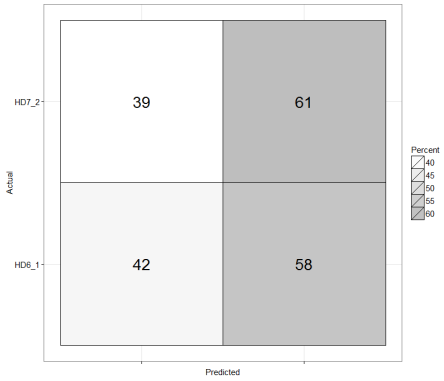
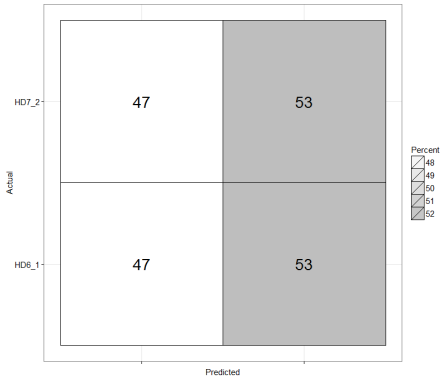


Figure 3.36: Typical confusion matrices per channel for HDAC 6 and 7.

(a) Channel 3



(b) Channel 4



(c) Channel 5

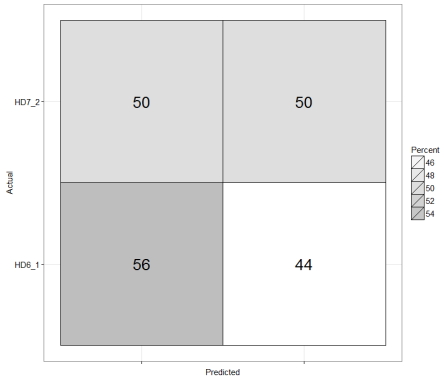


Figure 3.37: Typical confusion matrices per channel for HDAC 6 and 7.

The difference in performance between channel 1 and the others is less remarkable when looking at the confusion matrix. It is interesting to note that the patterns in 3.36a and 3.36b are similar: in the first case HDAC7 is well discriminated but HDAC6 is confused half of the time, and in the second it is the exact other way around . Channel 5 is in the exact same situation as channel 2. Results on channel 3 and 4 are very bad as HDAC6 is misclassified most of the time in both cases. The results of 10'000 random classifiers finally helped us determine the validity of the classification accuracy on each channel as shown in Table 3.23.

Table 3.23: Classification efficiency in percents on the different channels with randomized labels using PCA-LDA-CHARM-like.

Dataset	Mean	Standard Deviation	p-value
Channel 1	49.63	7.12	0.09
Channel 2	49.62	7.17	0.44
Channel 3	49.31	7.19	0.43
Channel 4	49.53	7.37	0.66
Channel 5	49.60	7.10	0.52

Classification appears to be strongly not significantly better than random on every channel. We cannot reject the random hypothesis for channel 1, but the p-value is closer to the threshold, which is not surprising as we observed in Table 3.22 that channel 1 obtained better results than the others. Classification per channel did therefore not really add value to our results.

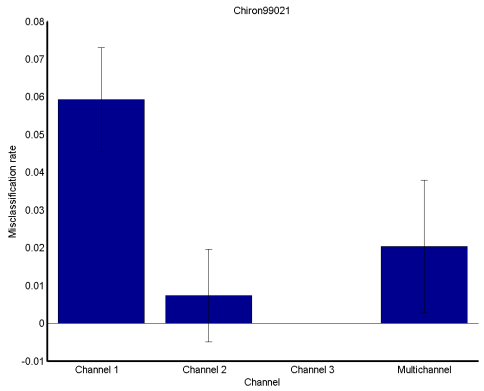
The results on the HDAC dataset are far from being as good as the ones obtained in the previous application examples. This however is not exceedingly surprising as these data are extremely challenging. The class characteristics we were looking for are not only very delicate, but we were also dealing with realistic cell images that have not been curated as the BBBC data were. A direct implication was that plate effects was potentially present. Such global bias is extremely detrimental to a whole-image based algorithm such as WND-CHARM or PCA-LDA-CHARM-like as they have no way to avoid capturing these global effects. We tried to classify our data using each channel separately, but this analysis did not yielded convincing result using the whole dataset. Best performances were achieved when simplifying the problem by bringing together different constructs for the same isoform and classifying HDAC isoforms only, but results were still too bad to use the classifier to identify profiles, and much worse than the reference results obtained on the same data using cell-based classification. Our experiment using only two HDAC isoforms, HDAC 6 and 7, yielded admissible accuracies using all channels together, but nothing of interest when analyzing channels separately. More work needs to be done in order to use our algorithm on data similar to the HDAC dataset. It is however difficult to make a global conclusion on the applicability of our method using results on these data as we recall that this dataset is part of an ongoing research.

3.3.4 Neuronal Outgrowth Datasets

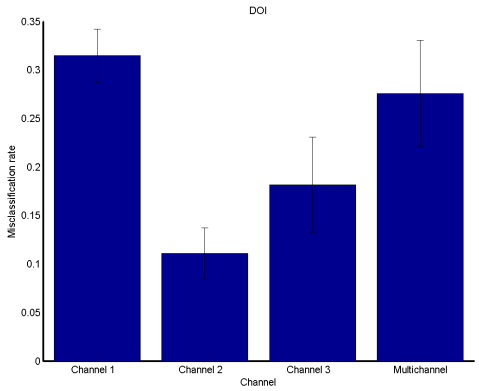
The Neuronal Outgrowth Datasets is a collection of data from an experiment looking at neuronal outgrowth when different chemicals are added to the neurons. The six different chemicals constitute the six datasets where in each case we aimed to classify positive versus negative phenotype (i.e. a two-class classification problem). Negative phenotype corresponded to the neuron in its natural state, and positive phenotype exhibited neuronal growth or inhibition of growth depending on the drug. For each image three fluorescence channels were acquired:

channel 1 detects cell nucleus, channel 2 dendrites (the shorter neurites), and channel 3 axons (the longer neurites). It was hence expected that most class difference would be determined using channels 2 and 3 as the effect of the drug should mostly affect neurites rather than cell nucleus.To investigate the discrimination power of these different channels we performed classification experiments using PCA-LDA-CHARM-like. Results averaged over 10 classification runs are presented in Fig. 3.38.

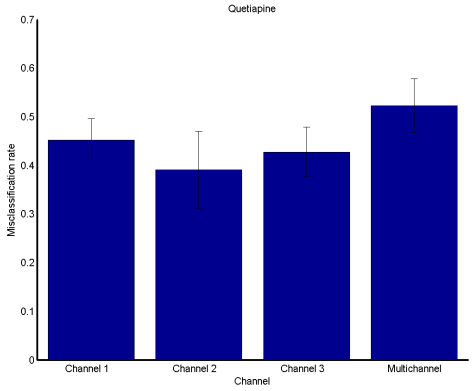
(a) Chiron99021



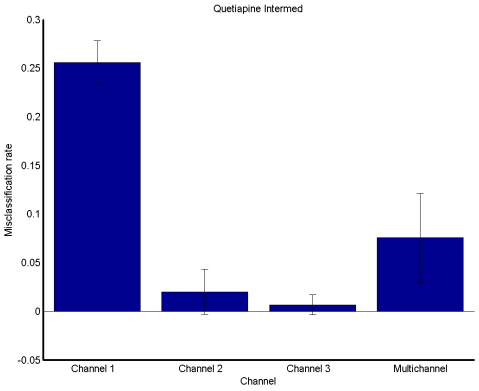
(b) DOI



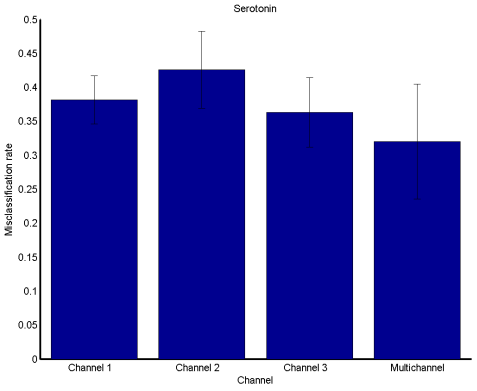
(c) Quetiapine



(d) Quetiapine Intermed



(e) Serotonin



(f) TDZD-8

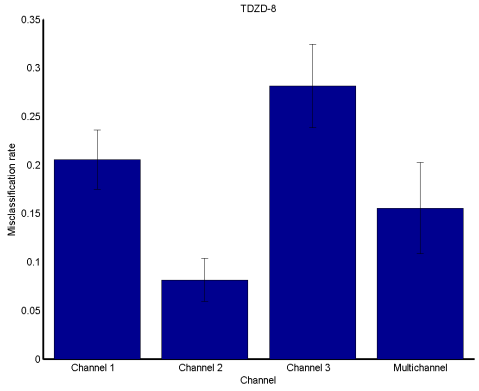


Figure 3.38: Mislassification rates using images from different channels of the Neuronal Outgrowth datasets. Results range from 0 (0%) to 1 (100%).

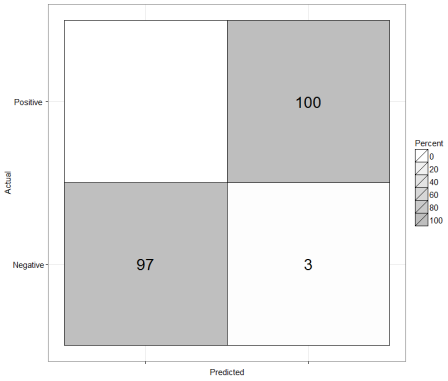
We notice that the perfect score is hit using channel 3 only in 3.38a, but we strongly suspect this to be a faulty result. As it can be observed in Fig. 2.12a, the third channel displayed in green is extremely noisy for samples of the positive phenotype. All images from this class are affected by noise, and we therefore suspect that the good performance obtained on channel 3 in 3.38a is due mostly to the fact that class discrimination is achieved by detecting the presence of noise rather than by looking at actual phenotypical differences. This is one of the drawbacks encountered when using whole-image based methods, as we already observed in the case of the HDAC dataset. Selecting channel 2 seems to benefit classification in the cases of 3.38b and 3.38f, while both 2 and 3 appear to yield better results than either channel 1 or multichannel on 3.38d. 3.38e and 3.38c display rather uniform results regardless of the channel used for classification. The global trend seem to be that channels 2 and 3 contain more information than channel 1, which was expected as we are looking for effect on the neurites and not necessarily on cell nucleus. The multichannel analysis did not appear to give the worst result in any case so it is probably not a bad option. We finally decided to perform analysis using all channels together as no single channel was found to yield significantly better results on all the datasets. Classification results for the multichannel analysis using PCA-LDA-CHARM-like are presented in Table 3.24.

Table 3.24: Classification efficiency in percent on the Neuronal Outgrowth datasets.

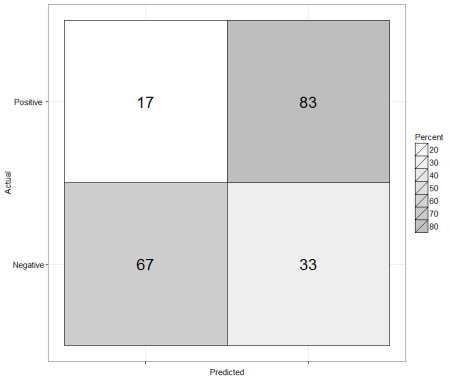
	Accuracy [%]	
Dataset	Mean	Standard Deviation
Chiron99021	97.96	1.75
DOI	72.41	5.46
Quetiapine	47.74	5.54
Quetiapine Intermed	92.44	4.57
Serotonin	67.96	8.45
TDZD-8	84.44	4.70

A wide range of results was obtained on these different situations. Classification on Chiron99021 and Quetiapine Intermed appear to yield very good accuracies with reasonable variance, but we remain suspicious about the results on Chiron99021 because of the noise present in samples from the positive phenotype. The performance obtained on Quetiapine Intermed is more likely to really represent an ability of our method to detect differences in cell phenotype. Results on DOI, TDZD-8 and Serotonin are less convincing but not catastrophic. What we obtained on Quetiapine is however much more concerning as the result looks completely random. These observations can be completed by looking at representative confusion matrices for each of the dataset, presented in Fig. 3.39.

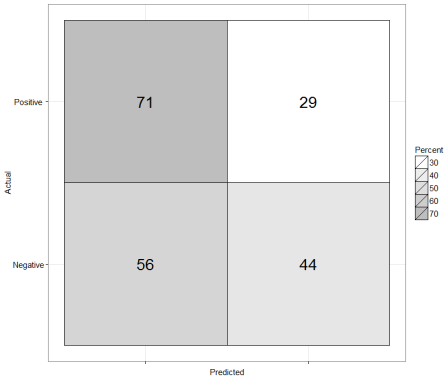
(a) Chiron99021



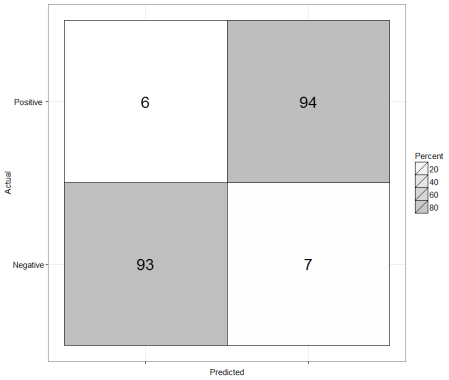
(b) DOI



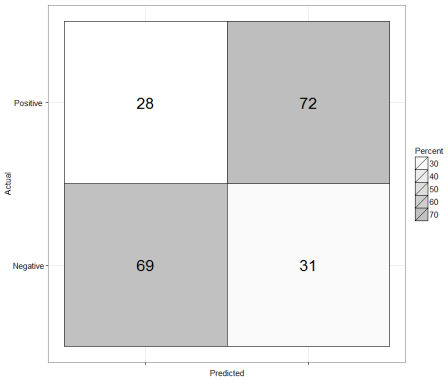
(c) Quetiapine



(d) Quetiapine Intermed



(e) Serotonin



(f) TDZD-8

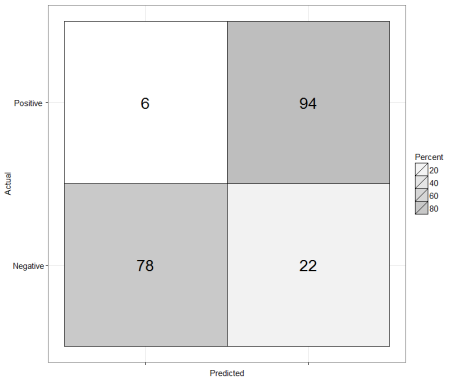


Figure 3.39: Typical confusion matrices for the Neuronal Outgrowth datasets.

The good results on Chiron99021 and Quetiapine Intermed are confirmed by the look of the confusion matrices 3.39a and 3.39d. Very good discrimination is achieved for each class in these two cases, but caution should still be taken when interpreting the good performances of Chiron99021. The situation is less favourable for DOI, Serotonin and TDZD-8 as we observe in 3.39b, 3.39e and 3.39f that in all three cases the negative phenotype is too often mistaken for the positive phenotype. The positive phenotype however appears to cause less problem and be less often misclassified. The random-looking result we obtained on Quetiapine is confirmed by 3.39c: the confusion matrix looks totally scrambled and for the positive phenotype a majority of elements are misclassified. The negative phenotype does not get much better results and is mostly confused as well. We randomized class identifiers in the training set on all the neurons datasets and executed 10'000 classification runs. The estimation of the efficiency distribution that can be obtained by chance helped us measuring the validity of the results from Table 3.24 by computing a p-value for each set, shown in Table 3.25.

Table 3.25: Classification efficiency in percents on randomized Neuronal Outgrowth datasets using PCA-LDA-CHARM-like.

Dataset	Mean	Standard Deviation	p-value
Chiron99021	51.86	7.87	0.0
DOI	51.11	7.67	0.0
Quetiapine	51.34	7.65	0.67
Quetiapine Intermed	50.10	8.57	0.0
Serotonin	51.53	7.75	0.02
TDZD-8	51.58	7.71	0.0

Classification on Quetiapine clearly does not appear to be significantly different than random as shown by its large p-value of 67%, which confirms the observation made on results from Table 3.24. Results on Serotonin are mediocre: with a p-value of 2%, classification on this set is better than what can be obtained by chance at 5% significance level, but not at 1%. In all other cases the situation is more reassuring as the p-value is lower than 0.01 or clearly equal to zero. We therefore managed to classify efficiently 4 out of the 6 datasets.

Results obtained on these datasets were globally favorable. It is important to recall that the effect on neuronal outgrowth is known to be subtle for some chemicals, making this problem much harder than the BBBC sets. The less good results obtained on Serotonin, TDZD-8 and DOI could hence indicate that the effect of these drugs on the cells is lighter than in the case of Quetiapine Intermed. We prefer remaining suspicious about the good accuracy obtained on Chiron99021 as it is likely to stem mostly out of the difference in data quality for the two classes. In spite of this PCA-LDA-CHARM-like managed to produce excellent results on one sets and good efficiencies on three others where image quality was observed to be more uniform across classes. It only appeared to be totally inefficient in one case.

4 Discussion

The analysis we performed on WND-CHARM gave us many information on the functioning of this classification algorithm. We were first able to reproduce the results previously obtained by Goldberg in [38] with the existing C++ implementation. As expected from the conclusions drawn in the original paper, the results were very good on all reference datasets. It served us as a reference to validate our implementations of WND-CHARM in Python and MATLAB. We also made sure our results were significantly better than random by performing "significance tests" where we shuffled class labels and classified each randomized dataset, thereby obtaining a measure of the random classification accuracy of WND-CHARM.

We then investigated in details the various steps of the algorithm. Our first major critics against WND-CHARM was the fact that it looks a bit too "brute force": the features vector seemed to be composed of as many feature as available image processing operations can extract out of an image, without explaining their presence. As a result the feature vector was highly complete and offered ways to discriminate properly many different sets of images, but its construction was hard to justify. Our analysis on features extraction followed several axis: we investigated the importance of the different features groups and features "levels" composing the CHARM vector. After reaching the conclusion that the construction of the vector was pertinent, we implemented several versions of a "CHARM-like" vector starting with a very small subset of features and iteratively adding more elements. Our goal in doing so was to design a features vector that we could defend with classification results showing that it was composed of a judicious subset of features that obtain good classification efficiency. Our iterative implementation helped us making sure we were not unnecessarily adding elements that would not improve results. In parallel we digressed on the features extracted on transforms of transforms as we wanted to better understand these complex features. We suggested these higher level features were only helping in classification by adding more dimensions to the features space, and tested this hypothesis through several experiments using linear and RBF-SVM on basic and higher levels features. Finally, we tried to mimic the effect of complex features by completing the vector composed of simple features only with random Fourier features. With these investigations we did not manage to fully grasp to which aspects of

the image compound transform features corresponded, but our results were nevertheless helpful as they allowed us concluding that these measurement were actually capturing some information in the image rather than simply adding random dimensions. To close our analysis related to features, we dedicated a part of our work to the analysis of the features selection process in WND-CHARM. We analyzed the content and stability of the "top 15%" features subset, a kind of analysis that has not been done before. We observed that the behavior of the selection process was difficult to understand and that the weights values used for classification did not seem to be stable. The second step of our analysis was dedicated to the classification method. After carrying on some parameter exploration experiments to get the best out of k-NN and PCA-LDA, we classified the CHARM vector using these two methods and compared results with WND. Classification using PCA-LDA yielded promising results that either outperformed or compared with the reference. Finally we did some research around the validation method. We tried to replace the "save 25%" validation method used in WND-CHARM by the standard 10-fold cross-validation, or by a k-fold cross-validation with a variable k parameter. The latter experiment did not obtain good results, therefore we dropped this idea. We observed that the 10-fold and "save 25%" validation methods yielded more or less equivalent classification accuracies, with the "save 25%" being slightly better but having a much larger variance, which confirmed our suspicions. We concluded that this latter method tends to overestimate classification efficiency and produce highly variable results across classification runs. For this reason we thought the use of 10-fold cross-validation would be preferable.

These observations helped us designing our own algorithm, "PCA-LDA-CHARM-like". We validated it against WND-CHARM on the reference datasets and on the IICBU collection. It obtained results competing with WND-CHARM's on both of these sources, but we thought it superior in various aspects. We conserved the idea of using groups of image-based features extracted from the original and transformed image, but the way we built our CHARM-like vector by iteratively adding features and observing results makes it more legitimate. We chose PCA for features selection, which brings many advantages over the weighting and subsequent thresholding present in WND-CHARM. First, the method is more reliable and commonly accepted, and then the parameters it requires can be related to the data distribution while the threshold value in WND-CHARM is purely empirical. Last but not least, the effect of thresholding was that only 15% of the features were actually used for classification, while about a thousand of them were computed. In this way most of the time required to execute WND-CHARM was dedicated to the computation of measurements that would never be used. When transforming features space using PCA, the resulting principal components are linear combinations of the features. As a result even if only a little number of principal components are kept, they are composed of a contribution of every features in the vector. The classification method we chose, LDA, also had the advantage of being more standard and trusted than WND. Finally, we left the option for both the "save 25%" and the 10-fold cross-validation method, but selected 10-fold as the default choice as this method appeared to yield more stable and reliable results. PCA-LDA-CHARM-like was designed to be user-friendly as intended.

Features extraction was therefore designed as a CellProfiler pipeline and classification was implemented in Python to make its integration into CellProfiler or related software easier. Finally our implementation choice also provides another advantage over the C++ version of WND-CHARM: our features extraction pipeline is modular and easily modifiable. The direct implication of this is an easier access to intermediate results and an enhanced possibility to perform further analysis. Extracting the CHARM vector out of the C++ implementation was not an easy task as this program was designed to be used as a black box. Our pipeline is on the contrary made to be openly editable: modules can be added or removed, and every parameter can be easily tweaked. In this way our CHARM-like vector is not a fixed entity: it can easily evolve and be modified by the user to fine tune results on a particular analysis.

Just as we did with WND-CHARM, we performed a brief analysis on our algorithm to identify possible ways of improving it in the future. On the feature vector side we observed that one of our earlier versions seemed to allow us obtaining sufficiently good results. As this vector is composed of less elements it requires a smaller execution time, which could be an advantage, but it also means that it captures less aspects of the image, which is an inconvenient in some cases. As the features that differ from these two versions are the higher level features we mentioned earlier, we decided to be cautious with this result. Future work focusing on the nature of these compound transforms features could help us determine how we are affecting the classifier's ability to characterize images by removing them. If we understand which aspect of the image these features are capturing we could predict on which kind of data a vector that does not contain them would fail. In this way we could design several versions our algorithm to be used depending on the nature of the images: a "lighter" one for cases where the compound transforms are not needed, and a full one for other applications. We also tried modifying the classification scheme for more refined methods. Penalized LDA did not produced better results than PCA-LDA. A similar situation was found when choosing LDA with other dimension reduction methods as a pre-processing step. We noticed that the use of SVM or Random Forests gave us excellent results, but as trade-off a prohibitively long execution time as well as large resources requirements for multiclass problems when using SVM which made it unusable on two of the reference sets. Random Forests and SVM therefore seemed to be promising alternatives, but in the latter case further work is needed to address time and memory requirements for this method if we aim to conserve the ability of the algorithm to be used in a wide range of situations. PCA-LDA still appears to us to be the best compromise between results and usability.

WND-CHARM was defined as a multi-purpose algorithm, but as we implemented it to be part of CellProfiler, creating an efficient classifier especially for biological images was of utmost importance. For this reason we dedicated the rest of our work in gathering images from various actual biological problems on which we could test our algorithm. We first used four of the BBBC datasets which are typical exampled of high-throughput cell-based assay images. We obtained excellent classification performances on these data. We also showed that the features we added to CellProfiler were useful in a whole image-based classification experiment like the one we performed as they improved the best Z' factor we could obtain on these data.

Our second application case concerned the identification of immunostaining patterns in human tissue images. This experiment was a success: we obtained good classification accuracies, and were able to identify the difficulties of this dataset by looking at typical confusion matrices. This set is in our opinion a perfect example of a case where our method fully exhibits its potential. As information is contained everywhere on tissue image data and little or no background is present the analysis can truly benefit from the construction of our CHARM-like vector. These images are also less prone to plate or population effects as it is the case in images featuring cell colonies. When present these global effects have a large risk of being captured by our whole-image feature extraction scheme, hence affecting classification in a bad way. This kind of situation would be an example of the limits of a WND-CHARM-like algorithm: as we extract information from the whole image our basic assumption is that the effects we measure are solely due to classes difference and do not come from external sources. This supposition might often not be reasonable for high-throughput experiments.

We tested our algorithm on a dataset composed of high-throughput cell-based images where the task was to identify the effect of different siRNA constructs targeted against various isoforms of Histone Deacetylase (HDAC) proteins. These data were expected to present a difficult classification problem because of the faint distinction between classes composed of different isoforms and constructs. The first results we obtained were indeed very bad. Since our data contained information from five different fluorescence channels, we then performed classification extracting features from each channel at the time. We noticed that we did not manage to obtain better results when focusing on certain channels than when considering them all together. This kind of analysis could be interesting to feedback to the experimental design: if it is observed that certain channels allow obtaining better results, the next set of images could be acquired using only the channels of interest, thus simplifying the imaging procedure. We also tried to simplify the problem by considering only different HDAC isoforms and ignoring the differences between constructs. This experiment gave us better performance, but the result was still outside the admissible range for a practical application. We also observed that our result on isoforms classification was much worse than what could be obtained using cell-based classification. We could attribute our bad results to the intrinsic complexity of the classification task, but the comparison with results obtained using per-cell features tend to indicate that our problem rather lies in the fact that cell-based information is truly important in these data. Finally we selected two HDAC isoforms and worked on this reduced subset as a further simplification of the problem. Mediocre results were obtained when using all channels together, and no better performance were obtained when considering channels separately. The results we obtain on the HDAC data give us indications on the relative strength of our method depending on the task to be performed. In the context of a profiling experiment like the full HDAC study where the goal is to identify signatures for the different isoforms, our whole-image based method appears to be inefficient probably because of a lack of sensitivity for small class differences, or due to a too large susceptibility to capture background information. However if the task is a simple classification experiment where we aimed to measure whether there is a statistically significant effect between different isoforms, or in other words

to identify if there is a drift between classes, our method could be helpful as we observed that we were able to detect an effect as our results were significantly better than random in most experiments. On these data segmentation-based classification will probably most often give better performance than whole-image based methods as per cell features seem to be very important. The reason for this can be found both in the fact that the patterns we are looking for might be so subtle they tend to be buried in background noise when the whole image is considered, and in the presence of global effects that are unavoidably captured by whole-image algorithms. This does not imply that our method is useless with cell-based data: the main point is to establish a trade-off between the effort needed for pre-processing and the desired results. If segmentation is extremely hard or impossible, a whole-image based classifier like the one we proposed can still be used to help classifying data, keeping in mind that the results might be only tolerable. While we do not recommend it for profiling experiments, our algorithm could still be used when it comes to a classification problem in which the task is to look for a statistically significant effect between classes.

Our last experiment concerned a set of images from neuronal outgrowth experiments. The different datasets corresponded to neurons treated with different drugs that act on neurite growth either by inhibiting or enhancing it. In every dataset the goal was to distinguish between phenotypes in the absence (negative, concentration of zero) and presence (positive, maximal concentration) of the drug of interest. We obtained varied results depending on the dataset: in a third of the cases the results were very good and in half of the datasets classification efficiency was satisfactory. In one dataset our algorithm appeared to be inefficient and was found to perform no better than a random classifier. One of the dataset which achieved excellent performances presented a large difference in image quality between the positive and negative phenotype class. We suggested that this good result should not be taken into account as it probably reflects the fact our algorithm is classifying based on image quality rather than on differences in cell phenotype between classes. Image quality was more conserved between classes in the others datasets, allowing us to be more confident on our results. We still recall that the images we were using we drawn from a single plate, and available positive control samples were known to be of quite poor quality. Better (and more reliable) results could therefore probably be achieved using images of better positive controls and more uniform image quality. In this setting it was indeed hard for us to determine if the acceptable results we got on half of the datasets truly represented the best performance we could obtain with our method in this problem, or if they were impacted by the bad quality of the data. Working with images of different quality depending on the class also caused us problems when interpreting the results as we couldn't be sure that our whole-image based classifier was actually capturing differences in phenotype rather than physical properties of the image. In any case as these data present a more complicated problem than the BBBC, we are quite satisfied by the results. Moreover, our results compare with what would be expected from cell-based classification: as the problem is subtle no excellent performances are obtained, but in most cases we are definitely able to identify an effect between the different classes. Whole-image based classification is very interesting in the particular case of such kind of neuronal data as in the very diverse

amount of neuron projects a common challenging part is typically the segmentation of the neurites. A complete and correct separation of the neuronal extensions from the background and subsequent identification of the corresponding cell body can be very difficult, but is needed in order to quantify the effect of a specific drug on neuronal outgrowth. Gathering features on the whole image could hence give a way to detect drug effect without requiring any segmentation.

PCA-LDA-CHARM-like therefore fulfills our initial goals: it performs at least as well as the algorithm that inspired it, it is designed in a neater way, and most of all it was observed to be efficient in different biological classification problems. Except on the HDAC dataset, we managed to achieve satisfying results while circumventing segmentation. This algorithm could certainly be further improved in several ways, but we are quite satisfied with its current form. Before modifying it further we think it is important to assess its ability to generalize by applying it to real problems.

As a closing remark we find important to discuss a potential pitfall of whole-image based algorithms as WND-CHARM and the one we propose. The main idea of these methods is to capture information on the whole image, which circumvent a possibly cumbersome segmentation step , but the direct result is that every effect present on the image is captured, even if it is not actual information. The resulting risk is a faulty analysis of the data: a classifier could for example perform very well in identifying different classes merely because the samples from each class were located in a different plate. The whole-image based algorithm would only capture plate effect, and would therefore fail when presented with images from the same classes but plated on a different device. In order words the problem with image-based classification comes from the fact we cannot be sure the difference in feature value we observe between classes is caused by the effect we want to detect. This risk is much more present when dealing with images of cell populations than for instance with tissue images. For this reason we think special attention should be dedicated to the analysis of classification results on cell-based experiments. Image-based algorithms can be used with more confidence on images where population or hardware effects are less likely to be influential and where image quality if constant over all images. However we would like to emphasize that there might be some cases in which this pitfall becomes a real strength over cell-based methods. In certain cell-based assays cells distribution is of extreme importance: the phenotype to be detected is sometimes determined by the way cells form colonies. A segmentation-based algorithm would be unable to capture such effect, while PCA-LDA-CHARM-like would be able to use this information. It is therefore important to understand the implication of whole-image based classification in order to avoid erroneous conclusions, but also to be able to fully benefit from its advantages.

5 Conclusion

We achieved several goals throughout this work. Our analysis of WND-CHARM helped us gain understanding in various fields of machine-learning theory. We investigated many aspects of WND-CHARM and observed the way it functioned at a refined level. The analysis we performed on the features extracted from compound transforms, or higher-level features, did not give us conclusive explanations on what these measurements mean at the image level, but it nevertheless helped us demonstrating their importance through our inability to mimic their effect using random Fourier features. Analyzing the meaning of the compound transforms features could be the subject for more research on the image processing side, as it is not immediately linked to biological applications. Overall we wanted to design our algorithm in a "wise" way: rather than simply re-implementing WND-CHARM, we wanted to create something we could justify and defend. We think we reached this goal as our PCA-LDA-CHARM-like was built using observations made on results of our WND-CHARM analysis. One of our major concerns was also to make the whole algorithm user-friendly. For this reason we built the whole features extraction process as a CellProfiler pipeline file which can be easily loaded and used. The classification part is at present state performed in a separate script, but it has been coded in Python to facilitate its integration into CellProfiler or another software from the CellProfiler suite (CellProfiler Analyst for instance). Most importantly we wanted the algorithm we designed to be useful. Making sure it allowed us obtaining results that competed with WND-CHARM was a first step, but we think the good results we obtained on the various real-life examples datasets make the most important point. A new method could be as nicely designed as possible, its actual value would always be measured through asking "does it work?". By testing our algorithm on several kind of biological data, we gave elements of answer to this question.

The fate of the classification part of our algorithm that now consists of an independent Python script is an open question. This issue is debated with the Imaging Platform and still needs to be settled. Future work could also be made to further improve our PCA-LDA-CHARM-like as we suggested in the discussion, and to yield a version 2.0 of the algorithm. This improvement process could be iterated at will to try to get the most of the various machine-

learning techniques available. Finally, we think the most important improvements could be made by asking for feedback from potential users. As this whole method was designed both in its substance and format to be practical and easy to use, taking into account the critics and comments of users to increase the algorithm's reach for the public would be of capital importance.

We hope this project gave a demonstration of the power of whole image-based classification, and most of all we raise the expectation that this work will contribute to make this technique more available to the biomedical research community, thereby providing new analysis tools.

A CellProfiler Modules Descriptions

A.1 Pre-existing Modules

Our CHARM-like vector is composed of measurements coming both from pre-existing and newly implemented CellProfiler modules. We only describe here the parameters we used for modules available in the current release, and invite the reader to consult CellProfiler online manual ([12]) for a description of the theory behind measurements extracted from each module.

A.1.1 ApplyThreshold

Our pipeline features three instances of the ApplyThreshold module with the following parameters:

- Otsu Global Thresholding, two classes, weighted variance minimized, binary output image. All other parameters are default.
- Otsu Global Thresholding, three classes, weighted variance minimized, pixels in the middle intensity class assigned to foreground, binary output image. All other parameters are default.
- Otsu Global Thresholding, three classes, weighted variance minimized, pixels in the middle intensity class assigned to background, binary output image. All other parameters are default.

A.1.2 EnhanceEdges

We used the EnhanceEdges modules with the original image as input and applied the Prewitt edge-finding method with all edge direction enhanced.

A.1.3 MeasureImageQuality

All parameters were kept to their default value in MeasureImageQuality, and one image list was added for each image we created (original, edge-filtered, three thresholded, three transformed and two compound-transformed images, a total of 10). Measurements of interests were subsequently selected in the ExportToSpreadsheet module. The mean and standard deviation of image intensity were extracted from all the 10 images. Maximal intensity was selected for all images except the three thresholding results, and the percentages of pixels with the maximal and minimal values in the image were computed for all images except the thresholded and edge-filtered ones.

A.1.4 MeasureTexture

Textures were computed on the original, transformed and compound-transformed images. For all of these, textures with scales 3 and 4 pixels were measured. Haralick textures features were computed in every possible direction (horizontal, vertical, diagonal and anti-diagonal). Finally, Gabor features were extracted using the default value (four angles).

A.2 New Modules

This section contains a description of the modules we created during this project. We explain the theory behind each measurement, and specify how it was actually implemented.

A.2.1 Histograms

Our Histograms module computes histograms with different numbers of bins of the image and returns the amount of pixels falling into each bin. The histogram-binning process is performed using the built-in `histogram` numpy ([37]) function and normalized in $[0,1]$:

```
import numpy as np
bins, edges = np.histogram(image, bins=b)
bins=np.array(bins, dtype=float)
bins=bins/np.max(bins)
```

`histogram` takes as input an image and a parameter `b`. `b` can either be a scalar indicating the number of bins, or an array indicating histogram bins edges. The function returns `bins`, a 1xb array containing the number of elements in each bins, and `edges`, a 1x(b+1) array with the values of bins edges.

In its current state the module computes 3, 5, 7 and 9-bins histograms of the image. We aim to eventually have the number of bins as a parameter that can be edited from CellProfiler's interface.

A.2.2 Moments

The Moments module was designed to easily extract moments statistics from an input image. It computes the first four moments about the mean.

Mean The first moment is simply computed using `mean` from numpy ([37]):

```
import numpy as np
mean=np.mean(pixels)
```

Standard Deviation Similarly the standard deviation, or second moment, is extracted with numpy's `std`:

```
import numpy as np
mean=np.std(pixels, ddof=1)
```

We set `ddof=1` in order to obtain the unbiased estimator of the standard deviation.

Skewness Skewness is the third moment defined as Eq. A.1 where μ is the mean and N the total number of elements, which in the case of an image corresponds to width times height. It measures the asymmetry of a distribution. A positive skewness indicates that the right tail is longer, i.e. that there are few high values, while a negative value means the opposite, i.e. that there are few low values. If the mean of the distribution is equal to its median, the skewness drops to zero.

$$Skewness = \frac{\frac{\sum_{i=1}^{N}(x_i-\mu)^3}{N}}{\left(\frac{\sum_{i=1}^{N}(x_i-\mu)^2}{N}\right)^{\frac{3}{2}}} \tag{A.1}$$

We strictly followed this definition in our implementation and used again `std` as skewness can be expressed using the biased estimator of the variance.

Kurtosis The fourth moment, kurtosis, is measured following Eq. A.2, using the same nomenclature as in A.1. This metric describes the morphology of peaks in the distribution. When the peak is acute around the mean and the distribution has large tails, kurtosis obtains a positive value. On the other hand when the peak is broader and the tails thin, kurtosis is negative.

$$Kurtosis = \frac{\frac{\sum_{i=1}^{N}(x_i-\mu)^4}{N}}{\left(\frac{\sum_{i=1}^{N}(x_i-\mu)^2}{N}\right)^{2}} \tag{A.2}$$

As it was the case for skewness, we implemented kurtosis following Eq. A.2 and used numpy's `std`.

A.2.3 Tamura

The aim of Tamura texture features is to propose a set of textural features that would correspond to *human visual perception*. The author's goal was to create texture features that would be *visually obvious* unlike the Haralick features for instance ([52]). Textures are properties of a region or a subimage (or macroscopic region), not of an isolated point. A texture is defined t as $t = R(e)$, where R is a *placement rule* and e an *element* ([52]). It is therefore a repetitive pattern of *elements* organized according to the *placement rule*. In natural textures, R and e are unknown and very variable, hence a statistical approach was suggested for local texture measurement. Tamura et al. proposed six features (Coarseness, Contrast, Directionality, Line-likeness, Roughness and Regularity) out of which only three were shown to be efficient: Coarseness, Contrast and Directionality. Our implementation was designed using both Tamura's original paper ([52]) and WND-CHARM's C++ code ([57]).

Coarseness

Coarseness is defined against fineness. When two images are composed of the same pattern but one is a scaled version of the other, the bigger ("magnified") version is coarser. In the situation where images are composed of different patterns, the bigger the size of the element composing the structure of the pattern and/or the less number of elements repetition are present, the coarser is appears to be. The idea is to use various-size operators to adapt the size of the area on which the measure is taken to the amount of coarseness of the texture, i.e. a large size is considered when dealing with a coarse texture, and a small one when a fine texture is present - we sort of measure the size of the texture element. The measure of coarseness is obtained through four steps.

Step 1: The averages of pixels intensities in regions of sizes $2^k \times 2^k$ are computed. The authors suggest using k=1...6. The average value over a region of size $2^k \times 2^k$ is defined as Eq. A.3 for each point (x,y) in the image. H, W are the height and the width of the image respectively, and I(i,j) is the intensity of the pixel (i,j) in image I.

$$A_k(x,y) = \sum_{i=x-2^{k-1}}^{x+2^{k-1}-1} \sum_{j=y-2^{k-1}}^{y+2^{k-1}-1} \frac{I(i,j)}{2^{2k}} \tag{A.3}$$

Step 2: For each position (x,y), *differences between pairs of average corresponding to pairs of non-overlapping neighborhoods just on opposite sides of the point in both horizontal and vertical orientations* ([52]) are computed. This amounts to compute Eq. A.4 (horizontal case)

and A.5 (vertical case).

$$E_{k,h}(x,y) = |A_k(x+2^{k-1},y) - A_k(x-2^{k-1},y)| \tag{A.4}$$

$$E_{k,v}(x,y) = |A_k(x,y+2^{k-1}) - A_k(x,y-2^{k-1})| \tag{A.5}$$

Step 3: For each position (x,y), S_{best} is defined as the size giving the highest value of E in either direction (horizontal or vertical), i.e. Eq. A.6 where k is such that $E_k = E_{max} = max(E_{1,h}, E_{1,v}, ..., E_{6,h}, E_{6,v})$.

$$S_{best}(x,y) = 2^k \tag{A.6}$$

Step 4: The measure of coarseness is finally defined as the average of S_{best} over the whole image as shown in Eq. A.7. If $E_{max} = E_i = E_j$ with $i \neq j$, then $E_{max} = E_{max\{i,j\}}$.

$$F_{coars} = \frac{\sum_{x=0}^{W}\sum_{y=0}^{H} S_{best}(x,y)}{H \times W} \tag{A.7}$$

In WND-CHARM's version, a matrix representing at each pixel the cumulative sum of all preceding pixels in the image (according to a certain scheme) is first computed such that the subsequent averages over regions of different size will be easily computed by subtracting boundary pixels and dividing by the size of the region. Steps 1 and 2 are computed using this cumulative sum. A and the two E are computed in two separate steps as suggested in [52]. Step 3 differs a little bit from the reference: S_{best} is defined as $S_{best} = k$, where k is the power of two yielding the highest value of E instead of 2^k. It is suggested in [23] to use k instead of 2^k since it is unlikely for textures to have a coarseness of 32 pixels. At this size the algorithm is hence mostly expected to detect noise. Using k instead of 2^k introduces a logarithmic scaling of the coarseness and lessens the influence of large scales, thus potentially reducing noise in the average measurement. This argument might however not be valid for biological data, where some textures can have a very wide range of coarseness values. The final measure of coarseness also differs from the original one. It is computed as Eq. A.8.

$$F_{coars} = \frac{\sum_{x=0}^{W}\sum_{y=0}^{H} S_{best}(x,y)}{(H-32) \times (W-32)} \tag{A.8}$$

In this implementation the measure of coarseness F_{coars} is returned, as well as a 3-bins histogram of S_{best} over the image. We implemented the two versions of coarseness as described in [52] and Eq. A.8, and we also included the 3-bins histogram of S_{best} in our module. All these elements are outputted from the module.

Contrast

Contrast is defined as high or low. It is composed of four factors: the dynamic range of grey-levels in the image, the polarization of the grey-levels histogram (or ratio between black and white areas), the sharpness of image edges, and the period of repeating patterns. The two first characteristics are the usual definition of contrast. F_{cont} takes into account the range and the dispersion of grey-levels (i.e. the variance σ^2) and the polarization of the grey-levels histogram (i.e. the kurtosis α_4). It is therefore defined as Eq. A.9 where σ is the standard deviation, α_4 the kurtosis defined as Eq. A.10, μ_4 the fourth moment about the mean, and μ the mean defined as $\mu = \frac{\sum_{x=0}^{W}\sum_{y=0}^{H} I(x,y)}{W*H}$ where W is the width and H the height.

$$F_{cont} = \frac{\sigma}{(\alpha_4)^n} \tag{A.9}$$

$$\alpha_4 = \frac{\mu_4}{\sigma^4} = \frac{\frac{\sum_{x=0}^{W}\sum_{y=0}^{H}(I(x,y)-\mu)^4}{W*H}}{\left(\frac{\sum_{x=0}^{W}\sum_{y=0}^{H}(I(x,y)-\mu)^2}{W*H}\right)^2} \tag{A.10}$$

The value of n is set to 0.25 ([52]), hence Eq. A.11.

$$F_{cont} = \frac{\sigma}{(\alpha_4)^{\frac{1}{4}}} = \frac{\sigma}{\left(\frac{\mu_4}{\sigma^4}\right)^{\frac{1}{4}}} = \frac{\sigma^2}{\mu_4^{\frac{1}{4}}} \tag{A.11}$$

Our implementation strictly follows Eq. A.11. A histogram of the grey level is used to compute the mean, the variance and the fourth moment about the mean, and the measure of contrast is returned.

Directionality

Directionality is defined against non-directionality. It is orientation-independent, meaning that two patterns that differ only in their orientation will have the same degree of directionality. In order to compute the total degree of directionality of the texture, it is proposed to use an *histogram of local edge probabilities against their directional angle* ([52]). The gradient's magnitude $|\Delta G|$ and edge direction $\theta \in [0,\pi[$ are defined as Eq. A.12 and A.13. θ is measured in the counterclockwise orientation such that the horizontal direction is equal to zero.

$$|\Delta G| = \frac{(|\Delta_H| + |\Delta_V|)}{2} \tag{A.12}$$

$$\theta = \arctan\left(\frac{\Delta_V}{\Delta_H}\right) + \frac{\pi}{2} \tag{A.13}$$

In order to compute the gradient (to detect edges in the image), the Prewitt operator composed of the two 3x3 difference filters Δ_H (horizontal, Eq. A.14) and Δ_V (vertical, Eq. A.15) is used.

$$\Delta_H = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \tag{A.14}$$

$$\Delta_V = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \tag{A.15}$$

The histogram H_D of edge probability is obtained by quantizing and binning by θ (edge angle) at each points satisfying $|\Delta G| \geq t$, where t is a arbitrary threshold that aims to remove directions that are not reliable enough as an edge. Eq. A.16 is hence obtained, where $k = 0, 1, ..., n-1$ and $N_\theta(k)$ is the number of points where $\frac{(2k-1)\pi}{2n} \leq \theta < \frac{(2k+1)}{2n}$. [52] suggests using n=16 and t=12.

$$H_D(k) = \frac{N_\theta(k)}{\sum_{i=0}^{n-1} N_\theta(i)} \tag{A.16}$$

The directionality is extracted from H_D by computing the sharpness of the peaks and defined as A.17 where n_p is the number of peaks, ϕ the quantized direction code (modulo 180°), ϕ_p the peak position of H_D, w_p the range of p the peak between valleys, and r a normalization factor.

$$F_{dir} = 1 - r\, n_p \sum_{p}^{n_p} \sum_{\phi \in w_p} (\phi - \phi_p)^2 H_D(\phi) \tag{A.17}$$

[52] only considers cases where at most two peaks are presents. Eq. A.18 where v_{ij} is the positions of valley from peak ϕ_i to peak ϕ_j can be used to determine whether two ($n_p = 2$) or only one ($n_p = 1$) peaks are to be considered.

$$n_p = \begin{cases} 2, & \text{if } \frac{H_D(v_{12})}{H_D(\phi_2)} < 0.5, \frac{H_D(v_{21})}{H_D(\phi_2)} < 0.5, \text{ and } \frac{H_D(\phi_2)}{H_D(\phi_1)} > 0.2 \\ 1, & \text{otherwise} \end{cases} \tag{A.18}$$

Directionality is computed in WND-CHARM is a different manner. First of all the gradient is determined using Sobel filters instead of Prewitt. At each point of the filtered image, the orientation θ is computed using Eq. A.13, and a sum rs is incremented, with $rs = rs + \Delta_H^2(x,y) + \Delta_V^2(x,y) + \theta^2(x,y)$. θ is then quantized in a 125-bins histogram H. The index b_{max} of the bin containing the maximum value of H is extracted, and a sum hs is computed, with $hs =$

$\sum_{b=0}^{125}(b+1-b_{max})^2 H(b)$. Using this, the directionality is obtained through Eq. A.19

$$F_{dir} = \left|\log\left(\frac{hs}{rs}\right)\right| = \left|\log\left(\frac{\sum_{b=0}^{125}(b+1-b_{max})^2 H(b)}{\sum_{x=0}^{W}\sum_{y=0}^{H}\Delta_H^2+\Delta_V^2+\theta^2}\right)\right| \tag{A.19}$$

The origin of this new definition is not explained in the code. It is however observed ([23]) that the directionality feature performs quite poorly since calculating the variance of the peaks of the histogram is complicated. A suggestion to improve the feature is then to *calculate the global variance of the histogram and use entropy* ([23]). As entropy can be defined as Eq. A.20 for a vector x of N elements, it might be the explanation for the presence of the log.

$$Entropy(X) = \sum_{n=0}^{N-1} p(x_n)\log(p(x_n)) \tag{A.20}$$

In our module we implemented both the directionality as described in [52] (Eq. A.21 where n_p are the peak values of the histogram, w_p are all the bins that include a given peak value p, and ϕ_p is the bin with the highest peak value) and as Eq. A.20. We used Prewitt's filters as suggested in [52]. The two versions of directionality are outputted from our Tamura module.

$$F_{dir} = \sum_{p}^{n_p}\sum_{\phi\in w_p}(\phi-\phi_p)^2 H_D(\phi) \tag{A.21}$$

A.2.4 Transforms

We implemented a Transform module that computes the Fourier, Wavelet or Chebyshev transform of an image. It outputs images, without any measurement made on them. These images can be further given as input to any existing measurement module. For the Fourier and Wavelet, not only the transform but also the reconstruction was implemented. In this way the user can process the transformed image and go back to the original domain if needed.

Fourier Transform

The Fourier Transform (FT) is very easily implemented using the `fft2` function from the `fft` class available in `numpy/scipy` ([37], [24]):

```
import numpy as np
ft_image=np.fft.fft2(raw_image)
```

As the Fourier-tranformed image is composed of complex values, we return its modulus using the numpy `abs` function:

```
mod_ft_image=np.abs(ft_image)
```

Haar Wavelet Transform

Just like the Fourier Transform, the Wavelet Transform (WT) is designed to express a signal using orthonormal basis functions. It differs however in the fact that it is able to localize a function precisely both in frequency and space. Many different wavelet basis function have been proposed and we here implemented the most classical one, the Haar wavelet transform ([20]). The discrete Haar Wavelet Transform can be described by the transformation matrix H shown in Eq. A.22.

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{A.22}$$

A signal x of size 2^N can be analyzed following Eq. A.23. This procedure can be also explained using the filter bank representation shown in Fig. A.1: the signal is first split in odd and even samples, which amounts to downsampling it by two. Odd samples are filtered using an edge detector and yield the wavelet coefficient at first scale ω_1. Even samples go through a rectangle filter, and the resulting low-passed image is used as input to start again the same procedure and obtain coefficients at second scale. At each scale the size of the image is therefore reduced by two. As a result the maximal possible number of scale is N for an image of size 2^N.

$$\begin{bmatrix} \omega_1(k) \\ y(k) \end{bmatrix} = H \begin{bmatrix} x[2k] \\ x[2k+1] \end{bmatrix} \tag{A.23}$$

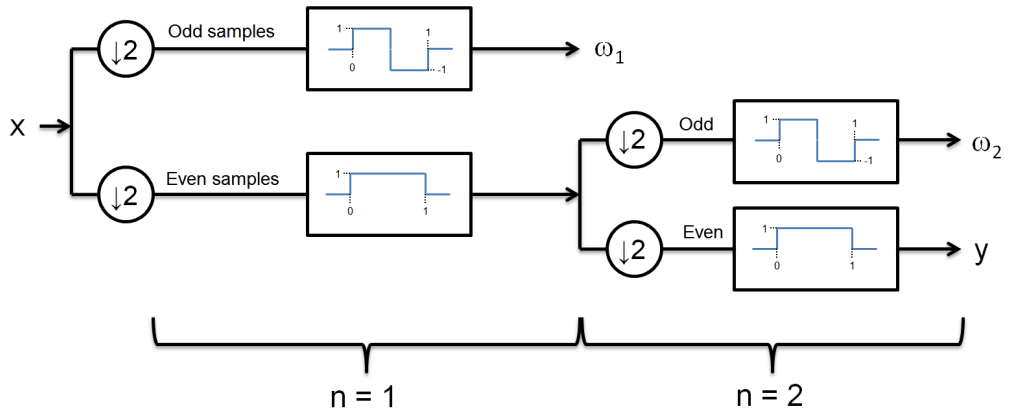


Figure A.1: Filter bank representation of the discrete Haar Wavelet Transform.

When working with 2-dimensional signals like images the transform can simply be applied in 1D first along the rows and then along the columns. The resulting Wavelet-transformed image is presented as shown in Fig. A.2. At the first scale, the top right square corresponds to the transformed image along rows, the bottom left square to the transformed image along

columns, and the bottom right square to the transformed image along both directions. The top left corner contains the lowpassed version of the image if only one scale is computed, or further decompositions if more than one scales are used.

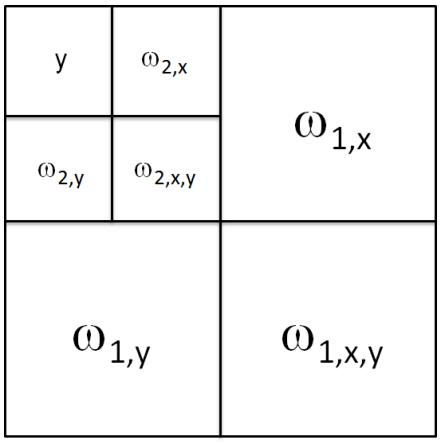


Figure A.2: Schematic representation of the wavelet-transformed image.

Our implementation strictly follows these explanations: at each scale we apply Eq. A.23 first along the rows and then along the columns.

Discrete Chebyshev Transform

The Discrete Chebyshev Transform (DCT) aims to approximate a smooth function u_n with Chebyshev Polynomials of the first kind of degree m, $T_m(x_n)$, weighted by expansion coefficients a_m as described in Eq. A.24.

$$u_n = \sum_{m=0}^{M-1} a_m T_m(x_n) \tag{A.24}$$

The Chebyshev Polynomials $T_m(x_n)$ are orthogonal, and the approximation coefficients can therefore be expressed as a scalar product between the function to be approximated and the Chebyshev Polynomials (Eq. A.25).

$$a_m = \langle u(x), T_m(x) \rangle = C_m \sum_{n=0}^{N-1} u(x_n) T_m(x_n) \tag{A.25}$$

In Eq. A.25 the C_m are defined as Eq. A.26.

$$C_m = \begin{cases} \frac{1}{N}, & \text{if } m = 0 \\ \frac{2}{N}, & \text{otherwise} \end{cases} \tag{A.26}$$

The Chebyshev Polynomials of the first kind can be expressed as $T_m(x) = \cos(m \cos^{-1}(x_n))$, and one can therefore see that $x_n \in [-1, 1]$. Using this definition, we can rewrite the approxi-

mation coefficient a_m following Eq. A.27.

$$a_m = C_m \sum_{n=0}^{N-1} u(x_n) \cos(m \cos^{-1}(x_n)) \tag{A.27}$$

The 2D version of the Chebyshev transform is given by Eq. A.28

$$u_{i,j}(x_i, y_j) \simeq \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} a_{mn} T_n(x_i) T_m(y_j) \tag{A.28}$$

The DCT is therefore separable and can be implemented efficiently by first computing the 1D transform along the rows, and then along the columns of the image. The Chebyshev Transform can also be implemented using the Discrete Cosine Transform, as described in [14].

Our implementation follows Ilya Goldberg's C++ code for WND-CHARM that can be found in `chevishev.cpp` ([57]). Our function is the image (i.e. a 2D signal), but since the Chebyshev Transform is separable as explained before, we work with the columns and then with the rows of the image (i.e. 1D signals). $u_n(x_n)$ is a 1xN vector, where N is either the width (if we are doing the transform along the rows) or the height (if we are doing the transform along the columns) of the image. u_n contains n elements (or pixels) with $n \in [0, N[$. Since $x_n \in [-1, 1]$ we can define $x_n = \frac{2(n+1)}{N} - 1$ with $n \in [0, N[$ such that $x_{n_{min}} = x_0 = \frac{2}{N} - 1$ and $x_{n_{max}} = x_{N-1} = \frac{2(N-1+1)}{N} - 1 = \frac{2N}{N} - 1 = 1$, and therefore $x_n \in [\frac{2}{N} - 1, 1] \subset [-1, 1]$. Using this definition, we have $n = \frac{N(x_n+1)}{2} - 1$. This change of variable amounts to approximating a function $g(x_n) = u\left(\frac{N(x_n+1)}{2} - 1\right)$ using Chebyshev Polynomials. The expression of the expansion coefficients therefore becomes Eq. A.29

$$a_m = C_m \sum_{n=0}^{N-1} g(x_n) T_m(x_n) = C_m \sum_{n=0}^{N-1} u(n) T_m\left(\frac{2(n+1)}{N} - 1\right) \tag{A.29}$$

In WND-CHARM's implementation, the whole coefficient is divided by 2 for unknown reasons. It amounts to approximating the function u_n divided by two, but this doesn't modify the problem since what is of interest is not to have the DCT representing faithfully the original function (since it does faithfully represents the function divided by two), but only to obtain new measurements to discriminate images. Hence if the same operation (here: division by two) is applied to all images, it doesn't matter. The final expression of the approximation coefficients is therefore Eq. A.30

$$a_m = C_m \sum_{n=0}^{N-1} g(x_n) \frac{T_m(x_n)}{2} = \frac{C_m}{2} \sum_{n=0}^{N-1} u(n) T_m\left(\frac{2(n+1)}{N} - 1\right) \tag{A.30}$$

A.2.5 ChebyHist

In our pipeline two Chebyshev transforms are computed on the image. The first one is a Chebyshev transform with order matching dimensions of the image, and is used as a new

image to compute additional features (Textures, Moments, etc.). The second one is a Chebyshev transform with order N=20. The resulting transformed image is therefore composed of 400 Chebyshev coefficients. These 400 coefficients are binned in a 32-bins histogram that compose the 32 *Chebyshev Histogram Statistics*. These features are computed on the original image and on the Fourier Transform of the original image (Fourier/Chebyshev compound transform). Our ChebyHist module performs binning and returns the *Chebyshev Histogram Statistics*.

B Code

B.1 WND-CHARM

B.1.1 Python

We implemented WND-CHARM in Python following both the description of the algorithm provided in [38] and [46], and the source code of the C++ version of the algorithm. In the code displayed in this section, `np` always corresponds to the `numpy` Python library ([37]).

Classifier training is performed by the `train` function of our `WNDCHARMClassifier` Python class. Its inputs are a `data` matrix containing training data, and a `labels` matrix containing the associated class labels. The features (i.e. columns in data matrix) are first normalized, and a weight is computed for each of them. Weights are then ordered by increasing value and selected using the 15% threshold as suggested in [46]. All weights that do not belong to the 15% best subset are set to zero and therefore become invisible for classification.

```
def train(self, labels, data):
    self.labels=np.array(labels)
    self.classes=np.array(np.unique(labels))
    self.data=np.array(data)

    for f in range(0, len(self.data[0])):
        self.data[:,f]=normalize(self.data[:,f])

    self.weight = np.array([])
    for f in range(0, len(data[0])):
        self.weight=np.append(self.weight,self.wnd_charm_weight(f))

    percent=0.15
    temp=np.sort(self.weight)
    index=len(self.weight)-np.round(percent*len(self.weight))
```

```
    if index>=len(self.weight):
        print 'Dataset contains too few information'
        sys.exit()

    thresh=temp[index]
    for f in range(0, len(self.weight)):
        if self.weight[f]<thresh:
            self.weight[f]=0.0
```

The `normalize` function used in `train` brings back every column in the data matrix to the $[0,100]$ interval as it is the case in WND-CHARM's C++ version.

```
def normalize(arr):
   arr=np.array(arr, dtype=float)
   max_val=float(arr.max())
   min_val=float(arr.min())
   if max_val!=min_val:
      res=np.array(100.0*((arr-min_val)/(max_val-min_val)))
   else:
      res=np.zeros([len(arr)])
   return res
```

Weights are computed following the C++ code and the explanations in [38]. Care is taken to avoid potential crashes or faulty results due to numerical errors.

```
def wnd_charm_weight(self, f):
    N=len(self.classes)

    features_f=np.array(self.data[:,f], dtype=float)
    if np.sum((features_f==False)*1.0)==len(features_f):
        return 0.0

    mean_f_c=np.array([])
    var_f_c=np.array([])

    for c in range(0,N):
        t_c=np.array(self.get_class_vectors(self.classes[c]), dtype=float)
        mean_f_c=np.append(mean_f_c, np.sum(t_c[:,f])/float(len(t_c)))
        temp_mean=np.sum(t_c[:,f])/float(len(t_c))
        var_f_c=np.append(var_f_c, np.sum(np.power(t_c[:,f]-temp_mean,2))/float(len(t_c)))
```

```
    mean_f=np.sum(mean_f_c)/float(N)

    if N>1:
        var_f=np.sum(np.power(mean_f_c-mean_f,2))/float(N-1)
    else:
        var_f=0.0

    inn_var_f=np.sum(var_f_c)/float(N)
    if inn_var_f==0:
        inn_var_f+=0.000001

    weight=var_f/inn_var_f
    return weight
```

Since it is required in weights computation to isolate samples from a particular class, we wrote a simple `get_class_vectors` function that allows extracting all samples belonging to a given class from the dataset.

```
def get_class_vectors(self, c):
    t_c=np.array([], dtype='S100')
    first_c=True
    for i in range(0, len(self.labels)):
        if self.labels[i] == c:
            if first_c:
                t_c=np.append(t_c, self.data[i,:])
                first_c=False
            else:
                t_c=np.vstack((t_c, self.data[i,:]))
    return t_c
```

When the training is done, a matrix `data` of samples from unknown classes can be given to the `classify` function. For each element of the `data` matrix the WND classifier is called and returns a predicted class.

```
def classify(self, data):
    data  = np.array(data, dtype=float)
    predicted_labels = []
    for f in range(0, len(data[0])):
        data[:,f]=normalize(data[:,f])
    for x in data:
        predicted_labels.append(self.WND(x))
    return np.array(predicted_labels)
```

Our `WND` function simply computes WND similarity between the sample to be classified and all classes in the training set. It returns the label of the class with maximal similarity.

```
def WND(self, x):
    dist = []
    for c in range(0,len(self.classes)):
        d=wnd_charm_similarity(x,self.weight,self.get_class_vectors(self.classes[c]))
        dist.append(d)
    dist=zip(dist,self.classes)
    return max(dist)[1]
```

Similarity is computed following [46] and the C++ implementation. The exponent value is set to -5.0 as suggested in [38].

```
def wnd_charm_similarity(x, w, c):
   if len(c)<1:
      print 'ERROR: Empty class'
      sys.exit()

   p=-5.0
   sum_t=np.array([])
   wnd=0.0

   for t in range(0, len(c)):
      if len(c.shape)>1:
         c_t=np.array(c[t,:], dtype=float)
      else:
         c_t=np.array(c[t], dtype=float)
      sum_f=np.sum(np.multiply(np.power(w, 2),np.power(x-c_t, 2)))
      sum_t=np.append(sum_t, np.power(sum_f, p))

   wnd=np.sum(sum_t)
   return wnd/len(c)
```

B.1.2 MATLAB

We designed a `classifyWND` function based on the existing `classify` and `classifykNN` MATLAB functions. The same inputs are required: a matrix of training data `XTRAIN`, the corresponding labels vector `YTRAIN`, and a matrix of test data with unknown labels `XTEST`. The method outputs a vector `YFIT` of predicted labels for the test set.

Following WND-CHARM's C++ code, we first bring back each features in the interval $[0, 100]$

using an ad-hoc `normalize` function:

```
function xnorm = normalize(x, upbnd, lowbnd)
    xmax=max(x);
    xmin=min(x);
    if xmin>=xmax
        xnorm=repmat(0.0,size(x,1),1);
    else
        xnorm=repmat(1.0/(xmax-xmin),size(x,1),1).*((repmat(upbnd-lowbnd,size(x,1),1).*
    end
end

nbfeats=size(xtrain,2);
for f=1:nbfeats
    xtrain(:,f)=normalize(xtrain(:,f),100.0,0.0);
    xtest(:,f)=normalize(xtest(:,f),100.0,0.0);
end
```

After this pre-processing step, the first operation in WND-CHARM's algorithm is to compute features weights following their definition in [38]:

```
classes=unique(ytrain);
nbclasses=size(classes,1);

weight=zeros(nbfeats,1);
classmean=zeros(nbclasses,1);
classvar=zeros(nbclasses,1);
for f=1:nbfeats
    for c=1:nbclasses
       classels=find(strcmp(ytrain,classes(c)));
       classmean(c)=mean(xtrain(classels,f));
       classvar(c)=var(xtrain(classels,f),1);
    end
    totalvar=var(classmean);
    meanvar=mean(classvar);
    if meanvar~=0
        weight(f)=totalvar/meanvar;
    else
        weight(f)=0.0;
    end
end
```

Weights are then thresholded, and the upper 15% is kept for classification. The other weights are not physically removed from the array, but their value is set to zero in order to render them ineffective in classification.

```
per=0.15;
temp=sort(weight);
tresh=temp(nbfeats-round(per*nbfeats));
for f=1:nbfeats
    if weight(f)<tresh
        weight(f)=0.0;
    end
end
```

Finally the actual classification step is performed using the definition of WND. For each element of the test set a predicted label corresponding to the class obtaining maximal similarity is added to the output vector.

```
nbtests=size(xtest,1);
yfit=cell(nbtests,1);
for i=1:nbtests
    sim=zeros(nbclasses,1);
    for c=1:nbclasses
        classels=xtrain(find(strcmp(ytrain,classes(c))),:);
        nbels=size(classels,1);
        for j=1:nbels
            dist=sum(power(weight',2).*power((xtest(i,:)-classels(j,:)),2));
            sim(c)=sim(c)+power(dist,-5);
        end
        sim(c)=sim(c)/nbels;
    end
    [val, ind]=max(sim);
    yfit(i)=classes(ind);
end
```

This function can be used in the same way as the existing `classify` MATLAB functions, and can for instance be fed into the `crossval` MATLAB method to perform k-fold cross-validation.

B.2 PCA-LDA

B.2.1 Python

We used functions from the scikit-learn Python library ([40]) to classify our whole-image features vectors using PCA-LDA in Python.

PCA

Since the columns of X (i.e. the features) are not known to be on the same scale we decided to first normalize and center each column. In order to do so we used the appropriate built-in `sklearn` function:

```
from sklearn.preprocessing import Scaler
scaler=Scaler()
X_norm=scaler.fit_transform(X)
```

X was explicitly defined as a float array, to avoid errors with `Scaler`.

Like most `sklearn` functions, PCA is performed on a NxM `numpy` array ([37]) of floats containing N samples (rows) of M features (columns). We explicitly casted the input array X as a `numpy` float array to avoid any type errors:

```
import numpy as np
X=np.array(X_raw, dtype='float')
```

PCA requires one argument `n_components`. If 0<`n_components`<1, PCA returns the number of principal components needed to explain a fraction of `n_components` of the variance in the original data. If `n_components`>1, PCA simply returns `n_components` principal components. Once PCA is instantiated, the analysis can be performed on the input matrix X:

```
from sklearn.decomposition import PCA
pca = PCA(n_components=0.98)
pca_data=pca.fit_transform(X)
```

A description of `sklearn`'s PCA utility can be found at [51].

LDA

LDA is also performed on a NxM `numpy` array of floats containing N samples (rows) of M features (columns). We explicitly cast the input array X as a `numpy` float array for the same reasons mentioned for PCA:

```
import numpy as np
formatted_training_data=np.array(training_data, dtype='float')
formatted_test_data=np.array(test_data, dtype='float')
```

For training, LDA requires the features matrix and the corresponding labels vector. The function is written such that the label vector must be composed of integer numbers. Since our classes did not always had numerical names we needed to recreate a vector of labels for LDA:

```
int_labels={}
for i in range(0, len(classes)):
   int_labels[classes[i]]=i

training_int=np.zeros([len(training_labels)])
for i in range(0, len(training_labels)):
   training_int[i]=int_labels[training_labels[i]]
```

This concludes the training part. Once the LDA instance was created, we could simply give to LDA's `fit` function our matrix of training data and the vector of corresponding labels:

```
import sklearn.lda as lda
classifier=lda.LDA()
classifier.fit(formatted_training_data, training_int)
```

We could hence test the efficiency of the classifier on our validation set. The `predict` function returns integer values as predicted labels, and we used the hash table `int_labels` we described before to know what the corresponding class was:

```
pred_int = classifier.predict(formatted_test_data)
pred_labels=np.array([])
for i in range(0, len(pred_int)):
   pred_labels=np.append(pred_labels, [key for key, value in int_labels.iteritems() if value
```

`pred_label` therefore contains the predicted classes for each element of the validation set.

A description of the LDA class is available online on `sklearn`'s website ([50]).

B.2.2 MATLAB

We implemented PCA-LDA in MATLAB following our Python code.

PCA

As a pre-processing step we centered and scaled the data along each column in order to bring different features to a common scale. We made sure to remove zero-variance features as they have no discrimination power.

```
X0 = X0(:,std(X0)~=0);
X0 = (X0 -repmat(mean(X0), size(X0,1),1) ) ./ repmat(std(X0), size(X0,1),1);
```

We then performed PCA using the MATLAB `princomp` function which returns the principal components coefficients (`c`), the representation of the data in PCA space (`s`), and the eigenvalues of the covariance matrix of the data matrix `X0` (`l`). When the `econ` option is used, only the non-zero eigenvalues of the covariance matrix are returned. In order to determine how much principal components are required to keep a portion of `thresh` of the variance, we computed a matrix `l0` containing the cumulative sums of eigenvalues divided by the total sum of eigenvalues and looked for the `neigs` values from this matrix that were above `thresh`. The `neigs` first principal components describing the data were then selected.

```
thresh=0.98;
[c s l] = princomp(X0(:,:), 'econ');
l0 = cumsum(l)/sum(l);
neigs = find(l0 > thresh, 1);
pcs(n,i)=neigs;
X = s(:,1:neigs);
```

LDA

In order to perform LDA we called the `classify` MATLAB function which takes as input a pxN matrix (where N is the number of features) `XTEST` of p unlabeled samples, a mxN matrix `XTRAIN` of m>>p training samples, and a mx1 matrix `ytrain` of m class labels corresponding to samples in `XTRAIN`. It returns a px1 matrix of predicted class labels for elements in `XTEST`. We fed this function to the `crossval` MATLAB method which directly performs 10-fold cross-validation as setup by the `cvpartition` MATLAB function. The final output is the misclassification rate obtained after cross-validation.

```
cp = cvpartition(y,'k',10);
classf = @(XTRAIN, ytrain, XTEST) (classify(XTEST,XTRAIN,ytrain));
cvMCR = crossval('mcr',X,y,'predfun',classf,'partition',cp);
```

Descriptions of both `princomp` and `classify` can be found in the online help of mathworks.com.

B.3 Validation methods

B.3.1 Python

In this section, `np` in the code always corresponds to the `numpy` Python library ([37]).

k-fold cross-validation

The Python code for k-fold cross-validation was used with the kind permission of its author, Auguste Genovesio at the Broad Institute.

```
def _k_fold_cross_validation_iterator(data, K=None):
   if K == None:
      K = data.shape[0]

   np.random.shuffle(data)
   for k in xrange(K):
      training = np.array([x for i, x in enumerate(data) if i \% K != k], dtype='S100')
      validation = np.array([x for i, x in enumerate(data) if i \% K == k], dtype='S100')
      yield training, validation
```

"save 25%"

We implemented a "save X%" validation method where X is by default set to 25%, but can be modified if desired. This method was written following WND-CHARM's C++ code and the description of "save 25%" given in [38].

```
def _saveX_validation_iterator(data, percent=0.25):
   if percent >= 1:
      print 'Error: training set must contains at least one element.'
      sys.exit()

   training=np.array([], dtype='S100')
   validation=np.array([], dtype='S100')
   first_v=True
   first_t=True

   classes=np.unique(data[:, 0])
   for c in range(0, len(classes)):
      t_c=np.array([], dtype='S100')
      first_c=True
      for i in range(0, len(data)):
```

```
            if data[i,0] == classes[c]:
                if first_c:
                    t_c=np.append(t_c, data[i,:])
                    first_c=False
                else:
                    t_c=np.vstack((t_c, data[i,:]))

        nbr=int(percent*len(t_c))
        np.random.shuffle(t_c)
        if nbr==1:
            print 'ERROR: Only one sample in test set.'
            sys.exit()

        for i in range(0, len(t_c)):
            if i<nbr:
                if first_v:
                    validation=np.append(validation, t_c[i,:])
                    first_v=False
                else:
                    validation=np.vstack((validation, t_c[i,:]))
            else:
                if first_t:
                    training=np.append(training, t_c[i,:])
                    first_t=False
                else:
                    training=np.vstack((training, t_c[i,:]))

    yield training, validation
```

B.3.2 MATLAB

10-fold cross-validation

Cross-validation was performed in MATLAB using the `crossval` function already mentionned in the PCA-LDA MATLAB section. More documentation can be found online at mathworks.com.

C Comparison Across Softwares

We present here results of similar experiments performed using different implementations. As we navigate through MATLAB, Python and R during our analysis we needed to ensure that we obtained the same results when performing identical experiments on different platforms.

C.1 C++/Python

WND-CHARM, "save 25%"

Fig. C.1 compares results averaged over 100 classification runs using the CHARM vector, WND, and "save 25%" for validation in the original C++ version and in our Python implementation.

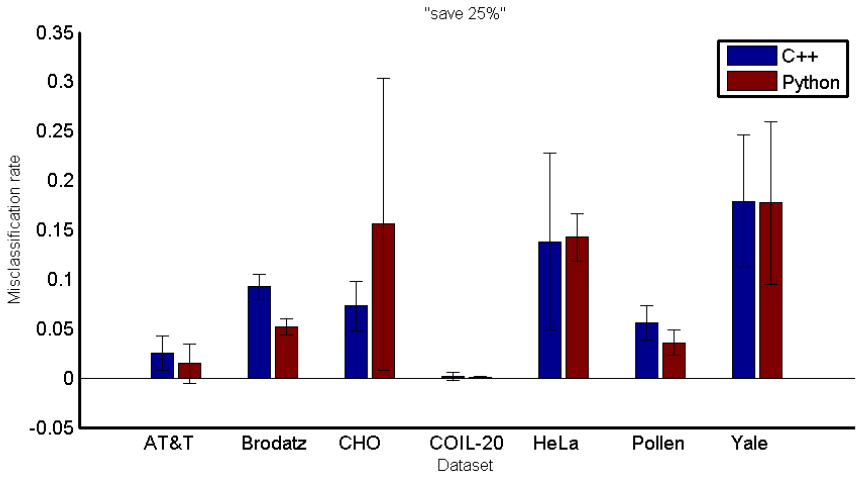


Figure C.1: Comparison of misclassification rates across versions using WND-CHARM and "save 25%" on the reference datasets. Results range from 0 (0%) to 1 (100%).

C.2 Python/MATLAB

WND-CHARM, 10-fold

Fig. C.2 compares results averaged over 10 classification runs using the CHARM vector, WND, and 10-fold cross-validation in MATLAB and Python.

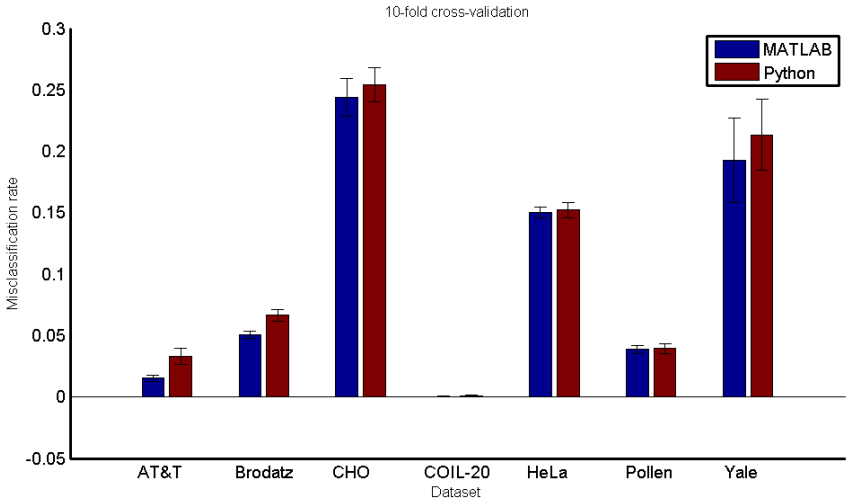


Figure C.2: Comparison of misclassification rates across versions using WND-CHARM and 10-fold cross-validation on the reference datasets. Results range from 0 (0%) to 1 (100%).

C.3 Python/MATLAB/R

PCA-LDA-CHARM, 10-fold

Fig. C.3 compares results averaged over 10 classification runs using the CHARM vector, PCA-LDA, and 10-fold cross-validation in MATLAB, Python and R.

PCA-LDA-CHARM-like, 10-fold

Fig. C.4 compares results averaged over 10 classification runs using our CHARM-like vector (v3.2), PCA-LDA, and 10-fold cross-validation in MATLAB, Python and R.

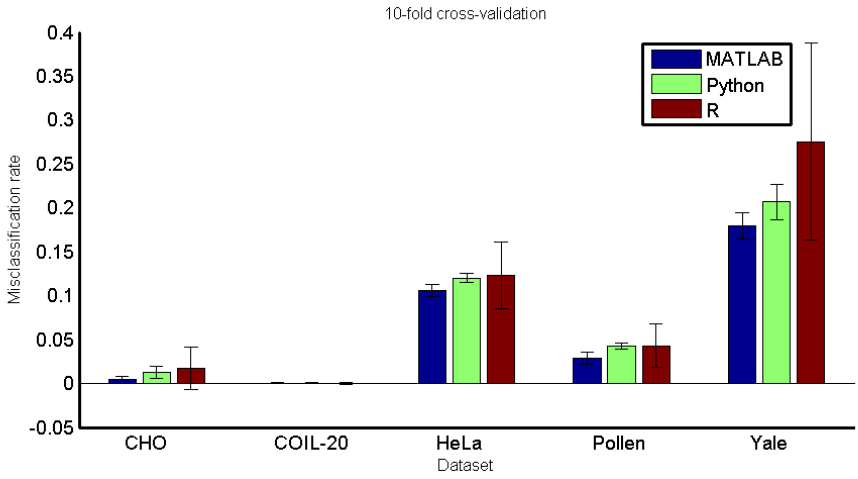


Figure C.3: Comparison of misclassification rates across versions using PCA-LDA-CHARM and 10-fold cross-validation on the reference datasets. Results range from 0 (0%) to 1 (100%).

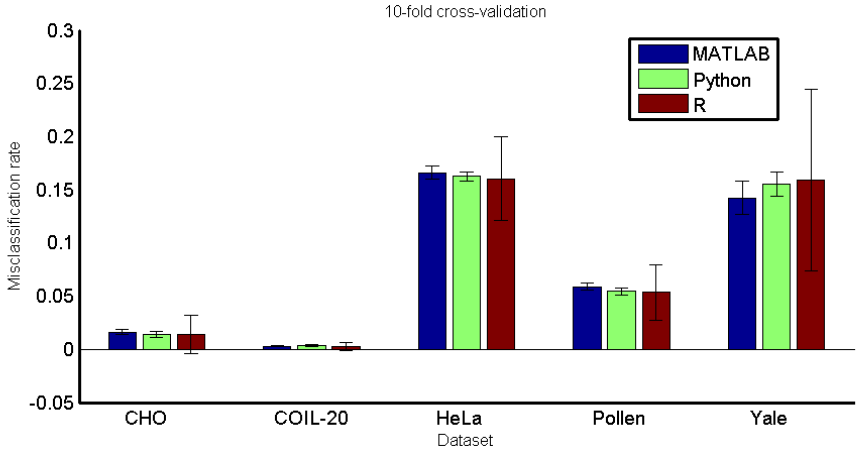


Figure C.4: Comparison of misclassification rates across versions using PCA-LDA-CHARM-like and 10-fold cross-validation on the reference datasets. Results range from 0 (0%) to 1 (100%).

D Contact information

The author of this thesis can be contacted at the following mailing address:

Virginie Uhlmann
Chataigneraie 17
1278 La Rippe (VD)
Switzerland

Home phone: +41 (22) 362 2419
Mobile phone: +41 (79) 726 8358

Electronic contact is also available using either of the following:

virginie (dot) uhlmann (at) epfl (dot) ch
uhlmann (dot) virginie (at) gmail (dot) com
virginie (underscore) uhlmann (at) bluewin (dot) ch

Bibliography

[1] A. Aizerman, E. M. Braverman, and L. I. Rozoner. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837, 1964.

[2] BBBC - Broad Bioimage Benchmark Collection. http://www.broadinstitute.org/bbbc/image_sets.html. Accessed: 5/24/2012.

[3] Christopher M. Bishop. *Pattern recognition and machine learning*. Springer, 1st ed. 2006. corr. 2nd printing edition, October 2006.

[4] M. V. Boland, M. K. Markey, and R. F. Murphy. Automated recognition of patterns characteristic of subcellular structures in fluorescence microscopy images. *Cytometry*, 33(3):366–375, Nov 1998.

[5] Michael V. Boland and Robert F. Murphy. A neural network classifier capable of recognizing the patterns of all major subcellular structures in fluorescence microscope images of hela cells. *Bioinformatics*, 17(12):1213–1223, 2001.

[6] L. Breiman and A. Cutler. Random Forests: Statistical Methods for Prediction and Understanding. http://www.stat.berkeley.edu/~breiman/RandomForests/. Accessed: 8/8/2012.

[7] P. Brodatz. *Textures: A Photographic Album for Artists and Designers*. Dover, 1966.

[8] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

[9] Anne Carpenter, Thouis Jones, Michael Lamprecht, Colin Clarke, In Kang, Ola Friman, David Guertin, Joo Chang, Robert Lindquist, Jason Moffat, Polina Golland, and David Sabatini. CellProfiler: image analysis software for identifying and quantifying cell phenotypes. *Genome Biology*, 7(10):R100+, October 2006.

[10] Computer Language Benchmarks Game. http://shootout.alioth.debian.org/u32q/benchmark.php?test=all&lang=gpp&lang2=python3. Accessed: 7/27/2012.

[11] T. Cover and P. Hart. Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13(1):21–27, January 1967.

[12] CellProfiler Help. http://www.cellprofiler.org/linked_files/Documentation/cp2_manual_11710.pdf. Accessed: 8/2/2012.

[13] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 1 edition, March 2000.

[14] Discrete Chebyshev Transform. http://en.wikipedia.org/wiki/Discrete_Chebyshev_transform. Accessed: 7/30/2012.

[15] A. W. G. Duller, G. A. T. Duller, France I., and Lamb H. F. A pollen image database for evaluation of automated identification systems. *Quaternary Newsletter*, 89:4–9, 1999.

[16] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Human Genetics*, 7(2):179–188, 1936.

[17] E. Fix and J. L. Hodges. Discriminatory analysis, nonparametric discrimination: Consistency properties. *US Air Force School of Aviation Medicine*, Technical Report 4(3):477+, January 1951.

[18] Athinodoros S. Georghiades, Peter N. Belhumeur, and David J. Kriegman. From few to many: Illumination cone models for face recognition under variable lighting and pose. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23:643–660, 2001.

[19] Estelle Glory, Justin Newberg, and Robert F. Murphy. Automated comparison of protein subcellular location patterns between images of normal and cancerous tissues. In *ISBI*, pages 304–307. IEEE, 2008.

[20] Alfred Haar. Zur Theorie der orthogonalen Funktionensysteme. *Mathematische Annalen*, 69(3):331–371, September 1910.

[21] Robert M. Haralick, K. Shanmugam, and Its'Hak Dinstein. Textural Features for Image Classification. *IEEE Transactions on Systems, Man, and Cybernetics*, 3(6):610–621, November 1973.

[22] H. Hotelling. Analysis of a complex of statistical variables into principal components. *J. Educ. Psych.*, 24, 1933.

[23] Peter Howarth and Stefan Ruger. Evaluation of texture features for content-based image retrieval. In *Proceedings of the International Conference on Image and Video Retrieval, Springer-Verlag*, 2004.

[24] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–.

[25] Thouis R. Jones, In Han Kang, Douglas B. Wheeler, Robert A. Lindquist, Adam Papallo, David M. Sabatini, Polina Golland, and Anne E. Carpenter. CellProfiler Analyst: data exploration and analysis software for complex image-based screens. *BMC Bioinformatics*, 9, 2008.

[26] Alexandros Kalousis, Julien Prados, and Melanie Hilario. Stability of feature selection algorithms. In *Proceedings of the Fifth IEEE International Conference on Data Mining*, ICDM '05, pages 218–225, Washington, DC, USA, 2005. IEEE Computer Society.

[27] Lee Kamentsky, Thouis R. Jones, Adam Fraser, Mark-Anthony A. Bray, David J. Logan, Katherine L. Madden, Vebjorn Ljosa, Curtis Rueden, Kevin W. Eliceiri, and Anne E. Carpenter. Improved structure, function and compatibility for CellProfiler: modular high-throughput image analysis software. *Bioinformatics (Oxford, England)*, 27(8):1179–1180, April 2011.

[28] Kernel-Machines.Org. http://www.kernel-machines.org/. Accessed: 7/17/2012.

[29] Max Kuhn. The "caret" Package. http://caret.r-forge.r-project.org/Classification_and_Regression_Training.html, 2009. Accessed: 7/18/2012.

[30] Murat Kunt, editor. *Reconnaissance des formes et analyse de scenes*. Traitement de l'information. Presses polytechniques et universitaires romandes [Paris], Lausanne, 2000.

[31] Michael R. Lamprecht, David M. Sabatini, and Anne E. Carpenter. CellProfiler: free, versatile software for automated biological image analysis. *BioTechniques*, 42(1):71–75, January 2007.

[32] Andy Liaw and Matthew Wiener. Classification and Regression by randomForest. *R News*, 2(3):18–22, 2002.

[33] David J. Logan and Anne E. Carpenter. Screening cellular feature measurements for image-based assay development. *Journal of biomolecular screening*, 15(7):840–846, August 2010.

[34] MATLAB - The Language of Technical Computing. http://www.mathworks.com/products/matlab/. Accessed: 7/16/2012.

[35] S. Nene, S. Nayar, and H. Murase. Columbia object image library (coil-100). *Technical Report CUCS00696*, 95(CUCS-006-96):223–303, 1996.

[36] S. Issac Niwas, Andreas Karsnas, Virginie Uhlmann, P. Palanisamy, Caroline Kampf, Martin Simonsson, Carolina Wahlby, and Robin Strand. Automated classification of immunostaining patterns in breast tissue from the Human Protein Atlas. To be presented at MICCAI 2012.

[37] Travis E. Oliphant. Python for scientific computing. *Computing in Science and Engg.*, 9(3):10–20, May 2007.

[38] N. Orlov, L. Shamir, T. Macura, J. Johnston, D. M. Eckley, and I. G. Goldberg. Wnd-charm: Multi-purpose image classification using compound image transforms. *Pattern Recognit Lett*, 29(11):1684–1693, Jan 2008.

[39] K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(6):559–572, 1901.

[40] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and Duchesnay E. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[41] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008. ISBN 3-900051-07-0.

[42] A. Rahimi and B. Recht. Random features for large scale kernel machines. *Proceedings of the 21st Annual Conference on Advances in Neural Information Processing Systems (NIPS)*, 2007.

[43] RStudio, Boston, MA, USA. *RStudio*, 2012. Retrieved: 7/16/2012.

[44] F. S. Samaria and A. C. Harter. Parameterisation of a stochastic model for human face identification. In *Proceedings of 2nd IEEE Workshop on Applications of Computer Vision*, Sarasota, FL, USA, December 1994.

[45] Lior Shamir, John D. Delaney, Nikita Orlov, D. Mark Eckley, and Ilya G. Goldberg. Pattern recognition software and techniques for biological image analysis. *PLoS Computational Biology*, 6(11), 2010.

[46] Lior Shamir, Nikita Orlov, D Mark Eckley, Tomasz Macura, Josiah Johnston, and Ilya Goldberg. Wndchrm - an open source utility for biological image analysis. *Source Code for Biology and Medicine*, 3(1):13, 2008.

[47] Lior Shamir, Nikita Orlov, D. Mark Eckley, Tomasz J. Macura, and Ilya G. Goldberg. Iicbu 2008: a proposed benchmark suite for biological image analysis. *Med. Biol. Engineering and Computing*, 46(9):943–947, 2008.

[48] Goldberg I. G Shamir L., Eckley D. M. Image tiling vs. cell segmentation - a case study. In *47th American Society for Cell Biology Meeting (ASCB'07)*, Washington, DC, 2007.

[49] Jonathon Shlens. A tutorial on principal component analysis. In *Systems Neurobiology Laboratory, Salk Institute for Biological Studies*, 2005.

[50] scikit-learn: LDA. http://scikit-learn.org/dev/modules/generated/sklearn.lda.LDA.html. Accessed: 7/30/2012.

[51] scikit-learn: PCA. http://scikit-learn.org/0.10/modules/generated/sklearn.decomposition.PCA.html. Accessed: 7/30/2012.

[52] Hideyuki Tamura, Shunji Mori, and Takashi Yamawaki. Textural Features Corresponding to Visual Perception. *IEEE Transaction on Systems, Man, and Cybernetics*, 8(6):460–472, June 1978.

[53] Mathias Uhlen, Per Oksvold, Linn Fagerberg, Emma Lundberg, Kalle Jonasson, Mattias Forsberg, Martin Zwahlen, Caroline Kampf, Kenneth Wester, Sophia Hober, Henrik Wernerus, Lisa Björling, and Fredrik Ponten. Towards a knowledge-based Human Protein Atlas. *Nature biotechnology*, 28(12):1248–1250, December 2010.

[54] Vladimir Vapnik. *Estimation of Dependences Based on Empirical Data: Springer Series in Statistics (Springer Series in Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982.

[55] Vladimir N. Vapnik. *Statistical learning theory*. Wiley, 1 edition, September 1998.

[56] Daniela M. Witten and Robert Tibshirani. Penalized classification using Fisher's linear discriminant. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(5):753–772, 2011.

[57] wnd-charm - Multi-purpose image classification and pattern recognition tool. http://code.google.com/p/wnd-charm/downloads/list. Accessed: 7/30/2012.

[58] Ji-Hu Zhang, Thomas D. Y. Chung, and Kevin R. Oldenburg. A simple statistical parameter for use in evaluation and validation of high throughput screening assays. *J Biomol Screen*, 4(2):67–73, April 1999.