Machine Learning regularity representation from biological patterns: a case study in a *Drosophila* neurodegenerative model

Sergio Díez Hermano
MSc in Bioinformatics and Biostatistics
Final Master Thesis – *Machine Learning*

Esteban Vegas Lozano

May 2017

FICHA DEL TRABAJO FINAL

Title:	*Machine learning regularity representation from biological patterns: a case study in a Drosophila neurodegenerative model*
Author:	*Sergio Díez Hermano*
Advisor:	*Esteban Vegas Lozano*
Co-advisors:	*Diego Sánchez and Lola Ganfornina*
Date:	05/2017
MSc title:	*MSc in Bioinformatics and Biostatistics*
Thesis area:	*Final Master Thesis in Machine Learning*
Manuscript language:	*English*
Keywords:	*Drosophila melanogaster, machine learning, classification*

Abstract (in English, 250 words or less):

Fruit fly compound eye is a premier experimental system for modelling human neurodegenerative diseases. Disruption of the retinal geometry has been historically assessed using time consuming and poorly reliable techniques such as histology or pseudopupil manual counting. Recent semiautomated quantification approaches rely either on manual ROI delimitation or engineered features to estimate the degeneration extent. This work presents a fully automated classification pipeline of bright-field images based on HOG descriptors and machine learning techniques. An initial ROI extraction is performed applying TopHat morphological kernel and Euclidean distance to centroid thesholding. Image classification algorithms are trained on these ROIs (SVM, Decision Trees, Random Forest, CNN) and their performance is evaluated on independent, unseen datasets. HOG + gaussian kernel SVM (0.97 accuracy and 0.98 AUC) and fine-tune pre-trained CNN (0.98 accuracy and 0.99 AUC) yielded the best results overall. The proposed method provides a robust quantification framework that addresses loss of regularity in biological patterns similar to the *Drosophila* eye surface and speeds up the processing of large sample batches.

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados y conclusiones del trabajo.*

El ojo de la mosca del vinagre es un modelo clásico de enfermedad neurodegenerativa. Históricamente la estimación del nivel de degeneración ha consistido en preparaciones histológicas o recuento manual en pseudopupila,

técnicas costosas en cuanto a tiempo de ejecución y de fiabilidad limitada. Recientemente se han desarrollado aproximaciones semiautomáticas a partir de imágenes tomadas a los ojos en luz blanca y por microscopía electrónica, que requieren de un paso previo de delimitación manual del área de interés (ROI), y una selección de variables prefijadas a partir de las cuales realizar la estimación. Con este trabajo se pretende proporcionar una herramienta totalmente automática de multiclasificación basada en la extracción de descriptores HOG y técnicas de *deep learning*. Se presenta un algoritmo de segmentación y extracción del ROI correspondiente al área del ojo, utilizando transformaciones morfológicas TopHat y filtraje por distancia al centroide del conjunto de píxels. Sobre estos ROIs se comparan diferentes algoritmos de clasificación (SVM, árboles de decisión, Random Forest). Los mejores resultados se obtienen mediante HOG+SVM con kernel gaussiano (precisión 0.97 y AUC 0.98) y CNN pre-entrenada (precisión 0.98 y AUC 0.99). Aplicándolos sobre un modelo de neurodegeneración que se apoya en el ojo de la mosca, con una geometría totalmente regular cuando está sano que se pierde a medida que progresa la enfermedad, es posible no sólo proporcionar un marco común capaz de analizar la pérdida de regularidad en otros patrones biológicos con simetrías similares, sino agilizar el procesado de grandes muestras de datos.

Table of contents

Figure list

Table list

1. Introduction

1.1 Context and motivations

Drosophila neurodegeneration model

Drosophila melanogaster stands out as one of the key animal models in today's modern genetic studies, with an estimated 75% of human disease genes having orthologs in flies [1]. Its growth as a powerful model of choice has been supported by the wide array of genetic and molecular biology tools designed with the fruit fly in mind [2], easing the creation of genetic deletions, insertions, knock-downs and transgenic lines. Fly biologists have greatly contributed to our knowledge of mammalian biology, making *Drosophila* the historical premier research system in the fields of epigenetics, cancer molecular networks, neurobiology and immunology [3]. The relative simplicity of *Drosophila* genetics (4 pairs of homologous chromosomes in contrast to 23 in humans) and organization (i.e. ~200,000 neurons in opposition to roughly 100 billion neurons in humans) makes the fruit fly an especially well-suited model for the analysis of subsets of phenotypes associated with complex disorders.

Specifically, the retinal system in *Drosophila* has been widely used as an experimental setting for high throughput genetic screening and for testing molecular interactions [4]. Eye development is a milestone in *Drosophila* life cycle, with a massive two-thirds of the essential genes in the fly genome required at some point during the process [5, 6]; thus constituting an excellent playground to study the genetics underlying general biological phenomena, from basic cellular and molecular functions to the pathogenic mechanisms involved in multifactorial human diseases, such as diabetes or neurodegeneration [7-9].

The fruit fly compound eye is a nervous system structured as a stereotypic array of 800 simple units, called ommatidia, which display a highly regular hexagonal pattern (**Fig. 1**). Precisely this strict organization allows to evaluate the impact of altered gene expression and mutated proteins on the external eye morphology, and to detect subtle alterations on the ommatidia geometry due to cell degeneration. One special type of cellular deterioration largely studied using *Drosophila* retina encompasses polyglutamine-based neurodegenerative diseases, namely Huntington's and the dominant Spinocerebellar ataxias (SCA) [10]

Polyglutamine disorders are caused by single gene mutations that lead to a toxic gain-of-function phenotype, primarily expansions of unstable CAG repeats, which translates to abnormally glutamine-enriched proteins that end up aggregating in the cell nucleus. Characteristic features in patients

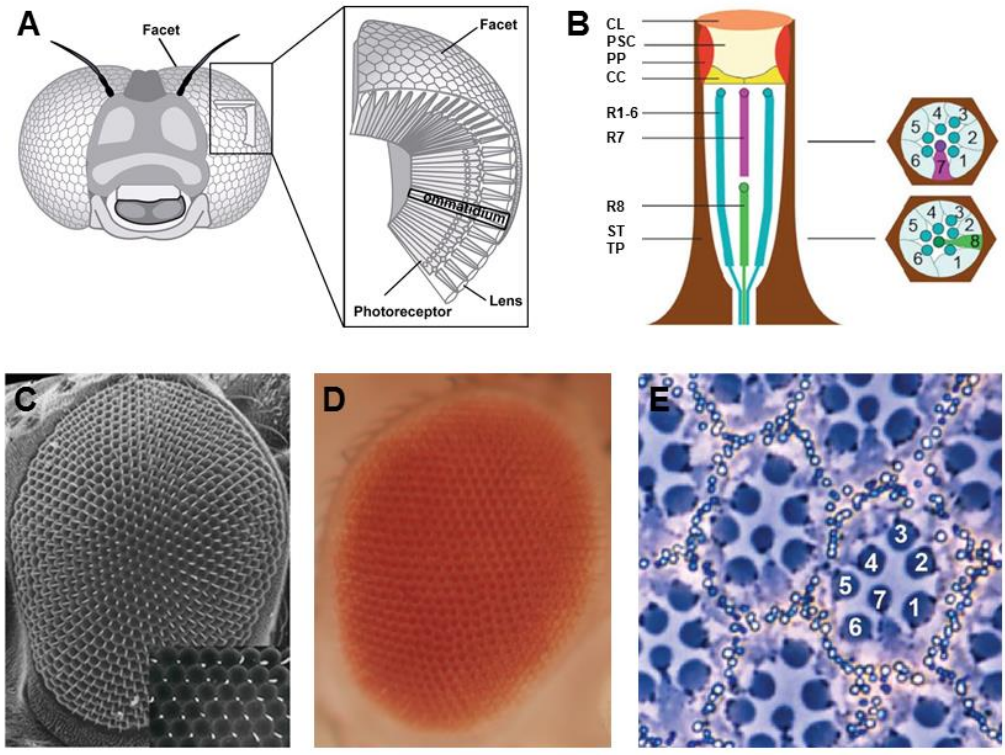


Figure 1. Drosophila compound eye structure. A Representation of the fruit fly eye surface and internal disposition of ommatidia. **B** Schematic of an ommatidium. Sagittal section (left), coronal sections at different planes (right). *CL* corneal lens; *PSC* pseudocone; *PP* primary pigment cells; *CC* cone cells; *R1-6, R7*, *R8* photoreceptor cells; *SP* and *TP* secondary and tertiary pigment cells. **C-E** Different eye imaging techniques demonstrating the hexagonal packing of the ommatidia and the trapezoidal arrangement of the photoreceptors, **C** scanning electron micrograph (SEM), **D** bright field microscope, **E** tangential section through the eye. Adapted from [6].

are motor impairment and age-dependent degeneration, and a varying age of onset inversely correlated with the CAG repeat length [10].

In *Drosophila,* it is possible to express endogenous and exogenous sequences in the tissue of interest using the UAS/*Gal4* binary system (**Fig. 2A**). This model exploits the P element, a transposable sequence naturally found in the fruit fly that can be engineered to manipulate genomic insertions of one or multiple transgenes in a tissue-specific manner. UAS/Gal4 system uses a yeast transcription factor, *Gal4*, that binds to the so-called Upstream Activating Sequence (UAS) enhancer element, triggering the expression of the inserted downstream transgene. Nor *Gal4* gene and UAS sites can be found in wild type *Drosophila* genome, neither fly transcription factors activate expression of UAS-controlled sequences. This transcriptional system is then a completely artificial tandem that allows for ectopic expression of a desired transgene, depending on the tissue-specific enhancer line used to express the *Gal4* gene.

Thus, overexpression of polyQ-expanded proteins via the UAS/*Gal4* system in the fly retina results in a depigmented, rough eye phenotype (REP) caused by loss of interommatidial bristles, ommatidial fusion and necrotic tissue (**Fig. 2B**). The vast majority of studies assessing the rough eye morphology relies on qualitative examination (*i.e.,* visual inspection) of its external appearance to manually rank and categorize mutations based on their severity [11-13]. Even though evident degenerated phenotypes are easily recognizable, weak modifiers or subtle alterations may go undetected for the naked eye. Quantitative approaches addressing this issue involve histological preparations from which evaluate the retinal thickness, regularity of the hexagonal array or scoring scales for the presence of expected features in the retinal surface [14-18]. Recently there have been efforts to fully computerize the analysis of *Drosophila* REP in bright field and SEM images in the form of ImageJ plugins, called *FLEYE* and *Flynotiper* [19, 20]. Whereas both methods propose automatized workflows, the former prompts the user to manually delimit the region of interest (ROI) to extract hand-crafted features from it, and the other relies

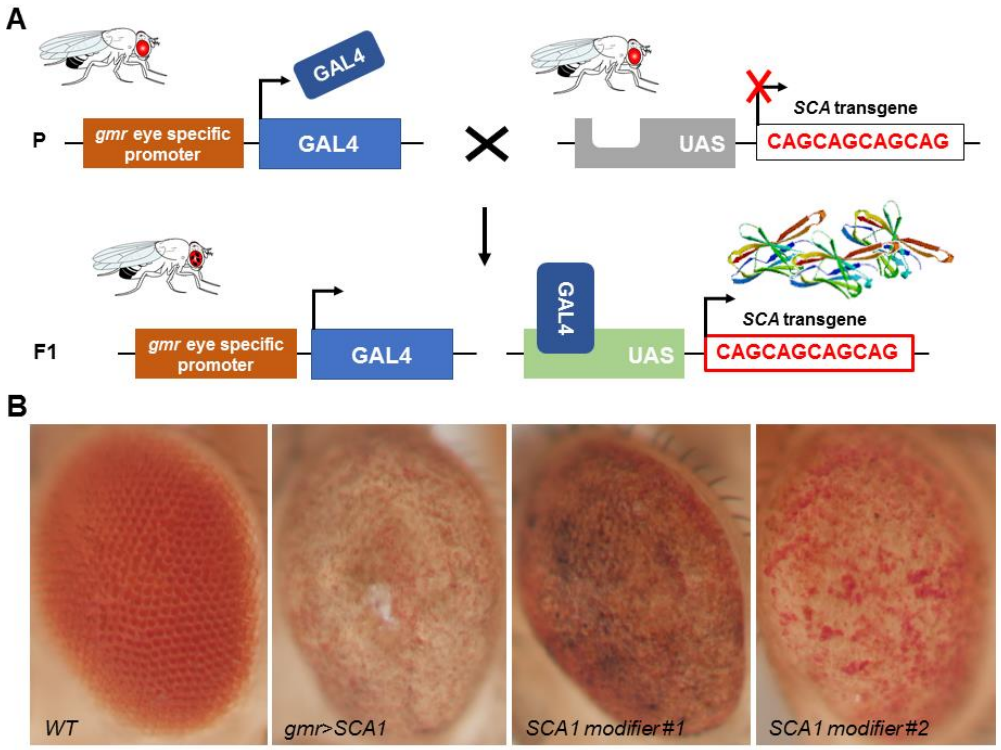


Figure 2. UAS/Gal4 system. A The Gal4 transgenic driver line expresses the transcription factor Gal4 in a specific spatiotemporal pattern, in this case under the *gmr* gene promoter, exclusive to the fly eye. The UAS transgenic line contains the Gal4 specific binding sites. Upon Gal4 binding to UAS sites, expression of downstream sequences is activated, which in our case leads to production of polyQ ataxin-related proteins in the fly eyes. For both systems to concur in a single fly, mating is required between transgenic lines for a Gal4 driver and a UAS construct, and only the F1 offspring will display Gal4 activated expression of UAS-controlled sequences. **B** Bright field images of REP. SCA1 gene modifiers can be tested on the fly eye via the UAS/Gal4 system. Complete loss of surface regularity and depigmentation can be appreciated between the WT and SCA1 phenotypes. SCA1 modifiers show intermediate levels of degeneration. Adapted from [8].

upon a single engineered feature and lacks any statistical background to support it.

Hence, there is a need to tackle a fully automatized, statistically multivariate assessment of *Drosophila* eye quantification, given its utmost relevance as a simple yet comprehensive model for testing general biology hypothesis and human neurological diseases.

Image classification pipelines

To this day, there is a surprisingly absence of studies that apply image classification techniques to the quantitative measurement of *Drosophila*

eye surface regularity. Particularly, machine learning algorithms have proven to be incredibly efficient image classifiers during the past decade [21], rapidly permeating in the fields of cell biology and biomedical image-based screening [22, 23]. Machine learning methods greatly ease the analysis of complex multi-dimensional data by learning processing rules from examples that can be later on generalized to classify new, unseen data.

The typical classification workflow consists in a pipeline of image pre-processing, object detection and features extraction, which are fed as input to the embedded machine learning algorithm. In the training phase, the principal goal of the learning task is to infer general properties of the data distribution from a few examples annotated according to predefined classes. This approach has been termed as 'supervised' learning [21, 24] and its successful application in bioscience ranges from high-content screening and drug development [25, 26] to DNA sequence analysis and proteomics [27, 28]. In opposition, there also exists an 'unsupervised' approach, that mainly attempts to cluster data points on the basis of a similarity measure and enables the exploration of unknown phenotypes, but we won't be focusing on it.

The general workflow for supervised image classification is depicted in **Fig. 3A**. The first pre-processing step usually aims at correcting uneven illuminations caused by camera artefacts, normalizing the intensity levels as these should not change with the position inside the picture. It may also include changes between different colour spaces. Next, the objects of interest need to be defined. Object detection is an inherently difficult task to generalize, and no single method exists to solve all segmentation problems in biological imaging. It is frequent to look for differences in region properties (*i.e.*, background removal by intensity thresholding) or edges and contours. Morphological transformations with varying kernel shapes and sizes may also help to enhance contrast between local regions.

Following segmentation, a feature extraction step is mandatory to translate each object into a quantitative vector, suitable as input for the classifier

algorithm. Descriptive features are derived from the raw pixel intensities in order to discard irrelevant information, such as spatial and spectral patterns exclusive to a single image [21, 29]. Features quantifying the pixel distribution have been extensively used and tend to measure texture properties, granularity, contour roughness or object circularity [30]. The histogram of oriented gradients (HOG) is another feature extractor that converts a pixel-based representation into a gradient-based one, calculating the frequency of a given intensity local change within the pixel array [31]. Until the arrival of deep learning, HOG in combination with classical machine learning classifiers was among the top performance techniques in object recognition [32-37]

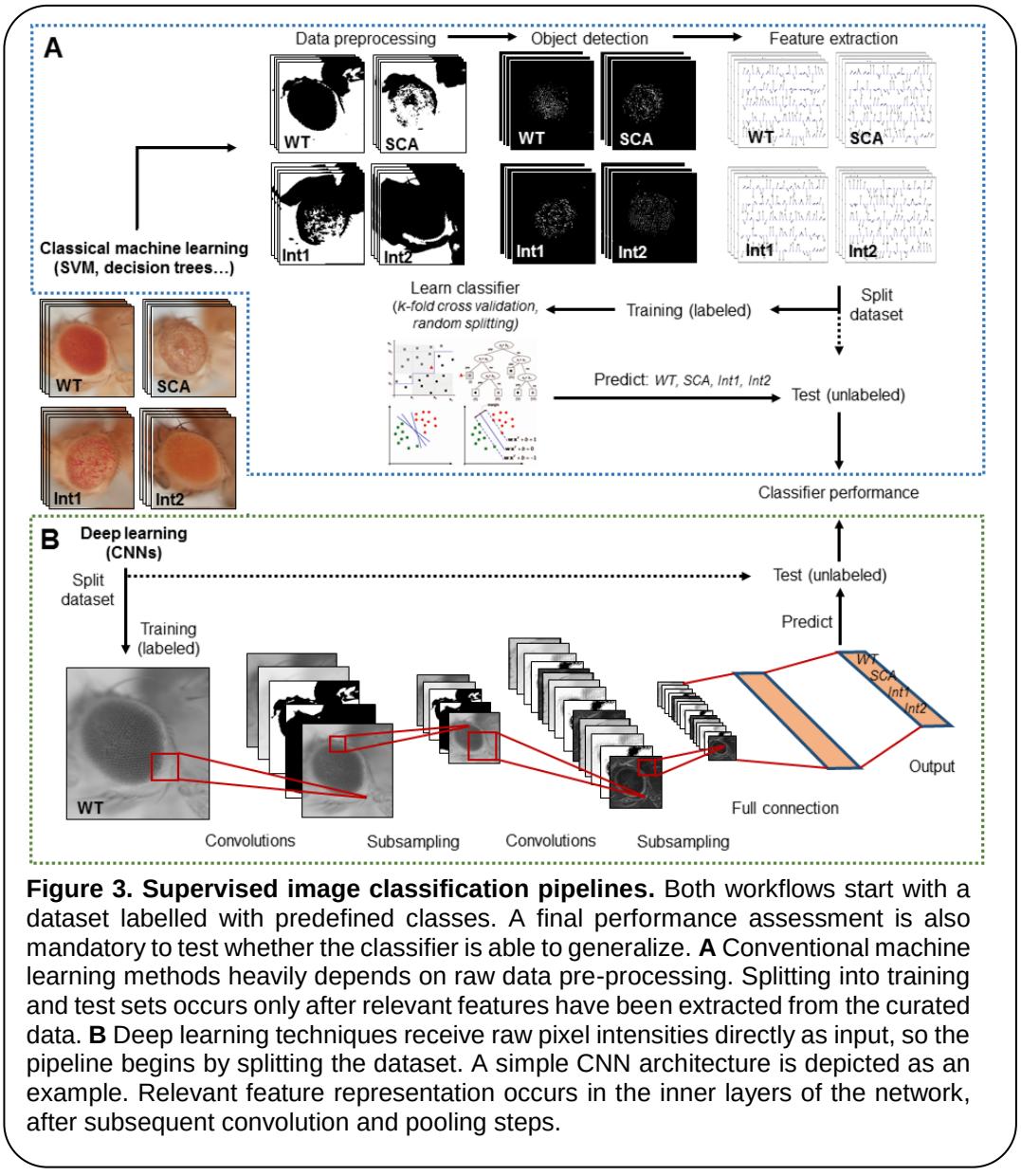


Figure 3. Supervised image classification pipelines. Both workflows start with a dataset labelled with predefined classes. A final performance assessment is also mandatory to test whether the classifier is able to generalize. **A** Conventional machine learning methods heavily depends on raw data pre-processing. Splitting into training and test sets occurs only after relevant features have been extracted from the curated data. **B** Deep learning techniques receive raw pixel intensities directly as input, so the pipeline begins by splitting the dataset. A simple CNN architecture is depicted as an example. Relevant feature representation occurs in the inner layers of the network, after subsequent convolution and pooling steps.

A labelled feature matrix serves then as the training input a classifier needs to automatically infer internal parameters of a learning model. This process is guided by an objective function that is subject to optimization and evaluates how well different combinations of parameters fit to the training data. It is essential to test the ability of the model to properly classify new examples not used for training, to ensure the learning rule is a generalized solution to the classification problem and the learner is not simply memorizing the training set. Splitting the dataset into learning and testing fractions may also follow diverse strategies (*i.e.,* k-fold cross-validation) [38, 39]. Machine learning techniques typically applied to image classification includes support vector machines (SVM) [27], decision trees (DT) [33], random forests (RF) [40] and neural networks (NN) [41], and will be explained in some detail in subsequent sections.

Alongside processing power and GPU-dedicated coding, deep learning methods have exponentially grown in importance during the last few years [42, 43]. Conventional machine learning algorithms aforementioned require data processing and feature enrichment prior to the training, as they are not suited to work with raw input. In contrast, deep learning procedures are general-purpose learners in the sense that they can be fed with raw data and automatically suppress irrelevant information and select discriminant characteristics, composing simple layers of non-linear transformations into a higher, more abstract level representation (**Fig. 3B**). In image classification, the input has the form of an array of pixel values, and the deeper a layer is in the network the more complex the features it learns. Superficial layers that directly receive the input extract general edge and orientation detectors, whereas final layers assemble motifs into larger combinations to represent defining parts of objects.

Convolutional neural networks (CNNs) are a well-known architecture for deep learning that have been continuously outperforming previous machine learning techniques, especially in computer vision and audio recognition [43]. With the increasing availability of large biological datasets, its popularity in bioinformatics and bioimaging has quickly escalated, and currently CNNs are addressing problems hardly resolvable

by former top-notch analysis techniques [44-48]. The striking advantage of these networks is that feature hand-crafting and engineering is completely avoided, as they implement intricate functions insensitive to perturbations thanks to the multilayer mapping representation of discriminant details. They have been proven to be a universal approximation algorithm. Crucial aspects of a CNN are the network depth (number of layers) and width (nodes in each layer), given the computational cost increases with the number of training parameters in higher dimensional feature spaces.

Large sized samples are a big helping hand in maintaining the trade-off between network complexity and accuracy, and that's where some areas in biology still fall short off. For this reason, transfer learning approaches are becoming a staple alternative when labelled training examples are limited or hard to gather [49, 50]. The concept of transfer learning refers to applying the knowledge acquired from CNNs already pre-trained on thousands of labelled natural images, to different but related tasks (*i.e.,* as feature extractors for smaller samples with categories not originally present in the network training). Source data don't even need to be related to the new dataset, as images share common patterns like edges and contours that can be modelled independently from the actual content of the image.

The novelty of the present work consists in applying and comparing the different image classification strategies mentioned so far in an extensively used biological model as *Drosophila melanogaster*, which has been scarcely addressed before and is in dire need of a state-of-the-art quantification framework.

Feature extraction and learning algorithms

We'll end this section by succinctly explaining the basic ideas behind the feature descriptor and the machine and deep learning algorithms used in the present work.

Histogram of Oriented Gradients. HOGs are local scale-invariant descriptors that result in a compressed and encoded version of an image

(**Fig. 4)**. Oriented gradients are vectors of both magnitude and direction of the intensity change in a given location. These are weighted by a Gaussian window. Then histograms of patched gradients are computed to obtain an estimated probability of preferred orientations within that patch. To preserve the spatial relation, these features are block-wise concatenated into a single vector ready to be used as input for a machine learning classifier (SVM, DT, RF, NN).

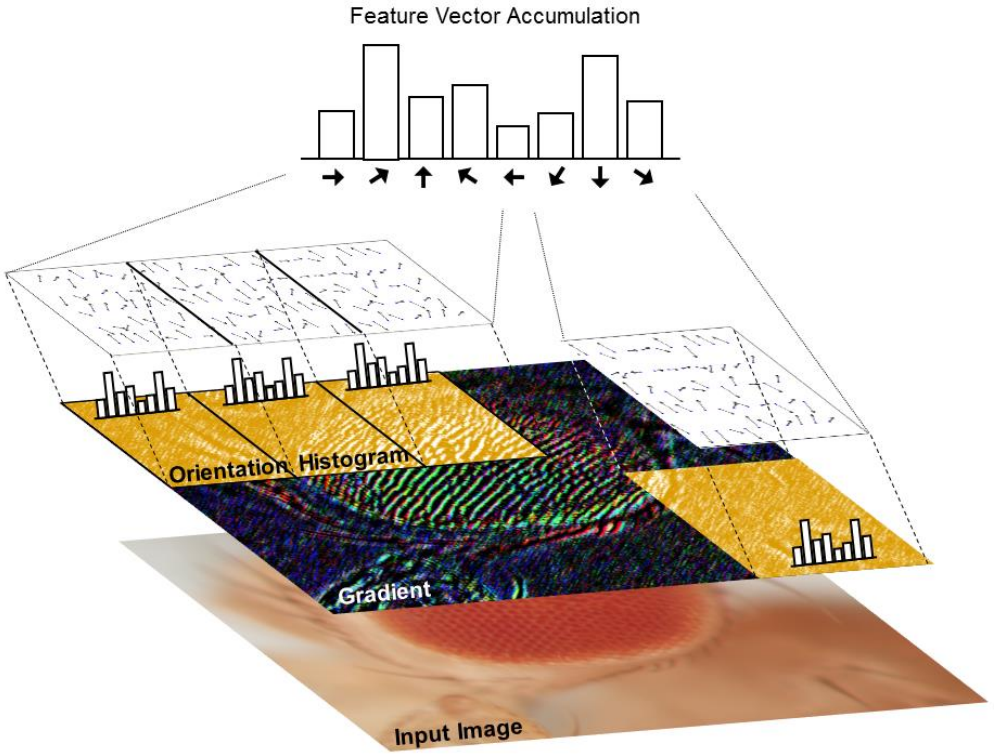


Figure 4. Histogram of Oriented Gradients. Input image is divided into smaller blocks. In these cells horizontal and vertical gradient filters are applied pixel-wise, so each pixel has an associated orientation and magnitude. A histogram for all the orientations in a block is built using discrete bins. Bigger magnitudes decide which is the preferred orientation in a given cell, and finally cell histograms are concatenated into a larger HOG descriptor vector.

Support Vector Machines. SVMs evaluates the goodness of all possible decision boundaries that can be constructed to separate data points belonging to two (line) or more (hyperplane) different classes. SVMs find the boundary with maximal margins to the nearest training samples (**Fig. 5A**), and the decision function derived is expected to generalize well as it minimizes the theoretical risk of error. Most SVM algorithms allows for

some percentage of misclassifications at a certain penalty parameter *C* (soft margins). When data is not linearly separable in the original input space, kernel transformations allow for more sophisticated boundaries in a higher-dimensional feature space (*i.e.,* Gaussian kernel or radial basis function, RBF) (**Fig. 5B**).

Decision Trees. DT classifiers are built using a heuristic called recursive partitioning. It iteratively splits the input space into smaller subsets, selecting at each node the feature and the threshold that will partition the sample, until the remaining data is sufficiently homogenous or a certain stopping criterion is met (pre-pruning). A common strategy to avoid overfitting involves reducing the size of the final tree (post-pruning). Usually the resulting boundaries are parallel to the feature axes (**Fig. 5C-D**). The final classifier is easy to interpret in most cases.

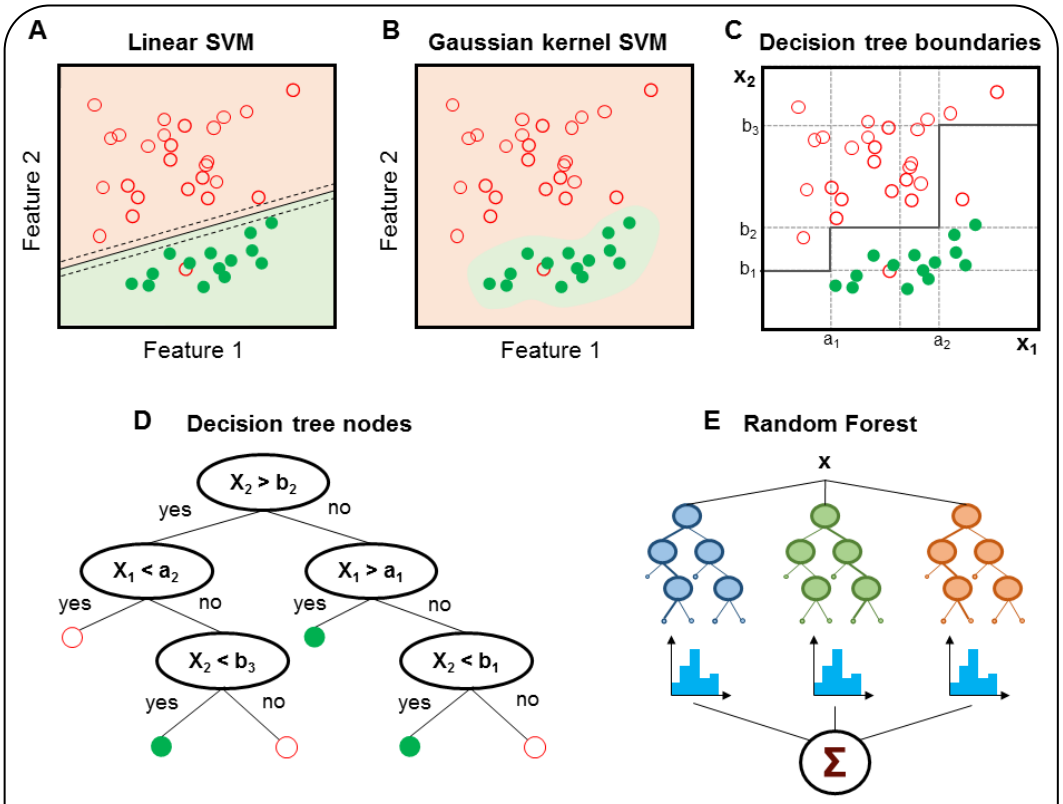


Figure 5. Conventional supervised machine learning classifiers. A SVM with linear kernel. Dotted lines represent the "support vectors" or maximal margins that determine the best linear boundary. They are "soft margins" as misclassification is allowed. **B** Non-linear kernels may induce SVMs to learn more intricate decision boundaries. **C** Binary decision tree boundaries are always parallel to the axis. Its corresponding splitting rules are represented in **D**, where internal nodes account for the feature thresholds and the terminal leafs for the final class predicted. **E** In random forest many weak DTs are randomly averaged to form a more robust class prediction. Adapted from [21, 24].

Random Forest. RFs are ensemble methods based on the notion that combining multiple weak learners results in an overall stronger classifier. The combination function to determine the final prediction may vary from a majority vote approach to averaging the outcome of randomly weighted decision trees (**Fig. 5E**). The resulting ensemble reduces the global variance while keeping the original bias low, and has an implicit feature selection that makes it robust and computationally efficient.

Convolutional Neural Network. CNNs are a type of multilayer perceptron (MLP) that reduces the degrees of freedom of spatially- correlated features. CNNs are composed of convolution, non-linear activation function and pooling layers. Convolutional layers perform an affine transformation of the input images using learned kernels in a local fashion, as calculations at each pixel involves only its surrounding neighbours (**Fig. 6A**). The computational cost of the convolutions depends on the number of filters and the stride (overlapping between kernel operations on the same image).

Convolutional layers greatly increase the number of hidden units as there are usually more output feature maps than input maps. The reasoning behind the local connectivity is that motifs are invariant to location, so they could appear anywhere within the image, hence the advantage of units at different locations sharing weights and detecting similar patterns. The result of this local weighted sum is passed through an activation non-linearity such as Rectified Linear Units (ReLu) (**Fig. 6B**). Subsequent pooling layers perform subsampling on the maps learned, merging semantically similar features into one (**Fig. 6C**). This reduces the dimension of the representation and gives the network some degree of invariance to shifts and distortions.

The arrangement of convolution, non-linearity and pooling forms a processing block that can be concatenated to the desired network depth. Lastly, a fully-connected layer encode the global patterns and the final output (*i.e.,* predicted classes) is retrieved. Central parameters of a CNN are the learning rate (small to avoid convergence problems), batch size, momentum, weight initialization (random, pre-defined), decay, dropout

rates and regularization. Tweaking these parameters is essential to prevent overfitting and find the adequate settings for the solving task.

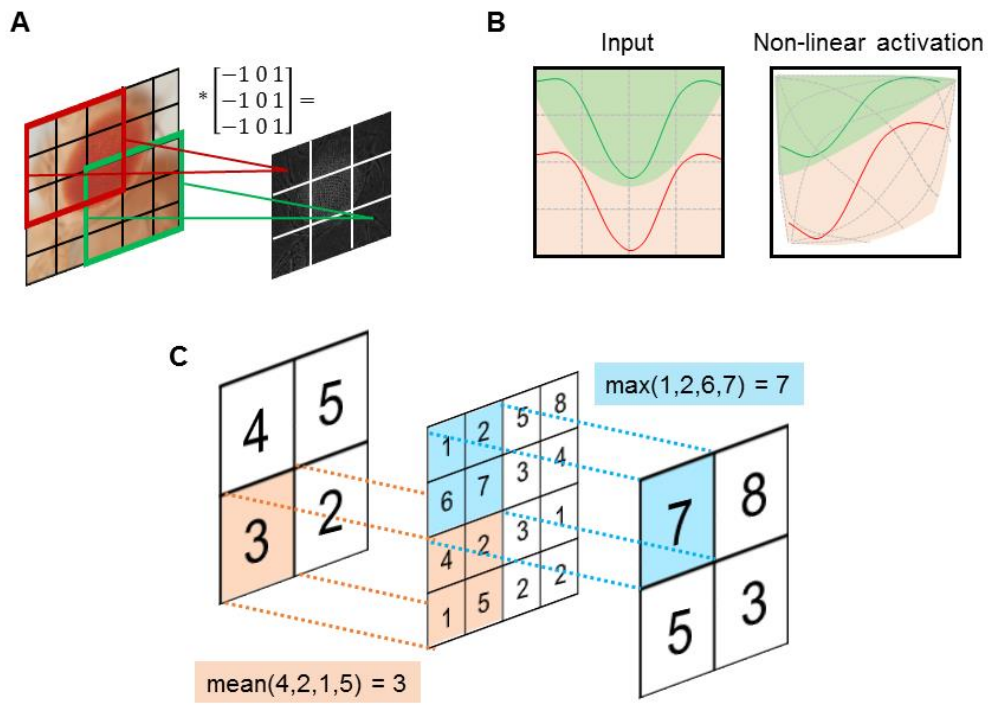


Figure 6. Computations in CNN inner layers. A Convolutional layers apply filters with fixed or varying size and different kernels to detect edges in many directions. The filtering is mathematically a discrete convolution. **B** Activation functions such as sigmoid, hyperbolic tangent or ReLU transform an initially non-linearly separable problem into a linear decision boundary. **C** Pooling strategies include “Max pooling” (highest local value) and “Average pooling” (mean local value).

1.2 Objectives

The main objectives in the present work are:

i) To automatically classify *Drosophila melanogaster* eye images with varying degrees of degeneration. To this extent, it is necessary to pre-process and segment the images in a way that allows for the extraction of non-manually drawn ROIs centred at the focal plane of the eye.

ii) To compare the most recent image classification techniques in terms of accuracy and global performance on independent fly eye image sets. This includes more classical machine learning algorithms (SVM, DT, RF) and state-of-the-art deep learning methods (CNN).

1.3 Materials and methods

Fly lines and maintenance

All stocks and crosses were grown in a temperature-controlled incubator at 25ºC, 60% relative humidity, under a 12 h light-dark cycle. They were fed on conventional medium containing wet yeast 84 g/l, NaCl 3.3 g/l, agar 10 g/l, wheat flour 42 g/l, apple juice 167 ml/l, and propionic acid 5 ml/l. To drive transgenes expression to the eye photoreceptor we used the line *gmr:GAL4.* REP was triggered using the *UAS:hATXN1^{82Q}* neurodegenerative transgene [51], and different UAS:modifier-gene constructs were used to test the system capability to recognize intermediate phenotypes.

External eye surface digital imaging

Digital pictures (2880x2048 pixels) of the surface of fly eyes were taken with a Nikon DS-Fi3 digital camera, in a Nikon SMZ1000 stereomicroscope equipped with a Plan Apo 1x WD70 objective. The flies were anesthetized with CO2 and their bodies immobilized on dual adhesive tape, with their heads oriented to have an eye in parallel to the microscope objective. Fly eyes were illuminated with a homogeneous fiber optic light passing through a translucid cylinder, so light rays were dispersed and didn't directly reach the eyes. Images taken with this method show a better representation of the surface retinal texture, in contrast to pictures where the light fall upon the eye and the lens reflection is captured by the camera, forming bright-spotted grids.

Additional settings include an 8x optical zoom in the stereomicroscope. A total of 308 image files were saved using NIS-Elements software in Tiff format.

ROI selection algorithm

All image analysis were performed using R programming language [52]. Eye images in RGB color space were first resized to a ¼ of their original resolution to help fit in memory and a White TopHat morphological transformation with a disc kernel of size 9 was applied using the package *EBImage* [53]. Transformed images are converted to grayscale and thresholded to keep only pixels with intensity > 0.99 quantile. Centroid of the remaining pixels is estimated using the Weiszfeld L1-median [54]. For each pixel, the Euclidean distance to the centroid is calculated, and those with distances > 0.8 quantile are discarded. A 0.90 confidence level ellipse is estimated on the final selected pixels, and its area is superposed to the original resized picture to extract the final ROI.

HOG descriptor and machine learning classifiers

Firstly, RGB ROIs were converted to grayscale maintaining the original luminance intensities. Histogram of gradients (HOG) features were extracted using the *OpenImageR* package [55]. A 5x5 cell descriptor with 5 orientations per cell was estimated in the gradient, resulting in a final 125-dimensional vector for each ROI.

SVM, DT and RF algorithms were trained on the extracted HOG features. Dataset was split into training and test sets with a 75/25 ratio using stratified random sampling to ensure class representation. The modelling strategy for all classifiers included cross-validation to assess generalization, grid search for parameter selection and performance evaluation on test set via confusion matrix, global accuracy, Kappa statistic and multiclass pairwise AUC [56]. We tested a RBF kernel SVM, DT, adaptative boosting DT and 1000-trees RF, using the R packages *kernlab, C50* and *caret* [57, 58].

Deep learning classifiers

Extracted ROIs were resized to a 224x224x3 RGB array and stored in vectorized form, resulting in a final data frame of 308x150528 dimensions. Dataset was split into training and test sets with a 75/25 ratio using stratified random sampling to ensure class representation. Two CNNs were trained on this data:

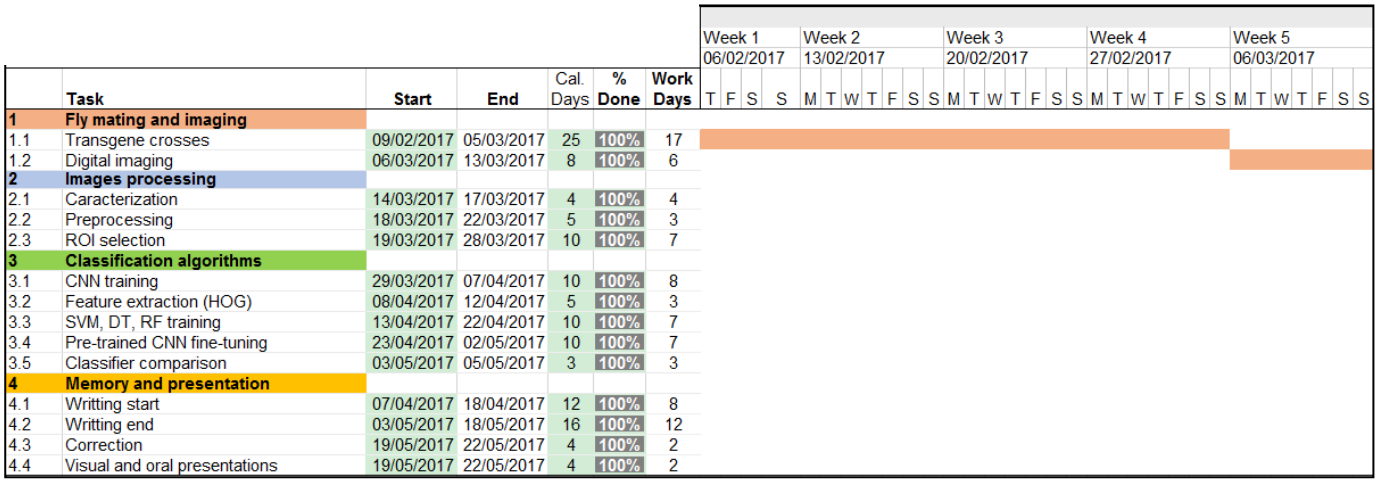
i) a simple CNN trained from scratch, with hyperbolic tangent as activation function, 2 convolutional layers, 2 pooling layers, 2 fully connected layers (200 and 6 nodes), 30 epochs and a typical *softmax* output. Each convolutional layer uses a 5×5 kernel and 20 or 50 filters, respectively. The pooling layers apply a classical "max pooling" approach. All the parameters in kernels, bias terms and weight vectors are automatically learned by back propagation with learning rate equal to 0.05 and stochastic gradient descent (SGD) optimizer to ensure the magnitude of the updates stayed small [59].

ii) a fine-tuned CNN using a *ImageNet* pretrained model with a batch-normalization network structure [60, 61], 30 epochs, a very slow learning rate (0.05) and a SGD optimizer. The final fully connected (6 nodes) and *softmax* output layers are tuned to fit the new fly eye ROIs.

For the CNN training the R package *MXNet* compiled for CPU was used [62]. Performance was assessed in terms of confusion matrix and global accuracy using *caret* package [63].

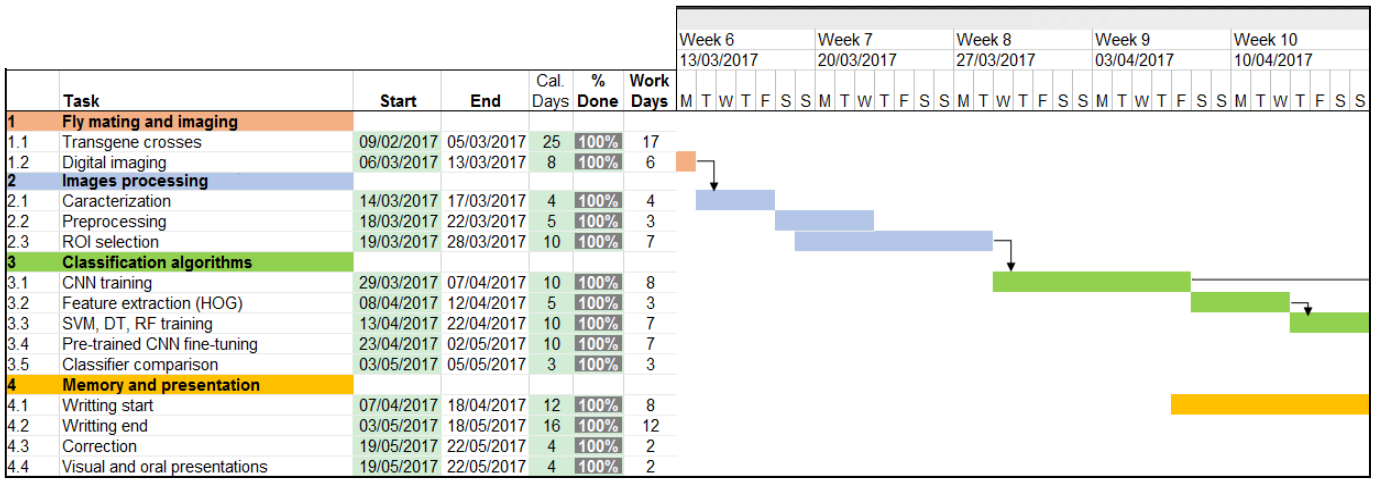
1.4 Task planning

For a comprehensive description of the resources required during this work, see *Materials and methods* section. The Gantt diagram shows the planned task calendar followed. Estimated days include installation and familiarization with the software, task resolution and mentor revision. Memory writing was planned to finish a week before the final due date to allow for mentor correction:



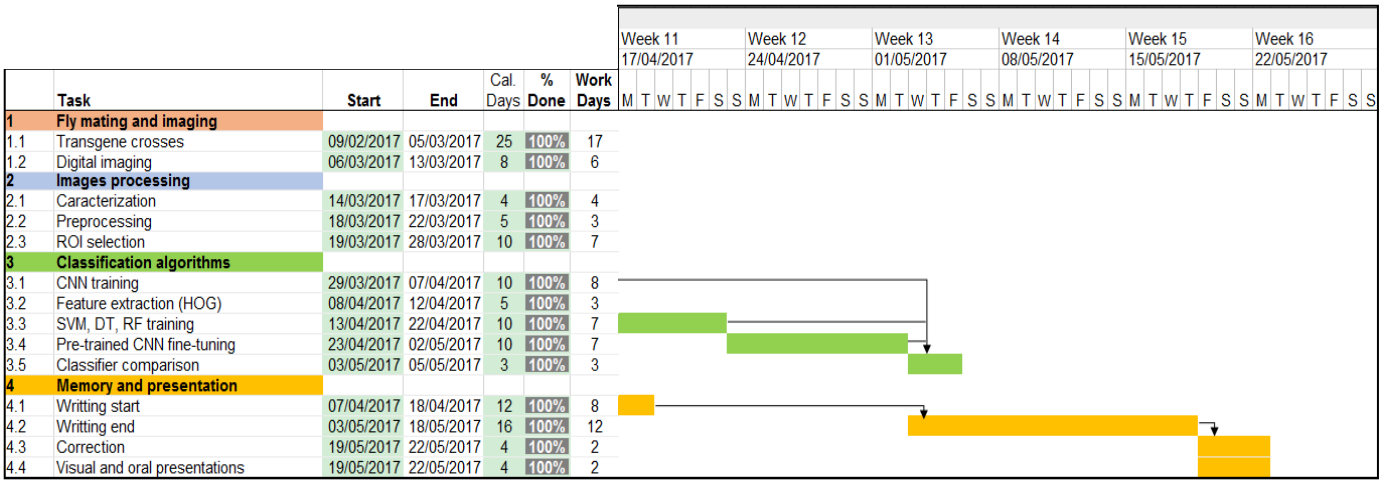
	Task	Start	End	Cal. Days	% Done	Work Days
1	**Fly mating and imaging**					
1.1	Transgene crosses	09/02/2017	05/03/2017	25	100%	17
1.2	Digital imaging	06/03/2017	13/03/2017	8	100%	6
2	**Images processing**					
2.1	Caracterization	14/03/2017	17/03/2017	4	100%	4
2.2	Preprocessing	18/03/2017	22/03/2017	5	100%	3
2.3	ROI selection	19/03/2017	28/03/2017	10	100%	7
3	**Classification algorithms**					
3.1	CNN training	29/03/2017	07/04/2017	10	100%	8
3.2	Feature extraction (HOG)	08/04/2017	12/04/2017	5	100%	3
3.3	SVM, DT, RF training	13/04/2017	22/04/2017	10	100%	7
3.4	Pre-trained CNN fine-tuning	23/04/2017	02/05/2017	10	100%	7
3.5	Classifier comparison	03/05/2017	05/05/2017	3	100%	3
4	**Memory and presentation**					
4.1	Writting start	07/04/2017	18/04/2017	12	100%	8
4.2	Writting end	03/05/2017	18/05/2017	16	100%	12
4.3	Correction	19/05/2017	22/05/2017	4	100%	2
4.4	Visual and oral presentations	19/05/2017	22/05/2017	4	100%	2

Table 1. Task planning weeks 1 to 5.



	Task	Start	End	Cal. Days	% Done	Work Days
1	**Fly mating and imaging**					
1.1	Transgene crosses	09/02/2017	05/03/2017	25	100%	17
1.2	Digital imaging	06/03/2017	13/03/2017	8	100%	6
2	**Images processing**					
2.1	Caracterization	14/03/2017	17/03/2017	4	100%	4
2.2	Preprocessing	18/03/2017	22/03/2017	5	100%	3
2.3	ROI selection	19/03/2017	28/03/2017	10	100%	7
3	**Classification algorithms**					
3.1	CNN training	29/03/2017	07/04/2017	10	100%	8
3.2	Feature extraction (HOG)	08/04/2017	12/04/2017	5	100%	3
3.3	SVM, DT, RF training	13/04/2017	22/04/2017	10	100%	7
3.4	Pre-trained CNN fine-tuning	23/04/2017	02/05/2017	10	100%	7
3.5	Classifier comparison	03/05/2017	05/05/2017	3	100%	3
4	**Memory and presentation**					
4.1	Writting start	07/04/2017	18/04/2017	12	100%	8
4.2	Writting end	03/05/2017	18/05/2017	16	100%	12
4.3	Correction	19/05/2017	22/05/2017	4	100%	2
4.4	Visual and oral presentations	19/05/2017	22/05/2017	4	100%	2

Table 2. Task planning weeks 6 to 10.



	Task	Start	End	Cal. Days	% Done	Work Days
1	**Fly mating and imaging**					
1.1	Transgene crosses	09/02/2017	05/03/2017	25	100%	17
1.2	Digital imaging	06/03/2017	13/03/2017	8	100%	6
2	**Images processing**					
2.1	Caracterization	14/03/2017	17/03/2017	4	100%	4
2.2	Preprocessing	18/03/2017	22/03/2017	5	100%	3
2.3	ROI selection	19/03/2017	28/03/2017	10	100%	7
3	**Classification algorithms**					
3.1	CNN training	29/03/2017	07/04/2017	10	100%	8
3.2	Feature extraction (HOG)	08/04/2017	12/04/2017	5	100%	3
3.3	SVM, DT, RF training	13/04/2017	22/04/2017	10	100%	7
3.4	Pre-trained CNN fine-tuning	23/04/2017	02/05/2017	10	100%	7
3.5	Classifier comparison	03/05/2017	05/05/2017	3	100%	3
4	**Memory and presentation**					
4.1	Writting start	07/04/2017	18/04/2017	12	100%	8
4.2	Writting end	03/05/2017	18/05/2017	16	100%	12
4.3	Correction	19/05/2017	22/05/2017	4	100%	2
4.4	Visual and oral presentations	19/05/2017	22/05/2017	4	100%	2

Table 3. Task planning weeks 11 to 16.

1.5 Brief product summary

The products that come out from the present work can be summarized as:

i) An automatic ROI selection algorithm from bright field images of *Drosophila* eyes, that is invariant to the method of illumination and eye position or orientation.

ii) Two classification algorithms based on Fine-tuned CNN (0.98) and HOG + rbf SVM (0.90-0.99), and a robust regularity index calculated from the model estimated class probabilities that allows direct comparison between different REP.

1.6 Brief description of the memory

The memory consists in a **Results** section, where the experiments performed will be explained and graphically portrayed; and a **Conclusion** section, where the results will be discussed in context with the expected outcome and the innovation brought to the current state of the art, as well as possible future research lines that may follow from the present work. A final **Supplementary Material** section includes the full images dataset used in this work as well as the complete R scripts.

2. Results

2.1 Detection of *Drosophila* eye from bright field images

The first step in the quantification workflow is the extraction of pixels corresponding to the fly eye from the rest of the image. One concern is that the eye is not flat but convex in morphology, so under white light only the central surface is at the camera focus. To address this issue, *white TopHat* morphological transformations were performed, defined as the difference between the input image and its opening by a structuring kernel. The opening operation involves an erosion followed by a dilation of the image, retrieving the objects of the input image that are simultaneously smaller than the structuring element and brighter than their neighbours.

Best results were obtained using a 9x9 disc-shaped kernel followed by a thresholding of pixels with intensities over the 0.99 percentile (**Fig. 7A**). Afterwards the centroid of the selected pixels was calculated as the L1-median, which is a more robust estimator of the central coordinates than the arithmetic mean. Points with Euclidean distance to the centroid greater than 0.8 percentile are more likely to lie outside the eye area and were discarded (**Fig. 7B**). A 0.90 confidence ellipse calculated on the selected pixels conforms the area of the final ROI, which was superposed and cropped from the original eye image (**Fig. 7C**). As can be appreciated in the example images, the method is invariant to the location of the eye within the image. Various combinations of the thresholds and centroid estimator were tested (**Fig. 8A**).

The proposed segmentation method also works well on bright field images where the light falls directly onto the ommatidium and the eye is seen as a

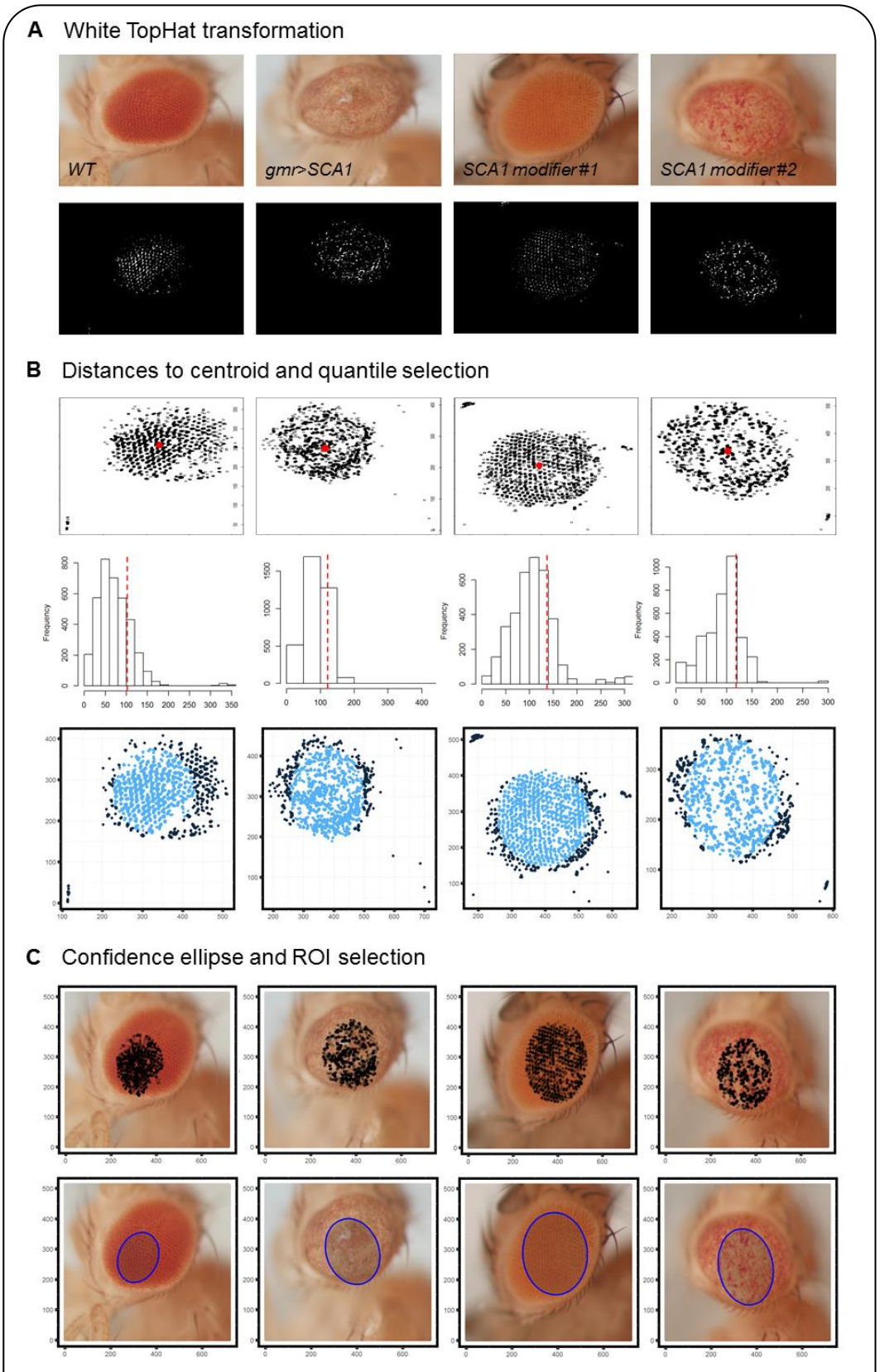


Figure 7. Drosophila eye detection strategy. Representative examples of healthy and REP eyes are shown. **A** Morphological transformation and intensity thresholding extract pixels mostly contained within the eye. **B** Euclidean distance to the centroid (red dot) and frequency histogram with 0.80 percentile marked as a red dotted line. Dark blue points are discarded as potential pixel outliers outside the eye limit. **C** Selected pixels are superposed to the original image and those within the area of a 0.90 confidence ellipse are extracted as the final ROI (blue shaded ellipse).

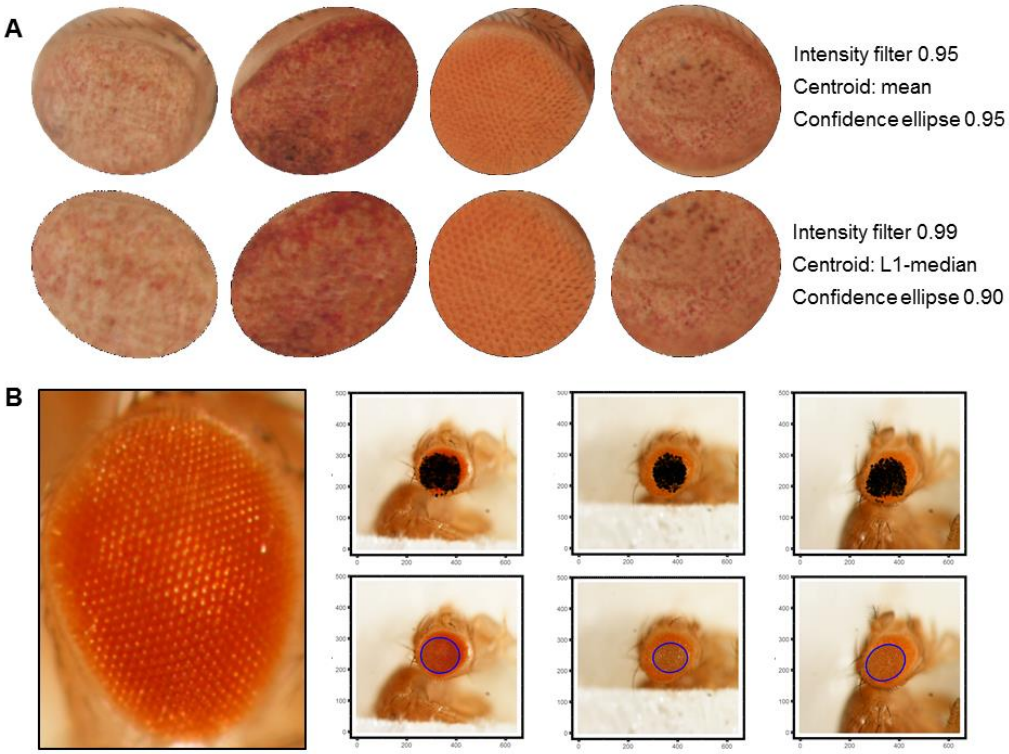


Figure 8. ROI selection optimization and extensibility. A L1-median centroid alongside stricter thresholds improve the eye area detection. **B** Bright-spotted fly eye images can also be successfully segmented using this method.

region enriched in reflection spots (**Fig. 8B**). The full array of final ROIs is represented in **Fig. S1**.

2.2 Histogram of Oriented Gradients features extraction

Prior to the HOG extraction, ROIs were transformed to grayscale preserving the luminance of the original RGB image. ROIs were divided into rectangular cells of 5x5 pixels, and for each cell an orientation histogram of 5 bins covering a gradient range of 0º to 180º is computed (**Fig. 9**). That means that a 125-dimensional feature vector is extracted for each ROI, representing the frequency of a certain gradient within the image (**Fig. 9**). The matrix formed by the 125-D vectors of all ROIs conforms the input for the machine learning classifiers.

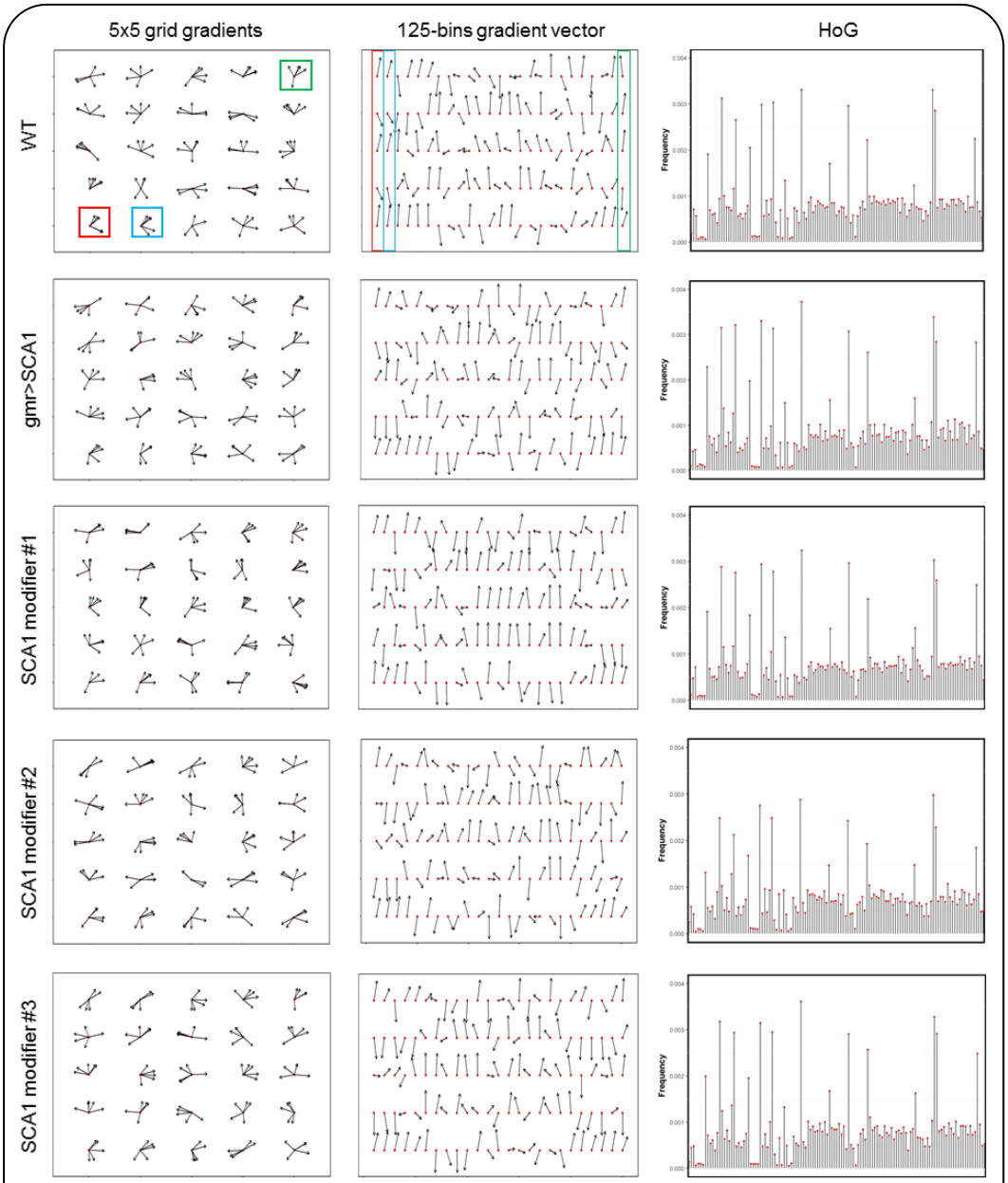


Figure 9. HOG feature extraction. For each phenotype mean values are represented. Each cell in the 5x5 grid contains the 5 major orientations within that patch. All gradients are concatenated in a 125-D vector (coloured blocks indicate between plots correspondence), and its frequency in the cropped ROIs is calculated, resulting in the final HOG. The SCA1 modifier#1 HOG is more similar to the WT than pure SCA1, and the inverse holds for modifier#3.

2.3 Machine learning classifiers comparison

SVM with RBF kernel, DT, AdaBoost DT and 1000-trees RF algorithms were tested on the HOG features extracted. Sample consisted in 308 fly eye images distributed in 5 different phenotype classes with varying degrees of retinal surface degeneration. Data was split using stratified

random sampling in 75% training and 25% test set. Optimal parameters for each classifier were found using 10-fold cross-validation on the training set. **Table 4** shows the confusion matrix and **Table 5** represents the global accuracy, Kappa statistic and multiclass AUC, defined as the average AUC of class pairwise-comparisons (**Fig. 10A**), calculated on the test data. Pairwise ROC plots are represented in **Fig. 10B**.

Color scheme: **SVM** **DT** **BoostDT** **RF**

Predicted \ Reference	WT	gmr>SCA	SCA Modifier#1	SCA Modifier#2	SCA Modifier#3
WT	20 16 20 20	0 2 2 2	1 4 3 1	0 0 0 1	0 4 3 1
gmr>SCA	0 2 0 0	11 7 7 8	0 2 0 0	0 1 1 0	0 2 3 0
SCA Modifier#1	0 0 0 0	0 0 2 0	12 7 10 11	0 0 0 0	0 0 0 0
SCA Modifier#2	0 1 0 0	0 0 0 0	0 0 0 1	14 12 11 12	0 3 0 0
SCA Modifier#3	0 1 0 0	0 2 0 1	0 0 0 0	1 2 3 2	16 7 10 15

Table 4. Machine learning classifiers confusion matrix.

Metric \ Classifier	SVM RBF	DT	AdaBoostDT	1000 RF
Accuracy	0.973 (0.907-0.997)	0.653 (0.535-0.760)	0.773 (0.662-0.862)	0.880 (0.784-0.944)
Kappa	0.966	0.560	0.711	0.847
Multiclass AUC	0.978	0.665	0.763	0.906

Table 5. Machine learning performance evaluation metrics on test data. Best results are shaded in grey.

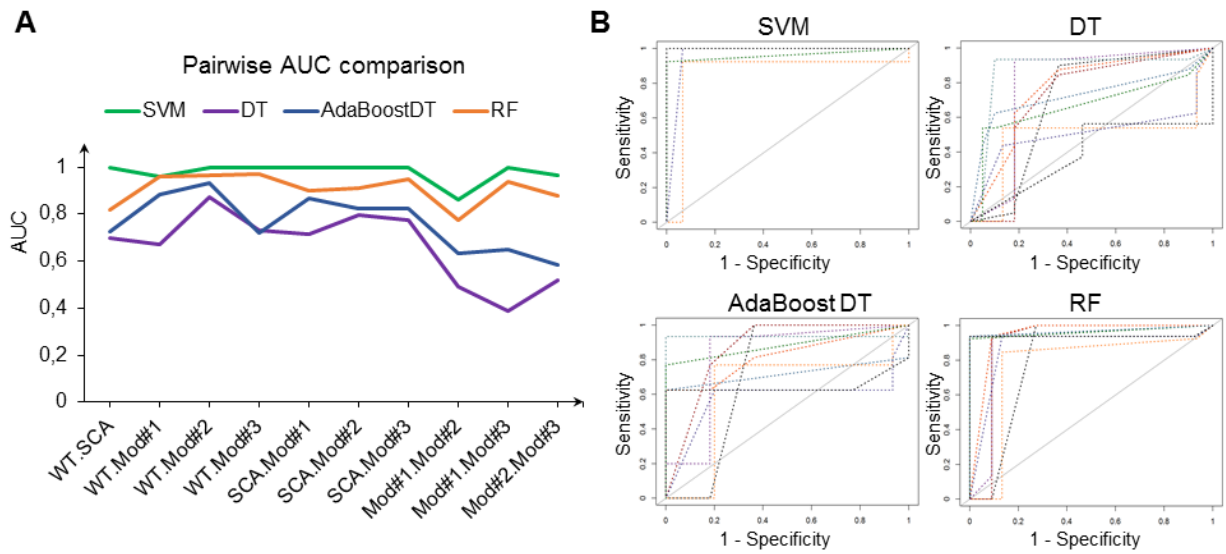


Figure 10. Class pairwise AUC and ROC. A SVM with RBF kernel outperforms the other classifiers in all comparisons. **B** ROC plots corresponding to the AUCs in **A**.

In general, the four classifiers performed fairly well on unseen data. Both DT algorithms fell on the low spectrum either in accuracy and AUC (<0.80), whereas RF achieved a remarkable AUC of 0.90. Overall, SVM accomplished the best results among all the error metrics evaluated, with a global accuracy of 0.97 (0.90-0.99), Kappa of 0.96 and a multiclass AUC of roughly 0.98. Parameters that yield these results were a Gaussian kernel (radial basis function), a cost penalty = 1 and sigma = 0.005. WT eyes were the most correctly classified phenotype by the four methods.

From the SVM estimated class probabilities it is possible to derive a regularity index, IREG, that ranges from 0 (total degeneration) to 1 (healthy eye) [19]. It is based on the knowledge of degeneration intensity of the phenotypes involved in the model: WT < Modifier#1 < Modifier#2 < Modifier#3 < SCA, from absence to full presence of REP. IREG is then calculated as:

$$\boldsymbol{IREG} = \frac{\mathbf{4} \cdot \boldsymbol{P}(\boldsymbol{eye} = \boldsymbol{WT}) + \mathbf{3} \cdot \boldsymbol{P}(\boldsymbol{eye} = \boldsymbol{Mod}\#\mathbf{1}) + \mathbf{2} \cdot \boldsymbol{P}(\boldsymbol{eye} = \boldsymbol{Mod}\#\mathbf{2}) + \boldsymbol{P}(\boldsymbol{eye} = \boldsymbol{Mod}\#\mathbf{3})}{\mathbf{4}}$$

When estimated on the test data, IREG distribution fits to the expected values and properly reflects the intrinsic variability of the fly model and the rough eye phenotype (**Fig. 11**).

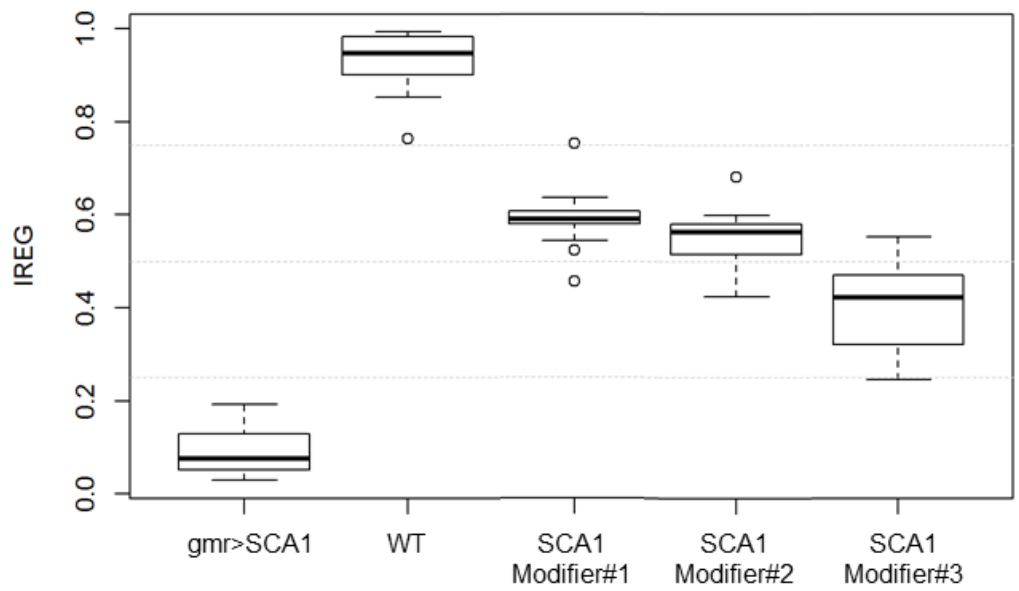


Figure 11. IREG boxplots. WT and SCA1 eyes show opposing IREG values and no distribution overlap. SCA modifiers show intermediate REPs and slight distribution overlapping, but the median and central boxes differentiate them. Grey dotted lines mark 0.25, 0.5 and 0.75 IREG values.

2.4 Deep learning classifiers

In contrast with the previous machine learning classifiers, that needed a transformation of the cropped images into an enriched feature space (HOG), deep learning algorithms use directly the ROIs pixel intensity arrays as input. Features are automatically learned during the learning process, from gross edge and contour detection to fine details discrimination the deeper the layer is in the network.

Two different strategies were followed to train the deep networks: learning a *de novo* model and transfer learning. The latter approach takes advantage of CNNs pre-trained on very large samples, which is especially well-suited for classifying new small datasets, as the majority of patterns

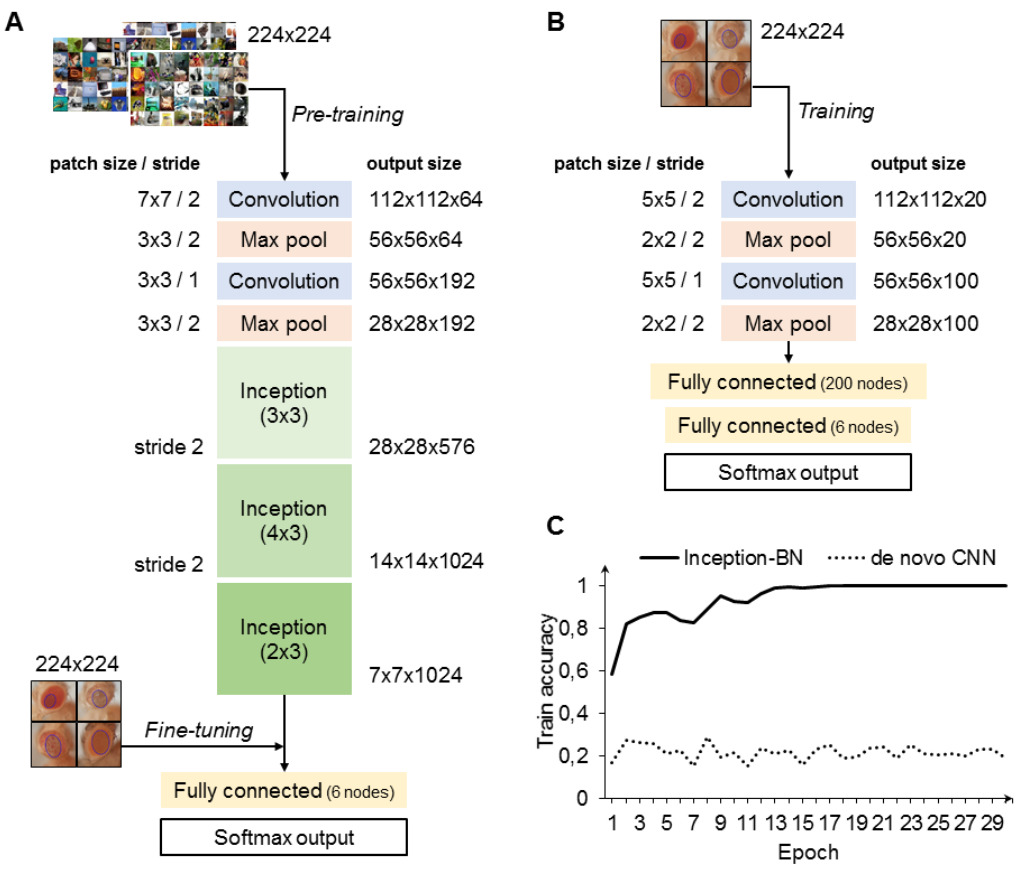


Figure 12. CNN architectures and learning curve. A Inception-BN is a 15-layered CNN pre-trained on thousands of natural images. A 6-nodes fully connected layer and softmax output are trained with new fly eye images on top of the Inception blocks. **B** *De novo* CNN with 5 layers and a 6-nodes fully connected and softmax output. **C** Train accuracy in the pre-trained model starts pretty high and quickly rises in the first few epochs. In contrast, *de novo* model accuracy remains low and fluctuates around the initial value with no apparent signs of improving.

and motifs commonly found in images are already known to the model internal representation. Thus, it is only necessary to fine tune the final layers to learn the particularities of the new images, which is many times faster than training a CNN from scratch and doesn't require thousands of labelled examples. The architectures of both *de novo* and pre-trained CNN are depicted in **Fig. 12A-B**. The pre-trained model chosen uses the *Inception* structure, characterized by including mini-batch normalization (BN) for each training epoch, which allows for high learning rates and acts as regularizer. In comparison, the *de novo* CNN is much shallower due to computational constraints.

Color scheme: **Inception-BN** ***de novo* CNN**

Predicted \ Reference	WT	gmr>SCA	SCA Modifier#1	SCA Modifier#2	SCA Modifier#3
WT	**20** 20	**0** 11	**0** 13	**0** 15	**0** 16
gmr>SCA	**0** 0	**11** 0	**0** 0	**0** 0	**0** 0
SCA Modifier#1	**0** 0	**0** 0	**13** 0	**0** 0	**0** 0
SCA Modifier#2	**0** 0	**0** 0	**0** 0	**14** 0	**0** 0
SCA Modifier#3	**0** 0	**0** 0	**0** 0	**1** 0	**16** 0

Table 6. CNN classifiers confusion matrix.

Metric \ Classifier	Inception-BN	*de novo* CNN
Accuracy	0.986	0.146
Multiclass AUC	0.997	0.5

Table 7. CNN performance evaluation metrics on test data. Best results are shaded in grey.

Accuracy during the training phase is usually a reliable indicator of a CNN capability to learn discriminative features with the available sample size (**Fig. 12C**). *De novo* CNN curve is a clear sign that either the network is not deep enough or the training sample is too small for the complexity of the classification task at hand. One major concern with the pre-trained Inception-BN was the possibility that the network was memorizing the training set, given the few epochs it needed to achieve perfect training accuracy. Performance assessment in an independent test set of unseen

images gave impressive accuracy and AUC values closer to 1 (**Tables 6, 7**), refuting the possibility of overfitting. The CNN trained from scratch predicted every eye to be WT, indicative of the weak classifying rule learned in training. Thus, transfer learning with pre-trained Inception-BN model is arguably the top performer classifier among all the methods tested in this work.

3. Conclusion

The present work provides a novel and fully automated method to quantitatively assess the degeneration intensity of the fruit flies compound eye, using reliable and robust state-of-the-art machine learning techniques. This new method consists in the acquisition of bright-field images from the external retinal surface, the extraction of a ROI enriched in information of the eye morphology and a classification algorithm built around a pre-trained deep learning algorithm, fine-tuned to the particularities of REP degeneration images. Additionally, a model based on the combination of HOG features extraction and Gaussian kernel SVM offered performance on par with the CNN and in fact required much less training time.

In contrast with previous quantification approaches [16, 19, 20], this method does not rely on patterns created by light reflecting in the eye lenses and can be applied to extract ROIs from a variety of illumination conditions. The proposed pipeline can process a 2880x2048 resolution image in less than 10 seconds, and batches of 50 images in approximately 90 seconds, depending on the hardware it runs on.

One of the major goals of this work was to analyse the potentiality of deep learning techniques to extract feature maps directly from the raw pixel array, that could be fed as input to other conventional machine learning algorithms (*i.e,* SVM). Due to computational constraints, it was not possible to tune up GPU-compiled versions of the software utilized, and the prohibitive CPU computational time and memory usage in its absence made the evaluation of the former objective not feasible. HOG was chosen as an alternative descriptor given its successful application in object

detection [32, 33, 36], and ended up resulting in a surprisingly powerful classifier in combination with conventional SVM.

Another drawback of CNNs is the staggering amount of labelled training examples they need to learn adequate internal representations of image patterns and motifs. Although sample size in *Drosophila* experiments ranks among the largest of any animal model in genetics, it is still a titanic effort to go beyond one thousand images in a typical fruit fly assay. This limitation affected the performance of the *de novo* CNN, which led to the alternative strategy of transfer learning. Using *Inception-BN,* a CNN pre-trained on millions of natural images [61], proved to be a well-thought solution that definitely opens up the field of deep learning to small scale biology setups.

Future lines of work include developing the fly eye detection algorithm further to make it extensible to other image capturing techniques (*i.e.,* SEM). A more immediate priority is the creation of a user-friendly Shiny application [64] that will allow the researcher to tweak the ROI selection parameters to fit the peculiarities of its own dataset prior to the degeneration quantification. Depending on the particular hardware settings, the app may also offer the user the possibility to train its own SVM or deep learning model.

The highlighted strengths of the proposed framework will enhance the sensitivity of high-throughput genetic screens based on rough eye phenotypes and demonstrates fly eye imaging is a top-notch technique for quantitative modelling human diseases.

4. Glossary

AdaBoost	Adaptative Boosting
AUC	Area under the ROC
BN	Batch normalization
CNN	Convolutional Neural Network
DT	Decision Tree
gmr	glass multimer reporter
HOG	Histogram of Oriented Gradients
IREG	Regularity index
MLP	Multilayer Perceptron
NN	Neural Network
PolyQ	Polyglutaminated
RBF	Radial Basis Function
RGB	Red, Green, Blue (colorspace)
ReLU	Rectified Linear Unit
REP	Rough eye phenotype
RF	Random Forest
ROC	Receiver Operating Curve
ROI	Region of Interest
SCA	Spinocerebellar ataxia
SEM	Scanning electron micrograph
SVM	Support Vector Machine
Tanh	Hyperbolic tangent
UAS	Upstream Activating Sequence
WT	Wild type

5. References

1. Reiter, L.T., Potocki, L., Chien, S., Gribskov, M., and Bier, E. (2001). A systematic analysis of human disease-associated gene sequences in Drosophila melanogaster. Genome Res *11*, 1114-1125.
2. St Johnston, D. (2002). The art and design of genetic screens: Drosophila melanogaster. Nat. Rev. Genet. *3*, 176-188.
3. Wangler, M.F., Yamamoto, S., and Bellen, H.J. (2015). Fruit flies in biomedical research. Genetics *199*, 639-653.
4. Thomas, B.J., and Wassarman, D.A. (1999). A fly's eye view of biology. Trends in genetics : TIG *15*, 184-190.
5. Thaker, H.M., and Kankel, D.R. (1992). Mosaic analysis gives an estimate of the extent of genomic involvement in the development of the visual system in Drosophila melanogaster. Genetics *131*, 883-894.
6. Treisman, J.E. (2013). Retinal differentiation in Drosophila. Wiley interdisciplinary reviews. Developmental biology *2*, 545-557.
7. Garcia-Lopez, A., Llamusi, B., Orzaez, M., Perez-Paya, E., and Artero, R.D. (2011). In vivo discovery of a peptide that prevents CUG-RNA hairpin formation and reverses RNA toxicity in myotonic dystrophy models. Proceedings of the National Academy of Sciences of the United States of America *108*, 11866-11871.
8. Lenz, S., Karsten, P., Schulz, J.B., and Voigt, A. (2013). Drosophila as a screening tool to study human neurodegenerative diseases. Journal of neurochemistry *127*, 453-460.
9. He, B.Z., Ludwig, M.Z., Dickerson, D.A., Barse, L., Arun, B., Vilhjalmsson, B.J., Jiang, P., Park, S.Y., Tamarina, N.A., Selleck, S.B., et al. (2014). Effect of genetic variation in a Drosophila model of diabetes-associated misfolded human proinsulin. Genetics *196*, 557-567.
10. Ambegaokar, S.S., Roy, B., and Jackson, G.R. (2010). Neurodegenerative models in Drosophila: polyglutamine disorders, Parkinson disease, and amyotrophic lateral sclerosis. Neurobiology of disease *40*, 29-39.
11. Roederer, K., Cozy, L., Anderson, J., and Kumar, J.P. (2005). Novel dominant-negative mutation within the six domain of the conserved eye specification gene sine oculis inhibits eye development in Drosophila. Developmental dynamics : an official publication of the American Association of Anatomists *232*, 753-766.
12. Bilen, J., and Bonini, N.M. (2007). Genome-wide screen for modifiers of ataxin-3 neurodegeneration in Drosophila. PLoS genetics *3*, 1950-1964.
13. Cukier, H.N., Perez, A.M., Collins, A.L., Zhou, Z., Zoghbi, H.Y., and Botas, J. (2008). Genetic modifiers of MeCP2 function in Drosophila. PLoS genetics *4*, e1000179.

14. Jonshon, R.I., and Cagan, R.L. (2009). A Quantitative Method to Analyze Drosophila Pupal Eye Patterning. PLoS ONE *4*.
15. Jenny, A. (2011). Preparation of adult Drosophila eyes for thin sectioning and microscopic analysis. Journal of visualized experiments : JoVE.
16. Caudron, Q., Lyn-Adams, C., Aston, J.A.D., Frenguelli, B.G., and Moffat, K.G. (2013). Quantitative assessment of ommatidial distortion in Drosophila melanogaster. Dros. Inf. Serv. *96*, 136-144.
17. Mishra, M., and Knust, E. (2013). Analysis of the Drosophila compound eye with light and electron microscopy. Methods in molecular biology *935*, 161-182.
18. Song, W., Smith, M.R., Syed, A., Lukacsovich, T., Barbaro, B.A., Purcell, J., Bornemann, D.J., Burke, J., and Marsh, J.L. (2013). Morphometric analysis of Huntington's disease neurodegeneration in Drosophila. Methods in molecular biology *1017*, 41-57.
19. Diez-Hermano, S., Valero, J., Rueda, C., Ganfornina, M.D., and Sanchez, D. (2015). An automated image analysis method to measure regularity in biological patterns: a case study in a Drosophila neurodegenerative model. Molecular neurodegeneration *10*, 9.
20. Iyer, J., Wang, Q., Le, T., Pizzo, L., Grönke, S., Ambegaokar, S., Imai, Y., Srivastava, A., Troisí, B., Mardon, G., et al. (2016). Quantitative Assessment of Eye Phenotypes for Functional Genetic Studies Using Drosophila melanogaster. G3 (Bethesda) *6*, 1427-1437.
21. Bishop, C.M. (2006). Pattern recognition and machine learning. Springer.
22. Sommer, C., and Gerlich, D.W. (2013). Machine learning in cell biology - teaching computers to recognize phenotypes. Journal of cell science *126*, 5529-5539.
23. Chessel, A. (2017). An Overview of data science uses in bioimage informatics. Methods *115*, 110-118.
24. Tarca, A., Carey, V.J., Chen, X.W., Romero, R., and Draghici, S. (2007). Machine learning and its applications to biology. PLOS Comput Biol *3*.
25. Castoreno, A.B., Smurnyy, Y., Torres, A.D., Vokes, M.S., Jones, T.R., Carpenter, A.E., and Eggert, U.S. (2010). Small molecules discovered in a partway screen target the Rho pathway in cytokinesis. Nat Chem Biol *6*, 457-463.
26. Millard, B.L., Niepel, M., Menden, M.P., Muhlich, J.L., and Sorger, P.K. (2011). Adaptative informatics for multifactorial and high-content biological data. Nat Methods *8*, 487-492.
27. Ben-Hur, A., Ong, C.S., Sonnenburg, S., Schölkopf, B., and Rätsch, G. (2008). Support vector machines and kernels for computational biology. PLOS Comput Biol *4*.
28. Reiter, L.T., Rinner, O., Picotti, P., Hüttenhaim, R., Beck, M., Brusniak, M.Y., Hengartner, M.O., and Aebersold, R. (2011). mProphet: automated data processing and statistical validation for large-scale SRM experiments. Nat Methods *8*, 430-435.

29. Mikolajczyk, K., and Cordelia, S. (2005). A performance evaluation of local descriptors. IEEE Transactions on Pattern Analysis and Machine Intelligence *27*.
30. Liu, S., Mundra, P.A., and Rajapakse, J.C. (2011). Features for cells and nuclei classification. Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Annual Conference *2011*, 6601-6604.
31. LLowe, D.G. (2004). Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision *60*, 91-110.
32. Dalal, N., and Triggs, B. (2005). Histograms of Oriented Gradients for Human Detection. IEEE Computer Vision and Pattern Recognition *Computer Society Conference*
33. Orrite, C., Gañán, A., and Rogez, G. (2009). HOG based decision tree for facial expression classification. n: Araujo H, Mendonça A, Pinho A, Torres M (eds) Pattern Recognition and Image Analysis *5524*, 176-183.
34. Dahmane, M., and Meunier, J. (2011). Emotion recognition using dynamic grid-based HoG features. IEEE Automatic Face & Gesture Recognition and Workshops (FG).
35. Doan, T., and Poulet, F. (2011). Large-scale Image Classification: Fast Feature Extraction and SVM Training. Advances in Knowledge Discovery and Management *527*, 152-172.
36. Li, S., Xiabi, L., Ling, M., Chunwu, Z., Xinming, Z., and Yanfeng, Z. (2012). Using HOG-LBP Features and MMP Learning to Recognize Imaging Signs of Lung Lesions 25th IEEE International Symposium on Computer-Based Medical Systems (CBMS).
37. Mao, Y., Liu, H., Ye, R., Shi, Y., and Song, Z. (2014). Detection and segmentation of virus plaque using HOG and SVM: toward automatic plaque assay. Bio-medical materials and engineering *24*, 3187-3198.
38. Rodriguez, J.D., Perez, A., and Lozano, J.A. (2009). Sensitivity Analysis of k-Fold Cross Validation in Prediction Error Estimation. IEEE Transactions on Pattern Analysis and Machine Intelligence *32*, 569-575.
39. Dobbin, K.K., and Simon, R.M. (2011). Optimally splitting cases for training and testing high dimensional classifiers. BMC medical genomics *4*, 31.
40. Schroff, F., Criminisi, A., and Zisserman, A. (2008). Object Class Segmentation using Random Forests. Proc. British Machine Vision Conference (BMVC).
41. Giacinto, G., and Roli, F. (2001). Design of effective neural network ensembles for image classification purposes. Image and Vision Computing *19*, 699-707.
42. LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. Nature *521*, 436-444.
43. Po-Hsien, L., Shun-Feng, S., Ming-Chang, C., and Chih-Ching, H. (2015). Deep Learning and its Application to general Image Classification.

International Conference on Informative and Cybernetics for Computational Social Systems (ICCSS).

44. Angermueller, C., Parnamaa, T., Parts, L., and Stegle, O. (2016). Deep learning for computational biology. Molecular systems biology *12*, 878.
45. Chen, C.L., Mahjoubfar, A., Tai, L.C., Blaby, I.K., Huang, A., Niazi, K.R., and Jalali, B. (2016). Deep Learning in Label-free Cell Classification. Scientific reports *6*, 21471.
46. Kraus, O.Z., Ba, J.L., and Frey, B.J. (2016). Classifying and segmenting microscopy images with deep multiple instance learning. Bioinformatics *32*, i52-i59.
47. Spanhol, F.A., Oliveira, L.S., Petitjean, C., and Heutte, L. (2016). Breast Cancer Histopathological Image Classification using Convolutional Neural Networks. International Joint Conference on Neural Networks (IJCNN).
48. Yang, X., Yeo, S.-Y., Hong, J.M., Wong, S.T., Tang, W.T., Wu, Z.Z., Lee, G., Chen, S., Ding, V., Pang, B., et al. (2016). A Deep Learning Approach for Tumor Tissue Image Classification.
49. Zhang, W., Li, R., Zeng, T., Sun, Q., Kumar, S., Ye, J., and Ji, S. (2015). Deep Model Based Transfer and Multi-Task Learning for Biological Image Analysis. 1475-1484.
50. Zou, N., Baydogan, M., Zhu, Y., Wang, W., Zhu, J., and Li, J. (2015). A Transfer Learning Approach for Predictive Modeling of Degenerate Biological Systems. Technometrics : a journal of statistics for the physical, chemical, and engineering sciences *57*, 362-373.
51. Fernandez-Funez, P., Nino-Rosales, M.L., de Gouyon, B., She, W.C., Luchak, J.M., Martinez, P., Turiegano, E., Benito, J., Capovilla, M., Skinner, P.J., et al. (2000). Identification of genes that modify ataxin-1-induced neurodegeneration. Nature *408*, 101-106.
52. R Core Team (2016). R: A language and environment for statistical computing. R Foundation for Satistical Computing, Vienna, Austria.
53. Pau, G., Fuchs, F., Sklyar, O., Boutros, M., and Huber, W. (2010). EBImage - an R package for image processing with applications to cellular phenotypes. Bioinformatics *26*, 979-981.
54. Vardi, Y., and Cun-Hui, Z. (2000). The multivariate L1-median and associated data depth. PNAS *97*, 1423-1426.
55. Mouselimis, L. (2017). OpenImageR: An Image Processing Toolkit. R package version 1.0.5.
56. Ferri, C., Hernandez-Orallo, J., and Salido, M.A. (2003). Volume Under the ROC Surface for Multi-class Problems. Exact Computation and Evaluation of Approximations. Proc Of 14th European Conference on Machine Learning *108-120*.
57. Karatzoglou, A., Smola, A., Hornik, K., and Zeileis, A. (2004). kernlab - An S4 Package for Kernel Methods in R. Journal of Statistical Software *11*, 1-20.

58. Kuhn, M., Weston, S., Coulter, N., and Culp, M. (2015). C50: C5.0 Decision Trees and Rule-Based Models. .
59. Bottou, L. (2010). Large-Scale Machine Learning with Stochastic Gradient Descent. Proceedings of COMPSTAT'2010. *Physyca-Verlag HD*.
60. Deng, J.,et al. (2009). Imagenet: A large-scale hierarchical image database. Computer Vision and Pattern Recognition *CVPR 2009*.
61. Ioffe, S., and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv:1502.03167.
62. Tianqi, C., et al. (2015). MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems. arXiv:1512.01274.
63. Kuhn, M. (2012). caret: Classification and Regression Training. R package version 5.15-044. Contributions from Jed Wing, Steve Weston, Andre Williams, Chris Keefer and Allan Engelhardt
64. Winston, C., Joe, C., Allaire, J.J., Yihui, X., and McPherson, J. (2017). Shiny: Web Applicaion Framework for R. R package version 1.0.2.

6. Supplementary material

6.1 Supplementary figures

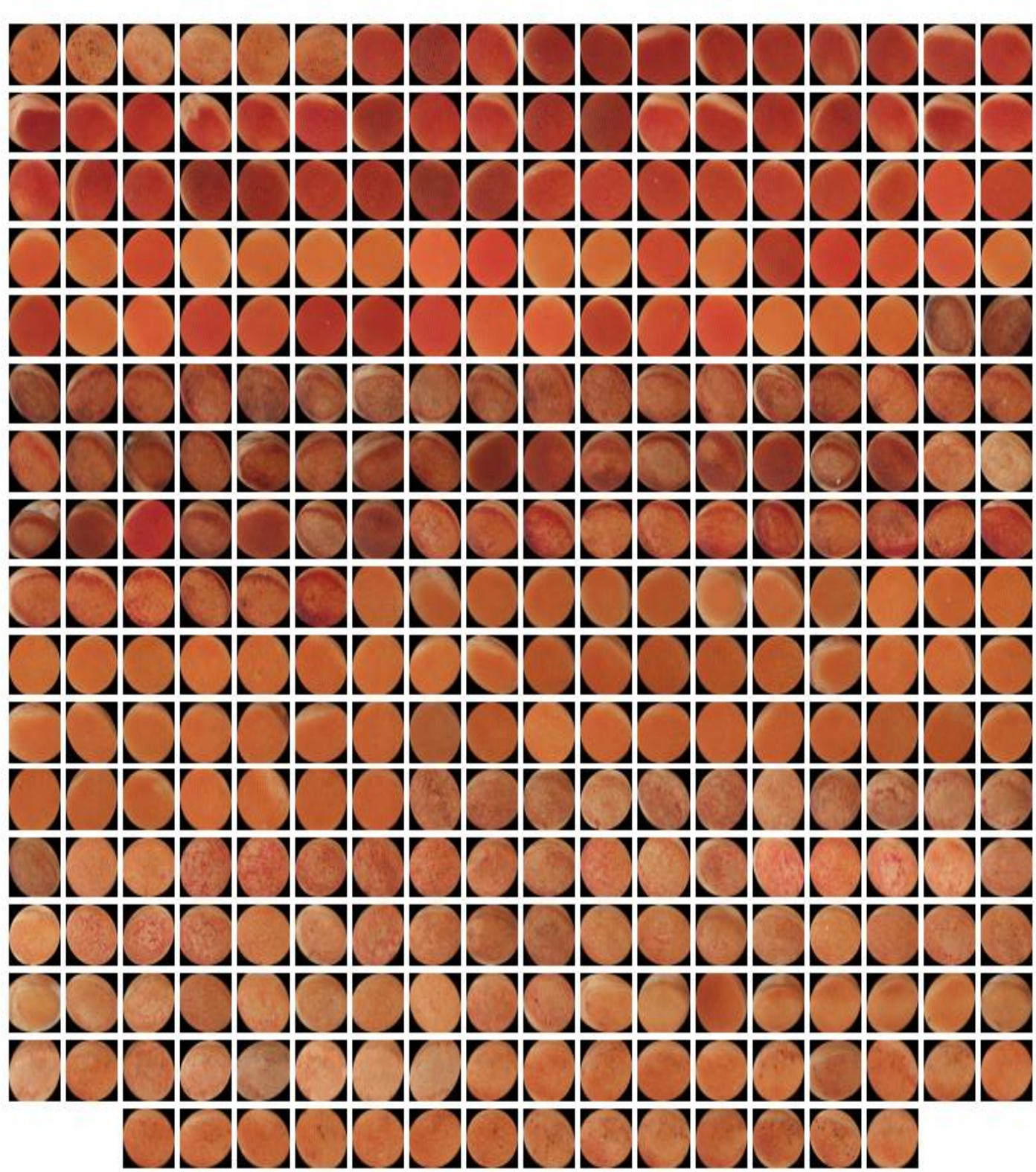


Figure S1. Full dataset of final ROIs. Indirect dispersed light illumination method (308 pictures). Sample used for training and testing the models through stratified random splitting.

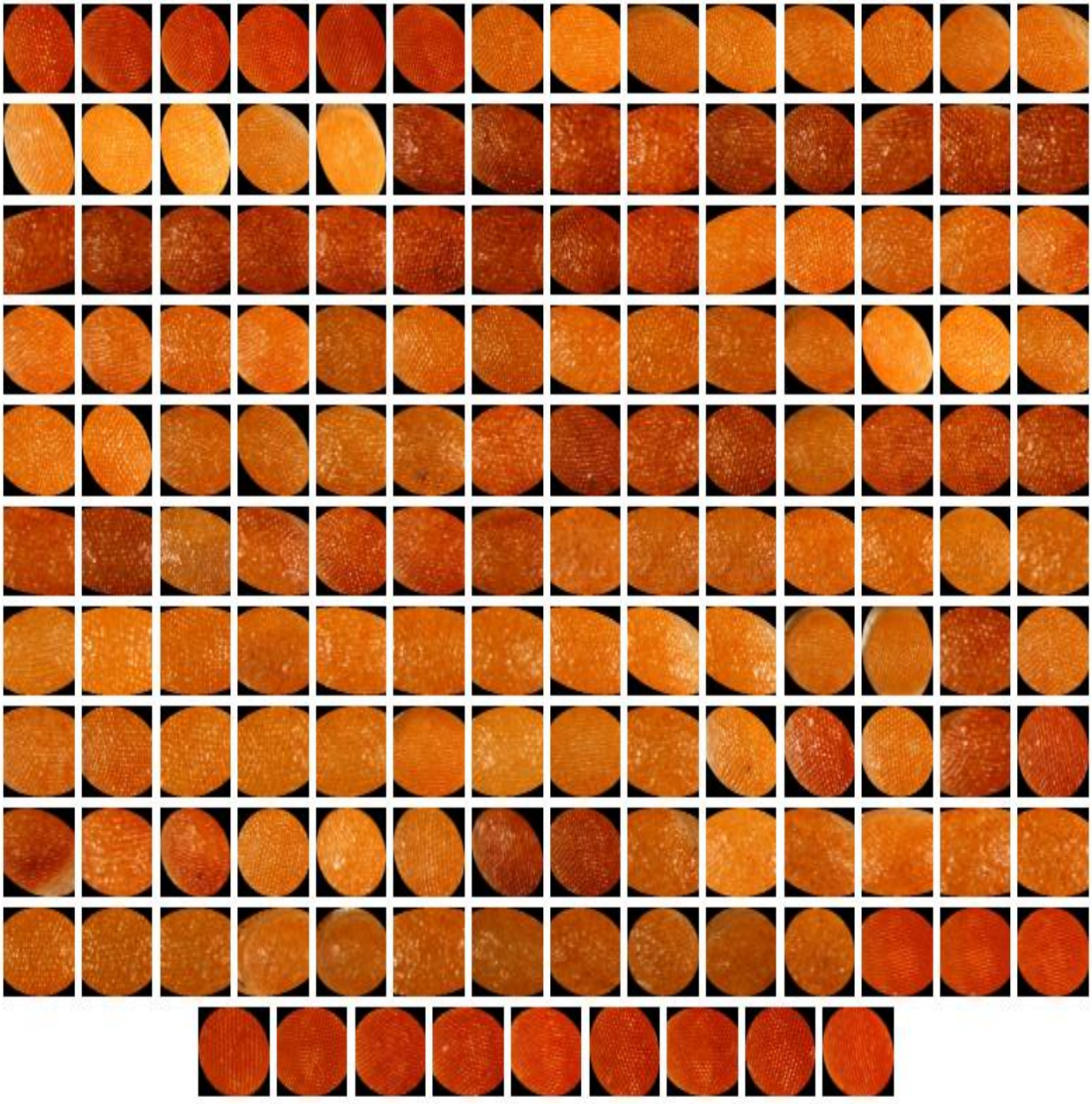


Figure S2. Direct light illumination method ROIs. 149 surface pictures of REPs similar to the ones used to train the models.

6.2 R Code

Image segmentation

```
#########################
# 1. Image segmentation #
#########################

#############
# Functions #
#############

# ggplot theme to be used
plotTheme <- function() {
  theme(
    panel.background = element_rect(
      size = 3,
      colour = "black",
      fill = "white"),
    axis.ticks = element_line(
      size = 2),
    panel.grid.major = element_line(
      colour = "gray80",
      linetype = "dotted"),
    panel.grid.minor = element_line(
      colour = "gray90",
      linetype = "dashed"),
    axis.title.x = element_text(
      size = rel(1.2),
      face = "bold"),
    axis.title.y = element_text(
      size = rel(1.2),
      face = "bold"),
    plot.title = element_text(
      size = 20,
      face = "bold",
      hjust = 0.5)
  )
}

# resize function
resFunc <- function(x) {
  resize(x, dim(x)[1]/4)
}


# Store RGB into data frame
RGBintoDF <- function(x) {
  imgDm <- dim(x)
  #Assign original image RGB channels to data frame
  imgOri <- data.frame(
  x = rev(rep(imgDm[1]:1, imgDm[2])),
```

```
  y = rev(rep(1:imgDm[2], each = imgDm[1])),
  R = as.vector(x[,,1]),
  G = as.vector(x[,,2]),
  B = as.vector(x[,,3])
  )
  return(imgOri)
}

# Store Gray channel into data frame
GintoDF <- function(x) {
  imgDm <- dim(x)
  #Assign original image RGB channels to data frame
  imgOri <- data.frame(
  x = rev(rep(imgDm[1]:1, imgDm[2])),
  y = rev(rep(1:imgDm[2], each = imgDm[1])),
  G = as.vector(x)
  )
  return(imgOri)
}

# White TopHat morphological transform
wTopHat <- function(x, y, z) {
  imgGrey <- channel(x, "green")
  imgTop <- whiteTopHat(imgGrey,
                        kern=makeBrush(y, shape = z))
}

# Display images
dispImg <- function(x) {
  display(x, method="raster")
}

# Select pixels with intensity > 0.99 quantile
dispImgT <- function(x, y) {
  display(x > quantile(x, y), method="raster")
}

# Add ellipse to plot
ellPlot <- function(z, w) {
  p <- ggplot(z, aes(x, y)) +
  geom_point() +
  labs(title = "Selected pixels") +
  stat_ellipse(level=w) +
  plotTheme()
  return(p)
}
```

```
# Create image from ROI
roitoImg <- function(z) {
  R <- xtabs(R~x+y, z)
  G <- xtabs(G~x+y, z)
  B <- xtabs(B~x+y, z)
  imgROI <- rgbImage(R, G, B)
  return(imgROI)
}

##################
# Load packages #
##################
library(lattice)
library(ggplot2)
library(sp) #for points.in.polygon
library(raster) #for pointDistance
library(tiff)
library(EBImage)
library(Gmedian)


#########################################################
# Morphological transformation and centroid distances #
#########################################################

## Read images and transform

# Read original images
tiffFiles <- list.files(pattern="*tif$", full.name=F)
tiffList <- lapply(tiffFiles, readImage)

# Resize to fit memory
tiffRes <- lapply(tiffList, resFunc)
rm(tiffList); invisible(gc()) # free memory space

# Assign resized images RGB channels to data frames
tiffOri <- lapply(tiffRes, RGBintoDF)

# White TopHat morphological transform
tiffTop <- lapply(tiffRes, wTopHat)

# Display example images
par(mfrow=c(3,4))
invisible(lapply(tiffRes[1:4], dispImg))
invisible(lapply(tiffTop[1:4],   function(x)   dispImgT(x,
0.99)))
```

```
## Threshold and centroids

# Assign gray channel to data frame
tiffG <- lapply(tiffTop, GintoDF)

# Threshold to keep pixels with intensity > 0.99
# quantile
tiffThres <- lapply(tiffG,
                    function(x) {x[x$G > quantile(x$G,
                    0.99), ]})
names(tiffThres) <- seq(1:length(tiffThres))

# Estimate images centroid
centroids <- lapply(tiffThres,
                    function(x) Weiszfeld(x[,1:2]))

# Plot examples
thresXY <- lapply(tiffThres,
                  function(x) x[,1:2, drop=FALSE])
par(mar=c(0.1,0.1,0.1,0.1), mfrow=c(1,4))
for (i in 1:4) {
  plot(thresXY[[i]])
  par(new=T)
  points(centroids[[i]]$median, col="red", pch=19)
}

## Distances to centroide

#Calculate distances to centroid
distCent <- list()
for (i in 1:length(thresXY)) {
   pdist <- pointDistance(p1=thresXY[[i]],
                          p2=centroids[[i]]$median,
                         lonlat=F)
   # Mark distances > 0.8 quantile, as they belong to
   # points outside the eye boundary in their majority
   pLogic <- pdist < quantile(pdist, 0.80)
   pp <- cbind(distCent = pdist, selected = pLogic)
   distCent[[i]] <- pp
}

# Plot example histograms
par(mfrow=c(2,2))
for (i in 1:4) {
  hist(distCent[[i]][,1])
  abline(v=quantile(distCent[[i]][,1], 0.8), col="red",
lty="dashed", lwd=2)
}
```

```
# Join thresholded and distances lists
thresDist <- mapply(cbind, tiffThres, distCent,
                    SIMPLIFY = FALSE)

# Discard points with distance > quantile 0.8
distSelect <- lapply(thresDist,
                    function(x) x[x$selected == 1, ])

# Plot examples
plotThres <- list()
for (i in 1:4) {
  p <- ggplot(data=thresDist[[i]], aes(x=x, y=y,
               color=selected)) +
       geom_point(show.legend = FALSE) +
       plotTheme()
  plotThres[[i]] <- p
}
lay <- rbind(c(1,2), c(3,4))
gridExtra::grid.arrange(grobs=plotThres, layout_matrix =
lay)


# Overlay to image
plotOverlay <- list()
for (i in 1:4) {
  p <- ggplot(data = tiffOri[[i]], aes(x = x, y = y)) +
       geom_point(colour = rgb(tiffOri[[i]][c("R", "G",
                                              "B")])) +
       labs(title = "Original Eye selected Points",
            cex=0.5) +
       xlab("x") +
       ylab("y") +
       geom_point(data=distSelect[[i]], alpha=0.2) +
       plotTheme()
  plotOverlay[[i]] <- p
}
lay <- rbind(c(1,2), c(3,4))
gridExtra::grid.arrange(grobs=plotOverlay, layout_matrix
= lay)

## Subset and confidence ellipse

# Add ellipse to plot
ellPlots <- lapply(distSelect, function(x)
ellPlot(x,0.90))
```

```
# Extract components
build <- lapply(ellPlots,
                function(x) ggplot_build(x)$data)
ells <- lapply(build, function(x) x[[2]])

# Select original image points inside ellipse
origpixList <- list()
for (i in 1:length(thresXY)) {
  imgOri <- tiffOri[[i]]
  ell <- ells[[i]]
  origEll <- data.frame(imgOri[,1:5],
          in.ell = as.logical(point.in.polygon(imgOri$x,
          imgOri$y, ell$x, ell$y)))
  origPix <- origEll[origEll$in.ell==TRUE,]
  origpixList[[i]] <- origPix
}

# Plot examples
plotOverlay2 <- list()

for (i in 1:4) {
  p <- ggplot(data = tiffOri[[i]], aes(x = x, y = y)) +
       geom_point(colour = rgb(tiffOri[[i]][c("R", "G",
                                              "B")])) +
       labs(title = "Original Eye final ROI", cex=0.5) +
       xlab("x") +
       ylab("y") +
       geom_polygon(data=ells[[i]][,1:2], alpha=0.2,
                    size=1, color="blue") +
       plotTheme()

  plotOverlay2[[i]] <- p
}

lay <- rbind(c(1,2), c(3,4))
gridExtra::grid.arrange(grobs=plotOverlay2,
                        layout_matrix = lay)

## Create image from final ROI
roisImg <- lapply(origpixList, roitoImg)

# Store ROIS as images
dir.create("ROIS2"); setwd("ROIS2")
for (i in 1:length(roisImg)) {
  writeImage(roisImg[[i]], tiffFiles[i], "jpeg")
}
```

HOG extraction and machine learning classifiers

```
#######################################
# 2. Machine learning classification #
#######################################

#############
# Functions #
#############

# Obtain IREG
IREG <- function(x) {
  return ((4*x[2]+ 3*x[4] + 2*x[3] + x[5])/4)
}

#################
# Load packages #
#################

library(lattice)
library(tiff)
library(jpeg)
library(EBImage)
library(caret)
library(pROC)
library(kernlab)
library(C50)

######################
# Preprocessing data #
######################

# Read original images
imgFiles <- list.files(pattern="*tif$", full.name=F)
imgList <- lapply(imgFiles, function(x) readImage(x,
                  type="jpeg"))

# To grayscale preserving RGB luminance
imgLum <- lapply(imgList, function(x)
channel(x,"luminance"))

############################
# HOG features extraction #
############################

# Calculate HOG descriptor

imgHOG <- do.call(rbind.data.frame,
          lapply(imgLum, function(x) OpenImageR::HOG(x,
                 cells=5, orientations=5)))
```

```
colnames(imgHOG) <- paste(rep("HOG",dim(imgHOG)[2]),
                          seq(1,dim(imgHOG)[2]),sep="")

# Add labels to images
fenotype <- read.csv("labels.csv", header = T)
new <- cbind(fenotype, imgHOG)

# Plot HOG features Example
gridHOG <- data.frame(x = rev(rep(1:25, 5)), y =
rev(rep(1:5, each = 25)))
exHOG <- t(new[c(1,83,145,200,265),-1])
classHOG <- c("Healthy", "IntA", "IntB", "IntC", "Deg")
colnames(exHOG) <- classHOG

par(mfrow=c(5,1))
for (i in 1:5) {
  plot(gridHOG$x, gridHOG$y, pch=19, cex=0.5,
       col="Blue",xlab="",ylab="", main=classHOG[i],
       xlim=c(0.5,25.5), ylim=c(0.5,5.5))
  length <- 0.6
  arrows(gridHOG$x, gridHOG$y,
         x1=gridHOG$x+length*cos(10^4*exHOG[,i]),
         y1=gridHOG$y+length*sin(10^4*exHOG[,i]),
         length=0.05, col="Black")
}

####################
# Train/test split #
####################

# Seed for reproducibility
set.seed(12345)

# Stratified random sampling 75/25
inTrain <- createDataPartition(y = new$Class, p= .75,
                               list = FALSE)

# Store partitions
train <- new[inTrain,]
test <- new[-inTrain,]

# Check class representation
prop.table(table(train$Class))
prop.table(table(test$Class))
```

```
#################
# SVM algorithm #
#################

## Model training

# Build the classifier with gaussian kernel, 10-fold CV
set.seed(12345)
SVMrbf <- ksvm(Class ~ ., data = train, kernel = "rbf",
               prob.model=TRUE,
               cross=10,
               kpar="automatic")
save(SVMrbf, file = "SVMrbf.rda") # save model
SVMrbf # show basic data

## Evaluating performance

# Make predictions
SVMrbf.pred <- predict(SVMrbf, test)
SVMrbf.probs <- predict(SVMrbf, test,
                        type="probabilities")
all <- data.frame(Class = test$Class,
                  Pred = SVMrbf.pred, SVMrbf.probs)

# Check first results
head(SVMrbf.pred)

# Comparison
(SVMrbf.confmat <- confusionMatrix(SVMrbf.pred,
test$Class))

## IREG estimation

# Estimate IREG for the test set
iregscore <- apply(SVMrbf.probs, 1, IREG)

# Plot distribution
result <- data.frame(Class = test$Class, ireg=iregscore)
boxplot(ireg ~ Class, result, ylab="IREG")


############################
# Decision Trees algorithm #
############################

## Model training
```

```
# Build the classifier
ctgC50 <- C5.0(train[,-1], train$Class)
ctgC50

## 10 trials boosting

# 10 trials
ctgC50.boost <- C5.0(train[,-1], train$Class,
                     trials = 10)
ctgC50.boost

## Evaluating performance

# 1 trial predictions
ctgC50.pred <- predict(ctgC50, test)

# 10 trials predictions
ctgBoost.pred <- predict(ctgC50.boost, test)

# Confusion matrix
C50.confmat <- confusionMatrix(ctgC50.pred, test[,1])
C50boost.confmat <- confusionMatrix(ctgBoost.pred,
                                    test[,1])
C50compar <- data.frame(C50.confmat$overall,
C50boost.confmat$overall)
colnames(C50compar) <- c("1-trial", "10-trials")

C50.confmat$table
C50boost.confmat$table
round(C50compar[c(1,2,3,4,6),],3)


############################
# Random Forest algorithm #
############################

## Model training

# Build the classifier
set.seed(12345)

# Specify options
ctrl <- trainControl(method = "repeatedcv", number = 10,
                     repeats=10,
                     classProbs = TRUE,
                     summaryFunction = multiClassSummary)
```

```
# Specify parameter tuning
grid <- expand.grid( .mtry = c(2, 8, 15, 21, 50))

# Train the model
ctgrndFor2 <- train(x = train[2:126], y = train$Class,
                    method = "rf",
                    tuneGrid = grid,
                    trControl = ctrl,
                    metric= "ROC", preProc = c("range"))
ctgrndFor2

## Evaluating performance

ctgrnd2.pred <- predict(ctgrndFor2, test)

# Confusion matrix
rndFor2.confmat <- confusionMatrix(ctgrnd2.pred,
                                   test$Class)
rndFor2.confmat$table


####################
# Model comparison #
####################

globalcompar <- data.frame(SVMrbf.confmat$overall,
                           C50.confmat$overall,
                           C50boost.confmat$overall,
                           rndFor2.confmat$overall)
colnames(globalcompar) <- c("SVMrbf", "C50", "C50boost",
                            "1000 RandomForest")
round(globalcompar[c(1,2,3,4,6),],3)

## Multiclass AUC
svmAUC <- multiclass.roc(test$Class,
          as.numeric(predict(SVMrbf, test,
          type="response")))
c50AUC <- multiclass.roc(test$Class,
          as.numeric(predict(ctgC50, test,
          type="class")))
c50boostAUC <- multiclass.roc(test$Class,
          as.numeric(predict(ctgC50.boost, test,
          type="class")))
randForCVAUC <- multiclass.roc(test$Class,
          as.numeric(predict(ctgrndFor2, test,
          type="raw")))
listauc <- sapply(list(svmAUC, c50AUC, c50boostAUC,
randForAUC, randForCVAUC), auc)
```

Fine-tune pre-trained Inception-BN CNN

```
######################
# 3. Pre-trained CNN #
######################

#################
# Load packages #
#################

library(tiff)
library(jpeg)
library(EBImage)
library(caret)
library(pROC)
library(mxnet)

######################
# Preprocessing data #
######################

##Read image and transform

# Read original images
imgFiles <- list.files(pattern="*tif$", full.name=F)
imgList <- lapply(imgFiles,
               function(x) readImage(x, type="jpeg"))

# Resize to square form
imgRes <- lapply(imgList,
                 function(x) resize(x, 224, 224))

# Save to disc
dir.create("rgb224")
setwd("rgb224")
for (i in 1:length(imgRes)) {
  writeImage(imgRes[[i]], imgFiles[i], "jpeg")
}

# Output file
out_file <- "fleyesrgb224.csv"

# List images in path
images <- list.files("rgb224", pattern="*tif$",
                     full.name=T)

# Set up df
df <- data.frame()

# Set image size. In this case 224*224*3
```

```
img_size <- 224*224*3

# Loop over each image
for(i in 1:length(images))
{
    # Read image
    img <- readImage(images[i], type="jpeg")
    # Coerce to a vector
    vec <- as.vector(unlist(img))
    # Bind rows
    df <- rbind(df,vec)
}

# Set names
names(df) <- c(paste("pixel", c(1:img_size)))

# Write out dataset
write.csv(df, out_file, row.names = FALSE)


####################
# Train/test split #
####################

## Test and train random shuffle and split

# Load datasets
fleyes <- data.table::fread("fleyesrgb224.csv",
                            header = T)
fenotype <- read.csv("labels.csv", header = T)
new <- cbind(Class = fenotype, fleyes)

# Shuffle new dataset
set.seed(123456)
shuffled <- new[sample(1:dim(new)[1]),]

# Train-test split
# Seed for reproducibility
set.seed(12345)

# Stratified random sampling
inTrain <- createDataPartition(y=new$Class, p= .75, list
= FALSE)

# Store partitions
train <- new[inTrain,]
trainNolab <- train[,-1]
test <- new[-inTrain,]
```

```
testNolab <- test[,-1]

# Fix train and test datasets
training <- data.matrix(trainNolab)
train_x <- t(training)
train_y <- train[,1]
train_array <- train_x
dim(train_array) <- c(224, 224, 3, ncol(train_x))

testing <- data.matrix(testNolab)
test_x <- t(testing)
test_y <- test[,1]
test_array <- test_x
dim(test_array) <- c(224, 224, 3, ncol(test_x))


####################
# Transfer learning #
####################

## Adapted from Dog vs Cats Kaggle competition

# Set seed for reproducibility
mx.set.seed(100)

# Load pretrained model
setwd("Inception-BN")
inception_bn <- mx.model.load("Inception-BN",
                              iteration = 126)

symbol <- inception_bn$symbol

# Check symbol$arguments for layer names
internals <- symbol$get.internals()
outputs <- internals$outputs

flatten <- internals$get.output(which(outputs ==
                                "flatten_output"))

new_fc <- mx.symbol.FullyConnected(data = flatten,
                                   num_hidden = 6,
                                   name = "fc1")
new_soft <- mx.symbol.SoftmaxOutput(data = new_fc,
                                    name = "softmax")
arg_params_new <- mxnet:::mx.model.init.params(
    symbol = new_soft,
    input.shape = c(224, 224, 3, ncol(train_x)),
    initializer = mxnet:::mx.init.uniform(0.1),
```

```
     ctx = mx.cpu(0)
    )$arg.params

fc1_weights_new <- arg_params_new[["fc1_weight"]]
fc1_bias_new <- arg_params_new[["fc1_bias"]]

arg_params_new <- inception_bn$arg.params

arg_params_new[["fc1_weight"]] <- fc1_weights_new
arg_params_new[["fc1_bias"]] <- fc1_bias_new

## Fine tune
model <- mx.model.FeedForward.create(
  symbol             = new_soft,
  X                  = train_array,
  y                  = train_y,
  ctx                = mx.cpu(0),
  eval.metric        = mx.metric.accuracy,
  num.round          = 30,
  learning.rate      = 0.05,
  momentum           = 0.9,
  wd                 = 0.00001,
  kvstore            = "local",
  array.batch.size   = 20,
  epoch.end.callback =
mx.callback.save.checkpoint("inception_bn"),
  batch.end.callback = mx.callback.log.train.metric(20),
  initializer        = mx.init.Xavier(factor_type =
                       "in", magnitude = 2.34),
  optimizer          = "sgd",
  arg.params         = arg_params_new,
  aux.params         = inception_bn$aux.params
)

## Evaluate performance
predict_probs <- predict(model, test_array)
predicted_labels <- max.col(t(predict_probs)) - 1

# Confusion matrix
table(test[,1], predicted_labels)

# Accuracy
sum(diag(table(test[,1],
         predicted_labels)))/dim(test)[1]

# Multiclass AUC
multiclass.roc(test[,1], predicted_labels)
```

De novo CNN

```
##################
# 4. De novo CNN #
##################

#################
# Load packages #
#################

library(tiff)
library(jpeg)
library(EBImage)
library(caret)
library(pROC)
library(mxnet)

######################
# Preprocessing data #
######################

##Read image and transform

# Read original images
imgFiles <- list.files(pattern="*tif$", full.name=F)
imgList <- lapply(imgFiles,
               function(x) readImage(x, type="jpeg"))

# Resize to square form
imgRes <- lapply(imgList,
                 function(x) resize(x, 100, 100))

# Save to disc
dir.create("rgb100")
setwd("rgb100")
for (i in 1:length(imgRes)) {
  writeImage(imgRes[[i]], imgFiles[i], "jpeg")
}

# Output file
out_file <- "fleyesrgb100.csv"

# List images in path
images <- list.files("rgb224", pattern="*tif$",
                     full.name=T)

# Set up df
df <- data.frame()

# Set image size. In this case 100*100*3
```

```
img_size <- 100*100*3

# Loop over each image
for(i in 1:length(images))
{
    # Read image
    img <- readImage(images[i], type="jpeg")
    # Coerce to a vector
    vec <- as.vector(unlist(img))
    # Bind rows
    df <- rbind(df,vec)
}

# Set names
names(df) <- c(paste("pixel", c(1:img_size)))

# Write out dataset
write.csv(df, out_file, row.names = FALSE)


####################
# Train/test split #
####################

## Test and train random shuffle and split

# Load datasets
fleyes <- data.table::fread("fleyesrgb100.csv",
                            header = T)
fenotype <- read.csv("labels.csv", header = T)
new <- cbind(Class = fenotype, fleyes)

# Shuffle new dataset
set.seed(123456)
shuffled <- new[sample(1:dim(new)[1]),]

# Train-test split
# Seed for reproducibility
set.seed(12345)

# Stratified random sampling
inTrain <- createDataPartition(y=new$Class, p= .75, list
= FALSE)

# Store partitions
train <- new[inTrain,]
trainNolab <- train[,-1]
test <- new[-inTrain,]
```

```
testNolab <- test[,-1]

# Fix train and test datasets
training <- data.matrix(trainNolab)
train_x <- t(training)
train_y <- train[,1]
train_array <- train_x
dim(train_array) <- c(100, 100, 3, ncol(train_x))

testing <- data.matrix(testNolab)
test_x <- t(testing)
test_y <- test[,1]
test_array <- test_x
dim(test_array) <- c(100, 100, 3, ncol(test_x))


##############################
# Deep learning from scratch #
##############################

## Model
data <- mx.symbol.Variable('data')

# 1st convolutional layer 5x5 kernel, 20 filters
conv_1 <- mx.symbol.Convolution(data= data,
                                kernel = c(5,5),
                                num_filter = 20)
tanh_1 <- mx.symbol.Activation(data= conv_1,
                                act_type = "tanh")
pool_1 <- mx.symbol.Pooling(data = tanh_1,
                            pool_type = "max",
                            kernel = c(2,2),
                            stride = c(2,2))

# 2nd convolutional layer 5x5 kernel, 50 filters
conv_2 <- mx.symbol.Convolution(data = pool_1,
                                kernel = c(5,5),
                                num_filter = 50)
tanh_2 <- mx.symbol.Activation(data = conv_2,
                                act_type = "tanh")
pool_2 <- mx.symbol.Pooling(data = tanh_2,
                           pool_type = "max",
                           kernel = c(2,2),
                           stride = c(2,2))

# 1st fully connected layer
flat <- mx.symbol.Flatten(data = pool_2)
```

```
fcl_1 <- mx.symbol.FullyConnected(data = flat,
                                  num_hidden = 200)
tanh_3 <- mx.symbol.Activation(data = fcl_1,
                               act_type = "tanh")

# 2nd fully connected layer
fcl_2 <- mx.symbol.FullyConnected(data = tanh_3,
                                   num_hidden = 6)

# Output
NN_model <- mx.symbol.SoftmaxOutput(data = fcl_2)

# Set seed for reproducibility
mx.set.seed(12345)

# Device used
device <- mx.cpu()

# Model training
model <- mx.model.FeedForward.create(NN_model,
          X = train_array,
          y = train_y,
          ctx = device,
          num.round = 30,
          array.batch.size = 20,
          learning.rate = 0.05,
          momentum = 0.9,
          wd = 0.00001,
          eval.metric = mx.metric.accuracy,
          epoch.end.callback =
     mx.callback.log.train.metric(20),
          optimizer="sgd")

## Evaluating performance

predict_probs <- predict(model, test_array)
predicted_labels <- max.col(t(predict_probs)) - 1

# Confusion matrix
table(test[,1], predicted_labels)

# Accuracy
sum(diag(table(test[,1],
               predicted_labels)))/dim(test)[1]

# Multiclass AUC
multiclass.roc(test[,1], predicted_labels)
```