Model Based System for Automated Analysis of Biomedical Images

María del Rocío Aguilar Chongtay

Ph.D.
University of Edinburgh
1997

To Emma Chongtay Yepez,
for the inherited dreams.

**With each step forward, with each problem to solve,
we not only discover new and unsolved problems,
but we also discover that where we believed
that we were standing on firm and safe ground,
all things are, in truth,
insecure and in a state of flux.**

K.R. Popper

Abstract

Today, digital images are used routinely in many areas of biomedical research and clinical practice. The need for reliable quantitative and qualitative analysis of very large numbers of these images, has produced an increasing interest in automated Biomedical image processing systems. Traditional image analysis methods (*TIA*) are built to exploit and take advantage of image data properties, as a result they tend to be fragile with respect to any changes in those properties. For this reason, *TIA* have been found to be ineffective for the recognition and classification of the inherently irregular and variable biological objects. A potentially more robust and reliable alternative are the Model-based methods as the underlying idea is to use previously identified and explicitly represented properties of the image data in order to establish the best possible interpretation.

This thesis presents the development of a probabilistic formulation of model-based vision using generalised flexible template models. It includes the design and implementation of a system which extends flexible template models to include grey level information in the object representation for image interpretation. This system was designed to deal with microscope images where the different stain and illumination conditions during the image acquisition process produce a strong correlation between density profile and geometric shape.

This approach is based on statistical knowledge from a training set of examples. The variability of the shape-grey level relationships is characterised by applying principal component analysis to the shape-grey level vector extracted from the training set. The main modes of variation of each object class are encoded within a generic object formulation constrained by the training set limits. This formulation adapts to the diversity and irregularities of shape and view during the object recognition process. The modes of variation are used to generate new object instances for the matching process of new image data. A genetic algorithm method is used to find the best possible explanation for a candidate of a given model, based on the probability distribution of all possible matches.

This approach is demonstrated by its application to microscope images of brain cells. It provides the means to obtain information such as brain cells density and distribution. This information could be useful in the understanding of the development and properties of some Central Nervous System (*CNS*) related diseases, such as in studies of the effects of the human immunodeficiency virus (*HIV*) in the *CNS* where neuronal loss is expected.

The performance of the *SGmodel* system, presented here, was compared with manual neuron counts from domain experts. The results shown no significant difference between *SGmodel* and manual neuron distribution estimates. The observation of larger differences between the counts of the domain experts underlines the automated approach importance to perform an objective cell distribution analysis. A practical benefit of this model based system is that it is adaptable to new conditions and allows rapid prototyping for different applications.

Acknowledgements

Many pages would be needed to thank enough so many people that have played an important role in my life during these years.

Firstly I would like to thank my supervisor, Dr. R. Baldock, for his time, contributions, comments and supervision throughout the project. I would also like to thank Dr. J. Bell for her assistance and supervision concerning the biomedical application of this project. I'm also grateful to her and Dr. J. Ironside for the manual counting of neurons. To Andrew Carothers for his advice in the statistical analysis of the results.

Deep thanks are due to Marcia Hiriart, Aphrodite Karnassopoulou, Amaia Bernaras, Marisa Barja, Tim Smithers, Duncan Davison, John Hallam and Margarita Sordo-Sanchez who were patient enough to read a draft of the thesis and provided useful comments on this dissertation.

Thanks are due to Bill Hill, Margaret Stark, John Ireland and Ian Poole who were always willing to answer my questions and to all the DA members for the special working environment at the Human Genetics Unit, MRC. The exchange of ideas with fellow students and researchers at the Artificial Intelligence Department from the University of Edinburgh, contributed to my understanding in many problems. I would also like to thank them.

I gratefully acknowledge the financial support provided by the National Autonomous University of México (UNAM). Thanks are also due to ESPRAM and the Human Genetics Unit, MRC for supporting financially the last months of this project.

I thank my family and all those who contributed in my life during these years through their friendship, in Mexico and in Edinburgh, their confidence in me always seemed grater than my own. In particular to Clotilde Coello and Pilar Segarra, for being a source of inspiration. To Amaia Bernaras, Tim Smithers, Marisa Barja and Tore Bratvold for being so supportive and helped me to overcome a difficult time. Hugh and Ilona Morison who have saved me more than once during critical times. Margarita and Hector Estrada have been a constant presence and a source of strength in good and difficult moments. To Laura Margarita Granada, Santiago Negrete, Enrique Filloy, Margarita Sordo, Dora Alexopoulou, Arturo Espinosa, Emma Patricia Tamayo and Merida Sotelo who are always willing to listen and help.

Finally, but not least, I would like to express my deepest gratitude to Marcia Hiriart, her enthusiastic support and unconditional friendship has played a key role to go through these challenging years. To Aphrodite Karnassopoulou for her openness, honesty and sincere friendship which has taught me about quality in life. To Rikke Magnussen for her deep contact that has played an important role in this transitional time and for her contagious enthusiasm which keeps a link to my own dreams. To Sarah Gingell for the fantastic breaking through team and for sharing an especial way of seen life. But that is not the end, as always the story must be continued and I would like to thank John Hertz, Sara Solla and Predrag Cvitanovic who helped me during the search for links to the future.

Declaration

I hereby declare that the work presented in this thesis was carried out by myself at The University of Edinburgh and that it describes my own research, except where due acknowledgement is made.

M. del Rocío Aguilar Chongtay
Edinburgh
November 3, 1997

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Automated image analysis methods are being increasingly applied to deal with quantitative assessment for Biomedical applications in which the need for accurate analysis of very large number of complex images is important (Sterio 1984). The motivation for this thesis research originated from the need to automate the analysis of microscope images from brain tissue to provide the means for obtaining information such as cell density and distribution (see figure 1.1). This information is intended to help in the understanding of some Central Nervous System (*CNS*) related diseases, such as in the study of the human immunodeficiency virus (*HIV*) effects in the *CNS* where neuronal loss is expected (chapter 2). Many changes have been reported to occur in the brain during infection by *HIV* but the mechanisms of how *HIV* mediates neurological damage, and the effects of *HIV* in the *CNS* in general are still not well understood (Sharer 1992; Elovaara 1995). Neuronal loss is reported in the *CNS* in people who have died with *HIV* infection (Ketzler et al. 1990; Wiley et al. 1991; Everall and Lantos 1991). Qualitative and quantitative studies of these changes at different stages of *HIV* infection require the assessment of large numbers of microscope images from brain tissue. For this reason a semi-automatic method for microscope image analysis could represent a significant contribution to the understanding of the development and properties of this disease and any other neurodegenerative diseases.

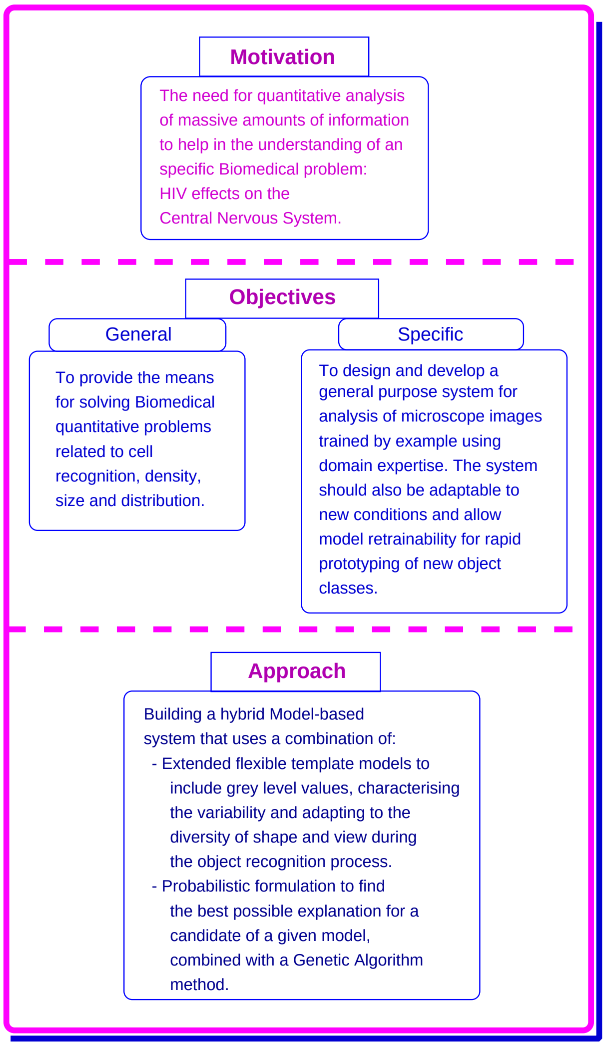

Figure 1.1: Motivation, Objectives and Approach.

1.2 Approach

The main advantage of automated image analysis is; the ability to objectively process, large amounts of material in a shorter period of time. However, the complexity and variability commonly found in biomedical structures makes difficult to automate these analyses (Brian and Marchevsky 1994), (Heus and Diegenbach 1992), (Miller et al. 1979). Model-free techniques, such as traditional image analysis (*TIA*), are often specifically designed to solve these complexities (chapter 3). *TIA* is based on sequences of image operations, carefully designed and tuned for a particular problem. These highly specialised systems are built to exploit and take advantage of details of image data, as a result of this they tend to be fragile with respect to any changes in those details, and also require considerable intervention by the domain expert (*e.g.*, the pathologist). Model-based techniques (*MB*), on the other hand, are less dependent on the details of the image data for their functioning, and thus less sensitive to changes and variation on these details. *MB* use prior knowledge to construct a set of assumptions and expected properties or predictions of the image structure thus improving the reliability and robustness of the analysis. However, modelling complex image objects (*e.g.*, biomedical objects) from different domains requires that their specific properties are included in an efficient representation in order to develop a robust and reliable system.

In image analysis, one of the most common types of model used is geometric models which describes elements with inherent geometrical properties such as spatial layout and shape, size, connectivity of components, view parameters etc. However geometric models are often too rigid to represent some classes of objects with the variability of shape that biomedical objects could present. Flexible models (*FM*) are considered more suitable for biomedical image analysis because of their capabilities such as moving or deforming to fit biological structures, learning from examples and generalising their knowledge so that it can be applied to new and different circumstances. *FM* are based on the statistical knowledge of object features and their main modes of variation characterised, for example, by applying a Principal Component Analysis (*PCA*) to a training set of object examples. In this thesis, the approach presented for microscope image analysis is based on flexible models.

The objects of interest in the domain of application in this thesis are irregular, and there is a relationship between their density profile and geometric shape given the nature of the light transmission images (*i.e.*, microscope images). In order to use all the knowledge about the appearance of the object, it is important that a model has the ability to recognise direct links between shape and grey values. In this approach a more complete model is proposed by extending the usual flexible model feature space to include grey values directly in a shape grey-value model for which the name *SGmodel* (chapter 4) has been coined. This hybrid approach involves processing images at different analysis levels, such as *low-level* (*e.g.*, image formation) to *high-level* (*e.g.*, image interpretation), (Gonzales and Wood 1992; Sonka et al. 1993; Castleman 1996). *SGmodel* was defined within a probabilistic formulation to find the best possible explanation for an object candidate of a given model, combined with a Genetic Algorithm optimisation method. One of the underlying features of this formulation is that it was designed to solve the normalisation problem produced by including non-spatial data dimensions and it allows classification when the feature space is different for each class.

The algorithms developed in this approach provide a rapid prototyping tool for new object classes and were designed to be usable by a domain expert, in this case the neuropathologist (chapter 5). An important contribution of this approach is a more generic model for the domain application which, in turn, implies that the model can be tuned to specific study cases (chapter 6), without the need for specialist image-processing and programming expertise, to incorporate new models for each object class. This results in a high adaptability to new conditions and allows rapid modelling of new object classes.

Finally the *SGmodel* system performance was tested in both synthetic and real data. The modes of variation, obtained by applying PCA to the training set of an artificially generated cell (chapter 5), were used to illustrate how the object variability is characterised by PCA. In this test, the model parameters were recovered exactly by the PCA (chapter 7). The small number of parameters used to define this artificial cell made it possible to observe how the first modes of variations captured the feature variability.

The *SGmodel* system was trained for *neuron* and *NOneuron* models for testing purposes with real data. The model training and testing was performed in randomly sampled images of brain tissue from five case groups involved in the *HIV-CNS* project (chapter 7). At individual cell level there were found cases of misclassification, however the comparison between manual and automatic *neuron* global counts over a testing set of images show no significant differences, this suggest that the algorithms developed in this thesis, *i.e., SGmodel* system, provides a consistent and reliable tool for cell population estimation in microscope image analysis.

1.3 Thesis Organisation

The thesis organisation described below is summarised in figure 1.2. In **Chapter 2** the domain of application of this thesis is presented namely analysis of microscope images of brain cells, and its role within the project "Brain Bank for Research on *HIV* Infection and the Nervous System", a project from the Neuropathology Laboratory, University of Edinburgh supported by the Medical Research Council, UK. This chapter also includes a description of *HIV* effects on the Central Nervous System (*CNS*) and some of the reported quantitative *CNS* cell studies related to *HIV* infection. A brief overview of some computerised cell image analysis is also described. This chapter describes how a quantitative analysis by an assessment of brain cells populations can be useful to determine *CNS* damage at different stages of *HIV* infection, and why extensive comparative research is needed for the understanding of the development and properties of this disease.

Chapter 3 reviews, within a model-based system framework, different flexible model approaches applied to problems where images are characterised by the presence of irregular objects. Their advantages and disadvantages with respect to this applications are discussed. A particular emphasis is put on Point Distribution Models *(PDM)* as this thesis approach (*SGmodel*) is based on them.

In **Chapter 4** the theoretical bases for the *SGmodel* approach is presented. The *SGmodel* is presented within a general image analysis framework with the specific needs of biomedical image analysis and designed in a meaningful way for the domain expert. This chapter describes the extension to *PDM* in the *SGmodel* and how new proposed model acts as an abstraction of the high-level prior knowledge about the objects. This chapter also presents the description and elements included in the probabilistic formulation for the matching process and to solve normalisation problems produced by the inclusion of non-spatial data dimension in the feature vector, by differences in dimensionality between class models and by the use of the candidates locator. For the candidates locator implemented in this work the name *cue finder* was coined and used in several parts of this thesis.

Chapter 5 presents the design of the functional and operational characteristics used to implement the *SGmodel* in a computational system for biomedical image analysis. The design is presented in terms of different levels of image analysis. This chapter describes the combination of theoretical and practical approaches involved in the implementation of the *SGmodel* system listed as follows: traditional image analysis procedures (*i.e.*, thresholding, dilation and erosion used for the *cue finder*), flexible model procedures (*PCA* based flexible model, *i.e.*, the extended *PDM*) within a probabilistic formulation, optimisation procedures (*i.e.*, *GAs* for the optimised search of the best possible match) and a Graphical User Interface (*GUI*) in which the models are represented in a domain expert language which allows rapid prototyping of new object classes and tuning without the need for an image-processing and programming expert. The definition of an artificial cell is included in this chapter and its artificially generated training set and how this could be used to understand the behaviour of the modes of variation obtained by *PCA*.

Chapter 6 describes the Biomedical material used to obtain the digitised images and the set of images used for the analysis. The methodologies used at different stages during the image analysis, from pre-processing to classification, are also presented. These include the image acquisition process, including microscope settings and shading correction. The model training process is presented and illustrated with an example of a neuron model. The procedures to obtain and visualise the modes of variation from a training set and their use during the matching process are also discussed.

In **Chapter 7** the results obtained from the *SGmodel* system performance using synthetic and real data are presented. The possible application of the artificially generated modes of variation is described and the graphical representation of their behaviour is included. The global and individual results obtained with the *SGmodel* probability class estimations and the comparison with those obtained by domain experts are discussed.

Chapter 8 Summarises the material presented in this thesis and presents a discussion of the achievements, contribution, and limitations of the work presented, and the general conclusions reached. Finally, a number of directions for future research are suggested.

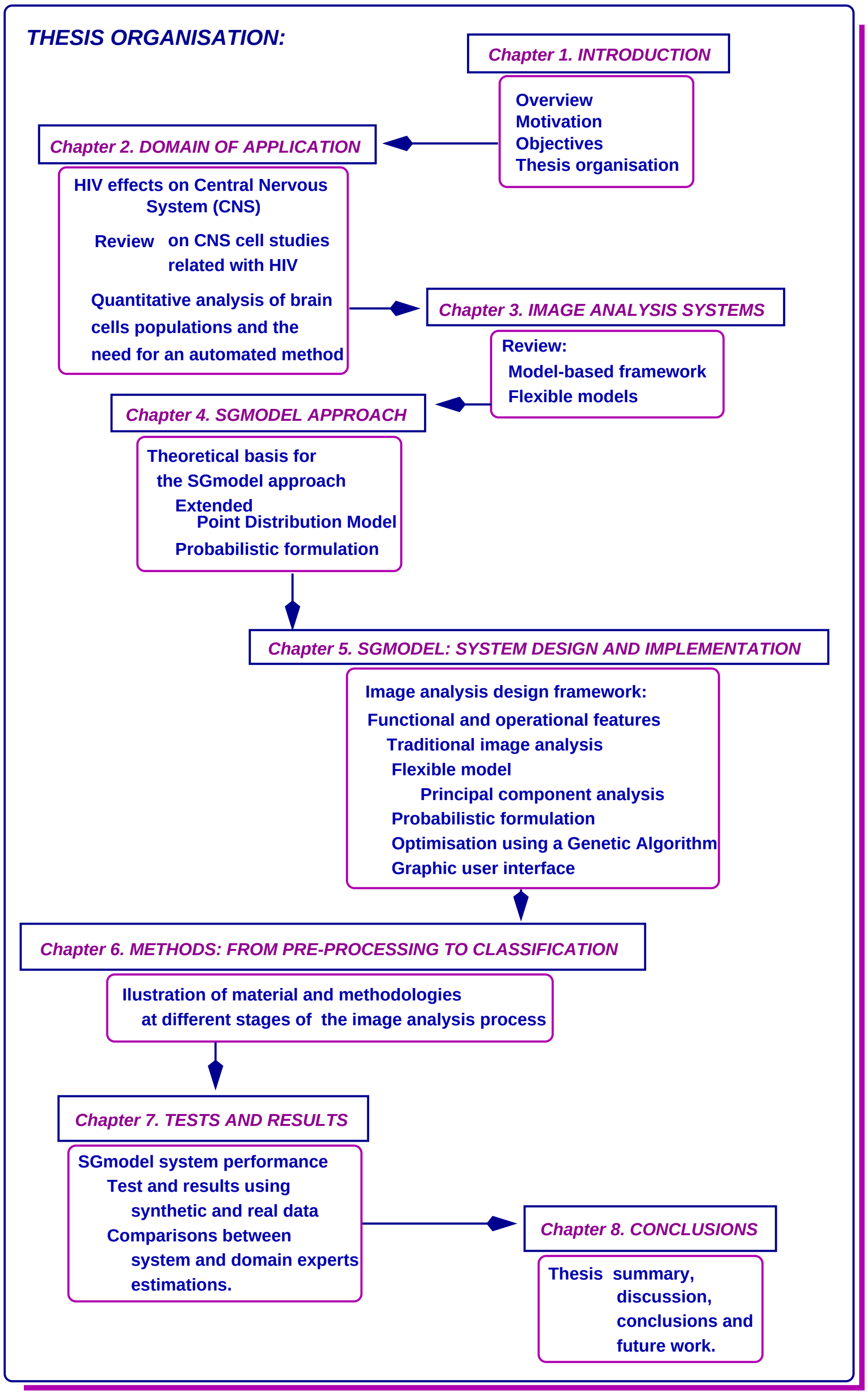

Figure 1.2: Thesis organisation summary.

Chapter 2

Domain of Application

2.1 Introduction

The domain of application of this thesis is concerned with the analysis of microscope images of brain cells. These provides the means to obtain information such as brain cell density and distribution. This information is intended to help in the understanding of some Central Nervous System (*CNS*) related diseases, such as in studies of the effects of *HIV* in the *CNS*, where neuronal loss is expected.

The application of this research is within the project "Brain Bank for Research on *HIV* Infection and the Nervous System"[1]. There is clinical and pathological evidence of Central Nervous System (*CNS*) damage related to the human immunodeficiency virus (*HIV*) infection known as *HIV* encephalitis (brain disease) (Sharer 1992). During the late stages of *HIV* infection, a dementia syndrome is frequently observed (Elovaara 1995). Many changes have been reported to occur in the brain during the infection but the mechanisms of how *HIV* mediates neurological damage are still not well understood. Neuronal loss is reported in the *CNS* of people who have died with *HIV* infection (Ketzler et al. 1990; Wiley et al. 1991; Everall and Lantos 1991). Qualitative and quantitative studies of these changes at different stages of *HIV* infection would help in the understanding of the development and properties of the disease and could be useful for treatment purposes. In this chapter the application context is presented, including a description of some *HIV* effects in the *CNS* and some related quantitative

[1] A project from the Neuropathology Laboratory, University of Edinburgh supported by the Medical Research Council, UK.

cell studies. The significant contribution that a semi-automatic method of quantitative cell analysis could make for the study of neurodegenerative diseases is also discussed.

2.2 HIV effects in the Central Nervous System

HIV affects the *CNS* in different ways at different infection stages. In the early stages, despite the lack of clear neuropathological changes after weeks of infection, there have been observed aseptic meningitis (inflammation of the brain membranes) and acute *HIV*-1 encephalitis (severe brain inflammation) (Elovaara 1995). In the *HIV* late stages, known as Acquired Immune Deficiency Syndrome (*AIDS*), brain atrophy can be found together with an encephalitis characterised by the infiltration of multinucleated giant cells (Vinters and Anders 1990).

It is also in the late stages of the *HIV* infection, that some people display an encephalopathy termed *HIV* associated dementia complex (*ADC*) or *AIDS* dementia, characterised by clinical manifestations such as cognitive changes, lethargy, social withdrawal and psychomotor retardation with a marked dementia (Brew et al. 1995). There has been intensive research in the possible correlation between *HIV* and the encephalopathy (Sharer 1992; Elovaara 1995). The understanding of *HIV* neuropathogenesis and the neurobiology of the resultant brain dysfunction at cellular and molecular levels would be very useful for treatment purposes (Vinters and Anders 1990).

Some of the changes in the *CNS* from people who have died with *AIDS* are due to severe opportunistic infections, resulting from clinical immunosuppression, these include:

- Viral infections, such as cytomegalovirus (CMV), herpes simplex (HSV), varicella-zoster, progressive multifocal leukoencephalopathy (PML);
- Fungal infections, such as cryptococcus and candida; and
- Parasitic infections, like toxoplasmosis (caused by *Toxoplasma gundii*).

Neoplasms (tumours) also occur in the *CNS* during *AIDS* and these are more commonly high grade lymphoma (tumours of lymphoid tissue). It has been suggested that some of the specific *HIV* effects within the brain could result from the *HIV* presence in certain cell types (Vinters and Anders 1990). *HIV* has been found free within the cytoplasm of multinucleated giant cells which are an accepted marker for *HIV* infection (*HIV*-GC). It has also been found that macrophages actively synthesise viral ribonucleic acid (RNA) and produce progeny virions within the brain (Everall and Lantos 1991). It is widely accepted that the only *CNS* cell types which are unequivocally capable of infection with *HIV* are macrophages and microglial cells (Price et al. 1988; Koenig et al. 1986; Everall and Lantos 1991; Glass et al. 1995; Brew et al. 1995; Dickson and Lee 1996). *HIV* presence has been described also within micro-vascular endothelial cells which constitute the morphologic substrate of the blood barrier. In a small number of cases there are reports of low levels of infection detected in neuroectodermal cells, including neurons and oligodendrocytes (Giulian et al. 1990).

Neuronal loss has been reported as one of the *CNS* changes in studies of *AIDS* neuropathology; Ketzler et al. (1990) found 18% neuronal loss in the frontal cortex in patients who died of *AIDS*; Wiley et al. (1991) reported 30-50% loss in cases with *HIV* encephalitis, and Everall et al. (1991) found about 38% loss in cases with minor pathology and *HIV* encephalitis. A wider description of these studies is included in section 2.3.

The following are some of the many questions still to be answered in relation to the effects of *HIV* in the brain.

- What are the mechanisms of *HIV* neuropathogenicity (Elovaara 1995)?
- Why do some people with typical *ADC* have no neuropathological evidence of *HIV* encephalitis (Brew et al. 1995)?
- Why do some patients with a heavy viral burden in the *CNS* have few neuropathological changes (Sharer 1992)?

While some of the changes found in the *CNS* could result from the direct effects of *HIV* on brain cells (Luthert et al. 1995), indirect effects result from other pathogenetic mechanisms which are poorly understood, (Wiley et al. 1991). If the infection could be detected in neurons or oligodendrocytes (neuroectodermally-derived cells), it could explain the changes in the grey and white matter as an *HIV* direct effect, but there is no convincing evidence of infection in these types of cells. It has been reported that *HIV*-infected macrophages are a primary source of indirect injury of nerve cells, as these are damaged by toxic substances released by the macrophages (Koenig et al. 1986; Giulian et al. 1990). Some of the tissue damage is due to the excess production of substances such as tumour necrosis factor alpha (TNFα), which may be toxic in life since it has been found to be toxic to oligodendrocytes *in vitro* (Giulian et al. 1990).

2.3 Quantitative cell studies

2.3.1 Computerised cell image analysis

Research for the cytology automation has a long history that goes back to 1950's (Hutchinson and Zahniser 1994). Blood cell counting, cervical cytology and chromosome analysis are between the main applications that require a fast and accurate cell counting. The large workload of these applications increased the need for experts skilled with the highly specialised knowledge required for the different cell studies. Research and funding have been devoted to automate this process and simulate the human expert performance with the advantage of a fast and constant accuracy that is not affected by factors such as fatigue or the possible variation between different experts. However, the complexity of biological objects has proven too difficult to develop completely automated systems that meet the high performance standards needed in these applications. A more realistic goal that is being set in some approaches is to assist the human experts with semi-automatic systems (Bartels 1994; Hutchinson and Zahniser 1994).

Intense research effort and development of computerised cervical cytology automation have been reported (Banda-Gamboa et al. 1992; Hutchinson and Zahniser 1994). The cervical screening systems analyse specimens to distinguish between normal and abnormal cells to assist cancer diagnosis (Husain et al. 1974; Tucker 1979). Some of the critical factors for automated quantitative analysis often reported in these works and also found in *CNS* cell analysis are:

- difficulties to standardise the staining,

- insufficient contrast between objects and background,
- lack of clear boundaries in objects and their components, and
- wide interspecimen variation.

In cervical cytology automation, some of these problems have been minimised by disaggregation of cells which also ease the image segmentation into its components (Banda-Gamboa et al. 1992). In general most of the segmentation methods applied to cell image analysis are based on the nature of the particular application and require *a priori* information of the objects of interest (Bartels 1994). In this application, from the segmented objects, simple measurements such as nuclear size, nuclear-cytoplasmic area, cytoplasmic, nuclear optical density, and nuclear DNA content have been reported useful for discriminating between normal and abnormal cells (Tucker 1979; Banda-Gamboa et al. 1992). Some of the main differences between the cervical tissue and *CNS* tissue which make difficult to apply the cervical cytology automation methods to *CNS* cell analysis are:

- object shape and size variation are wider within the same type of cell,
- in some *CNS* cells such as neurons, the cell boundary is not always detectable, and
- the *CNS* tissue has more non-cell objects and is generally more complex than the cervical smears.

Research and development in chromosome analysis (karyotyping) automation have also been intense (Lloyd et al. 1987). In this application the search for cells suitable for karyotyping has some similar problems to those found in cervical cell analysis such as the overlapping objects during the cell location. Chromosome analysis involve identifying each chromosome as normal or abnormal based on its size, the position of a visible morphological feature known as centromere and a precise banding pattern produced by the stain (Piper et al. 1989). Some of the critical factors for the automation of chromosome analysis are:

- location of individual chromosome from clusters of overlapped ones,
- chromosome deformation due to slide tissue preparation, and
- variability in chromosome shape depending on the stage of cell division,

One of the main differences between chromosome analysis and *CNS* cell analysis is that despite of the variation described above, chromosomes from the same class are expected to be rather similar while in some of the *CNS* cells there is high interclass variability. Piper et al. (1989) proposed a knowledge-based approach for chromosome analysis automation, and presented a problem-tailored system design for this application.

2.3.2 Cell studies in Neuropathology

In Neuropathology, cellular populations have been measured using manual and semi-automatic cell counting techniques, for example, based on nuclear size and shape to classify different kinds of cells. Immunocytochemistry and fluorescence, histological techniques which mark specific structures, are also used in cell identification. The following are some quantitative cell studies related with the effects of *HIV* in the *CNS*.

- Changes in neuronal size and density have been investigated by Ketzler et al. (1990) using morphometric analysis. They evaluated the brains of 18 people who died with *AIDS*, regardless of neuropathological diagnosis. They analysed 20-μm-thick, paraffin-embedded sections stained with cresyl violet. The results were compared with age-matched controls without neuropathological alterations reporting an 18% reduction in neuronal density and a perikaryon volume fraction reduced by 31%. As the brains were examined regardless of the neuropathological diagnosis, these results could reflect either direct effects of the *HIV* in the *CNS* or indirect, often multiple, pathologies.
- Neocortical damage was reported by Wiley et al. (1991) in 32 cases. They analysed blocks of tissue from the mid-frontal, superior temporal, and inferior parietal regions of the cortex. They counted cortical width and cell size on 20-μm-thick, paraffin-embedded sections, stained with cresyl violet. Wiley et al. (1991) used a statistical comparison of cases with *HIV* encephalitis and control

cases without significant neuropathology. In this work, a 30-50% neuronal loss was found and this was suggested to be an indirect effect of *HIV* infection of the *CNS*, as they did not find an association of these changes with the presence of *HIV* antigens.

- Eleven brains with *HIV* encephalitis, with no evidence of opportunistic infections or neoplasms were studied by Everall et al. (1991). The blocks of tissue analysed were from the frontal lobe and the quantitative cell assessment was random. The sections used were of 20 μm thickness stained with cresyl-violet. Neurons were identified by the presence of a distinct nucleolus or Nissl substance. Everall et al. (1991) performed three dimensional measurement using a stereological technique known as '*the dissector method*' which has been designed to optimise sampling of cells that are dissected by two optical planes of identical area separated by some distance (Sterio 1984). Everall et al. (1991) compared statistically *HIV* and control cases, obtaining similar results of 35-38% neuronal loss by using manual methods and with computer-assisted image analysis. They found a similar proportion of neuronal loss in cases with minor pathology and in those with *HIV* encephalitis. Given the lack of evidence of neuronal *HIV* infection, neuronal loss was suggested to be an indirect effect of the virus.

Despite the fact that *HIV* has been reported to cause serious nervous system disease in people with *AIDS* (Price et al. 1988), the virus relation with *ADC* is still unclear. In general mental disorders are often ascribed to nerve cell death. As described above, there is no clear evidence that *HIV* infects nerve cells directly but it is thought that cell death could be an indirect effect of *HIV* infection (Elovaara 1995).

2.4 The need for comparative studies

In order to contribute to the understanding of *HIV* infection and its specific expression in the *CNS*, it is necessary to characterise the tissue injury, for example by the quantitative assessment of neuronal loss. This information could be used to determine the relationship between the virus distribution and *CNS* damage. It is also important to assess tissue injury from different brain regions to evaluate *HIV* infection impact (direct or indirect) per region. Figure 2.1 shows the interaction between quantitative and qualitative information in a comparative study within the application context namely *Brain Bank for Research on HIV Infection and the Nervous System.*

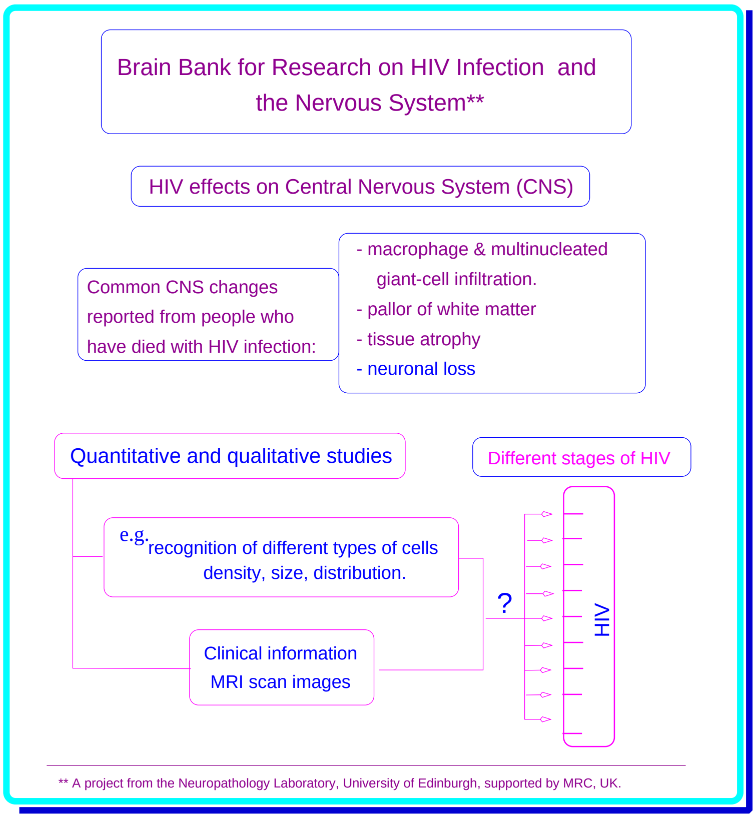

Figure 2.1: Application context

Extensive comparative research is then needed on many aspects. This thesis is concerned with quantitative assessment of brain cell populations; Counting neurons, which is known to be a complex task (Warren 1992). There are morphometry techniques to estimate cell numbers indirectly in order to avoid extensive sampling, given the large amount of material that has to be analysed from histological sections in studies of this magnitude. These estimates can be influenced by the cell class distinct size and shape and normally need correcting methods to be included. With the approach proposed in this thesis, no correction is needed as it provides direct counts of cell numbers for two dimensional sections that could be also used in a combination of other techniques, such as *dissector* (Sterio 1984) in order to estimate spatial relationships.

This research is intended to make a significant contribution by providing a means to perform a semi-automatic quantitative analysis of cell populations in *CNS* tissue from people who died at different stages of *HIV-AIDS*. The quantitative analysis method presented in this thesis would also be useful to a number of common neurodegenerative diseases characterised by cell loss, for example Alzheimer's disease and other dementias, motor neuronal diseases, etc.

A comparative study of distribution and effects of *HIV* in the *CNS* (*HIV-CNS*) is possible thanks to the availability of cases in the Medical Research Council, Edinburgh *AIDS* Brain Bank[2]. This *AIDS* Brain Bank has a wide range of material[3] for examination. The general morphometry parameters that concern neuropathologists are those of cell density, size and distribution of different cell types. The material analysed can be linked with clinical information and/or magnetic resonance images (MRI). Accurate assessment of neuronal loss, at different stages, could assist in the understanding of *AIDS* related dementia. This could also be useful for treatment planning. For example, some of the drugs used for treatment produce secondary effects such as toxic myopathy, which has been reported to be produced with long-term use of zidovudine (AZT). Therefore, it is important to characterise at which disease stage the treatment should be applied.

As part of the *HIV-CNS* project, a comparative semi-automatic assessment of astrocytes and microglial cells, from the groups listed below, was performed using brain tissue, with no *CNS* opportunistic infections or lymphomas (Roberts et al. 1997).

- $[AE]$ - *AIDS* Encephalitis drug users.
- $[ANE]$ - *AIDS* Non-Encephalitis drug users.
- $[PA]$ - HIV^+ Pre-*AIDS* drug users.
- $[DU]$ - HIV^- drug users.
- $[C]$ - Control HIV^-, non-drug users.

[2] At Neuropathology Laboratory, University of Edinburgh.

[3] Such as foetal, paediatric and adult brains of different ages and risk groups i.e. drug abusing, homosexual and bisexual and at different stages of *HIV* infection, i.e. pre-*AIDS* and *AIDS*.

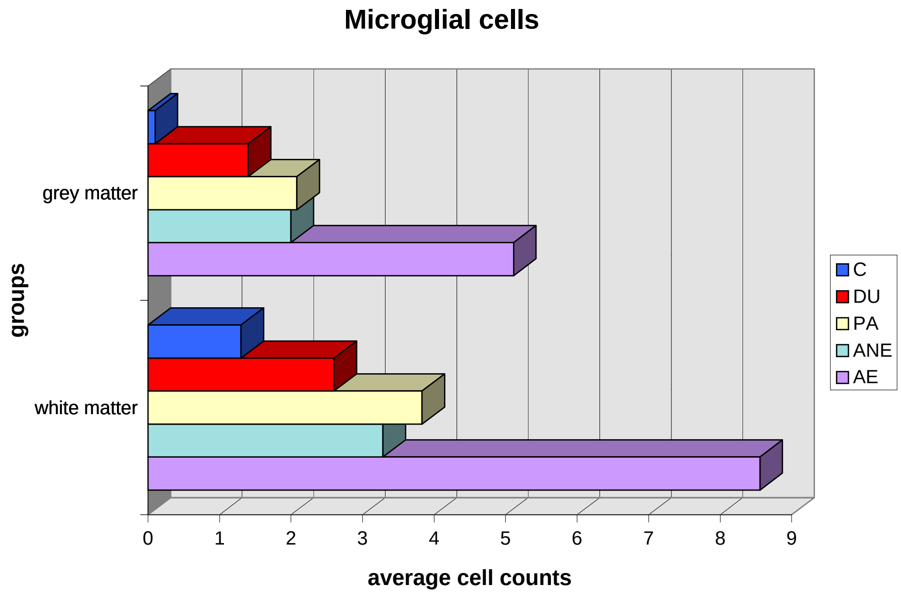

Figure 2.2: A comparative semi-automatic counting of microglial cells.

The results showed an increase in the averaged counts of microglial cells from both HIV^+ and HIV^- drug users, *i.e.*, *DU*, *PA*, *ANE* and *AE*, (see figure 2.2). The highest increase was found in the *AE* group. Figure 2.3 shows that the group of Pre-*AIDS* drug users (*PA*), had a slight increase in the astrocytes averaged counts in white matter but that *AIDS* Encephalitis drug users (*AE*), is the only group that shows a clear increase in both grey and white matter averaged counts[4]

A semi-automatic counting of neurons using '*the dissector method*' has also been performed on over 20 of the 45 cases analysed in this thesis. The results obtained by this alternative approach were compared to the findings from this thesis and are described in chapter 7.

[4] A table with the original cell counts is included in appendix C.

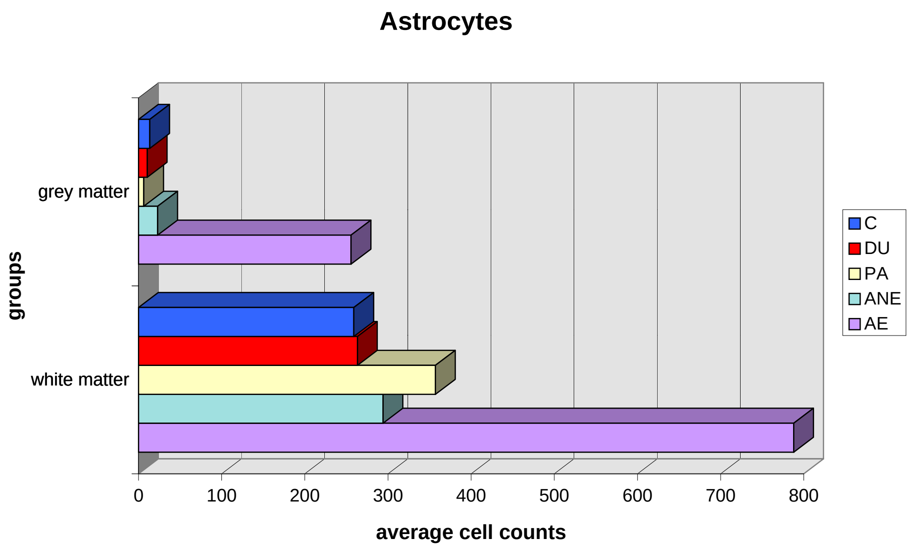

Figure 2.3: A comparative semi-automatic counting of astrocytes.

2.5 Summary

This chapter has presented the domain of application of this thesis, *i.e.*, the analysis of microscope images of brain cells. A review of the *HIV* effects in the *CNS* has also been presented and how this research provides some means for the understanding of neurodegenerative diseases. A number of quantitative cell studies were also presented. The diseases characterised by cell loss require an extensive comparative assessment of brain cell populations, for this reason this research is intended to make a significant contribution by providing an effective means to perform a semi-automatic quantitative analysis of cell populations.

Chapter 3

Image Analysis Systems

3.1 Introduction

Image analysis typically involves the following operations: image acquisition, preliminary processing, segmentation of the image into its components, region labelling, high level identification and interpretation. These image analysis operations are normally classified into different levels ranging from low-level to high-level (Gonzales and Wood 1992; Sonka et al. 1993; Castleman 1996) (figure 3.1). This classification of operation or methods varies between authors and also depends on their use in specific applications. One of the main differences between levels is the data used. Low-level processing normally uses the brightness values of the original image, in some cases little knowledge about the content or subject matter of images is used, some examples of low-level operations are: image acquisition, image compression, noise filtering, etc. High-level processing is related with recognition and interpretation tasks. At this level the analysis is guided by knowledge of the image content, for example size and shape of objects in the image or more complex object properties such as the relation between objects (Ballard and Brown 1982). In this chapter an overview of traditional image analysis (*TIA*), knowledge-based, and model-based image analysis is presented. This serves as a theoretical framework to support the approach of this thesis, namely the *SGmodel*.

Model-free techniques, such as *TIA*, are based on sequences of image operations, especially designed to solve specific problems (*e.g.*, X-ray image enhancement) (Parker 1994). These highly specialised systems are not very robust to changes in conditions, and also require considerable intervention by the domain expert to reset the appro-

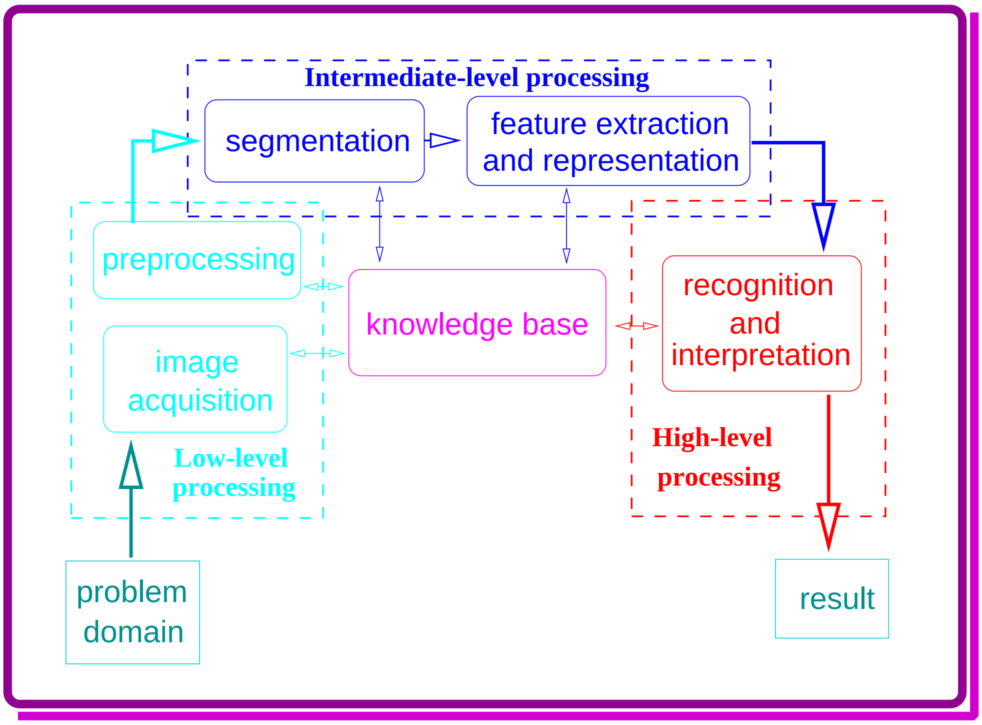

Figure 3.1: Image analysis levels, after Gonzales (1992).

priate parameters each time that the conditions change. In some cases the system has to be modified by the programming expert. In the bottom-up (data driven) data flow, usually found in *TIA*, the image information is made explicit without *a priori* information, and its representation is based on the data implicit in the original images, see figure 3.2 (Sonka et al. 1993). In order to expose constraints needed to accomplish the image interpretation task, the explicit representation of the image information is required.

In computer vision, Model-Based Systems (*MBS*) are goal oriented and prior knowledge is used to construct a set of assumptions and expected properties or predictions of the image structure. These predictions are verified in a top-down direction (model driven), at the different processing levels, from knowledge to the original image data (Ballard and Brown 1982; Gonzales and Wood 1992; Sonka et al. 1993). In this top-down mechanism the image interpretation is a process of hypothesis generation and verification. Thus the model is updated each time that a generated hypothesis is successfully tested. Figure 3.2 shows a simple representation of bottom-up and top-

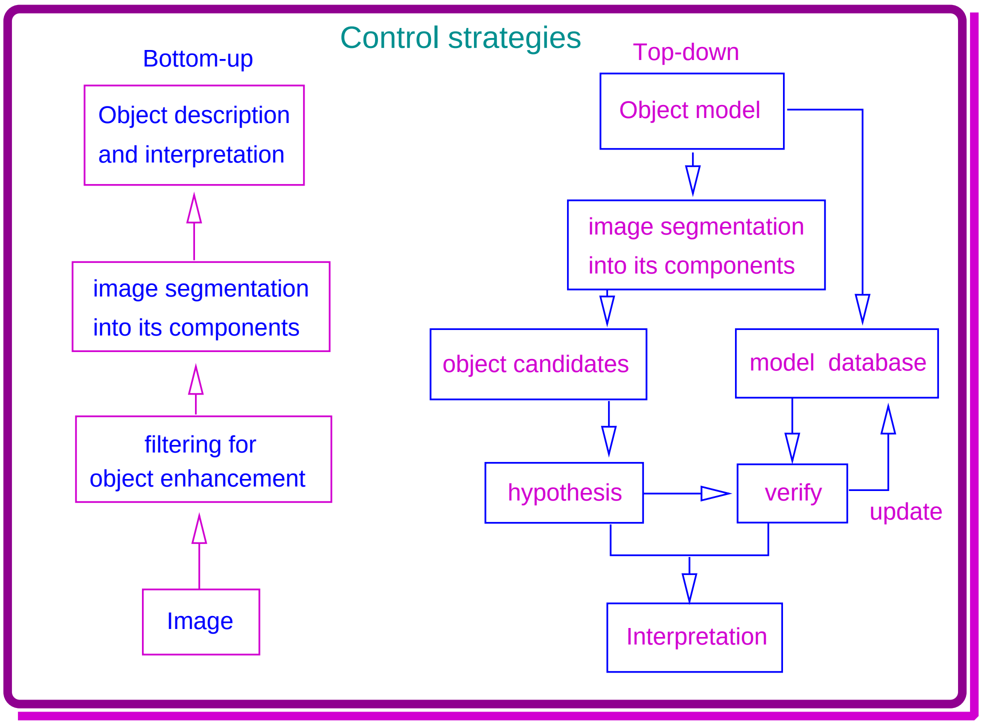

Figure 3.2: A simple representation of the image understanding control strategies: Bottom-up and Top-down.

down control strategies for image interpretation. Combinations of both strategies are commonly found in many applications, an example presented by Collins et al. (1991) is an approach to automated coronary border detection in angiographic images.

Modelling complex image objects (*e.g.*, chromosomes) from different domains requires that their specific properties are included in the representation (Piper et al. 1989). This is true especially in the representation of biomedical objects for recognition and classification given their high shape and size variability and usually large size of datasets (*e.g.*, *CNS* cells). The information required for the image interpretation includes, object representation (*e.g.*, shape information) and domain information (*e.g.*, clinical information, age, sex, etc.).

In image analysis, one of the most common type of models used is the geometric models which describe elements with inherent geometrical properties such as spatial layout and shape, size, connectivity of components, view parameters etc. However geometric models are too rigid to represent some classes of objects with the variability of shape that biomedical objects could present (*i.e.*, an specific model for each shape variant need to be defined). Flexible models (*FM*) are considered more suitable for biomedical image analysis because they are capable of deforming to fit image objects, learning from examples, and generalising their knowledge so that it can be applied to new and different circumstances.

This chapter presents a review of some flexible model image analysis methods. It is not intended to be exhaustive, but is nevertheless representative of the avenues investigated in this field. Particular emphasis is put on Point Distribution Models *(PDM)* as the approach presented in this thesis, the *SGmodel*, is based on them. The *SGmodel* approach is described in chapter 4.

3.2 Flexible Models: a review

Flexible models, also known as *deformable, adaptative or dynamic models*, have been successfully used in segmenting, matching and tracking non-rigid objects (Yuille et al. 1992; Lanitis et al. 1995; Onyango and Marchant 1996; Solloway et al. 1996). The flexibility is provided by object characterisation from a wide coverage on non-homogeneous data sets. This coverage is used to characterise the object variability and is included as an internal property of these models. Flexible models combine two basic control strategies, namely constraints derived from the image data (*bottom-up*) and *a priori* object knowledge, such as shape properties (*top-down*). These models are commonly based on energy-minimisation principles, statistical knowledge, or on a combination of techniques.

3.2.1 Snakes

The snakes or active contours approach proposed by Kass et al. (1987) is one of the first flexible models proposed for image processing applications. Snakes are defined as energy-minimising splines. The snake is a closed contour defined around a region of an image which deforms to minimise its energy. Figure 3.3, illustrates how the snake moves towards image features driven by a potential energy or image forces which couple the snake to the image data.

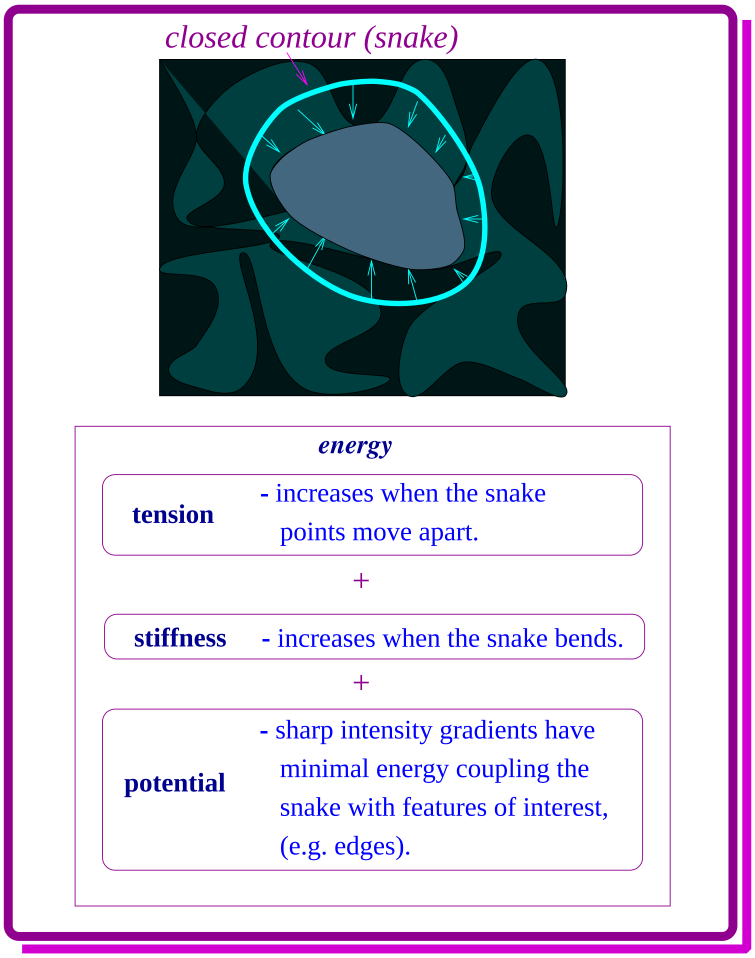

Figure 3.3: The snake (active contour model) deforms to minimise its energy driven by image forces.

The contour deformation is constrained by combining the internal stretching and bending forces (*i.e.*, tension and stiffness factors) driving the spline to approximate the locations and shapes of object boundaries based on the assumption that the boundaries of the image object are continuous or smooth. This approach is good for detecting high contrast features but has the disadvantage of not being able to find weak edges (*e..g.*, in X-ray, ultrasound and microscope images, objects not always have well defined boundaries). Moreover, it requires an approximate shape, and a starting position somewhere near the desired contour. Snakes are also influenced by a external constraint which is defined to avoid the snake to settle in an incorrect local energy minimum.

3.2.2 Deformable balloons

Deformable balloons were developed as a modified snake approach by Cohen (1991). Snakes only encode local spatial information, *i.e.*, no global shape, and the potential is calculated as the sum of forces around the snake. This could produce several problems: the snake may not be attracted to an edge because its initial position is not close enough to it; when the snake is not exposed to image forces (*e.g.*, the edge is too small or weak), in this case the snake shrinks on itself collapsing to a point. Cohen (1991) tackled this problem by extending the energy model with an internal pressure mechanism (deformable balloon) which results in the snake expansion. In this mechanism, the internal pressure pushes the snake outwards, preventing it from being attracted by weak edges, making the snake less sensitive to the initial conditions (figure 3.4).

Cohen (1991) has applied this method to the detection of crania and heart regions in magnetic resonance images. One of the disadvantages of this approach is the assumption that the object solutions are smooth. Another disadvantage of the balloons method is the increase in computation time as it uses a new parametrisation by sampling at a one pixel distance from the snake. The increase in time could be very high if the initial snake position is far from the solution. The general disadvantages of the snake approach also apply in this case, such as balloons are not suitable for detecting weak edges where the external forces are not enough to attract the snake. However the use of an internal pressure does introduce a global constraint because in the absence of image forces an additional inflation force is used. The model needs an initial position but no longer needs to be close to the solution to converge. This can be considered as a global active model. Cohen (1991) presents a successful application of *deformable ballons* extracting a ventricle from NMR images of the heart.

3.2.3 Statistical snakes

A variation of the active contours method was proposed by Porril and Ivinis (1994). *Statistical snakes* are also based on energy minimisation mechanisms for searching for edges or regions. The snake grows until its elements find pixels with intensities within two or three standard deviations from the mean of the original sample, whereas

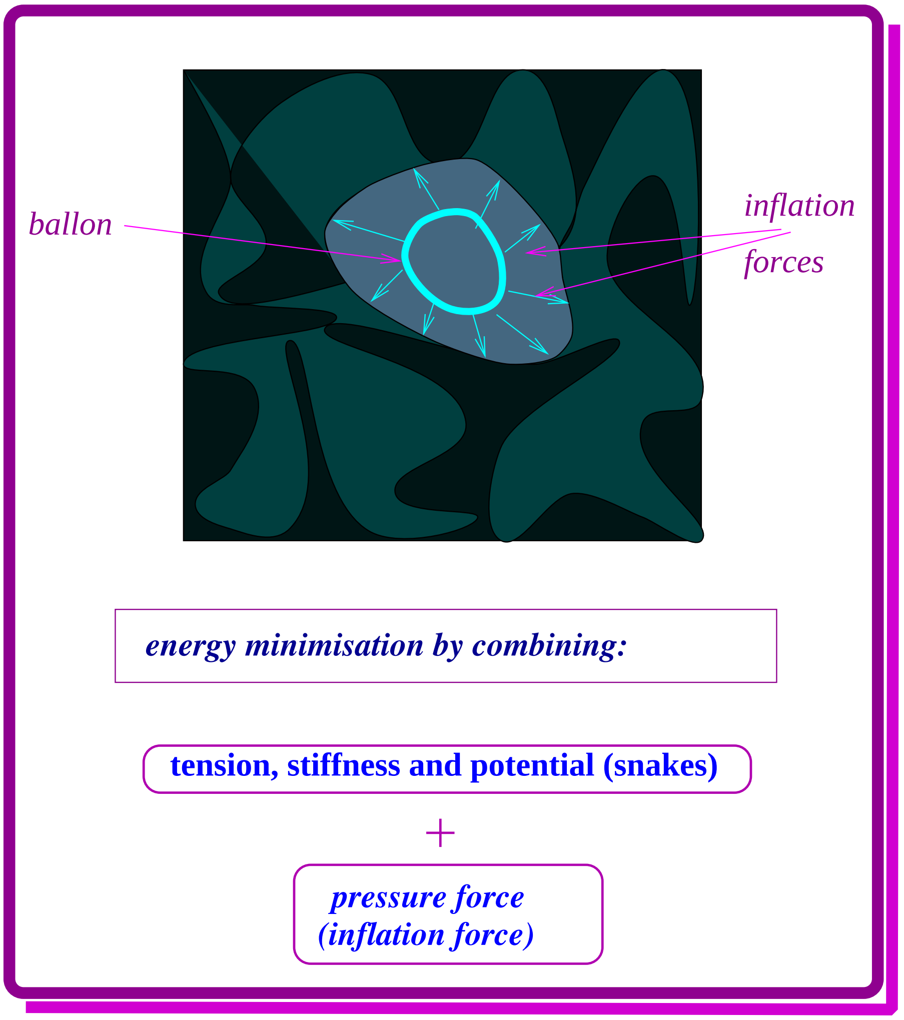

Figure 3.4: A simple representation of an active balloon model. The contour behaves like a balloon pushed by an inflating force.

the snake shrinks at the points where the image intensity exceeds the statistical limits. *Statistical snakes* expand, contract and deform to fit the object of interest. The process is performed as follows: a seed region is defined near the structure of interest, where the pixels inside the region model are sampled and their mean and standard deviation are calculated; the boundary of the seed is then converted into a statistical snake which expands until its elements find pixels which lie outside of the limits predefined by the statistics of this seed. If the model is copied to another image and exposed again to new forces, the model then deforms to fit the new data (figure 3.5).

Statistical snakes have been applied to anatomical structures segmentation and 3D

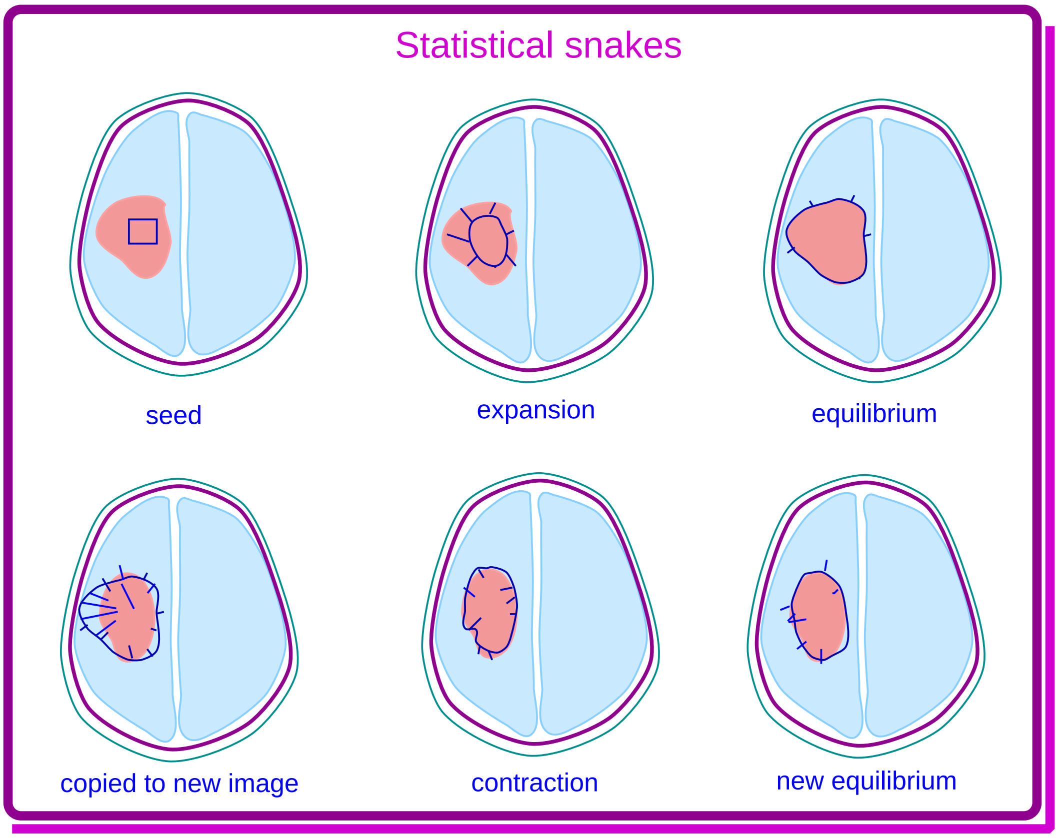

Figure 3.5: Diagrammatic representation of an statistical snake applied to two different image regions, after Porril and Ivinis (1994).

reconstruction from magnetic resonance (MR) and CT scan images, Ivinis and Porril (1994) present an example applied to tumour segmentation in brain NMR images. These active region models work well in some apparently homogeneous tissue, but cannot distinguish regions with similar statistical characteristics.

3.2.4 Bayesian approach

A Bayesian approach to dynamic contours was proposed by Storvik (1994). He uses Bayesian statistical theory to incorporate *a priori* information which is built into the active contour object representation. In this approach the image representation is done under the assumption that the image consist in only one simply connected object. Then he uses a conditional probability density function to combine the observed image with the contour. Figure 3.6 includes the object contour representation and the steps of the Bayesian paradigm to construct a posterior distribution from which the inferences about the contour are made.

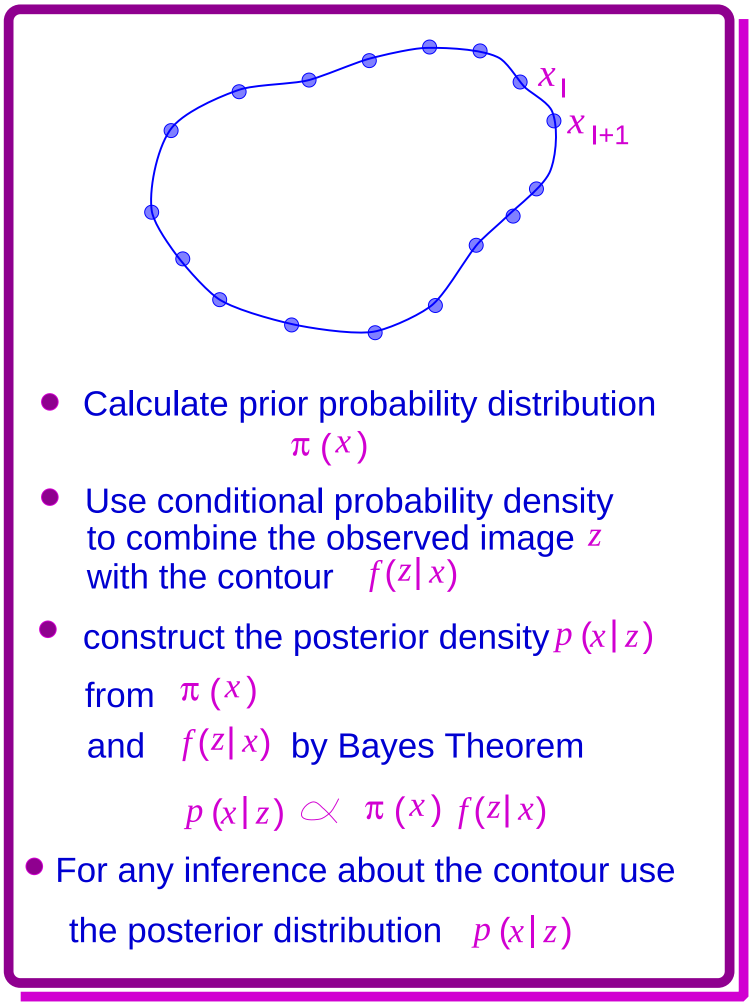

Figure 3.6: Representation of the a connected object contour and steps for the Bayesian paradigm, after Storvik (1994)

The energy is minimised by maximising the posterior distribution, *i.e.* by finding the maximum *a posteriori* (MAP) estimate. Then the curve moves dynamically towards the global minimum energy configuration by applying the optimisation method of simulated annealing. Storvik (1994) applied this approach to ultrasound images of the left ventricle and magnetic resonance images (MRI) of the human brain. Some of the disadvantages in this approach are the assumption that the image consists of only one object and its high computational requirements.

3.2.5 Snake generalisation

The dynamic contour approach proposed by Lai and Chin (1994) is a snake generalisation or *g-snake*. The *g-snake* shape formulation encodes any arbitrary contour in a shape matrix which is invariant to effects of rigid motions such as scaling, rotation, stretching and dilation. This shape formulation is defined in equation 3.1,

$$AU^T = 0 \tag{3.1}$$

A represents a contour vector containing an ordered set of points $V = [v_1, v_2, ..., v_n]$. U is a vector which represents any point displacement from an arbitrary reference point g, *i.e.* $u_i = v_i - g$. The invariant shape matrix is used for the global changes due to rigid motions, and combined with the use of Markov random field to exert control over local deformations for non-rigid objects. To extract variable contours from noisy images, Lai and Chin (1994) used a maximum *a posteriori* estimation under a Bayesian framework $(p(U, g))$. This is equivalent to the energy minimisation of the *g-snake*. This approach shows good results in classifying variable objects, however the *g-snake* initialisation is critical in images with severe clutter and deformation.

3.2.6 Template models

A deformable template approach, using a parameterised template for geometric object representation, was reported by Yuille et al. (1989), Yuille (1991), Yuille et al. (1992). These deformable templates provide *a priori* information about the expected object features. The parameters are changed in a flexible way and interact dynamically with the image to match themselves with the image data. The parameter set is used in an energy function which measures how well the template fits the image. The best fit corresponds to the minimum of the energy function, and therefore the detection process is guided by the energy function minimisation which attracts the template to salient image features (*e.g.*, edges). This approach specifies a probabilistic detection model in terms of the possible object deformation (imaging model) corresponding to the variation of the parameters (prior probabilities). One of the disadvantages of this approach is the need for preprocessing to set the initial values of the template parameters. This approach has been applied to face recognition (Yuille et al. 1989; Yuille 1991; Yuille et al. 1992) and the number of parameters used in the templates designed is relatively small (11 for the eye and 10 for the mouth). This could represent a problem in applications where more parameters are needed given that the computational cost increases exponentially with the parameter dimensionality.

3.2.7 PCA based template models

Bennett and Craw (1991) presented a combination of geometric parameterised templates and the statistical knowledge derived from the varied nature of the shape features. In this approach the search is performed only to feasible representations determined by this statistical knowledge, reducing in this way the large search space. This approach, and those reported in Craw et al. (1992) and Costen and Craw (1996), have been applied mainly to facial features. Craw et al. (1992) developed a face recognition system by using a polygonal template outline which is deformed to fit the data. The deformation is driven by the statistical knowledge about the shape variation and it is optimised by simulated annealing. They measured the overall fit (*f*) of the head outline to the image as the product of the shape (outline) and edge (grey-scale) parameters, applying the simulated annealing interactively when each new outline is generated. For the face internal features they combined random search techniques and simulated annealing to avoid non-global extrema with a fitness function similar to Yuille's approach described in section 3.2.6, using information about the edge and grey-level information. In a comparative study for automatic face recognition, Costen and Craw (1996) obtained eigenfaces (eigenvalues and eigenvectors) by applying PCA to normalised face models (34 landmarks manually marked excluding the hair). They applied an active shape model where new models were generated from the mean model up to two standard deviations. The normalised model obtained was then used to get the best possible fit. One of the matching choices that they tried was the Mahalanobis distance which was found effective in combination with the eigenface formulation and its variance properties. Mahalanobis distance is a way of determining the similarity between an unknown sample and a set of values measured from a collection of known samples (Duda and Hart 1973). Since the Mahalanobis distance is measured in terms of standard deviations from the mean of the training samples, the reported matching values give a statistical measure of how good the matching between samples is (Bishop 1995).

3.2.8 Point Distribution Models (PDM)

This approach is based on the statistics of aligned shape templates and their main modes of variation, characterised by applying Principal Component Analysis (*PCA*) to a set of object examples. With this technique, a compact object representation is built within the limits imposed by the training set (Cootes et al. 1992; Cootes and Taylor 1992; Cootes et al. 1992; Cootes et al. 1995). These compact representations include inner-class variability, derived from the statistical analysis of the object class descriptors from a set of examples (Lanitis et al. 1995). The descriptors used to generate *PDM* are tailored shape point positions (*model landmarks*) equivalent to all the objects of the same class. Figure 3.7 illustrates an example of a training set for an object class and how the object descriptors are represented as a feature vector.

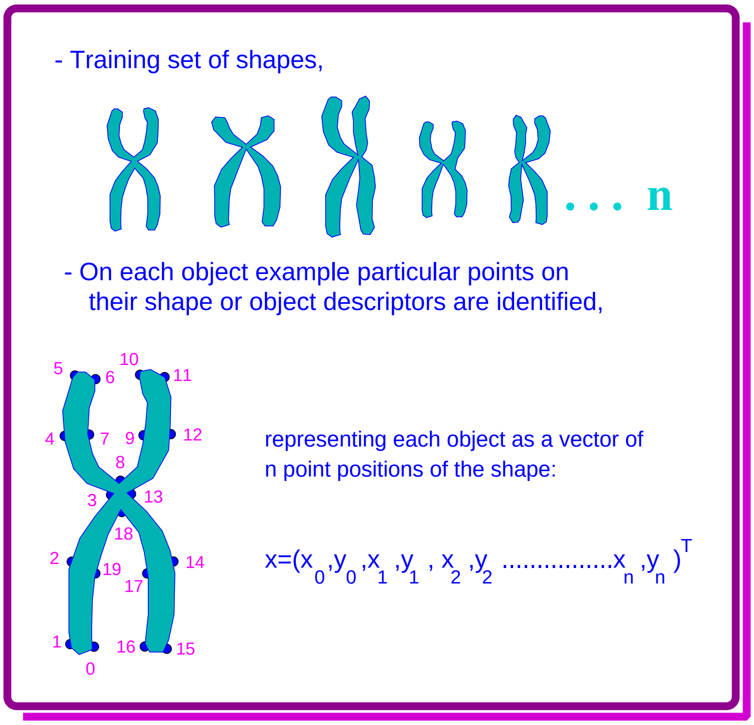

Figure 3.7: Representing the object examples in the training set as feature vectors.

The covariance matrix $\mathcal{C}$ of the feature vectors in the training set is calculated by using the distances between mean vector and each instance as follows (Bishop 1995):

$$\mathcal{C} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}_i - \overline{\mathbf{x}})(\mathbf{x}_i - \overline{\mathbf{x}})^T \tag{3.2}$$

The covariance matrix is then used to obtain the eigenvectors and eigenvalues during the calculation of principal components analysis (PCA)[1]. This technique is considered a classical procedure of multivariate statistics to find a lower-dimensional representation which accounts for the variance of the features while preserving as much of the relevant information as possible (Duda and Hart 1973; Jolliffe 1986; Bishop 1995). PCA considers p variables $(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_p)$ and generates linear combinations to produce indices or components $(\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_p)$, which are uncorrelated and therefore measure different dimensions and express the statistical variability in the data. The indices are ordered so that the first component $\mathbf{z}_1$ corresponds to the largest eigenvalue λ_1 and describes the largest amount of variation. Most of the variability in the data set can be normally represented by the first few components especially if the original variables are highly correlated. A detailed description of this technique and a full discussion can be found in Jolliffe (1986).

New object examples can be generated from the model using the derived mean feature vector and the main modes of variation, in the notation of Cootes et al. (1992):

$$\mathbf{x} = \overline{\mathbf{x}} + P\mathbf{b}, \tag{3.3}$$

where $\mathbf{x}$ is the instance feature vector, $\overline{\mathbf{x}}$ represents the mean shape, $P = (\mathbf{p}_1, \mathbf{p}_2, ...\mathbf{p}_t)$ is a matrix of the t orthonormal unit eigenvectors of the covariance matrix and $\mathbf{b}$ is a vector of shape parameters which defines the instance (see figure 3.8). By generating new instances, the model deforms to find the best fit with the data. The limits for these deformations are defined together by the training set of object examples.

This generic approach has shown better results in speed of convergence and optimality of solution than other approaches, but is applied mostly to shapes. In the case where

[1] This technique is also known as eigenvector, Hotelling or Karhunen-Loève transform (Gonzales and Wood 1992).

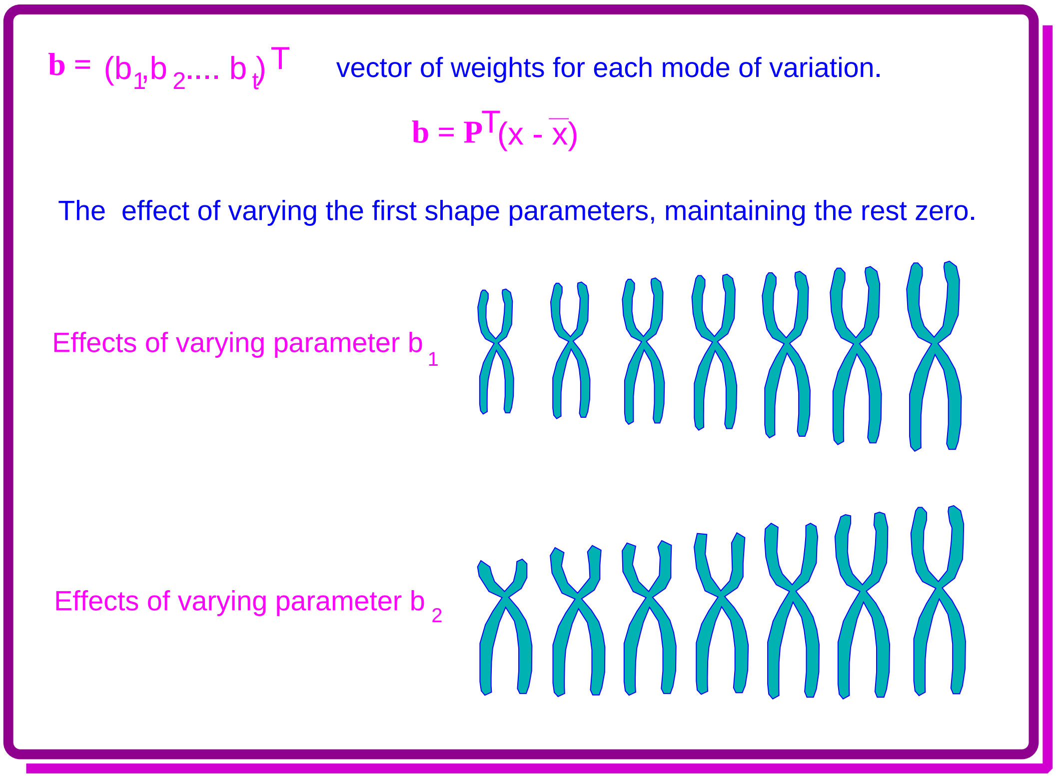

Figure 3.8: New object instances can be generated by modifying the set of shape parameters **b** in the above equation.

grey level information was included, the model was applied in images of simple natural objects such as bananas and human eyes (Cootes and Taylor 1994) where they did not found a strong link between shape and grey level. *PDM* have been tested with some shapes within biomedical images (Hill et al. 1994; Cootes et al. 1994). A key step in these approaches is the location of identifiable points in the object shape to generate the *PDM*. In the case of the nervous system, cells have highly variable boundaries, and therefore their shape is difficult to characterise by using a *PDM*.

3.3 Summary

This chapter has reviewed the model-based framework, with particular attention to flexible models. Despite the fact that flexible models have been successfully used for some non-rigid object recognition applications, they have several disadvantages in some biomedical applications. For example, some of the methods described require a starting model position near the object of interest in order to find the solution, this would result in a highly interactive process. As described in chapter 2, for this thesis application a semi-automatic method is needed given the massive amount of material to be analysed. A number of the flexible models, presented in this chapter, work with single object images; this characteristic is also incompatible with the need for semi-automatic analysis. In several of the approaches described here, a well defined shape contour is needed. These are not suitable for some biomedical applications where the objects to be recognised often present weak edges in images and have poorly defined shape contours. In the following chapter the bases of this thesis approach (*SGmodel*) are presented. This novel generic model representation for recognition and classification integrates different methodologies to tackle most of the problems described here, with the capability of handling the inherent variability of irregular objects within a probabilistic context.

Chapter 4

SGmodel Approach

4.1 Introduction

The objects of interest in the domain of application in this thesis are irregular, and there is a relationship between their density profile and geometric shape given the nature of the object staining. The staining process is related to the total quantity of the substance for which the stain is specific and therefore the grey-level profile is strongly related to the projected shape when imaged in transmission mode.

Figure 4.1 shows two examples of microscope images from brain tissue, from the images used in this thesis application. The objects in these examples vary in shape and size within classes and their boundaries are not always well defined. The background also presents high variability in noise and grey values.

In order to use all the knowledge about the appearance of the object, it is important that a model has the ability to recognise direct links between shape and grey values. As described in chapter 3, *PDM* is a generic approach which has been successfully used with shapes. In this chapter a more complete model is proposed by extending the *PDM* feature space to include grey values directly in a shape grey-value model namely *SGmodel*. The indirect use of the grey value information has been reported in several works such as Cootes et al. (1992), Cootes et al. (1993), Cootes et al. (1994). In these works, *PDM* has been used in combination with grey value information around each *model landmark* to suggest the direction towards a better fit of the model with the image object. The statistics of the grey levels around a given point corresponds to a particular pattern on the object shapes in images of different examples, and therefore they search for the area where the image best matches the grey-level environment model for each point. Cootes and Taylor (1994) have also included grey value information in their deformable models but for their applications the correlation between the shape and image intensity was weak. The importance of this technique will be apparent when the correlation between shape and image value is strong. The main features that make *SGmodel* suitable for microscope image analysis are also discussed in this chapter.

4.2 Shape-Grey Flexible Models

In *SGmodel*, statistical knowledge of shape and grey values of biomedical objects is used to produce a generic model representation for detection and classification. The main difference between the *SGmodel* approach and earlier *PDM* applications is that the feature vector contains all the spatial coordinates plus grey-level information. Another difference is that, given the variable or poorly defined boundary shape of the objects to be classified, the shape knowledge is based in the grey values distribution sampled in a radial template designed for these models (see section 5.5.1). The grey-level data could be the actual pixel values or the result of some kind of prior image processing, for example, averaging, gradient operator or some specialised technique such as profile analysis. The model feature vector $\mathbf{x}$ (in eq. 3.3) is augmented by the grey-value information to a general feature vector denoted $\mathbf{f}$

$$\mathbf{f}_i = \overline{\mathbf{f}} + P\mathbf{b}_i. \tag{4.1}$$

Where $\overline{\mathbf{f}}$ represents the mean of spatial coordinates and grey values calculated over a training set. The matrix of the main modes of variation is built by selecting the most significant n eigenvectors: $P = (\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, ..., \mathbf{e}_n)$. The first modes account for most of the variation in the original variables, the number n is selected according with the number which accounts for the 90% of the total variation. The vector $\mathbf{b}$ represents the weights for each mode of variation and is calculated as follows: $\mathbf{b} = P^T(\mathbf{f} - \overline{\mathbf{f}})$.

SGmodel is intended to deal with microscope images[1]. In this type of images the objects

[1] It could be used with any transmission type images such as X-ray, computed tomography (CT) and

(or their components) can be enhanced from the background or from other objects by the stain technique used in the slide preparation (Parker 1994). The staining combined with the illumination conditions[2] during the image acquisition process, produces a strong link between density profile and geometric shape. Finding the relationships between shape and grey level of the object, and providing the model with the ability to recognise them, allows the use of more information about the appearance of the object in the image analysis process, and thus deal with the shape variability.

The model acts as an abstraction of the high-level prior knowledge about the objects, designed in a meaningful way for the domain expert. The imaging process is also defined within the model so that, in principle, it is possible to generate instances of the objects as they would appear in the images by selecting model-parameter values.

This model can be thought of as an objective function with two main elements; the first measures the deviation from the mean shape-grey values; the second evaluates the match of the flexible model with the underlying image. A genetic algorithm (GA) is used as an optimisation tool to maximise or minimise the overall objective function. GA based methods are robust and suitable for the type of search optimisation problem found in this thesis (see section 5.6.4). However other optimisation techniques can be also used.

In model-based vision, the matching process seeks the highest probability of the model variables (*i.e.* the best possible explanation) with respect to the image data. This can be calculated using Bayes rule (Duda and Hart 1973; Gonzales and Wood 1992; Parker 1994; Bishop 1995):

$$Pr(M_i|I,C_i) = \frac{Pr(I|M_i,C_i)Pr(M_i|C_i)}{Pr(I|C_i)} \tag{4.2}$$

Where M_i is a model of the class C_i of an object observed within the image data I. For the purposes of matching, the denominator can be ignored since it is independent of the instance parameters.

MR images but in these cases the staining intensity (density) is not related to shapes.

[2] The illumination comes from behind the object being digitised in the optical microscopes, producing images known as transmission images.

We expect the image data to be a combination of values from two sources: the object or foreground and the remainder or background. In the case of transmission light microscopy images, we expect the contributions to the measured optical density of foreground and background to be additive. In other imaging models a strict partition might be appropriate, even though blurring (point spread) and digital approximation, the pixels around the edge of an object will always have contributions from both foreground and background. Therefore we divide the image into two parts:

$$I = I_M + I_B \tag{4.3}$$

where I_M and I_B are the model and background pixel values respectively, and therefore

$$\begin{aligned} Pr(I|M_i, C_i) &= Pr(I_M, I_B|M_i, C_i) \\ &= Pr(I_M|M_i, C_i)Pr(I_B|I_M, M_i, C_i) \end{aligned} \tag{4.4}$$

Assuming that the foreground and background image values are produced independently, then

$$Pr(I_B|I_M, M_i, C_i) = Pr(I_B|M_B) \tag{4.5}$$

where M_B is the background model. The instance probability is:

$$Pr(M_i|I, C_i) = \frac{Pr(I_M|M_i, C_i)Pr(I - I_M|M_B)Pr(M_i|C_i)}{Pr(I|C_i)} \tag{4.6}$$

It should be noted that the estimation of what evidence arises from the background depends on the model instance, and finding the best fit instance requires a background as well as a foreground model.

If the distribution of the principal modes of variation is assumed to be normal, then the model instance probability can be written, in terms of the Mahalanhobis (see section 3.2.7) distance D_{M_i}:

$$Pr(M_i|C_i) \propto e^{-D^2_{M_i}}; \qquad D^2_{M_i} = \sum_{j=1}^{t} \frac{b_j^2}{\lambda_j^{C_i}}, \tag{4.7}$$

where t is the number of principal components used to define the model space and $\lambda_i^{C_i}$ is the eigenvalue of the jth eigenvector for model class C_i.

The model training (see detailed description in 6.5) is performed by sampling grey-level values through object examples in a set of images. The sampling is done by using the defined model template and the feature vector per object includes spatial and grey-level information. The inclusion of non-spatial data dimensions brings to the foreground a problem which is present in the usual *PDM* applications but has been ignored so far, namely the normalisation of coordinates. This becomes a problem when the original space of variation is truncated to the most important principal axes. The truncation is performed on the basis of the observed variation which is clearly dependent on the relative normalisation of the spatial and grey-level feature values. The truncation in this approach takes the number of modes of variation which account for the 90% of the total variation. In fact this problem is inherent in any model interpretation of data (*linear regression analysis*) including the *PDM* applied to purely spatial coordinates for which, in general, the measured errors are not isotropic.

Another normalisation problem arises when models of different classes have different parameters and/or different number of dimensions. This means that the probabilities calculated by matching each independent class model to the data cannot be compared. One possibility is to attempt to establish all the contributing terms in equation 4.6 but this would prove extremely difficult. An alternative is to treat each matched model applied to new instances as a measurement feature and to build a classifier (see description next) based on the full set of features, *i.e.*, the model feature vector for each class concatenated to form an extended feature vector (*classifying model*).

To build this classifier the training sets used for the individual models can be re-used. The process is to match each member of a training set A with the models of all classes and thereby generate an extended feature vector for each member of each class, this is described in detail in chapter 5. These can then be used to derive a classifier. In this case *PCA* was used for each class and to calculate a probability in the Mahalanobis distances, but any suitable classifier could be used, for example, *K nearest neighbour* (Duda and Hart 1973), or *Neural nets* (Bishop 1995).

4.3 Conclusions

The *SGmodel* approach proposed in this thesis, presents features which make it suitable for biomedical image analysis. These features can be identified as follows.

- That the model acts as an abstraction of the high-level prior knowledge of the objects, designed in a way that is meaningful to the domain expert.
- That it is based on the statistical knowledge from a training set of examples. The main modes of variation of each object class are represented in a generic object formulation. During the object recognition process, this formulation adapts to the diversity and irregularities of shape and grey-values distribution.
- That the variability of the shape-grey relationships is characterised by applying principal component analysis (PCA) to the shape-grey level vector extracted from the training set. Given the modes of variation, it is possible to generate new examples to be used to test the training set distributions, followed by a test matching of new data.
- That during the matching process the obtained model is adjusted to fit the image data within the limits of variability from the training set.
- That it includes a mechanism to recover the normalisation given the inclusion of non-spatial data, different model parameters and/or different number of dimensions.
- That the retrainable nature of this model makes it adaptable to new conditions, allowing rapid prototyping of new object classes.

In most of the model-based image analysis applications, the required domain expert information is represented and integrated in a technical language. These models are usually implicit and unmodifiable by the domain user (Staib and Duncan 1992; Robinson et al. 1994).

An important contribution of this approach is a more generic model for the domain application which, in turn, implies that the model can be tuned to specific study cases, without the need for specialist image-processing and programming expertise, to incorporate new models for each object class. This contribution will become apparent in the implemented model which is described in the next chapter. The algorithms are developed not only using the domain knowledge but also in a way that makes the models explicitly accessible to the user in the domain language. This characteristic supplies the domain expert with a straightforward tool which eases the understanding of the models involved in the system, and provides the possibility to adjust and retrain them or to define completely new models. This results in a high adaptability to new conditions and allows rapid modelling of new object classes. Finally one of the the underlying features presented in this work, is that the probabilistic formulation provides the means to solve the normalisation problem produced by including non-spatial data dimensions and allows classification when the feature space is different for each class.

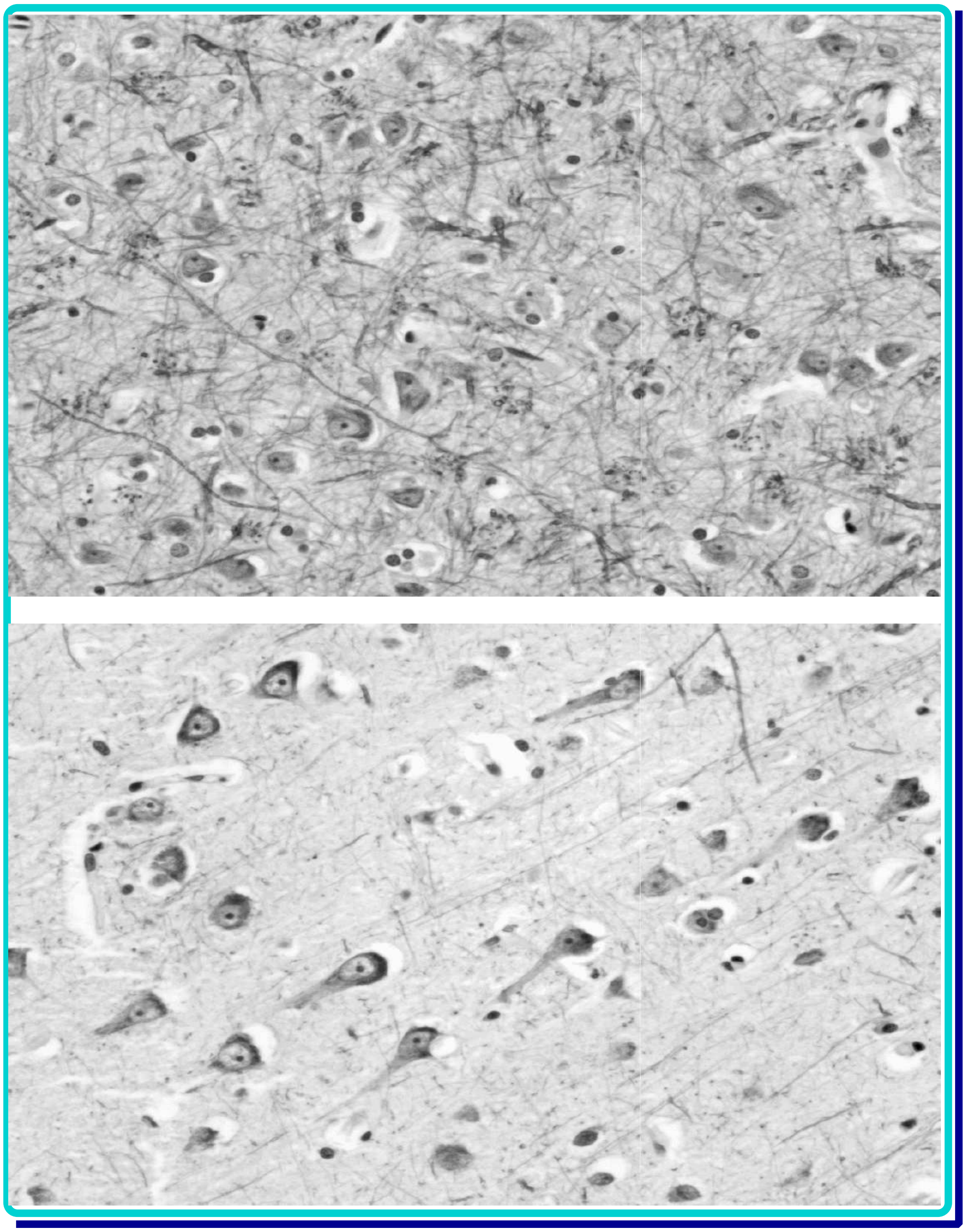

Figure 4.1: Two examples of the microscope images from brain tissue. There is high variability of shapes and sizes within object classes, and also in the background between images.

Chapter 5

SGmodel: system design and implementation

5.1 Introduction

This chapter presents the design of the functional and operational characteristics used to implement the *SGmodel* into a computational system for biomedical image analysis. The present study covers different aspects of image analysis which, as mentioned in chapter 3, are normally classified from *low-level* (*e.g.*, image formation) to *high-level* (*e.g.*, image interpretation), (Gonzales and Wood 1992; Sonka et al. 1993; Castleman 1996). The interaction of the *SGmodel* system at these different levels of image processing within a more general image analysis system is also described in the following sections. This chapter also presents an artificially generated cell model (section 5.5.5). This model was designed to observe, in a controlled fashion, the behaviour of the variation modes obtained by the *PCA* applied to a training set. The design and implementation of the system are presented according to the specific needs of the biomedical image analysis and in relation to the Shape-Grey values model described in the previous chapter.

5.2 The general image analysis context

The *SGmodel* system is intended to form part of a more general image analysis system shown diagrammatically in figure 5.1. In this figure the *SGmodel* modules, at the different levels *Low, Intermediate and High* of image analysis, and their interaction with the *Shape Grey value Model*, are represented within the dotted line. The purpose of this section is to give an overview of the general context of the *SGmodel* system. The following sections present a more detailed description of the *SGmodel* design.

- *Low-level*: vision is related to image formation and transformations such as noise reduction and image enhancement (Castleman 1996; Gonzales and Wood 1992). The *low-level* vision modules include image acquisition and storage, these provide the input material (digital images) to be analysed by the system. *SGmodel* does not include these as built-in processes, but the digitisation environment is set to optimise the input images. The digitisation conditions are described in the methodology chapter 6 and are included as a module in the proposed design of the general image analysis system shown in figure 5.1.

- *Intermediate-level*: The digitised images obtained in the *low-level* are the input information to the *intermediate-level*. This level deals with the extraction, characterisation and representation of features from the image. These operations are important because the recognition and interpretation processes depend on appropriate feature extraction and representation. The feature extraction in *SGmodel* involves the model's definition and training, and the characterisation of the model class variability with *PCA* (section 5.5). The feature representation is described in section 5.5.3, and an example can be found in section 6.5.

- *High-level*: The functions of detection and interpretation are performed in this level. The image information must be interpreted in the specific application context. The processes are often based on model representations, image based knowledge systems, search and optimisation techniques, decision support systems and other tools, some of which are used within *SGmodel* (section 5.6). At this level the output information is used in a feedback loop interacting with the module *Shape Grey value Model* towards a refinement process which ends with a final interpretation output.

The general *Model-Based System for Automated Biomedical Image Analysis* has also two external modules; *Pre-processing* and *Non-image information* described in the following sections.

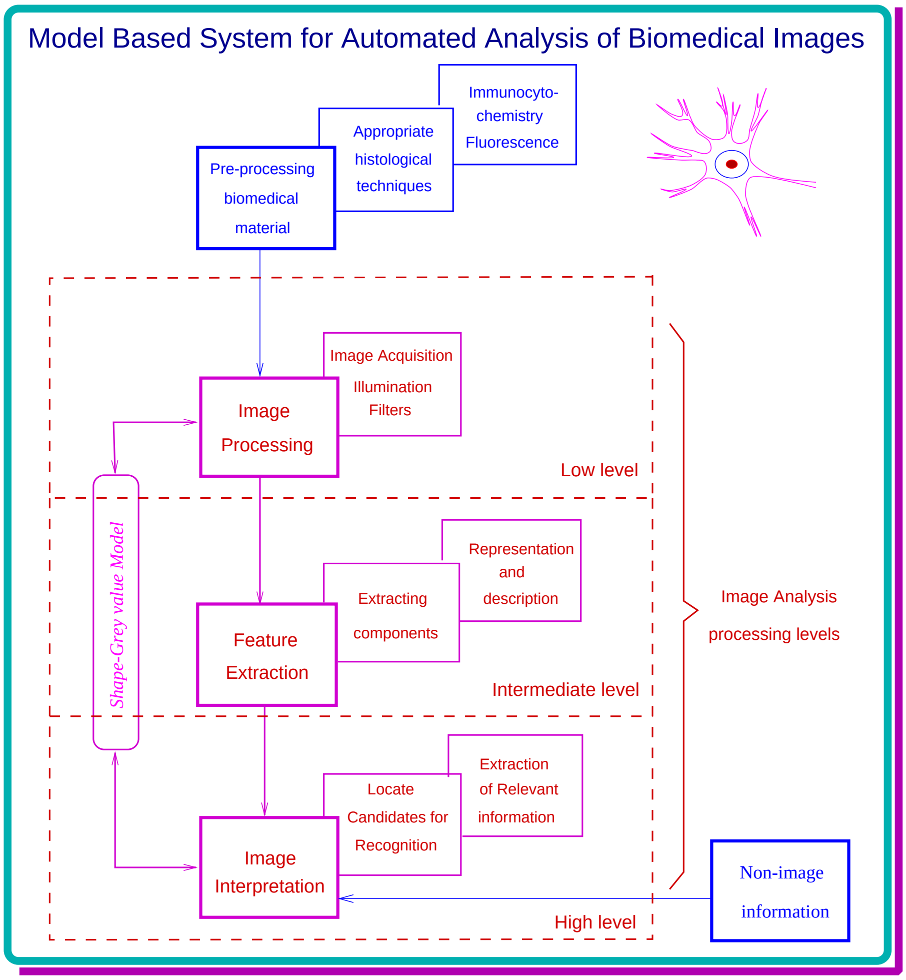

Figure 5.1: *SGmodel* in a general image analysis framework.

5.2.1 Pre-processing biomedical material

The pre-processing includes the tissue preparation into stained slides to be digitised. The microscope images in this study are transmission images where the absorption of the light, which has come through the tissue, at each point is a measure of the optical density, and in which measurements vary between tissue sections of different thickness (Marchevsky and Erler 1994). Consequently the thickness of the section plays an important role, as some features of the objects of interest can be enhanced or occluded as the tissue thickness varies (Warren 1992). For each application several tests should be done to choose the most appropriate size. For example, if the tissue for digitisation has very dense objects, then it is possible that a thinner section is going to be needed for a better contrast in the images. Staining also plays an important role as it could be specific to enhance different types of cells or structures of interest from the background. The tissue thickness and stain used for this work are described in chapter 6.

5.2.2 Non-image information

During the usage phase, interactions of the *SGmodel* with the three levels of processing are part of a loop of the generic hypotheses-test-refinement process within the feedback mechanism which allows for auto-optimisation. The refinement should combine all the information obtained at the different analysis levels together with non-image information (case history, patient data, histological processing, *etc.*) that is available at the high-level.

5.3 A user interface framework

In designing an automated model-based system for image analysis, targeted to process large amounts of material, it is convenient to include a user interface which eases the use of the processes available for the application user (Johnson 1989).

In this work, the system[1] developed was designed to have a menu-driven Graphical User Interface (*GUI*) for reducing, as much as possible, the effort required to activate the general system processes, such as display of the application's images on the screen, location and inspection of areas of interest, interactive model parameters settings, building training sets, classification performance on a displayed image, presentation of graphical results, and the possibility of corrective retraining when errors are detected. The following sections describe the design of the system image processes implemented in the three modules: *Image and data file manipulation*, *Model actions* and *Analysis actions*. The structure of the system design is summarised in figure 5.2. In this figure *GA* stands for Genetic Algorithm, an optimisation tool described in section 5.6.4.

[1] Appendix B includes a detailed description operational system.

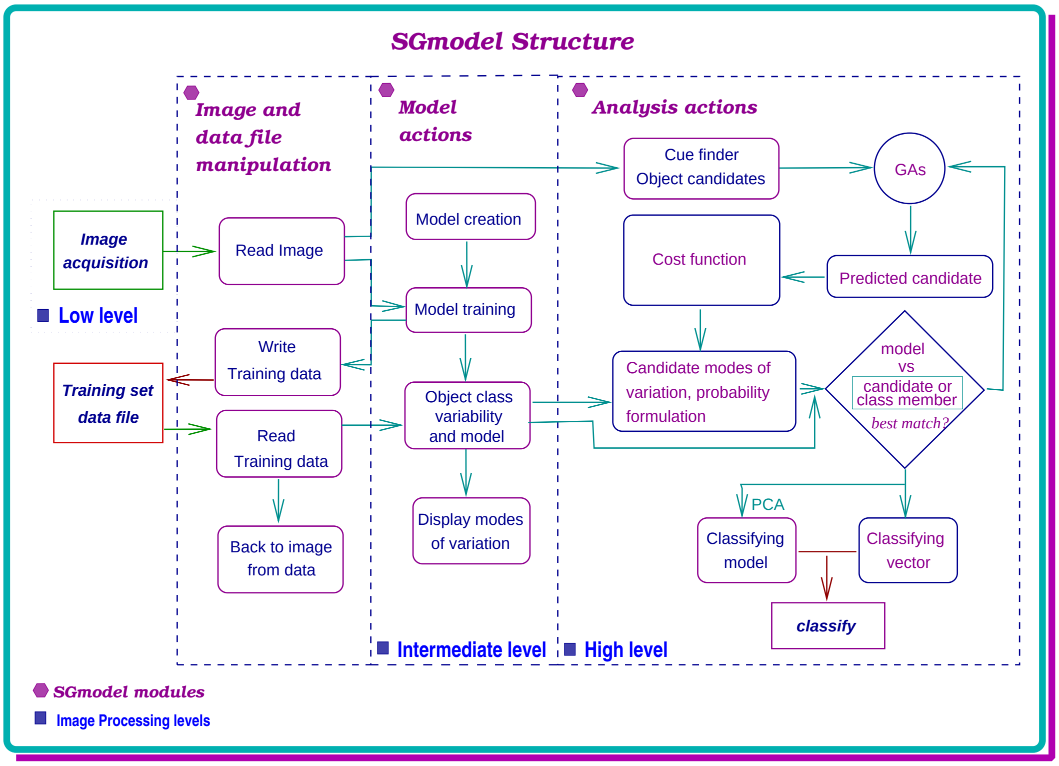

Figure 5.2: SGmodel structure.

5.4 *Low level*: Image and data file manipulation

The digitisation environment affects the image quality. Many image processing operations are procedures used for image correction and there are complete works dedicated to the improvement of images quality or correction of defective images (Rushford 1968). It is difficult to control all these factors and in many cases the image processing starts with already digitised images. In this work the digitisation conditions can be set in order to obtain an image quality which favours the image processing and analysis. Some of the factors which could affect the image quality are: the tissue thickness and stain (*pre-processing*), the microscope illumination, the use of complementary colour filters, the camera and the digitiser settings.

This module (see figure 5.2) was designed to allow the application user to:

- read and display digitised images;

- read and write the training data generated with the *SGmodel* system; and
- to go back to an image from the data information in order to visualise the set of object examples with which the model was trained and perform any adjustments, if needed.

5.4.1 Read Image

This process provides the user with the capability of reading a digitised image stored in a memory device and to display it in the main window of the system user interface. The displayed image is then used as input to the other modules, such as model training, matching, and classification.

5.4.2 Read/Write training data

An application program should provide a mechanism for storing and transferring data files to and from the memory device. *SGmodel* allows this for the data produced by the model training process. The information stored for a model training set should include: model name, model information, image filename, number of examples in the training set and instance features (see example in section 6.5). This information is stored in a text file to allow direct data checking and editing. In section 5.5 (*Model actions*) model information and instance features are described.

5.4.3 Back to image from data

The training set information is stored in a data file (section 5.4.2). The procedure *Back to image from data* reads the data file with the training set information and produces a list of the images used to train the model. From this list, an image can be displayed on the main window of the *SGmodel* system. The selected image displays the points sampled on each object. This feature allows to check how the sampling was performed and can be used as an aid to understand the results obtained, and to determine possible adjustments in the model settings. This procedure also allows automatic re-sampling which could be useful if different tests are needed. The automatic re-sampling could also be useful in the case that some filtering is required to enhance some features in the images.

5.5 *Intermediate level*: Model actions

The problem addressed in this work, of different object classes location and identification, is a pattern recognition problem. In pattern recognition, the two main components are feature selection and classification. The former is described in what follows, while the latter is presented in section 5.6 as part of the *Analysis actions* module (figure 5.2).

It is important to make an appropriate selection of characteristics which retain most of the relevant object information and which are going to be used to identify the objects. The classification process is based on these features which will lead to clear distinctions between different object classes (Bishop 1995). The knowledge of the application experts, *prior knowledge*, is the knowledge available about the sought solution which is often required in the selection of features, and can result in improving the classifier performance. The dimensionality of the features should be kept as small as possible because the complexity and the amount of data required to train the classifier increases as the number of features increases.

5.5.1 Shape-Grey values template

Model features were designed to be simple in order to allow quick training and testing while trying to retain most of the relevant object information.

Given the specific characteristics of the objects (cells), the model was defined as a radial template. The model features such as the number of radii and the number of points per radius to be sampled are determined at the beginning of the training process. The selection criterion is based on training the model with different number of features and evaluating its performance. The time needed to try different combinations of feature size depends on the data set size, this process is only performed once to obtain the optimal feature size to train the model.

The module *Model Create and Training* was designed for the radial model template definition by specifying the number of radii and the angles at which the grey values are going to be sampled for each object class. In both cases, radii and angles can be fixed or chosen manually by the domain expert during the training procedure using the selection process described before. In the fixed mode, the values given for the point positions for each radii represent a relative distance from the central template position. For the angles, the first angle is always chosen manually (normally according to a constant criterion such as the side with the shortest distance between the centre of the nucleolus and the nucleus boundary), and the rest of them in the fixed mode are set in a position relative to the first one at angles defined in the model, (figure 5.3). The default values are set with the model values defined for radii and angles. This model set is going to be used to obtain a collection of sampled objects or training set.

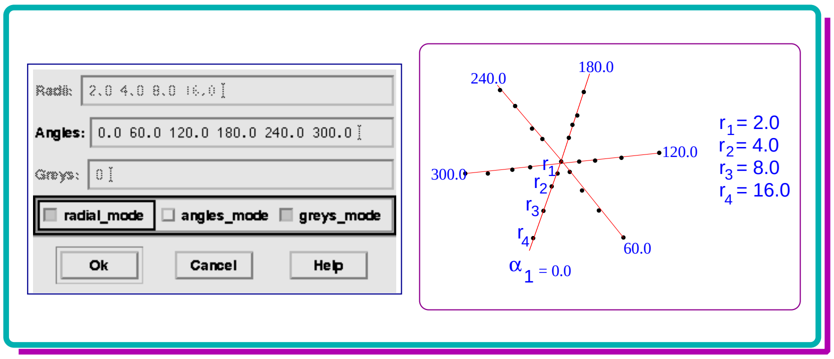

Figure 5.3: Example of a radial model template definition.

5.5.2 Model training

The model training module was designed to produce a training set by sampling object grey-level values in microscope images. The sampling is performed by using the radial model template described in previous section. When the *Model training* procedure is activated, an object in the image is selected by clicking on it with the left mouse button, and then displayed in a magnified window, see figure 5.4. The sampling with the specified radial template is accomplished in a "clicking-mouse" interactive mode. The training set created with this process consists in a set of feature vectors with the form $\mathbf{f}(r_1, g_1, r_2, g_2, ...r_n, g_n)^T$, where $r_i = (x_i, y_i)$ (radii coordinates from where the grey values are sampled in the image). The criteria to choose the sampling point positions would depend on the object to be characterised and would be defined by the domain expert, an example of training using a neuron model is described in 5.5.2.

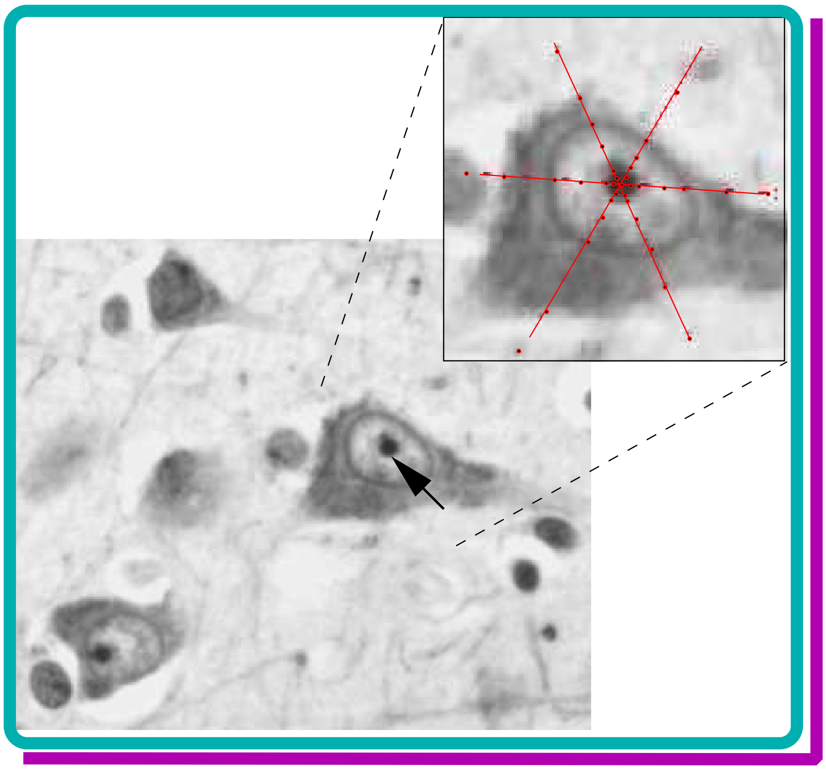

Figure 5.4: Example of training a neuron model with a radial template.

5.5.3 Calculating modes of variation

The *Variation modes* module was designed to characterise the variability of the shape-grey relationship by applying Principal Component Analysis (*PCA*) to the feature vector. *PCA* is, as mentioned in chapter 3, a statistical procedure to find a lower-dimensional representation which accounts for the variance of the features (Duda and Hart 1973; Jolliffe 1986). In other words if the object being analysed is described by a large number of parameters, *PCA* reduces it into a smaller set of new parameters which reflect the variability of the object. *PCA* involves the computation of the mean feature vector and covariance matrix, its eigenvalues and eigenvectors as follows.

The mean $\bar{\mathbf{f}}$ is calculated using the feature vectors $\mathbf{f}$ generated from the number of objects, N, in the training set, using

$$\bar{\mathbf{f}} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{f}_i. \tag{5.1}$$

The covariance matrix $\mathcal{M}$ is then calculated,

$$\mathcal{M} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{f}_i - \bar{\mathbf{f}})(\mathbf{f}_i - \bar{\mathbf{f}})^T, \tag{5.2}$$

and it is a measure of similarity between the i^{th} object values and the mean.

The result is a symmetric matrix $\mathcal{M}$ which has m eigenvectors ($\mathbf{e}$) and eigenvalues (λ) (here m corresponds to the number of features), which satisfy the equation

$$\mathcal{M}\mathbf{e}_i = \lambda_i\mathbf{e}_i. \tag{5.3}$$

The eigenvectors corresponding to the largest eigenvalues describe the modes in which the original values have their largest variance[2]. The information obtained from this process is integrated into the Shape-Grey Flexible model representation. As described in the previous chapter, it is possible to generate model instances by combining the mean shape, the eigenvectors matrix, and the derived shape-grey value parameters (*variation modes*). This in turn is used in several processes, such as *Model Display*, *Construct Classifying model* and *Classify*.

5.5.4 Visualising the object class variation

In order to visualise the *variation modes* behaviour, a *Model Display* module was included in the *SGmodel* design. The compact representation (see eq. 4.1), relates the shape-grey parameters with the modes of variation computed over the training set. This representation is used to generate new examples from the class of shape-grey value within the limits imposed by the training set. The initial information to be

[2] For the *PCA* calculation, the public domain PCA 2.8 code by A. Stolcke, and obtained from the ftp site icsi-ftp.berkeley.edu, was adapted for the *SGmodel* system.

displayed is taken from the training set mean values obtained, such as shape-grey parameters from the nucleolus and nucleus and grey values from the cytoplasm. The cytoplasm is not displayed because its shape shows high variability in form and size as it sometimes continues through long structures (axons and dendrites) which could exceed the image range.

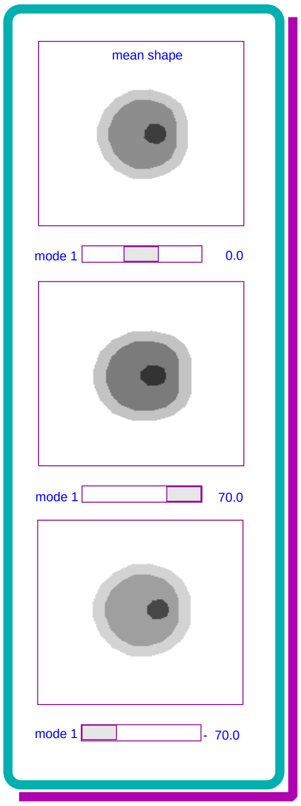

Figure 5.5: Visualising the *variation modes* over the mean shape-grey values.

The mean nucleolus and nucleus shape-grey values and the cytoplasm grey values, following the edge of the nucleus shape, are displayed on a window where it is possible to modify each mode of variation by moving the slide bars which represent each of them as it is illustrated in figure 5.5. This shows to see how the mean shape-grey values are affected by each mode of variation.

5.5.5 Artificial cell model

This module was designed in order to test the system behaviour. The generation of an artificial cell, where the cell parameters are controllable, provides a means to observe the behaviour of the modes of variation. This procedure is useful in understanding how the object variability is characterised by PCA. The model cell was designed to include dependency between some of the spatial features, and is defined with the following model parameters (see figure 5.6):

1. radius of the small circle (nucleolus);
2. distance between centres of the small circle and the ellipse (nucleus);
3. horizontal radius of the ellipse;
4. vertical radius of the ellipse;
5. grey value of the circle;
6. grey value of the ellipse; and
7. grey value of the ellipse boundary.

These parameters are illustrated in figure 5.6, the numbers without a circle denote, radii or distances and the numbers within a circle denote grey values.

A set of artificial cell examples was then generated by random sampling of the artificial model parameters. The sampling is performed as simulating a centre position grey value, 3 position features (parameter 3 and 4 are correlated) with their 3 corresponding grey values along 18 angles, producing a 109 dimensional feature space. The grey values were randomly generated numbers within the limits found in real images, *i.e.*, values from 0 to 255 for the ellipse and its boundary and from 100 to 255 for the small circle. These parameters were chosen to simulate the shape and grey level variation observed in the real cells and in this way to understand the modes of variation behaviour.

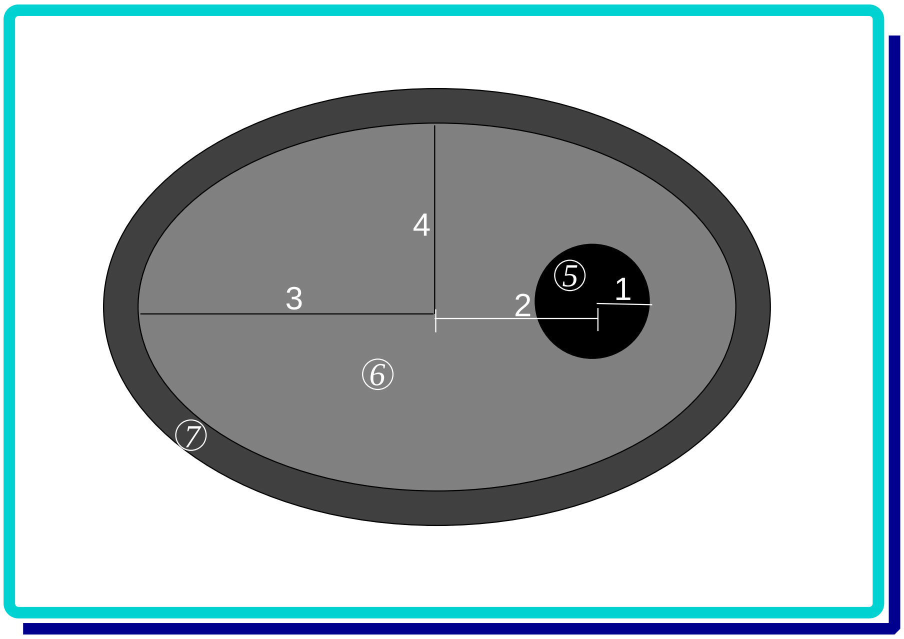

Figure 5.6: The artificial model cell parameters.

5.6 *High level*: Analysis actions

The last module of this thesis approach includes the *high level* image processing tasks such as object detection and classification (interpretation) see figure 5.2.

The analysis process to perform these tasks is divided in two main sections. In the first section an overview of the steps required to classify an object candidate is described, including the generation of *classifying models* in which the training sets of each model are merged together in order to obtain a normalised probabilistic formulation. The object candidate is compared with this formulation to determine the class in which it belongs. This comparison is not directly made with the object candidate but with a *candidate classifying vector*. The latter is built by concatenating the best possible fit between each *matching model* and the object candidate.

The second section describes the utilities used during the classification process. That includes an object candidates locator or *cue finder* and determination of closeness or *goodness of fit* (for the matching process) by computing the Mahalanobis Distances (*MD*) between the model and the candidates. This section also includes the interaction with an optimisation module to produce the best possible match, which is based on a Genetic Algorithm approach (GA, section 5.6.4).

5.6.1 Classification

The general purpose of the analysis process is to detect possible object candidates from a new image, and to calculate their class probabilities based on the models defined and trained, and also on the statistical knowledge obtained during the object class characterisation with PCA. This information, produced at the *intermediate level*, is going to be used by different *high level* procedures, for this reason two *intermediate level* procedures are included in the following sequence of steps that summarises the analysis procedure during the object classification (the step numbers are indicated within circles in the respective figures):

- **(1) Intermediate level** Training models. To produce training sets, for 2 or more models. An example with two models (model A and model B which produce TrSet A and TrSet B training sets) is presented in figure 5.7. These models (A and B) are also used to illustrate the rest of the classification steps.
- **(2) Intermediate level** Generating Shape-Grey level models. The training sets (TrSet A and TrSet B) are used to calculate the mean feature (matchingMean-FeatsA and MeanFeatB) and combined with the PCA calculations it is possible to generate the corresponding model matching, see figure 5.7.
- **(3A) High level** To generate a *classifying model* for each model for normalisation purposes, by considering each *matching model* for each class as a measuring instrument that can be applied to any image or image region. These measuring instruments are then applied to each of the training examples in each class. In the two model examples, an extended feature vector or *classifying model* A is generated. As illustrated in figure 5.8 a set of extended vectors is generated by concatenating the model A feature vectors with the best fit between model B and each object example in the training set for model A, PCA is then applied to this set and the *classifying model* [modelA-modelB] is obtained. The result is a set of features that are directly comparable and therefore can be used to generate a

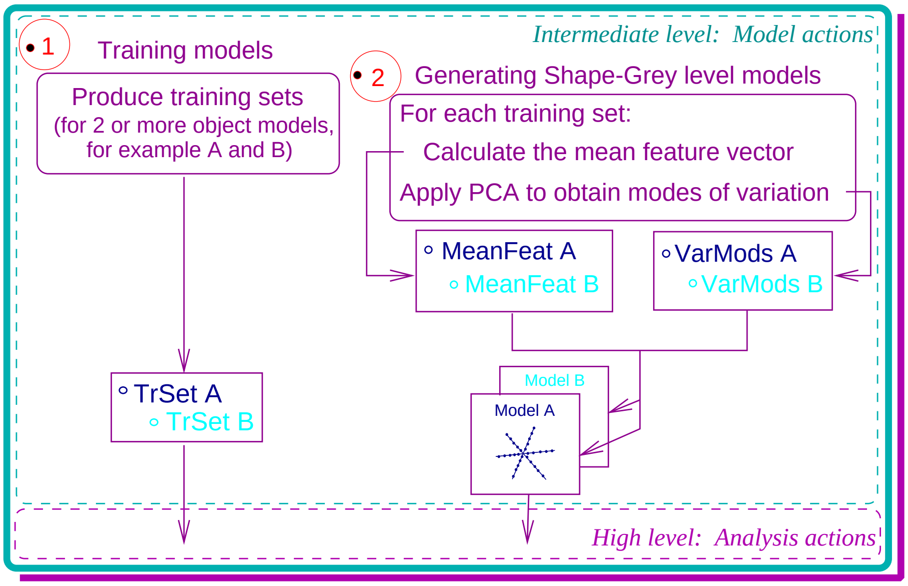

Figure 5.7: Model action producing information to be used at the High level: Analysis actions, *i.e.*, training sets and Shape-Grey value models (model).

classifier (*classifying model* [modelA-modelB]).

- **(3B) High level** In this step a similar procedure as the one described in the previous step (3A) is performed, an equivalent extended model or *classifying model* [modelB-modelA] is obtained, see figure 5.9.

- **(4) High level** Generating the *candidate classifying vector*. In order to perform this step, the *matching models* generated at the *Intermediate level* and the possible candidates found by the cue finder, are needed. For the classification of object candidates, an extended feature vector (*classifying vector*) is produced in a similar way used to generate the *classifying model*. The difference is that the *classifying vector* is generated by concatenating the best fit for each *class model* (*i.e., matching model*) with the candidate to be classified, see figure 5.10.

- **(5) High level** The *classifying vector* is then compared with the *classifying model*. For example, using *PCA*, and calculating the Mahalanobis distances between each *classifying model* and the *candidate classifying vector*, generating each class probability.

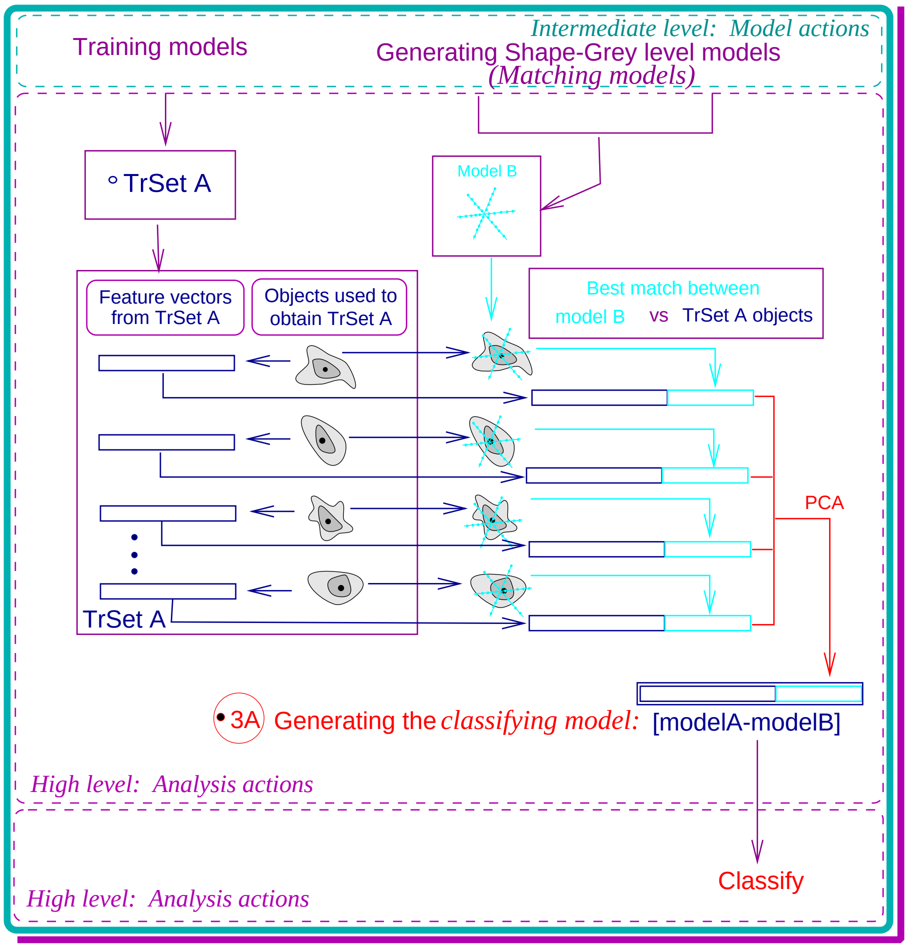

Figure 5.8: Generating a *classifying model*, an extended *classifying model* A is produced by concatenating the feature vectors (TrSet A), from the A training set of object examples, with the best possible match between the *matching model* B.

- **(6) High level** The candidate object is classified by directly comparing the resultant probabilities.

5.6.2 Object locator (*cue finder*)

Before the matching process starts, the object candidates within the image are located by performing some image processing procedures such as threshold computation, application of combined erosion and dilation operators and normalisation. This process was named *cue finder* as it gives the signals (object candidates) where the matching process has to act. Given the noisy nature of the application images, the object locator

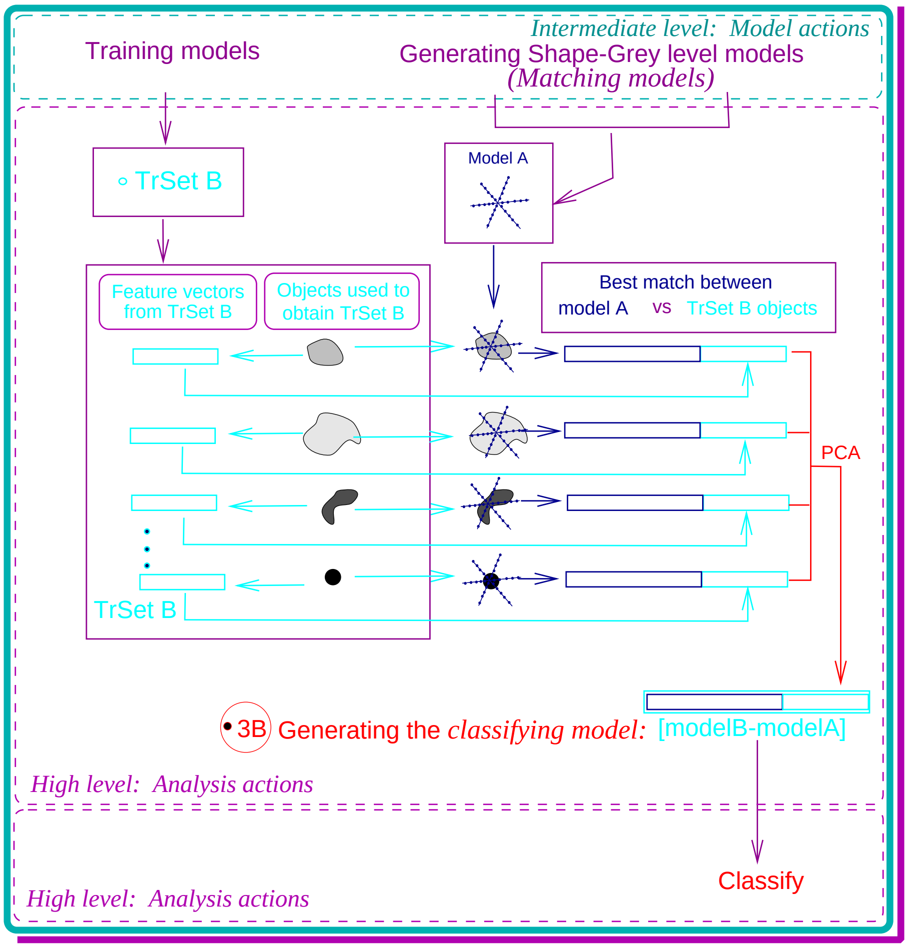

Figure 5.9: A *classifying model* should be generated for each model defined. An extended *classifying model B* is produced by concatenating the feature vectors (TrSet B) with the best possible match between the *matching model* A, in a similar way as described in 5.8.

approach combines several processes as follows (figure 5.12):

- Computing a threshold from a histogram (Nakagawa and Rosenfeld 1979). Several thresholding methods were tested by applying them in several images and observing if the objects of interest were located as object candidates. The tested methods[3] are integrated into the *SGmodel* system, in this way if the image conditions change, a different method can be chosen by the domain expert. Method **[1]** was found to give the best results and is the default method used by the

[3] The threshold calculation methods described here are those implemented in the Woolz image processing system developed at the MRC Human Genetics Unit, U.K. (Piper and Rutovitz 1985).

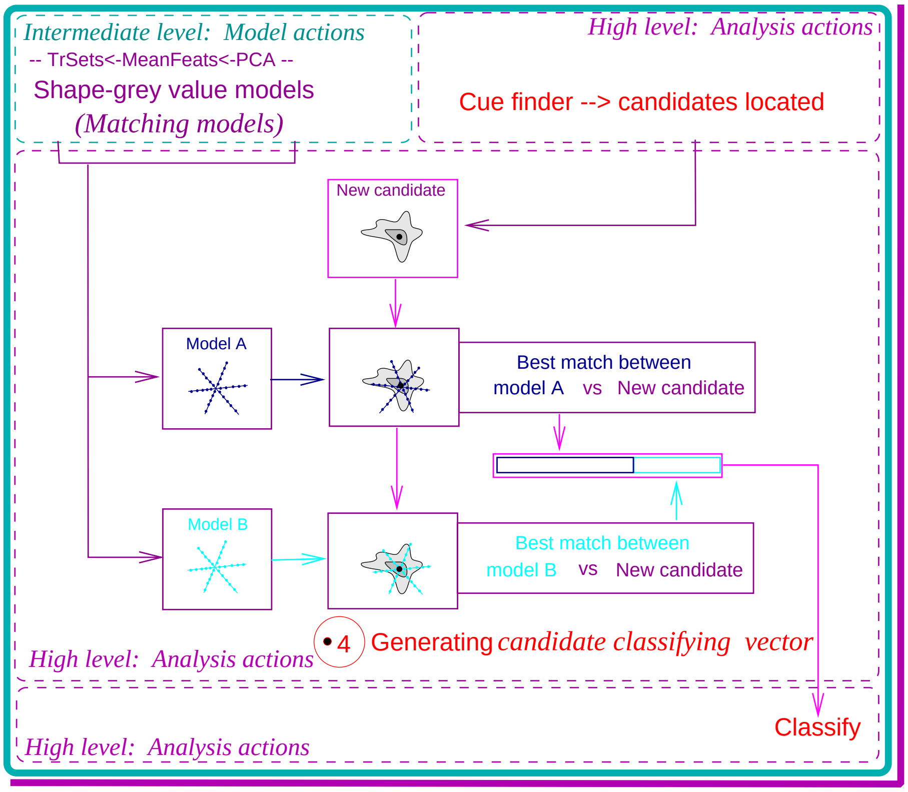

Figure 5.10: Generating the *candidate classifying vector*, by concatenating the best fit for each *class model* (*i.e.*, *matching model*) with the candidate to be classified.

system:

[1] In this method the threshold is defined as the x-intercept of a line fitted to the upper slope of the histogram main of the peak value. The line is fitted from the first point whose value is 90% or less of the peak value to the first point whose value is 33% or less of the peak value. Zero points in the histogram are ignored.

[2] The chord joining the histogram peak and the histogram right-hand end point is defined. Perpendicular to this chord, the furthest histogram point is the threshold.

[3] The slope of the histogram at each point above the histogram peak value is computed. The threshold is the first point at which this slope becomes zero or positive.

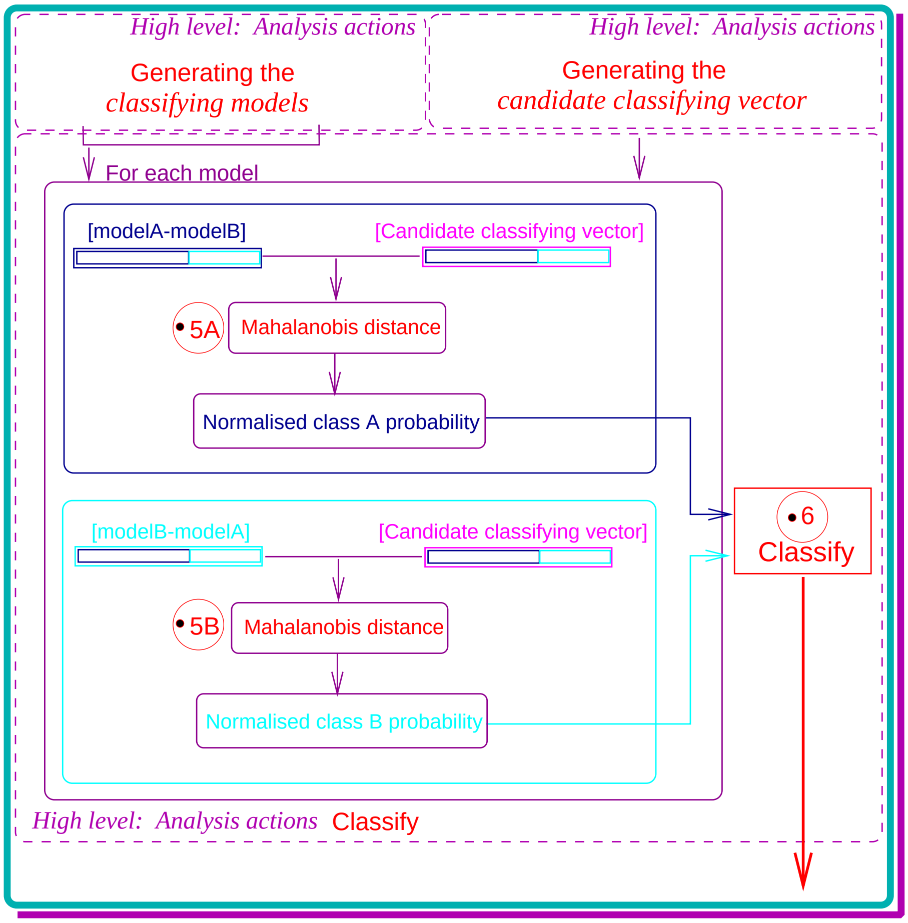

Figure 5.11: The classification is based on the class probability generated by directly comparing the *classifying vector* and the *classifying model* using the Mahalanobis distances.

[4] The threshold is at the first minimum above the histogram peak. A quantisation parameter is defined as 10% of the histogram mean value; a minimum must have maxima on either side which exceeds it by at least the value of this parameter.

[5] The entropy of the histogram is used to determine the threshold (for a more detailed description see (Kapur 1985)).

- To apply morphological operations such as erosion and dilation operators (Russ 1994; Sonka et al. 1993; Gonzales and Wood 1992), erosion removes pixels than should not be there, for example when two objects are slightly overlapped, the simplest kind of erosion is to remove any pixel touching another pixel that is

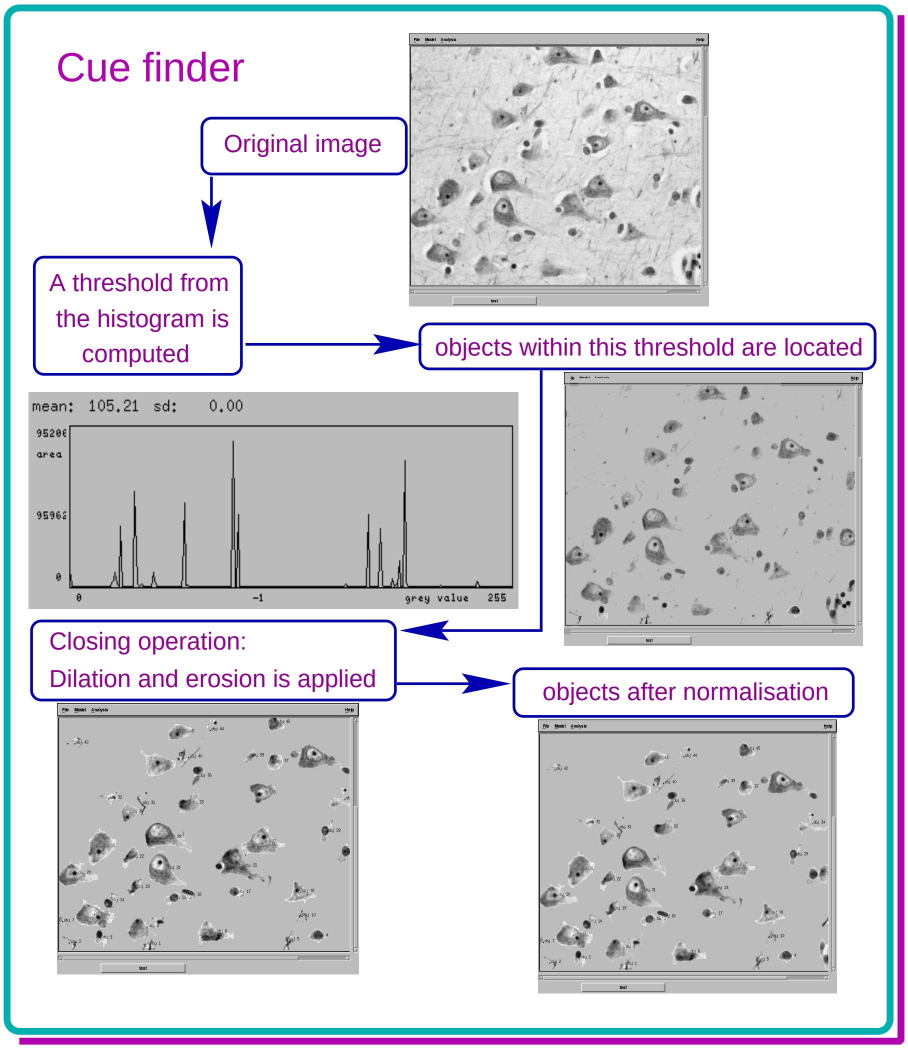

Figure 5.12: Traditional image processes during used to find the object candidates. In the *SGmodel* system, these are integrated in a module named *Cue finder*.

part of the background, similarly the simplest dilation set on pixels from the background that are touching the object of interest. Application of a closing operation (dilation followed by erosion) in order to fill small and thin gaps by connecting nearby objects. Additionally application of an opening operation, (erosion followed by dilation) in order to remove small regions by breaking objects at thin points. Both operations smooth the boundaries without changing significantly their area (Russ 1994).

- Finally, normalising the result from the above operations produces the region outlines of the object candidates.

In this approach, simple image segmentation techniques are used to obtain information such as the size and coordinates of the image objects, but during the matching process this information is used on the original non-segmented image in order to use background information as well (figure 5.13).

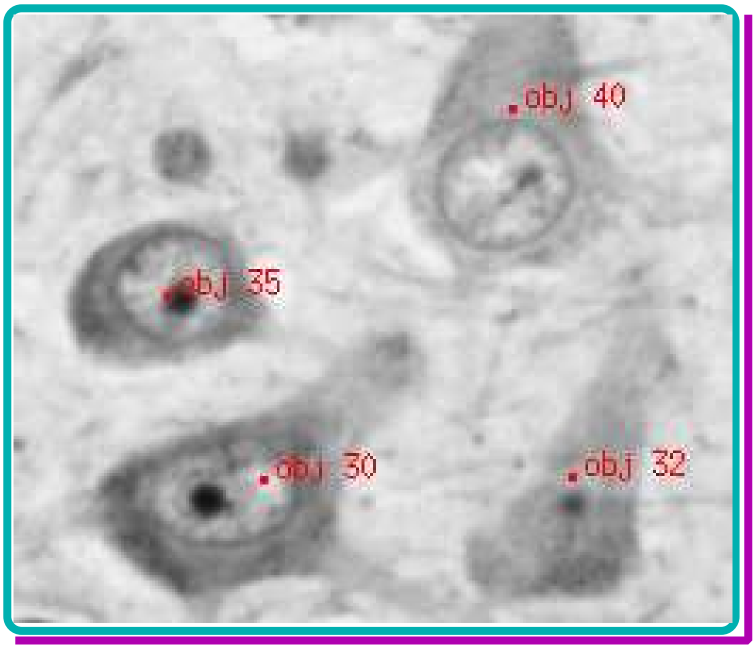

Figure 5.13: Example of region image with neuron candidates located.

The cue finding process is simply used to reduce search space. This influences the probability calculation which can be estimated as follows (using class A and B as examples of model classes):

$$Pr(C = A|I, cue) = \frac{Pr(A|cue, M_A)Pr(A|cue)}{Pr(A|cue, M_A)Pr(A|cue) + Pr(B|cue, M_B)Pr(B|cue)} \tag{5.4}$$

The values included in 5.4 are obtained as follows: the *cue finder* is used with specific set of images used to train a model (A, *e.g., neuron*). From that set the total number of class A members is known and the percentage of members found by the *cue finder* of classes A ($Pr(A|cue)$) and B ($Pr(B|cue)$) is then calculated.

Finally, it is important to emphasise that the final matching process is independent of the image features used by the *cue finder*.

5.6.3 Matching process

The matching process searches for the best possible fit between a model instance and an image object. The procedure to find this best possible match is as follows:

The model template is placed at different positions within the object region (see figure 5.14 and 5.15). At each position grey-level values are sampled, producing an *object instance*. The *goodness of fit* (determination of closeness using *MD*) between this *object instance* and the model is calculated. The position of the model radii and sampling points per radii are re-adjusted according to the main modes of variation calculated for the model. The grey-level values are sampled for the adjusted template producing a new *object instance* for which the *goodness of fit* is also calculated.

The iterative optimisation of *goodness of fit* was designed to use a Genetic Algorithm (*GA*, section 5.6.4) technique given the large search space for the best possible explanation. Each time that the model template is adjusted, the vector of the model weights and its probability density is recalculated. This process is repeated until the best explanation (best fit) for the object candidate is established.

5.6.4 Optimisation: A Genetic Algorithm guided search

The matching and classification processes described in this chapter include the search for the best possible match. For this search, an optimisation technique is applied. In this section a brief overview about optimisation algorithms is given as a framework for the Genetic Algorithm (*GA*) technique used in this work.

The optimisation process is then the maximisation or minimisation of an objective function of *goodness of fit*. There are several optimisation methods such as *calculus based methods* which use the gradient of the objective function to guide the direction for searching the maximum (hill climbing) (Sonka et al. 1993). The main problem of these methods is that they could easily end in a local maxima (minima), instead of the global one. In other words they are not robust, but they still could be useful in hybrid

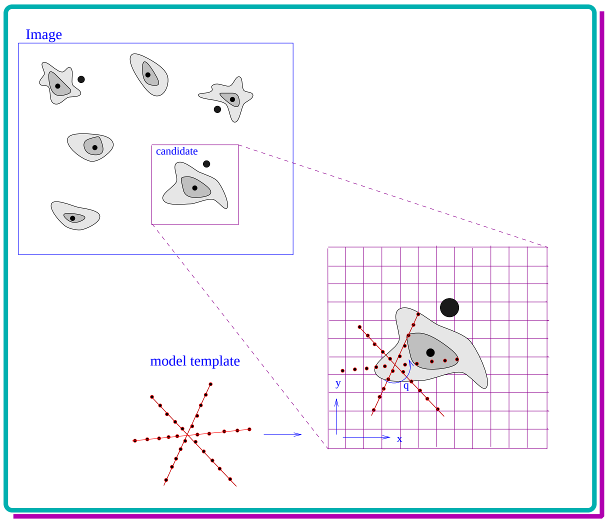

Figure 5.14: After the object candidates are found in an image, a search for the best possible match between the model template and each object candidate is performed within each object region.

strategies. There are some other methods which improve the probability of finding the global maxima by using some different strategies:

- By starting the hill-climbing at several points in the search space. If the search space is big this technique could become very inefficient.
- By applying searches by dynamic programming.
- By applying random searches, etc.

The *random searches* are nowadays among the most used strategies (Kirkpatrick et al. 1983). One of them, the Simulated Annealing (*SA*), uses random processes to guide its form of search for the minimum of an objective function (cost function). With *SA* it is not guaranteed that the global optimum will be found, but the solution is usually near-optimal. *GA* are another example of a directed search by random choice as a tool to guide the search through a coding of parameter space. The use of *GA* is recommended when the space to be search is not unimodal (*i.e.* more than one hill), not smooth and large or if the fitness function is noisy, (Mitchell 1996). The search method based on one solution per candidate at a time (as in the case of hill climbing) might result in an irrecoverable error in a noisy application. These characteristics are present in the search space of this application. A genetic algorithm method was chosen because of its robustness and also because of the public domain code availability[4], however other optimisation techniques can be also used .

GA based function optimisers have been proven to be useful over a wide range of difficult problems (De Jong 1992). Its mechanism for searching the maximum or minimum of an objective function is based on Natural Selection principles. In matching problems, it is common to find several feasible solutions that could be locally optimal and only one is best represented by the global maximum. *GAs* search for the optimum from many places from the search space, and from a population of solutions gives a better chance of finding the global optimum. A *GA* approach does not guarantee that the global maximum will be found, however it has been shown that normally gets very close to it, and is thus valuable for these tasks (Goldberg 1989). The search space in this recognition problem refers to a k number of images which have n number of predicted candidate regions. Each region has a j number of possible matches.

The search for new and better solutions depends on the values generated by the evaluation function which describes the *goodness of fit* (objective function), see figure 5.15. At each iteration, known as a generation, each candidate solution (individual) is evaluated and recombined with others on the basis of its overall fitness.

The following are the parameters to be used by *GA* in this approach and are coded as a finite-length string:

- x and y candidate coordinates;
- *x-size* and *y-size* of the candidate region to search;
- *theta*, the starting angle to place model template;
- *wstd*, the standard deviation of the model weight values.

By using simple genetic operators between candidates, the population evolves and the

[4] For the *GA* calculations the public domain PGAPack 1.0 code by D. Levine, and obtained from the ftp site ftp.mcs.anl.gov, was integrated into the *SGmodel* system, except for the evaluation function which was designed and developed specifically for this thesis application.

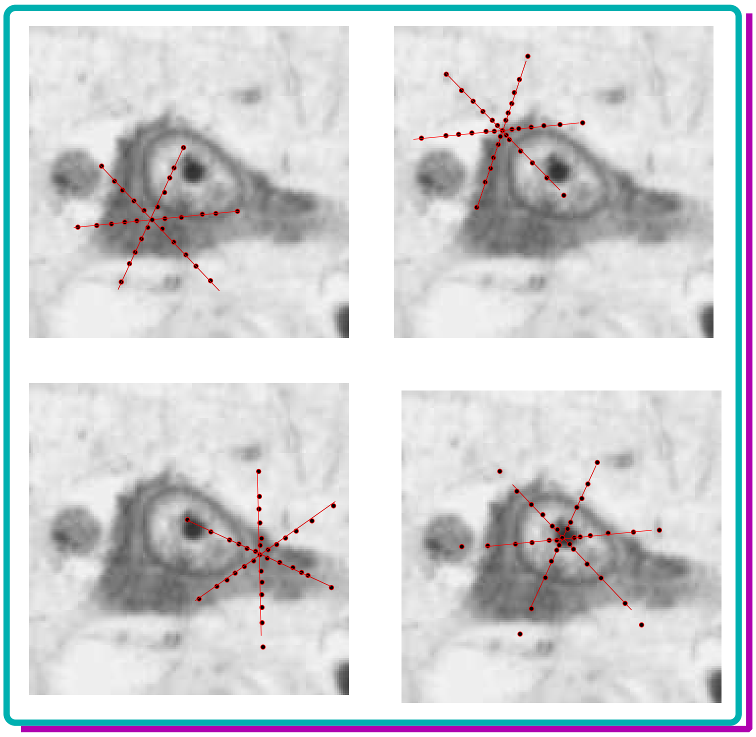

Figure 5.15: GA search, interactive evaluation of *goodness of fit* to find the best possible match.

best solutions with higher fitness (higher probability) are allowed to survive for the next generation of solutions. The solutions with poor fitness are excluded. The basic *GA* operators listed below are illustrated in figure 5.16 (Goldberg 1989; Mitchell 1996):

- Selection, which selects the fittest and removes the rest based on a probabilistic treatment.
- Crossover, in which parts of parent solutions are combined to create new ones.
- Mutation, in which part of the individual information is changed from time to time to prevent premature loss of possible solutions.

Evaluation function (cost function)

GA require a function that performs an evaluation of the *predicted candidate* and returns a fitness value. This is a measure, relative to the rest of the solutions population,

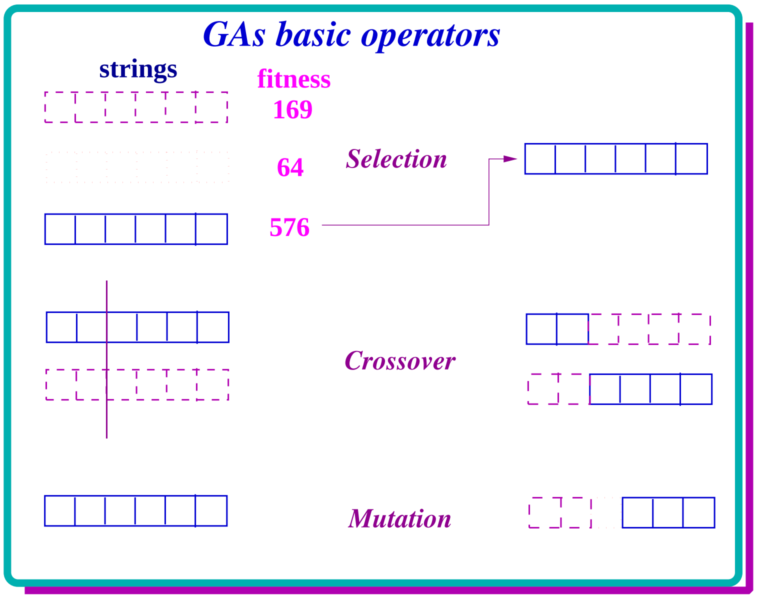

Figure 5.16: Schematic examples of GA basic operators

of how well the *predicted candidate* matches a class model (section 4.2). The matching process requires a set of values of the model variables to be specified, which have the highest probability with respect to the image data, as already mentioned in chapter 4. For this probability we make the same assumption as Haslam et al. (1994), that the density function can be modelled as a multivariate Gaussian, and use the Mahalanobis distances (to be minimised). In principle, the minimisation is applied to all the model space, but in practice limits must be set to provide the *GA* with ranges for each parameter. To achieve the latter, two standard deviations on either side of the mean values were used. The ranges of the position reset by the size of the object cue and the orientation is matched over 0-360 degrees.

The information used by *GA* to form the *predicted candidates*, as described before, are the coded parameters x and y candidate coordinates, *x-size* and *y-size* of the candidate region to search, *theta*, a starting angle to place the model template, and *wstd*, the standard deviation of the model weight values. With this information the ranges for the generation of the *predicted candidates* is set within the candidate region and the weight limits established by the training set (2 standard deviation). Each *predicted candidate* is derived using these ranges and the model representation (eq. 4.1). Then it is evaluated by the joint probability estimation (eq. 4.2) of the model modes of variation and the truncated modes of variation. The result is then used as the fitness value. Figure 5.17 illustrates this search process. *GA* use probabilistic transition rules from the current population of strings to a new and better population of strings based on the idea of supporting good strings with higher fitness and removing poor strings with lower fitness, thus allowing the best solutions to survive with a higher probability. This is achieved by applying the selection, crossover and mutation operators described before.

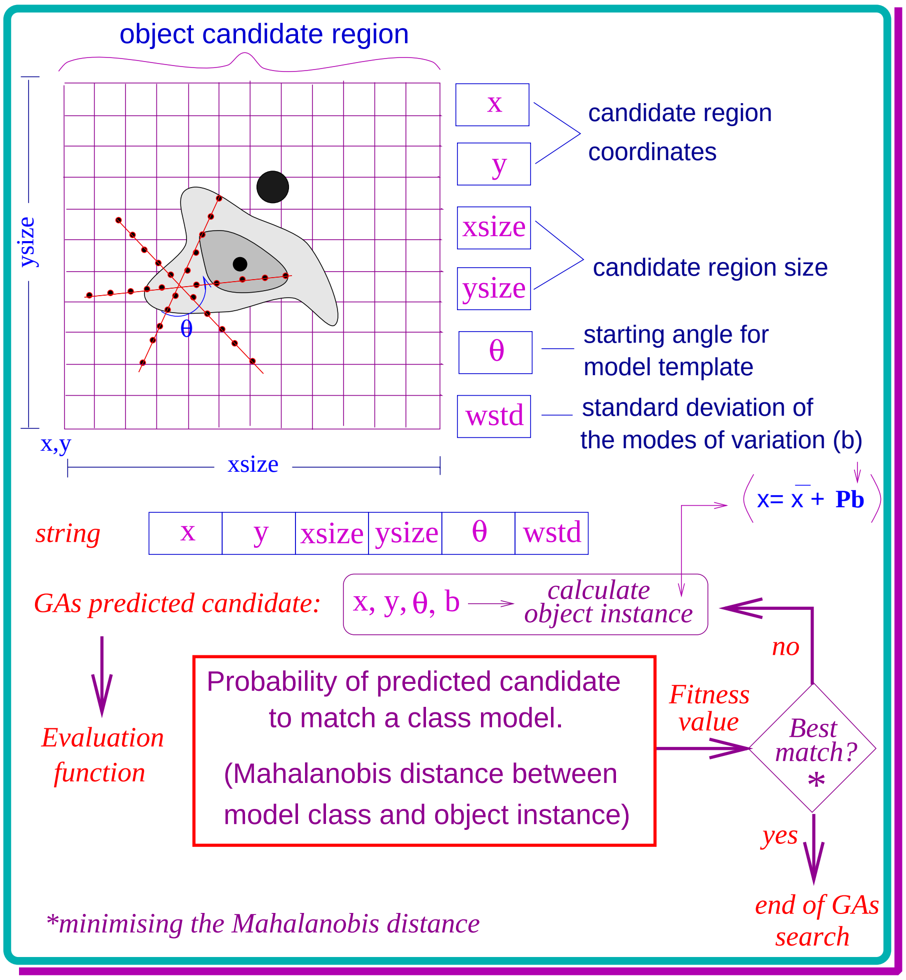

Figure 5.17: GA search within the candidate region, the *goodness of fit* is determined with respect of the minimised *MD* between model-candidate as described in chapter 4, eq. 4.7.

5.7 Conclusions

The implementation of *SGmodel* in a system gives the means for testing the approach proposed in this thesis. The *SGmodel* performance on synthetic data reached the objectives namely system testing and as an aid on the understanding of the *PCA* use for variability characterisation. The *SGmodel* use with real data provides a tool for solving Biomedical quantitative problems related with cell recognition, density, and distribution which was one of the motivations for this work. The *SGmodel* system represents in practical terms the main contributions of this thesis mentioned in chapter 4. The novel hybrid system implemented combines a wide range of theoretical and practical approaches such as traditional image analysis procedures (*i.e.*, threshold, dilation and erosion used for the *cue finder*), flexible model procedures (*PCA* based flexible model, *i.e.*, the extended *PDM*) within a probabilistic formulation, optimisation procedures (*i.e.*, *GA* for the optimised search of the best possible match) and a *GUI* in which the models are represented in a domain expert language which allows rapid prototyping of new object classes and tuning without the need for an image-processing and programming expert.

Chapter 6

Methods: from pre-processing to classification

6.1 Introduction

This chapter describes the methodologies used at different stages during the image analysis, from pre-processing to classification. These include the image acquisition process, including microscope settings and shading correction. The Biomedical material used to obtain the digitised images and the set of images chosen for the analysis are described. The model training process is presented and illustrated with an example of a neuron model. The procedures to obtain and visualise the modes of variation from a training set, and their use during the matching process are also discussed. A brief description of how the *PCA* information is used during the matching and classification procedures is presented.

6.2 Pre-processing

The material to be digitised, microscope sections of *CNS* tissue, was processed at the Neuropathology Laboratory, University of Edinburgh to have permanent preparations. The procedures to produce these permanent preparations of *CNS* tissue are: *fixation, embedding, sectioning (microtomy), staining, and mounting*, (Sumner 1969).

- *Fixation* preserves the tissue by chemical means. The tissue is normally fixed by immersion in a fixative (*i.e.*, formalin) or a solution of two or more substances.

- *In embedding* the tissue is infiltrated and surrounded with a hard substance which reduces the tissue distorting when is sectioned.
- *In sectioning* the tissue is cut into very thin slices, or sections, using a microtome.
- *Staining* involves colouring the tissue with dyes to show up different cellular components as a result of different chemical affinities.
- *Mounting* involves putting the stained slide of tissue in put on a glass slide and a covership is then glued on top.

Brains were obtained at autopsy from AIDS patients, and standard procedures for preserving and staining the tissue were used. These procedures include: brain tissue fixation in formalin for 2-3 weeks. Selected blocks of tissue were embedded into paraffin wax, cut at 3, 5 and 7 μm. The tissue was stained with cresyl violet, a basic dye that colours acidic tissue constituents such as nucleolus which makes neuronal recognition easier.

6.3 Image acquisition

The biological material (brain sections) was placed under an optical microscope, with a digital camera attached. A Zeiss Axioplan microscope with a Xillix Microimager 1400 camera, connected to a Sun Sparc 10 workstation, were used in this process (figure 6.1). Image acquisition is the process by which the analogue optical images from the microscope are converted to a digital image representation. This is performed by the camera which is a rectangular array of light sensitive cells (change-coupled device CCD). The digital representation is termed a grey-level image (Vernon 1991) which has light intensity sampled points (pixels) in a predetermined equally spaced grid. The digitisation environment is an important factor in the success of the image processing. An adequate image contrast could determine an accurate grey value characterisation and the appropriate segmentation during the process of locating object candidates. The contrast will be determined by the stain, microscope illumination and the use of complementary colour filters. The output obtained in this module are the digitised images to twelve bits of grey scale information (values - 4095) which are mapped to eight bits (0 - 255) via a hardware look up table.

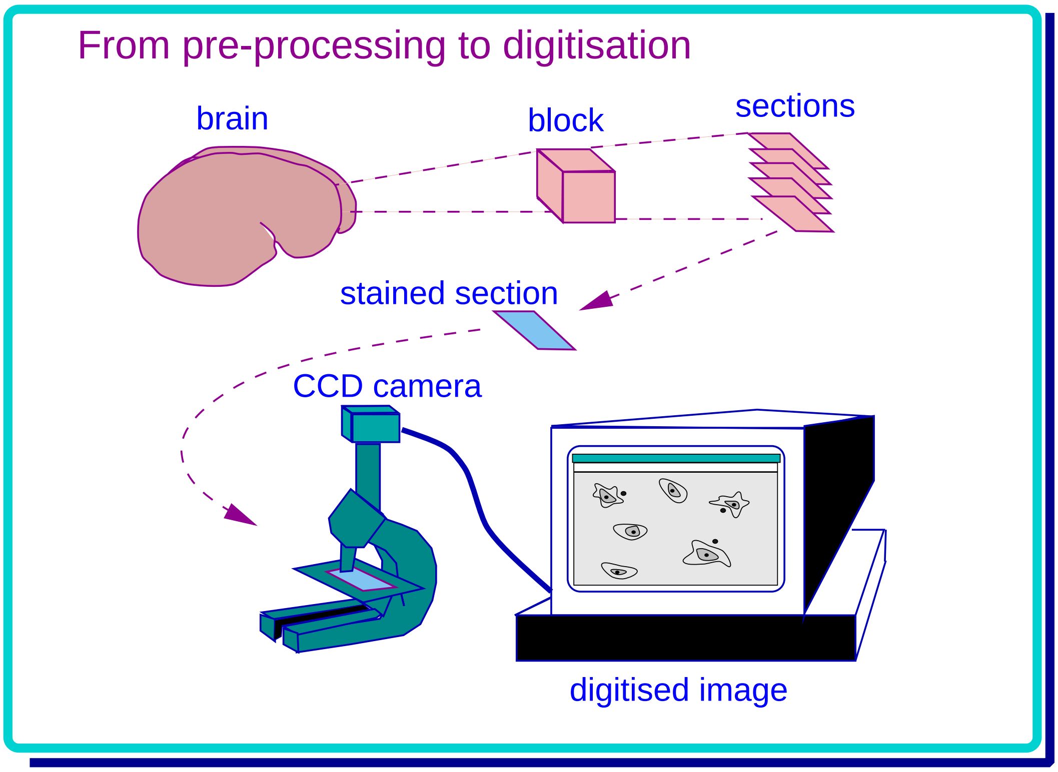

Figure 6.1: Selected blocks from the fixed brains were embedded into paraffin wax, sectioned, stained and digitised from a microscope with a CCD camera and digitiser.

The digitised images were randomly selected on each of the 45 brain tissue slides used, 25 images per slide were digitised. Figure 6.2 show two examples of brain tissue slides, the dots in the tissue represent the locations where the images were digitised from.

6.3.1 Microscope settings

The criterion for choosing the microscope magnification (25x), was based on the need of processing the greatest number of objects per image while keeping the object features clear enough for recognition and classification. Several filters were tested in the illumination source and assessed by checking the overall image contrast. It was found that the standard green filter of the Zeiss microscope produced good contrast and was optimal for the performance of the candidates object locator used in the matching

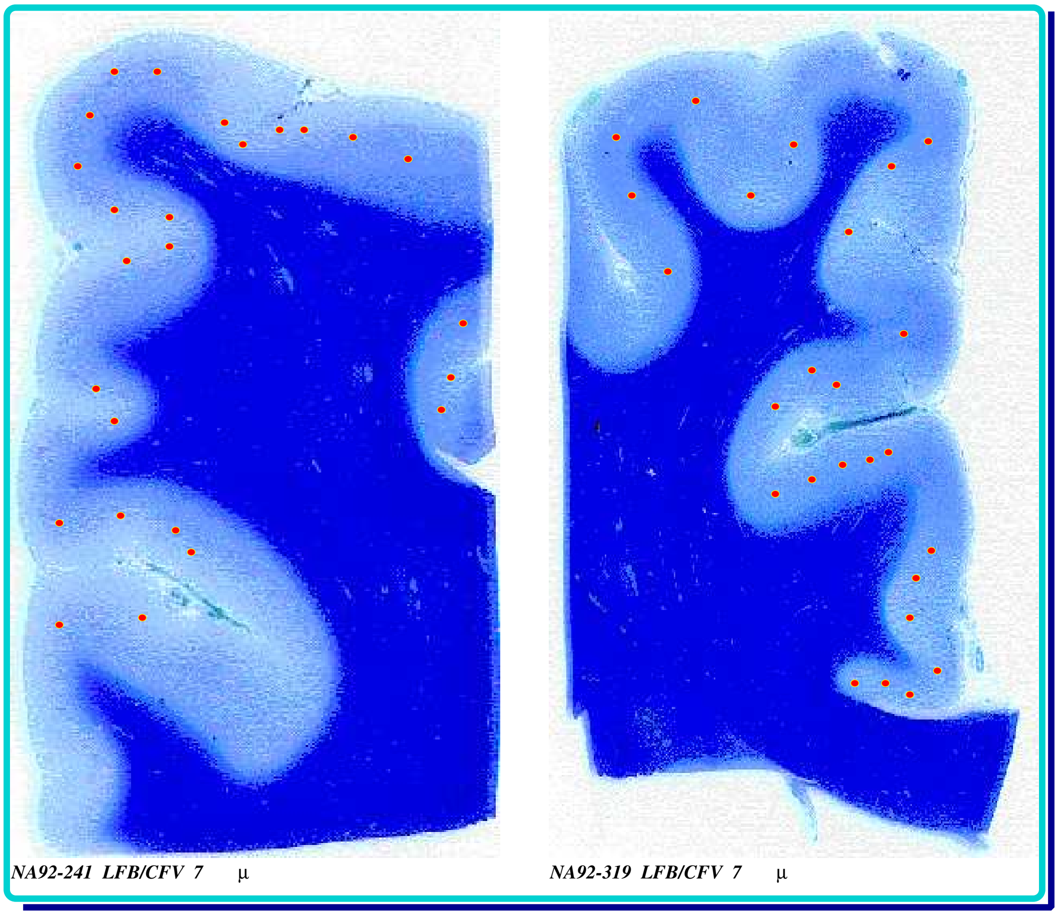

Figure 6.2: Examples of brain tissue sections from which images were randomly digitised, the dots represent digitisation points.

module. The camera used is very light sensitive therefore it was also necessary to use a dark neutral density filter in the illumination path. For each image acquisition session, the microscope optics should be adjusted to Köhler illumination[1] to obtain a good optical resolution, reduce glare and to reduce the variations of illumination intensity on the image field (*shading*) to ensure consistency, (Brian and Marchevsky 1994).

6.3.2 Correcting nonuniform illumination - shading

In many image analysis tasks, the brightness of regions is used as means of identification, counting and measurement. In this approach, we assume that the same type of

[1] A system of illumination for a light optical microscope that leads to highly uniform illumination of the field of view (Bacus and Grace 1987).

feature will have similar grey values wherever it appears in the field of view, and then these features can be used for identification. For this reason it is important to ensure uniform illumination during the image acquisition process. In principle the illumination will always vary over the field even with correct microscope set up, furthermore the camera light sensitive cells will all have slightly different responses to the same light intensity. To correct this variation it is necessary to know how a given range of light intensities, maps onto grey values for each CCD in the array. If the calibration function is known for each cell then this range can be applied onto standardised grey values (*e.g.* optical density). In this work we assume linear response to intensities, the shading correction should be performed each time that the microscope settings are adjusted or a slide is changed. Baldock and Poole (1993) describe how this correction can be achieved by the following steps:

- adjusting the microscope settings for the current slide;
- focusing the objective and condenser in a clear field of the slide then digitising several frames at different slide positions;
- computing a bright-field (I_B) image is computed by selecting the brightest value of these captured frames;
- finally a dark-field image (I_D) is captured by switching off or covering the microscope light source;
- the shade corrected image is calculated using

$$I' = \frac{(I - I_D)}{(I_B - I_D)} 255, \tag{6.1}$$

which is a number proportional to the transmission coefficient of the material. Multiplication by 255 renormalises the values to an 8 - bit range.

Figure 6.8 includes two examples of the final digitised microscope images from brain tissue.

6.4 Model definition

In chapter 5 the model radial templates were described. The parameters for each model, *i.e.*, the number of radii, the number of points per radius to be sampled, and the angles between radii, have to be determined at the beginning of the training process. For this process the information taken in account is the specific characteristics of the object and the prior knowledge from the expert user. For example an object neuron is recognised by the presence of a well defined dark circular object (known as the *nucleolus*) which normally has a centred position within the cell (see figure 6.3).

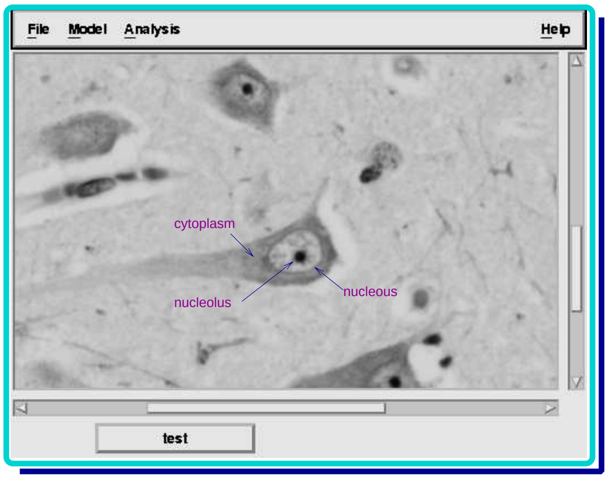

Figure 6.3: Image displayed in *SGmodel* user interface, the main parts of a *neuron* object are indicated. The centred dark structure is known as the *nucleolus* situated inside of an structure known as *nucleus*, and the latter is surrounded by the *cytoplasm*.

The model *neuron* was defined as follows:

- a sampling point in the *nucleolus* centre;
- six radii with an optional initial direction;
- fixed angles (of 60 degrees); and
- 6 points per radius to perform the grey level sampling, the criteria for the point positions are described in next section 6.5.

A model *NOneuron* was also defined with the same features, to model all the other image objects found by the cue finder. These objects lack constant identifiable structures therefore the central sampling point was put roughly in the centre of the object and the 6 sampling points per radius were set with fixed distances.

6.5 Model training

The model training process consists of sampling grey-level values through object examples in a set of digital images, using the defined model template. The training set obtained with this process is then used to characterise the object class shape and its grey-level values distribution, see section 6.6.

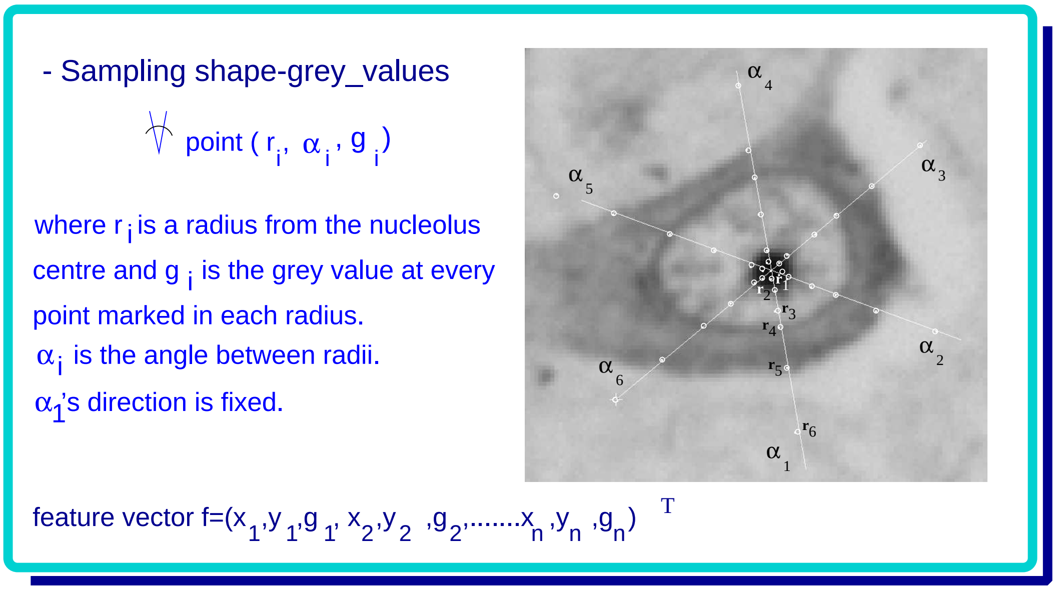

Figure 6.4: Sampling grey level values in an object neuron.

The grey-level sampling is carried out as follows, (see figure 6.4).

- Each image is displayed in the main window of the *SGmodel* user interface, as shown in figure 6.3 (see appendix B for the system description).
- Then, using the domain expert knowledge, the objects of interest are located.
- For each object the points sampled are chosen manually starting with the *nucleolus* centre point.

- The initial direction α_1 of the first radius is chosen following some constant criterion for all the objects in the training set, *e.g.*, the direction to the nearest edge.

- At each angle (from α_1 to α_6) the six points are sampled using the following criteria:

  r_1 at a fixed distance from the centre point within the nucleolus;

  r_2 at the nucleolus edge;

  r_3 between the r_2 and the nucleus edge;

  r_4 at the nucleus edge;

  r_5 at the cell body edge; and

  r_6 at a constant distance from r_5.

In figure 6.4, the information (r_i, α_i, g_i) related to each point sampled on the object neuron, is defined as:

- r_i, the i^{th} radius from the nucleolus centre;

- α_i, the i^{th} angle between radii (α_1 is fixed); and

- g_i, the i^{th} grey value at that position.

The feature vector has the form $\mathbf{f} = (r_1, g_1, r_2, g_2, ...r_n, g_n)^T$, where $r_i = (x_i, y_i)$ (radii coordinates from where the grey values are sampled in the image). An example of the information stored for each object in the training set is shown in table 6.5.

Training set for model:	neuron6					
num_radii:	6					
radii:	variable					
num_angles:	6					
angles:	60	120	180	240	300	360
num_grey:	1					
greys:	variable					
numData:	442					
filename: neu3m_001.gm						
centreXY:	479.833	0.1666				
angle:	-1.19599					
grey type:	3					
begin features:						
	249	1.95	250.17	4.86	229.56	7.06
	192.67	10.09	190.06	30.98	139.78	46.92
	56	2.03	248.64	4.88	209.17	10.61
	101.56	15.67	150.67	25.74	113.22	41.79
	118.17	1.9	244.33	4.48	194.33	11.61
	81.81	18.04	143	23.64	110.28	39.62
	70	1.95	243.33	5.08	176.95	11.58
	89.94	16.06	141	20.68	96.33	36.74
	75.56	2.03	245.83	5.37	173.08	14.33
	110.56	22.94	134.22	33.38	144.22	49.38
	49	1.9	247.78	5.19	199.11	9.14
	167.17	17.08	205.42	37.9	142.97	53.85
	63					
end features						

Figure 6.5: Information stored per each object sampled

The model has to be trained in a way that captures all the possible variations of an object class. Some examples of object neuron sampled for a training set are shown in figure 6.6.

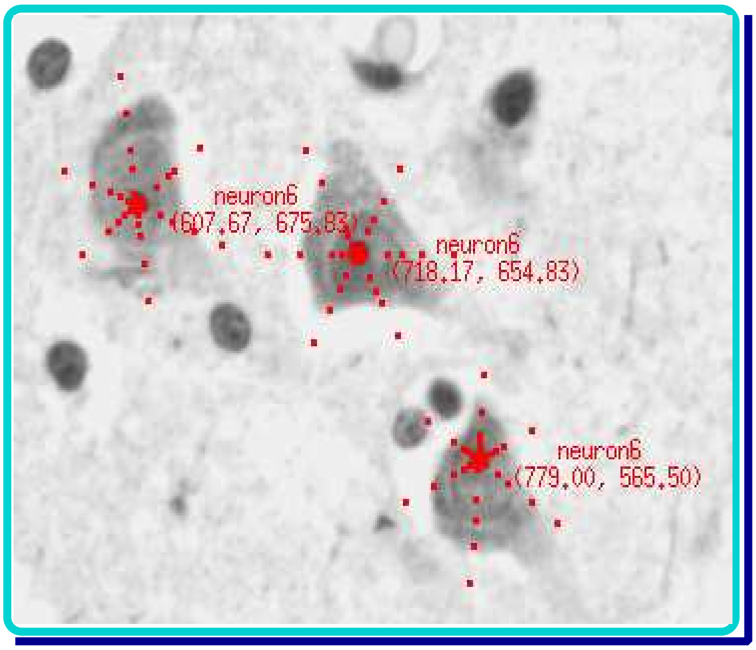

Figure 6.6: Examples of object neuron with sampled points used for a training set, neuron6 stands for a model neuron with 6 radii.

6.6 Variability characterisation

The variability of the shape-grey values is characterised by applying PCA to the feature vectors from the training set as described in section 5.5.3. The results from applying PCA are stored in a data file, and include the following information:

- number of features;
- number of object examples used;
- total number of variation modes;
- model modes;
- mean of the feature vector;
- eigenvalues and eigenvectors, calculated from the training set mean vector and covariance matrix, as explained in 3.2.8 and 5.5.3;
- standard deviation of shape-grey value parameters; and
- number and names of the images used.

This information is used to generate new model instances and obtain training set distributions which are used during the matching process.

6.7 Matching and classification

As described in sections 5.6.3 and 5.6.1, the matching procedure between class models and class members is used to generate a *classifying model*. This model is then compared with a *classifying vector* produced by using the same matching procedure as well, but in this case between class models and object candidates to be classified. A normalised class probability is the result of these processes. The following is the sequence of actions involved in classifying new image objects.

- For each class read training set from data files (see section 5.4.2).

- Create the *matching models* by applying PCA to each class training sets (see section 5.5.3).

- Generate a *classifying model* by concatenating the best fit of each class model with each training member, see sections 5.6.3 and 5.6.1. Then apply PCA to the set of extended vectors.

- For each candidate to be classified in a new image, produce a *classifying vector*, as described in section 5.6.1, this process is illustrated on the left hand side of figure 6.7.

- The normalised class probability is obtained by comparing each *classifying vector* with the *classifying model* using the matching procedure (section 5.6.3).

Figure 6.7 illustrates this process using an example with two model classes (neuron and NOneuron). In this figure the models are represented as radial templates, sections 5.5.1 and 6.5 describe the design of these model templates.

6.8 Conclusions

The general methodologies and settings in each step from pre-processing to classification have been presented in this chapter. These settings have been chosen to obtain an optimal performance of the *SGmodel* system for the application domain of this thesis. However different tuning actions can be performed if the application domain or the conditions change at any level. Tuning can be applied in the following procedures.

- Digitisation setting (*e.g.*, light, filters, etc.) for good contrast and enhancement of objects and structures of interest.

- The model definition (*e.g.*, number of radii, etc.). These parameters should be tuned to characterise the class shape-grey values distribution.

- Number of examples used to train the model. The system performance can be affected by the size of the training sets. If a training set is not big enough(at least 500 examples) to capture all the class variability then some objects in new

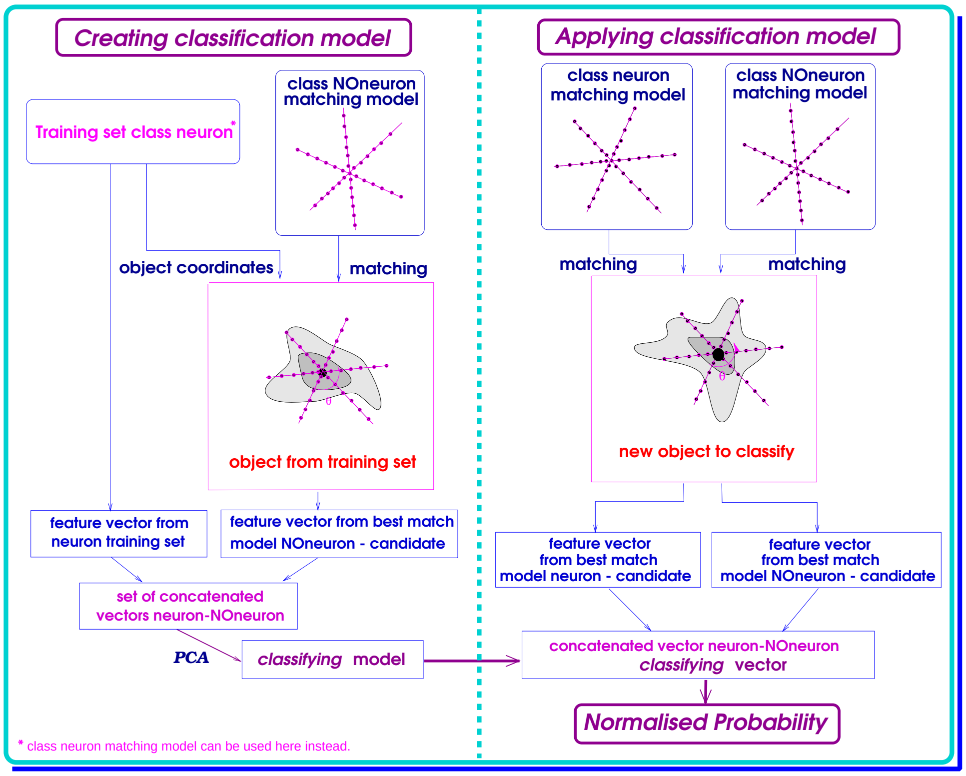

Figure 6.7: The classification process, example with two classes (neuron and NOneuron). In the training set, the feature vectors for class one (neuron) are concatenated with the feature vector of the best possible match between the class two model (NOneuron) and the image object (the models are represented as radial templates). The set of extended feature vectors from the training set is used to establish the *classifying model* which is compared with the extended *classifying vector* of a new object to calculate a normalised class probability.

images could be misclassified if their shape-grey values are not included within the training set distribution (*i.e.* producing more false-negative results).

- The thresholding method chosen. This can be modified in the *Analysis actions* for an optimum cue finding performance, (see description of the thresholding methods available in 5.6.2).

Several procedures presented here are time demanding, for the digitisation of the images once the microscope is set, it takes in average 20 minutes per slide (25 images), the model tuning and training would depend on the size of the training set selected, in average 120 examples were trained per day for a *neuron* training set with 1400 examples.

Most of these procedures are are only performed once, and thereafter the trained models can be applied directly to new sets of images.

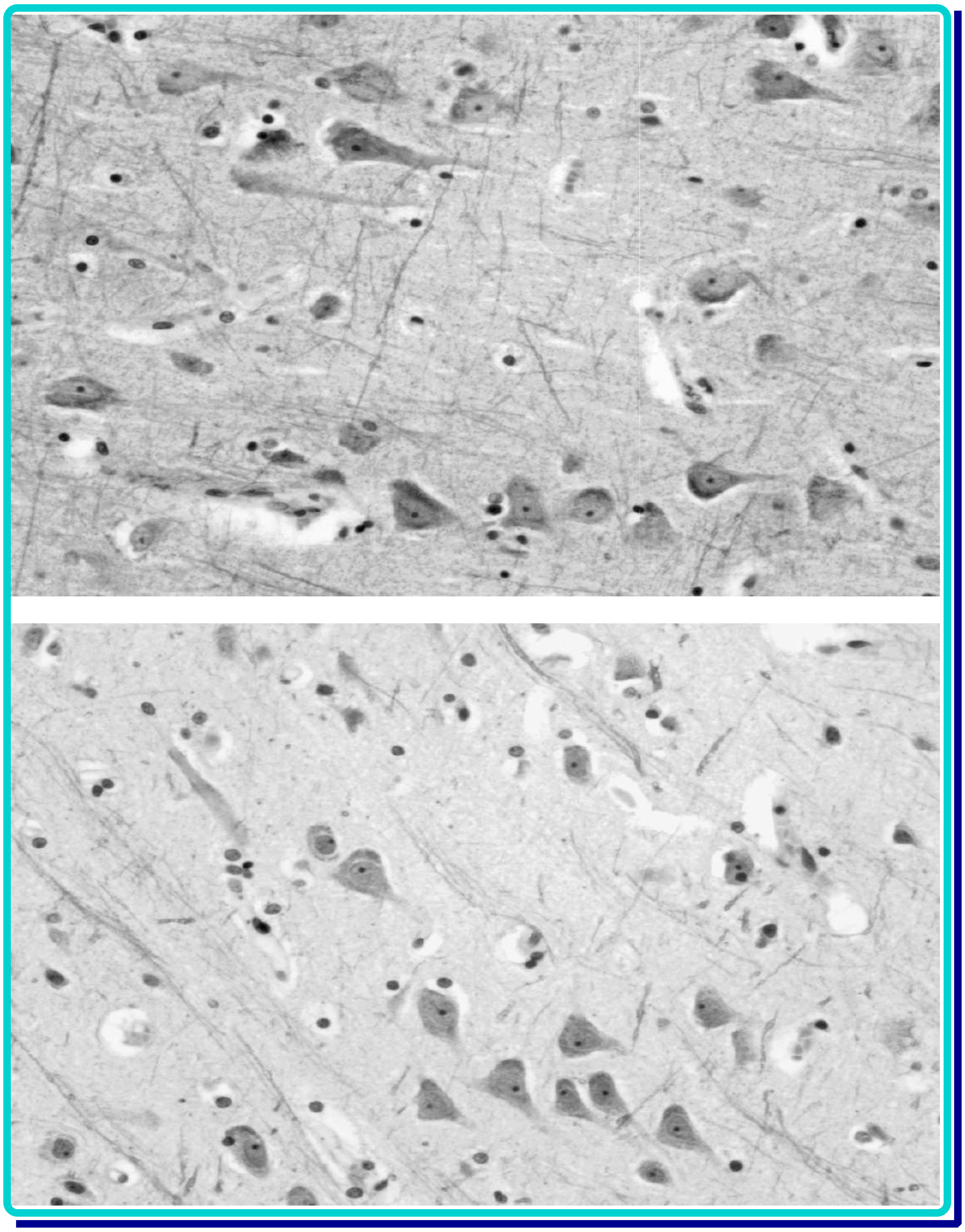

Figure 6.8: Two examples of the microscope images from brain tissue.

Chapter 7

Tests and Results

7.1 Introduction

This chapter is focused on the results obtained from the *SGmodel* system performance using synthetic and real data. The synthetic data were produced by an artificially generated cell model (section 5.5.5). The results of these observations illustrate how *SGmodel* characterises the shape-grey level variability from the object examples used to train the model. The performance of the *SGmodel* system with real data was tested by training models for *neuron* and *NOneuron* objects. The training and testing was performed in randomly sampled images of brain tissue from the five groups involved in the *HIV-CNS* project described in chapter 2. For the *SGmodel* system validation, the comparison between estimations obtained with the system and those from the domain experts' manual counts are presented. For this thesis application, analysis at the level of cell distribution estimates is required, and therefore analysis of the results is focused at this level, however the *SGmodel* performance at a case-by-case level is also discussed.

7.2 Artificial cell model

The generation of an artificial cell model and its set of examples was described in section 5.5.5. A training set of feature vectors was generated by random selection of each model parameter (radii and grey-values) and assumed to have a uniform distribution over a fixed range. *PCA* was applied to this set of artificial cells, each of which has a feature space of 109 dimensions. The variability of shape-grey values in the artificial cell is accurately captured by the first 6 modes of variation which correspond to the 7 parameters defined in the model (parameter 3 and 4 are correlated) which demonstrates the power of the technique for this type of application. In figure 7.1 the modes of variation behaviour is illustrated in relation to these controlled cell parameters (shown between brackets).

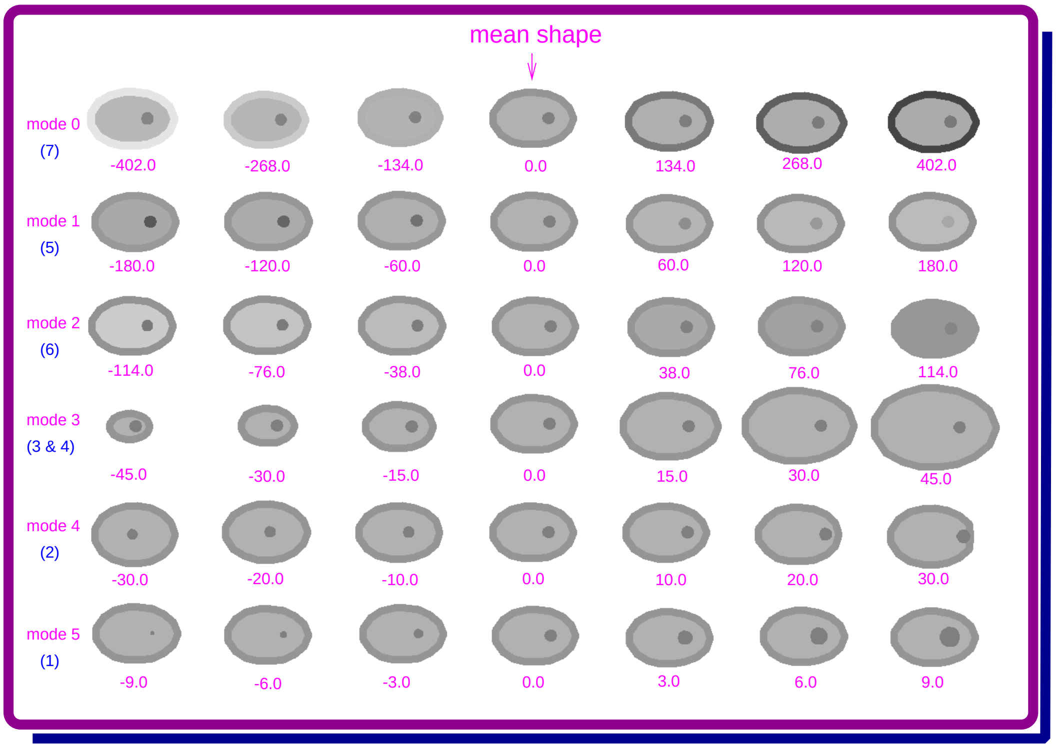

Figure 7.1: Modes of variation in an artificial cell

To observe the behaviour of each individual mode of variation, the starting point in figure 7.1 is the middle column which represents the mean shape. The values of each mode are modified while the rest are kept constant at the mean value. The results observed were as follows.

- When the value of mode 0 is modified, variation on the grey value of the ellipse boundary is observed, thus the variation captured by mode 0 corresponds to parameter (7) defined in the artificial cell model (section 5.5.5).
- Mode 1 is related with the variation in the grey value of the central circle and corresponds to the cell model parameter (5).

- The variation in the grey value of the ellipse is related to mode 2 and corresponds to parameter (6).
- Mode 3 corresponds to the combined variation of parameters (3) and (4). These parameters define the horizontal and vertical radius of the ellipse and are correlated.
- The distance between centres of the small circle and the ellipse is defined by the model parameter (2) and its variation is captured by mode 4.
- Finally, mode 5 is related with the variation of the radius of the small circle, *i.e.*, model parameter (1).

This example illustrates that the 109 dimensionality can be represented with a relatively high accuracy by projection onto the first 6 modes of variation (figure 7.1). However if only only the first principal components are used the error introduced by the dimensionality reduction has to be taken in account. In this case the model parameters are recovered exactly by the PCA which can not be expected for real cells, however the artificial cell shows how the underlying simplicity can be recovered. With this, the artificial cell model has proven useful in testing the *SGmodel* performance, and in the understanding of how the object variability is characterised by PCA on a set of examples.

7.3 Application domain: Model *neuron*

For testing the system with the application domain, *neuron* counts were estimated on images from the groups mentioned in chapter 2:

- [AE] - $AIDS$ Encephalitis drug users;
- [ANE] - $AIDS$ Non-Encephalitis drug users;
- [PA] - HIV^+ Pre-$AIDS$ drug users;
- [DU] - HIV^- drug users; and
- [C] - Control HIV^-, non-drug users.

7.3.1 Training models *neuron* and *NOneuron*

The set of images to be analysed should be typical for the application under study in the sense that they should encompass the class variability. To analyse these five groups 236 randomly digitised images were used to train the model *neuron* with 1400 examples of object neuron, and 215 images to train the model *NOneuron* also with 1400 examples of *NOneuron* objects. The criteria to choose the *neuron* examples was based on the prior knowledge from the expert user, *i.e.*, each object neuron is recognised by the presence of a well defined dark circular object (known as *nucleolus*) which normally has a centred position within the cell (see figure 6.3 in chapter 6). Then the *cue finder* was used over the set of images where the *neuron* examples were sampled, and those objects found by the *cue finder*, but not classed as *neurons* were chosen as *NOneuron* examples.

PCA was applied to both *neuron* and *NOneuron* training sets the eigenvalues obtained are shown in graphs 7.2 and 7.3 respectively, in these graphs the 73 values observed correspond to the total number of the model features. Most of the object variation for both models can be represented with the first 20 modes. For the model *neuron*, the first 20 eigenvalues represent 92.42% of the total variability in the examples of the training set. In the case of the model *NOneuron*, the first 20 eigenvalues represent 98.33% of the total training set variation. In this work, all the eigenvalues were used for classification. The advantage of the dimensionality reduction is only used during the matching process in which only the modes that represent 90% of the total variability are used to generate the *matching model*.

The variability in these training sets was characterised by applying *PCA* to the feature vectors. The information obtained is used to generate new examples by modifying the variation modes. The variation for the new examples is then limited by the variation found within the training set. The generation of new examples is used during the matching and classification processes as described in chapter 5. Figure 7.4 shows several examples generated by modifying the first 4 modes of variation. Some of the modes represent the variability of more than one feature, for example mode 0 in figure 7.4 is related with the position and size of the dark circle *nucleolus* but also to the grey-values

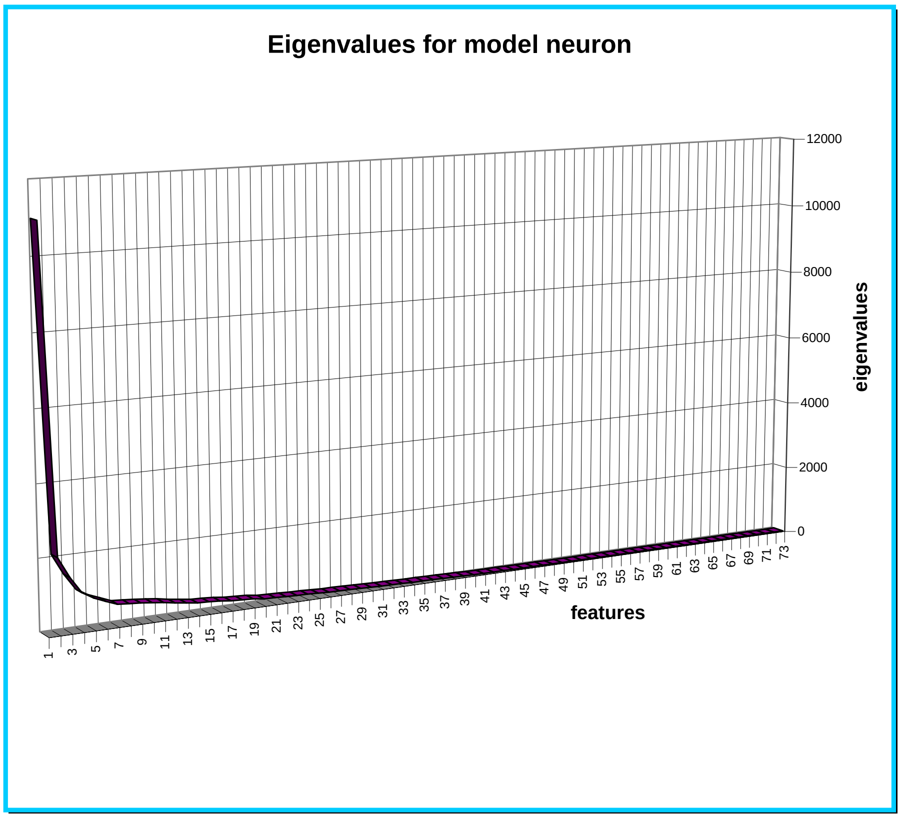

Figure 7.2: The eigenvalues obtained by applying *PCA* to the 1400 feature vectors on the *neuron* training set. Most of the total class variability is captured with the first 25 eigenvalues.

of the *nucleolus*.

The search for the best fit needed during the matching and classification processes is guided by an iterative optimisation process based on a *GA* (description in section 5.6.4). Figure 7.5 shows an example of the probability landscape produced by the *GA* in the search for the best match. The displayed values are the match Mahalanobis distances in the object candidate region, calculated during the search process and plotted in terms of the two parameters x and y. If more than one value are at the same position, the lower one has been displayed.

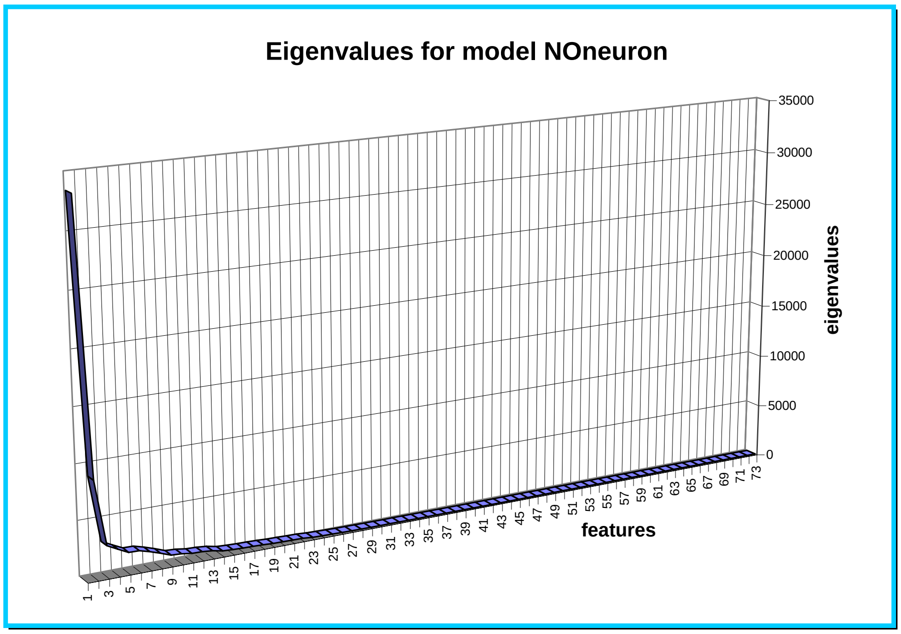

Figure 7.3: The eigenvalues for the model *NOneuron* were also obtained by applying *PCA* to the 1400 examples in the training set. Most of the total class variability is captured with the first 20 eigenvalues

7.3.2 *SGmodel* neuron estimates

A test set of 270 images was used to obtain the estimate of neuron counts for the *C, DU, PA, ANE* and *AE* groups. The images in this testing set were different from those used for training the models.

To obtain both training and testing sets, 45 brain tissue slides were used from the five groups of interest. The neuron estimates were calculated over 6 images per slide, the average per slide of these estimates is shown in figure 7.6. To calculate the final neuron probability it is necessary to know the values of the $Pr(neuron|cue)$ and the probability that a neuron will not be detected by the cue finder (false negative rate).

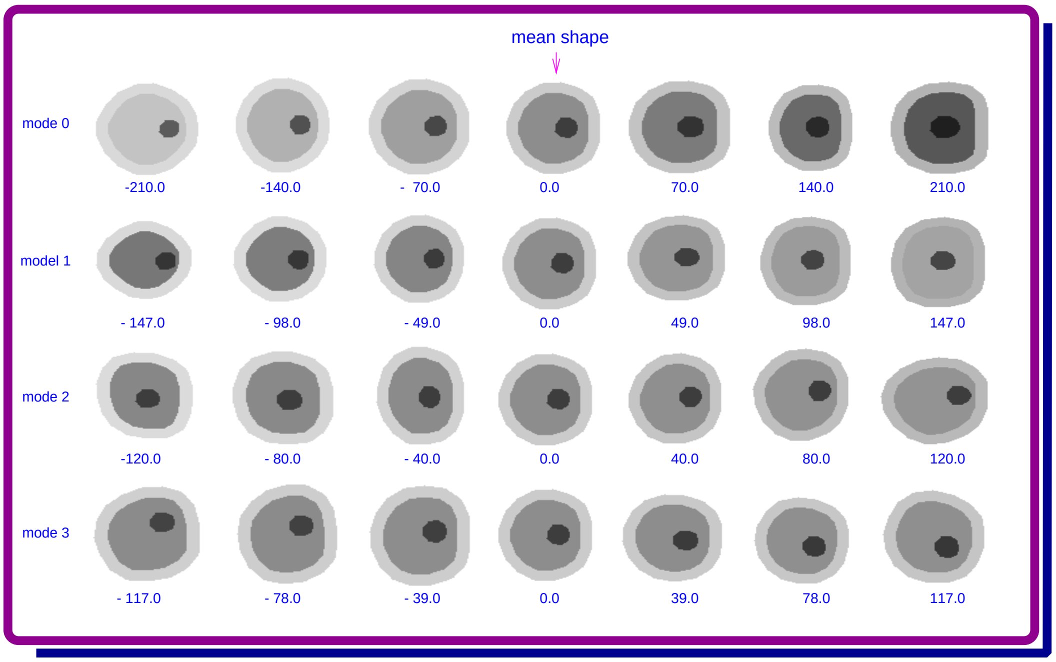

Figure 7.4: New examples of nucleus and nucleolus of the neuron, generated by modifying the first 4 modes of variation.

These were estimated by applying the cue finder to the training set images for the neurons with the result that $Pr(neuron|cue) = 0.1515$ and the false negative rate is 5.02%. These values are used in eq. 5.4 to calculate the final probabilities.

Figure 7.6 includes three tables. The first on the left hand side includes: the name of the image, the slide identifier, the group from which it was obtained and the average neuron counts per slide. The second table on the right hand side includes the total estimates distributed by group and includes the average neuron estimates per image per group which is also shown in figure 7.7. The third table, on the bottom of the right hand side in figure 7.6, shows the results of a statistical significance test known as the *T-test*. Taking in account that the numbers of images involved per group were different, the *T-test* was applied to evaluate the significance of mean differences between neuron distribution estimates from the control group *C* and the other groups *i.e., C-DU, C-PA, C-ANE* and *C-AE*.

The significance value obtained by the *T-test* is a number between zero and one. The common convention is to consider any value smaller than 0.05 as "*significant*". This convention is arbitrary and the use of extra criteria is also accepted for values near the

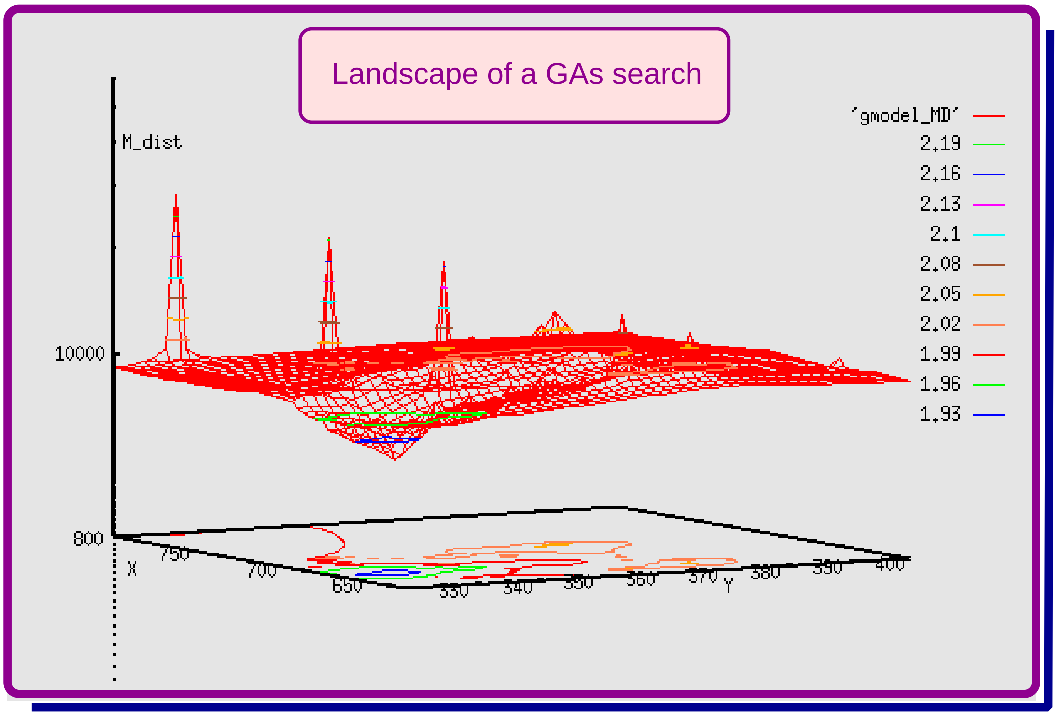

Figure 7.5: A space landscape example searched by the GA during the matching process.

significance level of 0.050 (*i.e.*, 0.048, 0.049, 0.051, 0.052, etc.) which can be almost identical but fall on different side of this decision mark. For this *T-test* comparison a *two tailed* test was required as the variation in values could happen in both directions of the values distribution.

The results obtained with the *T-test* suggest that the mean differences between the control group and drug users (*C-DU*) and between control and pre-*AIDS* (*C-PA*) are not significant but that there is a significant difference between the estimates from control and *AIDS* Non-Encephalitis drug users, and also from control and *AIDS* Encephalitis drug users (*i.e., C-ANE and C-AE*).

For the interpretation and possible use of these results, the neuropathologist needs to relate the cell estimates obtained with other related factors. For example how homogeneous and well preserved was the brain tissue from the different groups, and in particular if there has been a significant shrinkage of the tissue which will affect the distribution estimates. The *SGmodel* system performance was compared with manual

Image	Slide	grp	neu AVG
neu01	NA89-329	AE	12
neu02	NA90-260	C	8
neu04	NA91-102	AE	12
neu05	NA91-115	AE	9
neu06	NA91-127	ANE	11
neu07	NA91-155	C	6
neu10	NA91-399	C	8
neu11	NA91-401	C	9
neu12	NA91-402	C	7
neu13	NA91-417	C	7
neu14	NA91-458	ANE	9
neu15	NA91-472	C	9
neu16	NA92-58	DU	10
neu19	NA92-161	PA	11
neu20	NA92-222	ANE	13
neu21	NA92-241	DU	14
neu22	NA92-259	PA	6
neu24	NA92-290	C	9
neu25	NA92-319	DU	8
neu26	NA92-337	PA	6
neu27	NA92-338	PA	11
neu29	NA92-353	AE	11
neu30	NA92-365	C	12
neu31	NA92-387	DU	5
neu32	NA92-390	ANE	12
neu33	NA92-451	PA	13
neu35	NA93-07	DU	11
neu36	NA93-30	DU	9
neu37	NA93-52	ANE	11
neu38	NA93-68	PA	9
neu41	NA93-104	DU	8
neu42	NA93-120	ANE	15
neu43	NA93-127	DU	9
neu47	NA95-10	PA	12
s1	NA91-353	AE	12
s10	NA93-02	DU	10
s12	NA93-89	PA	10
s13	NA93-94	PA	11
s16	NA93-135	AE	11
s17	NA93-358	AE	15
s18	NA94-284	DU	13
s19	NA95-66	PA	9
s2	NA92-115	ANE	12
s4	NA92-278	PA	10
s7	NA92-345	PA	13

Group	C	DU	PA	ANE	AE
	8	10	11	11	12
	6	14	6	9	12
	8	8	6	13	9
	9	5	11	12	11
	7	11	13	11	12
	7	9	9	15	11
	9	8	12	12	15
	9	9	10		
	12	10	11		
		13	9		
			10		
			13		
tot/gps	*75*	*97*	*121*	*83*	*82*
avg/ima	*8*	*12*	*10*	*12*	*12*

T-test				
	C-DU	*C-PA*	*C-ANE*	*C-AE*
auto	*0.191*	*0.031*	*0.002*	*0.002*

Figure 7.6: Neuron average counts in the testing set.

estimates performed on a set of images by two domain experts, in the following section this comparison is presented.

7.4 Performance evaluation: comparing manual vs automatic estimations

The neuron estimates obtained by the *SGmodel* system performance on a set of images were compared with those by two domain experts who manually assessed the same set of images.

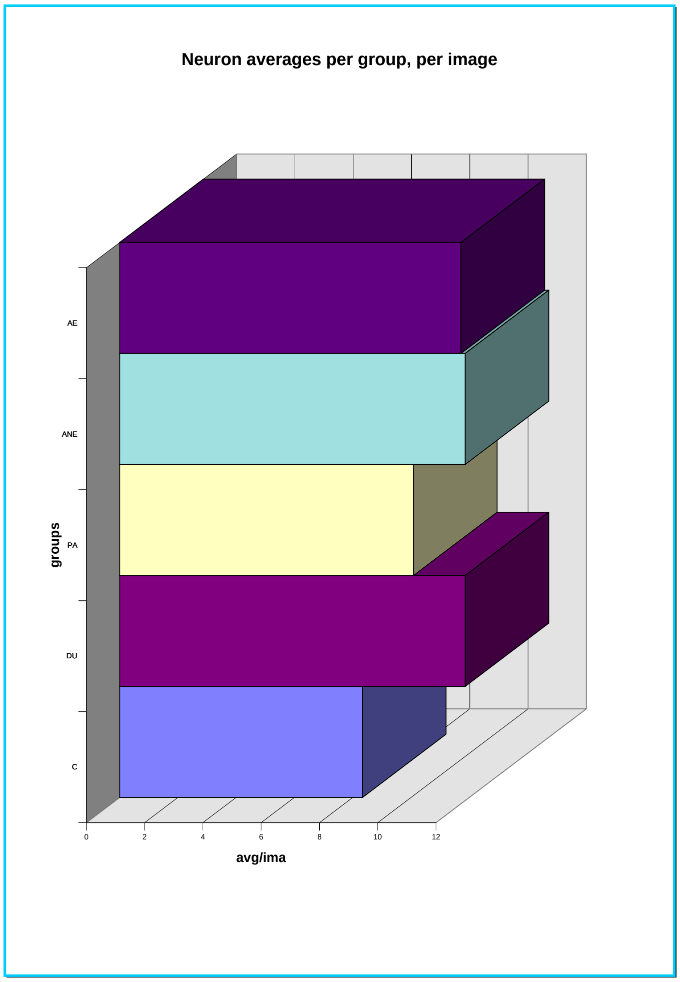

Figure 7.7: Average counts per group in the testing set

In order to evaluate if there were significant differences between neuron distribution estimates a paired *T-test* (where the individual values are equivalent) was applied. The *T-test* was applied to the estimates from the 45 images assessed manually (Manual1 and Manual2) and automatically (Predicted). In table 7.8 the counts per image are shown and the *T-test* results comparing Manual1 with Predicted (the *SGmodel* estimates), Manual2 with Predicted, and Manual1 with Manual2 are presented.

The *T-test* results show that there is no significant difference between the three global neuron estimates, but there is a bigger difference between Manual1 and Manual2 than between the *SGmodel* estimates and either of the manual counts. However in a case-by-case comparison, it was observed in some images higher variation between the auto-

	Image File	Slide ID	Cues	Manual1	Predicted1	Manual2
1	neu01-15	NA89-329	56	11	13	13
2	neu02-15	NA90-260	39	15	9	16
3	neu04-15	NA91-102	36	11	8	11
4	neu05-15	NA91-115	38	7	9	6
5	neu06-15	NA91-127	47	8	12	8
6	neu07-15	NA91-155	35	6	8	6
7	neu10-15	NA91-399	31	7	7	15
8	neu11-15	NA91-401	54	9	13	9
9	neu12-15	NA91-402	30	15	7	16
10	neu13-15	NA91-417	21	5	5	6
11	neu14-15	NA91-458	41	14	9	13
12	neu15-15	NA91-472	40	8	9	8
13	neu16-15	NA92-58	43	14	10	13
14	neu19-15	NA92-161	50	11	12	11
15	neu20-15	NA92-222	41	8	9	7
16	neu21-15	NA92-241	76	11	18	11
17	neu22-15	NA92-259	28	6	6	3
18	neu24-15	NA92-290	46	8	11	8
19	neu25-15	NA92-319	32	11	7	13
20	neu26-15	NA92-337	34	8	8	6
21	neu27-15	NA92-338	46	7	12	6
22	neu29-15	NA92-353	62	17	15	19
23	neu30-15	NA92-365	46	5	11	4
24	neu31-15	NA92-387	25	4	6	4
25	neu32-15	NA92-390	29	6	7	9
26	neu33-15	NA92-451	59	12	15	9
27	neu35-15	NA93-07	57	5	14	6
28	neu36-15	NA93-30	26	3	6	3
29	neu37-15	NA93-52	67	6	16	4
30	neu38-15	NA93-68	39	7	9	7
31	neu41-15	NA93-104	42	10	10	9
32	neu42-15	NA93-120	66	13	16	6
33	neu43-15	NA93-127	29	12	7	10
34	neu47-15	NA95-10	58	14	13	14
35	s1-15	NA91-353	44	8	11	6
36	s10-15	NA93-02	34	15	8	11
37	s12-15	NA93-89	40	15	9	13
38	s13-15	NA93-94	55	12	12	11
39	s16-15	NA93-135	42	16	10	13
40	s17-15	NA93-358	68	15	17	11
41	s18-15	NA94-284	53	12	12	10
42	s19-15	NA95-66	30	12	7	12
43	s2-15	NA92-115	49	11	12	11
44	s4-15	NA92-278	41	14	9	10
45	s7-15	NA92-345	65	15	16	11
	total counts		*1990*	*459*	*470*	*428*
	percentage			*100%*	*102.39%*	*93.25%*
				man1-auto	*man2-auto*	*man1-man2*
			T-Test:	*0.695517*	*0.190757*	*0.053962*

Figure 7.8: Evaluation results, comparing manual vs automatic estimation of neurons, total counts, averages and *T-test* results.

matic neuron estimate and any of the manual estimates (*e.g.* images: 2,5,9,16), see appendix D for a case-by-case descriptive statistical analysis. The case-by-case comparison suggests that in order to obtain better accuracy at individual levels, a more refined model definition would be necessary. This refinement can be done by either defining different or more features in the model[1] in order to fully capture the object variation, or if the object class variability is too big then by defining several subclass neuron models. For example, figure 7.9 shows two cases of misclassified objects. The false -ve case has a grey level distribution more similar to the background than to the

[1] *e.g.*, in this case it could be by selecting more radii or grey-level gradient values for the neuron model.

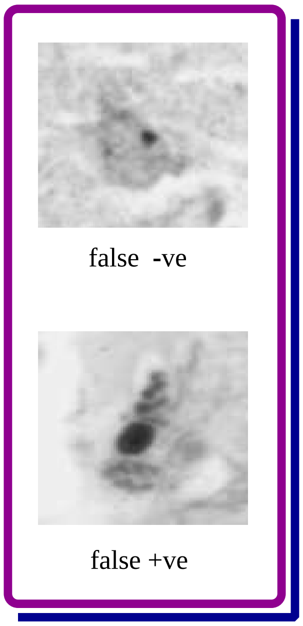

Figure 7.9: Examples of misclassified objects, on top a false -ve from image s19-15 and at the bottom a false +ve from image neu37-15.

distributions found in the training set. In the false +ve case, the grey level distribution is similar to some found in neuron examples.

The importance of this variation at individual level would depend on the application requirements. The results obtained by the *SGmodel* system would give useful information for cell distribution studies, which was the specification for this thesis application, therefore the analysis presented here is focused at this level.

The 45 counts were also compared per case group (*i.e., C, DU, PA, ANE* and *AE*) for each counting type, see table 7.10. The results of the *T-test* do not show significant difference between manual and automatic estimates, but there is a significant difference for the pre-*AIDS* group (PA) between Manual1 and Manual2 assessment. Figure 7.11 is the graphical representation comparing the results from the three different estimates (Manual1, Manual2 and Predicted).

With the information in table 7.10 it is possible to compare separately the distribution per group between Manual1 and Predicted estimates. As mentioned before, the differences found in the average per image estimates are not significant and the distribution

count	Man1	Pred	Man2	Man1	Pred	Man2	Man1	Pred	Man2	Man1	Pred	Man2	Man1	Pred	Man2
group	C	C	C	DU	DU	DU	PA	PA	PA	ANE	ANE	ANE	AE	AE	AE
	15	9	16	14	10	13	11	12	11	8	12	8	11	13	13
	6	8	6	11	18	11	6	6	3	14	9	13	11	8	11
	7	7	15	11	7	13	8	8	6	8	9	7	7	9	6
	9	13	9	4	6	4	7	12	6	6	7	9	17	15	19
	15	7	16	5	14	6	12	15	9	6	16	4	8	11	6
	5	5	6	3	6	3	7	9	7	13	16	6	16	10	13
	8	9	8	10	10	9	14	13	14	11	12	11	15	17	11
	8	11	8	12	7	10	15	9	13						
	5	11	4	15	8	11	12	12	11						
				12	12	10	12	7	12						
							14	9	10						
							15	16	11						
total	78	80	88	97	98	90	133	128	113	66	81	58	85	83	79
avg/im	9	9	10	10	10	9	11	11	9	9	12	8	12	12	11

Ttest man1-auto				
C	DU	PA	ANE	AE
0.887	0.95	0.68	0.25	0.83

Ttest man2-auto				
C	DU	PA	ANE	AE
0.66	0.59	0.25	0.19	0.73

Ttest man1-man2				
C	DU	PA	ANE	AE
0.25	0.23	0	0.356	0.37

Groups differences				
	C-DU	C-PA	C-ANE	C-AE
man1	1	0.149	0.675	0.1
man2	0.69	0.846	0.453	0.524
auto	0.549	0.163	0.115	0.072

Figure 7.10: Manual1, Manual2 and Predicted counts per case group.

of the estimates represented in graph 7.12 and 7.13 are quite similar, the only group that has slightly bigger difference is ANE, in fact this is the only group that has a different average number of neurons per image.

On the other hand, by comparing separately Predicted and Manual2 estimates there are small differences in all the averages per image numbers, but the difference is also bigger for group ANE, see graph 7.14. In this group the Manual1 and Manual2 were more similar. However the distributions from Predicted and Manual2 are also similar.

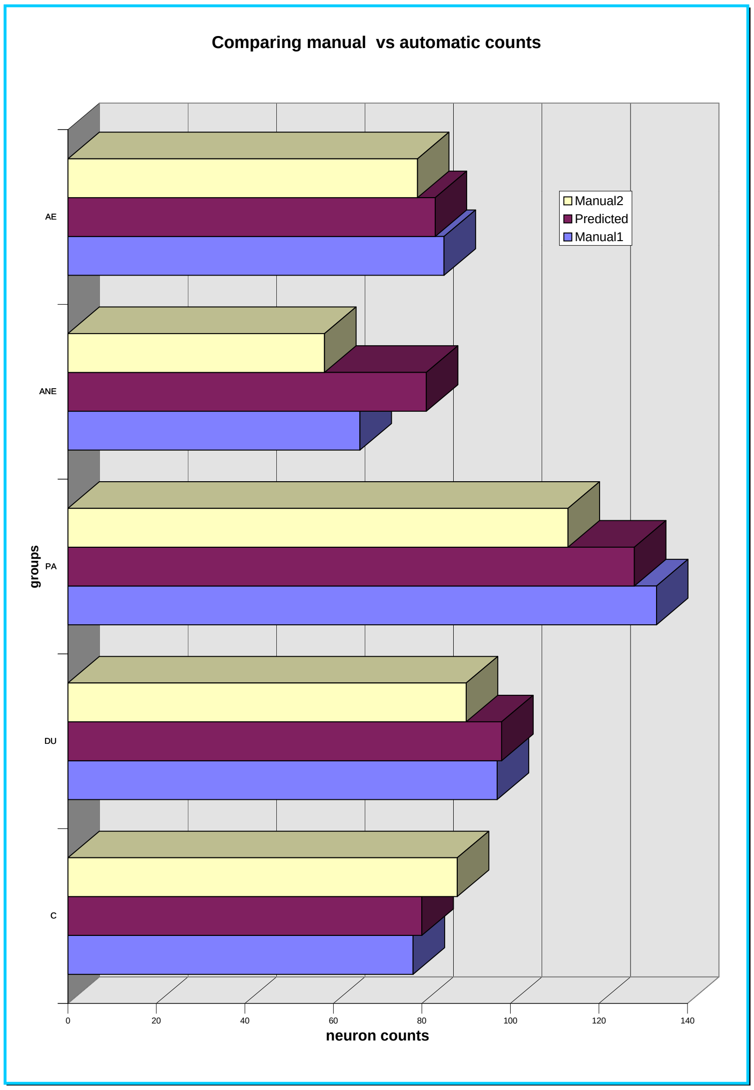

Figure 7.11: Comparing manual vs automatic counts on each case group.

7.5 Program timings

Table 7.15 provides measurement of average times used by *SGmodel* at different performance steps on a Sun Sparc 10 workstation. The averages were calculated at different times during the day to include possible variations in the machine workload. The system developed for this thesis was implemented to allow interactive testing and analysis but in the first instance without consideration of computing time. For reference purposes the time taken for typical usage are given with the realisation that a significant speed-up could be achieved for an installed system.

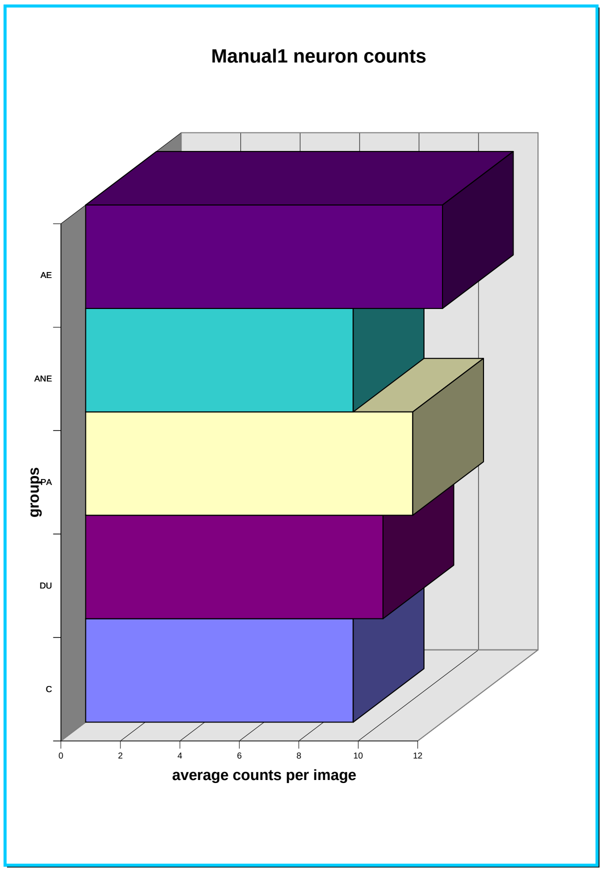

Figure 7.12: Manual1 neuron counts between groups.

7.6 Conclusions

The main modes of variation observed from real data involved in most of the cases both shape and grey-value correlation and gave some understanding about the variability characterisation by PCA. The possibility of generating new examples using the class model and within the training set limits has proven useful for testing not only training set distributions but also to obtain a class probability of new image objects.

In this thesis application, the analysis accuracy was focused at cell distribution estimates level. The results of comparing the neuron distribution estimates per group show more similarities between the automated system results and each of the manual estimates, except for group ANE. For this group the results between manual-manual estimates were more similar than those between either manual and automated estim-

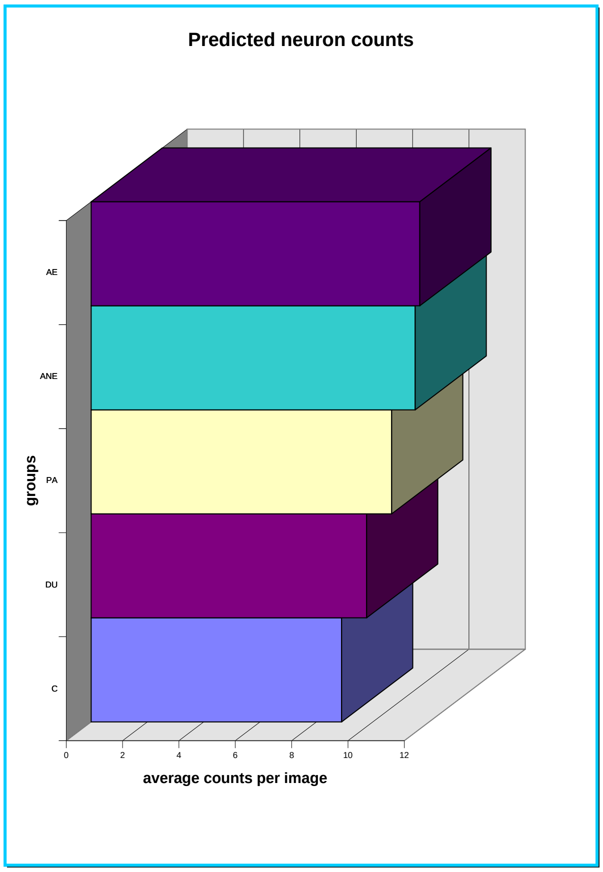

Figure 7.13: Comparing the *SGmodel* automatic neuron estimates between groups.

ates. This suggests that further analysis of the images from this group is required in order to find out the factors involved that make the *SGmodel* approach produce false +ve classifications. The differences between the manual-manual distributions, although not significant, show that there are variations between domain experts, further comparisons with a bigger number of domain experts would be required to reach more precise conclusions with respect to the estimated error. For this thesis a larger trial was not possible because of insufficient time and the number of available experts with suitable experience.

The manual-manual variations have been observed before and may be due to fatigue, etc., but are not the issue of this thesis. The smaller differences between manual and automated estimates show that the *SGmodel* system is a reliable tool for cell distribution estimates in microscope images. On the other hand in a case-by-case comparison,

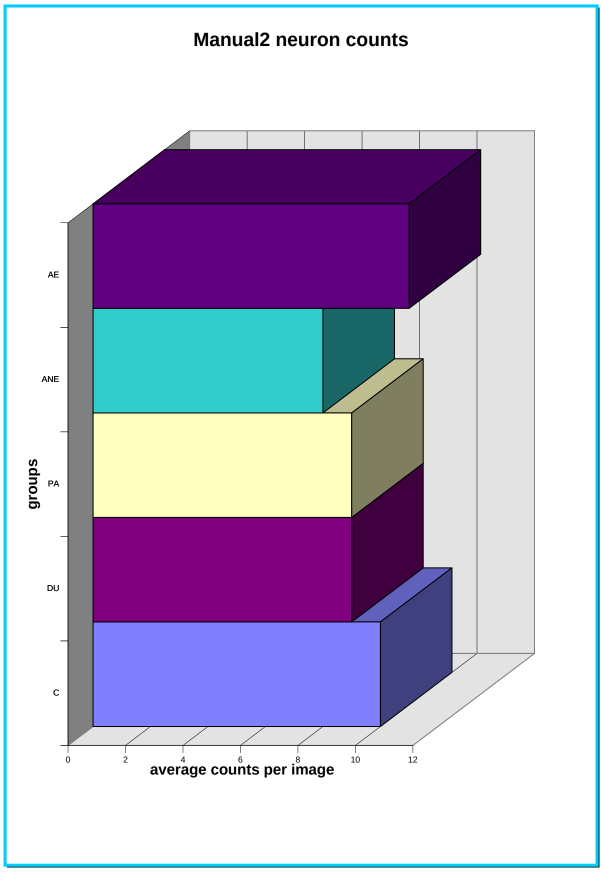

Figure 7.14: Manual2 neuron counts between groups.

it was observed in some images higher variation between the automatic neuron estimate and any of the manual estimates. The importance of this variation at individual level would depend on the application requirements. The results obtained by the *SGmodel* system would give useful information for cell distribution studies. However the misclassified cases observed suggest that better accuracy at individual levels could be obtained by either refining the model paramenters (modifying or increasing/decreasing the model features) to fully capture the object variability or by defining several subclass models where the class variation is too high to be captured in a single model.

Training set	45 images
total num. of	
objects found	
(cue finder)	1990
average number	
of objects per	
image	44
Average time for	
training a model	
per object	47 seconds
PCA over 1400	
examples	22 seconds
Building a	
concatenated	
classification	
model (set 1400)	3 hrs and 27 minutes
Average time for	
classification	
per object	0.71 minutes
Average time for	
classification	
per image	31.40 minutes

Figure 7.15: Performance timing at different processing levels using a Sun Sparc 10 workstation.

Chapter 8

Conclusions

This chapter presents the summary of the thesis, discusses conclusions and suggestions for future work. The main issues investigated in this thesis are reviewed in section 8.1, the conclusions drawn in each chapter are also included. Section 8.2 presents the thesis outcome with respect to the objectives and the thesis contributions are assessed. The limitations of this thesis are discussed in section 8.3, and some ideas for future work and research directions are outlined.

8.1 Thesis Summary

In chapter 1 a general introduction to the thesis is presented including the motivation of the thesis, which originated from the need for an accurate automatic quantitative analysis of biomedical images in order to assess the massive amount of information from the brain images. This quantitative analysis would help the understanding of some Central Nervous System (*CNS*) related diseases. The context of this motivation (*i.e.*, the *HIV* effects in the *CNS*) was described in chapter 2 as a review for the domain of application of this thesis. This review underlined the importance of (*SGmodel*) as the mechanism for a semi-automatic quantitative analysis of cell populations. The model-based framework described in chapter 3 provides the basis for the approach proposed in this thesis, and in particular some of the flexible models (*FM*) features. Despite the fact that *FM* have been successfully used for non-rigid object recognition, they present several disadvantages for their use in some biomedical applications. For example, some of the methods described require a starting model position near the object of interest, which would result in a highly interactive process. Other *FM* methods are not suitable for recognising weak edges. These problems were taken into account during the design of the approach proposed in this thesis namely the *SGmodel*. Chapter 4 provided a detailed description of the theoretical basis of the different elements that *SGmodel* integrates into a generic representation with the capability to characterise the object variability in the objects found in the domain application. The *SGmodel* has been developed in this thesis as an extended flexible modelling formalism, combined with statistical, probability and optimisation tools, to be a Model-based method for recognition and classification of biomedical objects. Chapter 5 described the combined elements used for the implementation of the novel hybrid *SGmodel* system such as traditional image analysis procedures (*i.e.*, threshold, dilation and erosion used for the *cue finder*), flexible model procedures (*PCA* based flexible model, *i.e.*, the extended *PDM*) within a probabilistic formulation, optimisation procedures (*i.e.*, *GAs* for the optimised search of the best possible match) and a *GUI*. The definition of an artificial cell for testing the system performance and to observe the modes of variation behaviour was also included in this chapter. The methodologies and step by step settings to apply the *SGmodel* from pre-processing to classification were presented in chapter 6. Chapter 7 presents the results obtained by the *SGmodel* performance on synthetic data which reached the objectives namely system testing and as an aid on the understanding of the *PCA* use for variability characterisation. The testing with real data involved the comparison between system-manual and manual-manual estimates, the results were discussed and represented in tables and graphs in this chapter as well. The results showed that the *SGmodel* performance met the specification of this thesis application, *i.e.*, to provide useful information at cell distributions level and suggested that in order to obtain better accuracy at individual level, model refinement would be required. This refinement can be achieved by modifying the model features selected (in number or specification of different features).

8.2 Achievements and contributions

The *SGmodel*'s novel generic model representation proposed in this thesis for recognition and classification, integrates different features which make it suitable for biomedical image analysis. These features are that:

- the model acts as an abstraction of the high-level prior knowledge of the objects, designed in a meaningful way to the domain expert;
- it is based on the statistical knowledge from a training set of examples. The main modes of variation of each object class are represented in a generic object formulation. During the object recognition process, this formulation adapts to the diversity and irregularities of shape and grey-values distribution;
- the variability of the shape-grey relationships is characterised by applying principal component analysis (PCA) to the shape-grey level vector extracted from the training set. Given the modes of variation, it is possible to generate new examples to be used to test the training set distributions, followed by a test matching of new data;
- during the matching process the obtained model is adjusted to fit the image data within the limits of variability from the training set;
- includes a mechanism to recover the normalisation given the inclusion of non-spatial data, different model parameters and/or different number of dimensions. The use of a *cue finder* affects the total probability, a normalisation to solve this is also included.
- the retrainable feature of this model makes it adaptable to new conditions, allowing rapid prototyping of new object classes.
- An important contribution of this approach is a more generic model for the domain application which, in turn, implies that the model can be tuned to specific study cases, without the need for specialist image-processing and programming expertise, to incorporate new models for each object class. For example if the application in turn requires better accuracy at individual object estimates, different combinations of model features can be selected and tested by the domain expert, including definition of subclass models if the object variation is too high to be capture by a single model.

- the *SGmodel* performance successfully met the goal of reproducing the neuron distribution estimates of the neuropathologist during the comparison with two human experts. No significant differences were found when a T-Test was applied to the *SGmodel*-manual estimates. Further comparisons with a bigger number of domain experts would be required to reach more precise conclusions with respect to the estimated error. Once optimised, the *SGmodel* system will be used for cell distribution estimates by the Neuropathology Laboratory, University of Edinburgh, different types of brain cells would be analysed (*e.g.*, astrocytes, microglia and neurons).

From this list it can be concluded that this approach provides the means for solving some of the quantitative problems related with cell distribution estimates with a generic system for analysis of microscope images, retrainable and adaptable to new conditions and therefore the objectives of this research have been achieved.

8.3 Future Work

The broad sweep of topics involved in this thesis leave many problems for the future work to address. The priorities for designing, implementing and testing this thesis approach were focused in reaching the objectives established at the beginning of this research. From the achievements and contributions mentioned in 8.2 it is possible to say that these objectives were reached. However many questions still remain unanswered in both the in the Biomedical and the Image Processing areas of this research. The following summarise the questions that have arisen during this research:

- To explore the modes of variation behaviour by introducing correlation between grey-level parameters into the artificial cell, making this simulation more similar to the real cells modes of variation.

- To explore the behaviour of the genetic algorithms module with different parameter settings. The *GAs* parameters currently used in the system are the result of informal testing and results reported in the *GAs* literature (Goldberg 1989). However they are not optimal and additional experimentation with other values will yield better performance on any given application.

- With the obtained feature vector to implement an artificial neural network approach to compare its performance in object recognition with the current statistical classifier. The performance of a well-trained neural network classifier is usually similar to that of a well-designed statistical classifier (Castleman 1996), the less detailed knowledge of the problem required for the development of a neural network classifier could result in a more generic system

- To establish the bases needed to include the present system into an image understanding framework. The image relevant information could be combined with the information of different applications by integrating a knowledge based system and a decision support system.

- To explore the use of different feature combinations (e.g. gradients) to achieve better classification at individual object levels.

- To compare the system performance with a bigger number of domain experts to obtain more precise conclusions with respect to the estimated error.

8.3.1 Biomedical Side

- To assess sets of brain tissue that could be relevant to other applications,

- To check the system reliability for a wider range of applications by comparing the *SGmodel* performance with domain experts for each different application.

The results obtained during the system testing are independent from those results arising from the neuropathologist interpretation of the estimates obtained. In chapter 2 it was mentioned that neuronal loss is expected as one of the effects of *HIV* in *CNS*. The comparison between the case groups assessed does not show this. Counting cells in the brain raises several problems related to the material and methodology, so these have to be carefully chosen in order to avoid erroneous conclusions. Some of the factors which could affect the counting are underlined by (Everall et al. 1991).

- Shrinkage produced during fixation and embedding of tissue block in paraffin-wax as part of the histology process, can alter the apparent number of cells.

- Given that the number of cells varies with age, it is important that the comparison with control material is with age-matched individuals.
- Opportunistic infections or neoplasms which have been reported coincidental with *HIV* infection themselves cause change in cell number. Study cases should be without these complications.
- The tissue section thickness should be chosen in a way to avoid, as much as possible, optical distortions which result in erroneous cell number and size.
- The sampling technique should be designed taking into account the orientation of the tissue in order to obtain a representative sample; otherwise results may easily be distorted by the complex structure and cellular organisation within the brain.
- It is important to perform the cell quantisation in a three dimensional way in order to have more realistic results, one technique often used for this is called "disector" (Sterio 1984), (Braendgaard and Gundersen 1986).

Bibliography

Bacus, J. and L. Grace (1987). Optical microscope system for standardized cell measurements and analyses. *Applied Optics 26*, 3280–3293.

Baldock, R. and I. Poole (1993, October). Video camera calibration for optical densitometry. *Journal of Microscopy 172*, 49–54.

Ballard, D. and C. Brown (1982). *Computer Vision*, Chapter Matching, pp. 352–382. Prentice-Hall, Inc.

Banda-Gamboa, H., I. Rickkets, K. Cairns, K. Hussein, J. Tucker, and N. Husain (1992). Automation in cervical cytology: an overview. *Analytical Cellular Pathology 4*, 25–48.

Bartels, H. (1994). Quantitation in histopathology. In A. Marchevsky and P. Bartels (Eds.), *Image Analysis: A primer for Pathologists*, pp. 1–28. Raven Press, Ltd.

Bennett, A. and I. Craw (1991). Finding image features using deformable templates and detailed prior statistical. In *British Machine Vision Conference*, pp. 231–239. Springer-Verlag.

Bishop, C. (1995). *Neural Networks for Pattern Recognition*, Chapter Pre-processing and feature extraction, pp. 295–331. Clarendon Press.

Braendgaard, H. and H. Gundersen (1986). The impact of recent stereological advances on quantitative studies of the nervous system. *Journal of Neuroscience Methods 18*, 39–78.

Brew, B., M. Rosenblum, K. Cronin, and R. Price (1995). Aids dementia complex and hiv-1 brain infection: clinical-virological correlations. *Annals of Neurology 38*, 563–570.

Brian, S. and A. Marchevsky (1994). Microphotometry in pathology. In A. Marchevsky and P. Bartels (Eds.), *Image Analysis: A primer for Pathologists*, pp. 181–206. Raven Press, Ltd.

Castleman, K. (1996). *Digital image processing*, pp. 447–545. Prentice-Hall Inc.

Cohen, L. (1991). On active contour models and balloons. *Comput. Vision, Graphics, and Image Processing: Image Understanding 53*(2), 211–218.

Collins, S., C. Wilbricht, S. Fleagle, S. Tadikonda, and M. Winniford (1991). An automated method for simultaneous detection of left and right coronary borders. In *Computers in Cardiology 1990*, pp. 7. IEEE.

Cootes, T., D. Cooper, C. Taylor, and J. Graham (1992). Training models of shape from sets of examples. In *British Machine Vision Conference*, pp. 9–18. Springer-Verlag.

Cootes, T., D. Cooper, C. Taylor, and J. Graham (1995). Active shape models their training and application. *Computer Vision and Image Understanding* (61), 38–59.

Cootes, T., A. Hill, C. Taylor, and J. Haslam (1994). The use of active shape models for locating structures in medical images. In *Information Processing in Medical Imaging, 13th Int. Conf. IMPI'93, Arizona, USA*, pp. 33–47. Springer-Verlag.

Cootes, T. and C. Taylor (1992). Active shape models - smart snakes. In *British Machine Vision Conference*, pp. 266–275. Springer-Verlag.

Cootes, T. and C. Taylor (1994). Modelling object appeareance using the grey-level surface. In *British Machine Vision Conference*, Volume 2, pp. 478–488. BMVA Press.

Cootes, T., C. Taylor, A. Lanitis, D. Cooper, and J. Graham (1992). Building and using flexible models incorporating grey-level information. In *British Machine Vision Conference*, pp. 1–30.

Cootes, T., C. Taylor, A. Lanitis, D. Cooper, and J. Graham (1993). Building and using flexible models incorporating grey-level information. In *Fourth International Conference on Computer Vision*, pp. 242–246. IEEE Computer Society Press.

Costen, N. and I. Craw (1996). Automatic face recognition: What representation? Submitted to the European Conference on Computer Vision ECCV'96 in Cambridge.

Craw, I., D. Tock, and A. Bennett (1992). Finding face features. In G. Sandini (Ed.), *Lecture Notes in Computer Science*, Number 58, pp. 92–96. Springer-Verlag.

De Jong, K. (1992). Are genetic algorithms function optimizers? *Parallel Problem Solving form Nature 2*, 3–13.

Dickson, D. and S. Lee (1996). Microglia in hiv-related cns neuropathology: An update. *Journal of NeuroAIDS 1*(1), 57–83.

Duda, R. and P. Hart (1973). *Pattern Recognition and Scene Analysis*, Chapter Unsupervised learning and clustering, pp. 189–297. John Wiley and Sons, Inc.

Elovaara, I. (1995). Hiv-1 infection of the brain: Which pathogenic mechanisms are relevant for tissue damage? *Medical Virology 5*, 105–119.

Elston, R. and W. Johnson (1994). *Essentials of Biostatistics*, pp. 35–134. F. A. Davis Company.

Everall, I. and P. Lantos (1991). The neuropathology of hiv: a review of the first 10 years. *International Review of Psychiatry 3*, 307–320.

Everall, I., P. Luthert, and P. Lantos (1991, May). Neuronal loss in the frontal cortex in hiv infection. *The Lancet 337*, 1119–1121.

Giulian, D., K. Vaca, and C. Noonan (1990). Secretion of neurotoxins by mononuclear phagocytes infected with hiv-1. *Science 250*, 1593–1596.

Glass, J., H. Fedor, S. Wesselingh, and J. McArthur (1995). Immunocytochemical quantitation of human immunodeficiency virus in the brain: Correlations with dementia. *Annals of Neurology* (38), 755–762.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimisation and Machine Learning*, Chapter Some applications of genetic algorithms, pp. 89–142. Addison-Wesley.

Gonzales, R. and R. Wood (1992). *Digital Image Processing*, pp. 571–658. Addison-Wesley.

Haslam, J., C. Taylor, and T. Cootes (1994). A probabilistic fitness measure for deformable template models. In *British Machine Vision Conference*, pp. 33–42. BMVA Press.

Heus, R. and P. Diegenbach (1992). The use of texture analysis for the discrimination of nissl substance in neurons. *Journal of Neuroscience Methods 44*, 209–215.

Hill, A., C. Taylor, and K. Lindley (1994). Medical image interpretation a generic approach using deformable templates. *Journal of Medical Informatics 1*(19), 47–59.

Husain, N., R. Allen, E. Hawkins, and J. Tylor (1974). The quantimet cytoscreen and the interactive approach to cancer screening. *Hystochem Cytochem 84*, 549–555.

Hutchinson, M. and D. Zahniser (1994). Cervical cytology automation. In A. Marchevsky and P. Bartels (Eds.), *Image Analysis: A primer for Pathologists*, pp. 309–334. Raven Press, Ltd.

Ivinis, J. and J. Porril (1994). Active region models for segmenting medical image. In *International Conference on Image Processing*, pp. 227–231. IEEE.

Johnson, E. (1989). Controlling the computer system, the user interface. In J. Capowski (Ed.), *Computer Techniques in Neuroanatomy*, pp. 265–283. Plenum Press.

Jolliffe, I. (1986). *Principal component analysis*, Chapter Principal component analysis for special types of data, pp. 199–234. New York Springer-Verlag.

Kapur, J. (1985). A new method for grey-level picture thresholding using the entropy of the histogram. *Computer Vision, Graphics, and Image Processing 29*, 273–285.

Kass, M., A. Witkin, and D. Terzopoulos (1987). Snakes: Active contour models. In *First International Conference on Computer Vision*, pp. 259–268. IEEE Computer Society Press.

Ketzler, S., S. Weis, H. Haug, and H. Budka (1990). Loss of neurons in the frontal cortex in aids brains. *Acta Neuropathologica 80*, 92–94.

Kirkpatrick, S., C. Gellat, and M. Vecci (1983). Optimization by simulated annealing. *Science* (220), 671–680.

Koenig, S., H. Gendelman, J. Orenstein, M. Dal Canto, G. Pezeshkpour, M. Yungbluth, F. Janotta, A. Aksamit, M. Martin, and A. Fauci (1986). Detection of aids virus in macrophages in brain tissue from aids patients with encephalopathy. *Science 233*, 1089–1093.

Lai, K. and R. Chin (1994). Deformable contours: Modeling and extraction. In *Pattern Analysis and Machine Intelligence*, pp. 601–608. IEEE press.

Lanitis, A., C. Taylor, T. Ahmed, and T. Cootes (1995). Classifying variable objects using a flexible shape model. In *5th International Conference on Image Processing and its Applications.*

Lanitis, A., C. Taylor, and T. Cootes (1995). Automatic face identification system using flexible appearance models. *Image and Vision Computing 13*(5), 393–401.

Lloyd, D., J. Piper, D. Rutovitz, and G. Shippey (1987). Multiprocessing interval processor for automated cytogenetics. *Applied Optics 26*(16), 3356–3366.

Luthert, P., M. Montgomery, A. Dean, R. Cook, A. Basketville, and P. Lantos (1995). Hippocampal neuronal atrophy occurs in rhesus macaques following infection with simian immunodeficiency virus. *Neuropathology and Applied Neurobiology 21*, 529–534.

Marchevsky, A. and B. Erler (1994). Morphometry in pathology. In A. Marchevsky and P. Bartels (Eds.), *Image Analysis: Aprimer for Pathologist*, pp. 125–179. Raven Press, Ltd.

Miller, A., R. Alston, and J. Corsellis (1979). The practical application of image-analysing system to neuropathology: principles and possible scope. In *Recent Advances in Neuropathology*, pp. 113–128. Churchill Livingstone.

Mitchell, M. (1996). *An introduction to genetic algorithms*, Chapter Implementing a Genetic Algorithm, pp. 155–179. MIT Press.

Nakagawa, Y. and A. Rosenfeld (1979). Some experiments on variable thresholding. *Pattern Recognition 11*, 191–204.

Onyango, C. and J. Marchant (1996). Flexible contour point distribution models. *Image and Vision Computing* (14), 703–708.

Parker, J. (1994). *Practical computer vision using C*, Chapter Image types and applications, pp. 17–20. John Wiley and Sons, Inc.

Piper, J., R. Baldock, R. Towers, and D. Rutovitz (1989). Towards a knowledge-based chromosome analysis system. In C. Lundsteen and J. Piper (Eds.), *Automation of Cytogenetics*, pp. 275–293. Springer-Verlag.

Piper, J. and D. Rutovitz (1985). Data structures for image processing in a c language and unix environment. *Pattern Recognition Letters 3*, 119–129.

Porril, J. and J. Ivinis (1994). A semiautomatic tool for 3d medical image analysis using active contour models. *Medical Informatics 19*(1), 81–90.

Price, R., B. Brew, J. Sidtis, M. Rosenblum, A. Scheck, and P. Cleary (1988). The brain in aids: Central nervous system hiv-1 infection and aids dementia complex. *Science 239*, 586–592.

Roberts, G., J. Bell, F. Grey, and I. Everall (1997). Neuronal loss in hiv. *Neuropathol. Appl. Neurobiol. 23*, 435.

Robinson, G., A. Colchester, and L. Griffin (1994). Model-based recognition of anatomical objects from medical images. *Image and Vision Computing 12*(8), 499–507.

Rushford, C. (1968). Restoration, resolution and noise. *J. Opt. Soc. Amer. 58*(4), 539–545.

Russ, J. (1994). *The Image Processing Handbook*, Chapter Segmenting and thresholding, pp. 347–405. CRC Press, Inc.

Sharer, L. (1992, January). Pathology of hiv-1 infection of the central nervous system. A review. *Journal of Neuropathology and Experimental Neurology 51*(1), 3–11.

Solloway, S., C. Taylor, C. Hutchinson, and J. Waterton (1996). Quantification of articular cartilage from mr images using active shape models. In *4th European Conference in Computer Vision, Cambridge University*, pp. 400–412. Cambridge University Press.

Sonka, M., H. V., and B. R. (1993). *Image Processing, Analysis and Machine Vision*, pp. 255–372. Chapman and Hall.

Staib, L. and J. S. Duncan (1992, Nov). Boundary finding with parametrically deformable models. *IEEE Transactions on Pattern Analysis and Machine Intelligence 14*(11), 1061–1075.

Sterio, D. C. (1984, May). The unbiased estimation of number and sizes of arbitrary particles using the disector. *Journal of Microscopy 134*(2), 127–136.

Storvik, G. (1994, Oct). A bayesian approach to dynamic contours through stochastic sampling and simulated annealing. *IEEE Transactions on Pattern Analysis and Machine Intelligence 16*(10), 976–986.

Sumner, A. (1969). *A Laboratory Manual of Microtechnique and Histochemistry.* Blackwell Scientific Publicatins, Oxford and Edinburgh.

Tucker, J. (1979). Preliminary results for automated cervical cytology using nuclear dna content. *Machine Perception of Patterns and Pictures (Inst. Phys. Conf. 44)*, 186–194.

Vernon, D. (1991). *Machine Vision*, pp. 28–42. Prentice Hall.

Vinters, H. and K. Anders (1990). *Neuropathology of AIDS*, pp. 6–11. CRC Press.

Warren, M. (1992). Simple morphometry of the nervous system. In M. Stewart (Ed.), *Quantintative Methods in Neuroanatomy*, pp. 211–247. John Wiley and Sons.

Wiley, C., E. Masliah, M. Morey, C. Lemere, R. DeTeresa, M. Grafe, L. Hansen, and R. Terry (1991, June). Neocortical damage during hiv infection. *Annals of Neurology 29*(6), 651–657.

Yuille, A. (1991). Deformable templates for face recognition. *Journal of Cognitive Sciences 3*(1), 59–70.

Yuille, A., D. Cohen, and P. Hallinan (1989). Facial feature extraction by deformable templates. In *CVPR*, pp. 104–109.

Yuille, A., H. P.W., and D. Cohen (1992). Feature extraction from faces using deformable templates. *International Journal of Computer Vision 8*(2), 99–111.

Glossary

AIDS Acquired Immune Deficiency Syndrome. The late stage of a disease caused by infection with the virus called HIV.

ADC AIDS Dementia Complex, disorder with progressive dementia.

A priori knowledge (information) Knowledge available before the search for a problem solution starts.

ASABI Automated System for Analysis of Biomedical Images.

ANOVA One way of variance analysis

AZT azidothymidine, zidovudine.

CMV Cytomegalovirus, pathogen agent which produces neurologic abnormalities by demyelination.

CNS Central Nervous System, composed by brain and spinal cord.

CV Computational Vision

GFAP Glial fibrillary acidic protein.

GA's Genetic Algorithms search.

HIV Human Immunodeficiency Virus.

HSV Herpes simplex, infectious agent.

Image Analysis Term used to embodied the idea of automatically extracting useful information from and image.

MGC Multinucleated Giant Cells.

Nissl substance aggregation of basophilic material in the cytoplasm of nerve cells.

PCA Principal Component Analysis, statistical analysis to reduce the dimensionality of data.

PDM Point Distribution Models.

Varicella zoster infectious agent.

WOOLZ MRC Image Processing System based on object oriented data structures.

acute Rapid in onset; severe, life-threatening.

antigen foreign substance.

astrocytes supporting cells of the nervous system.

axon neuron prolongation which carry impulses away from the cell body

candida fungal pathogen.

cresyl Stain used in histological techniques.

cryptococcus fungal pathogen.

disector a probe that samples objects in three-dimensional space irrespective of their size or shape.

encephalitis brain inflammation.

encephalopathy neurological abnormalities, brain disease.

glia supporting cells of the nervous system, astrocytes, oligodendrocytes and microglia.

SGmodel Shape-grey value Model.

gyral latter separate in the surface of the brain.

heuristic mothod of solving problems by evaluating past experience.

immunocytochemistry histological technique based in immunochemical reactions to mark specific structures.

leukoencephalopathy neurological disease caused by the infectious agent papovavirus.

lymphoma enlarged lymph nodes.

macrophages large phagocytes, cells that destroy bacteria or cell fragments.

meningitis inflammation of the membranes enclosing the brain and spinal cord.

microglia supporting cells of the nervous system.

morphometric measurements of form and structure.

myelin white fatty material which ensheath axons.

neoplasms tumour, a new growth of tissue serving no physiologic function (e.g. lymphoma).

neuron nerve cell.

neuroectodermal cells derivated from the wall of the neural tube (i.e. neurons, glial cells and ependymal cells).

neuropathy nervous system disorder.

neuropathogenesis origin of neurological abnormalities.

neuropil terminals and dendrites in neurons.

nuclei spherical structure commonly in a central position of the cell body.

nucleolus spherical prominent structure within the nucleus.

oligodendrocytes supporting cells of the nervous system.

pallor paleness.

pathogenetic referent of the nature of disease.

perikarion cell body.

toxoplasmosis infection caused by the infectious agent Toxoplasma gundii.

xgobi dynamic graphics program for data analysis for X windows.

Appendix A

Sampling information: slides, images and case groups

Histological material

The table in the following page show the images used in this thesis which were digitised from a set of 45 histological slides, 25 images per slide were obtained. For training purposes 236 images were used for model the *neuron* and 215 for the model *NOneuron*.

Image	slide-code	group
neu01	NA89-329	AE
neu02	NA90-260	C
neu04	NA91-102	AE
neu05	NA91-115	AE
neu06	NA91-127	ANE
neu07	NA91-155	C
neu10	NA91-399	C
neu11	NA91-401	C
neu12	NA91-402	C
neu13	NA91-417	C
neu14	NA91-458	ANE
neu15	NA91-472	C
neu16	NA92-58	DU
neu19	NA92-161	PA
neu20	NA92-222	ANE
neu21	NA92-241	DU
neu22	NA92-259	PA
neu24	NA92-290	C
neu25	NA92-319	DU
neu26	NA92-337	PA
neu27	NA92-338	PA
neu29	NA92-353	AE
neu30	NA92-365	C
neu31	NA92-387	DU
neu32	NA92-390	ANE
neu33	NA92-451	PA
neu35	NA93-07	DU
neu36	NA93-30	DU
neu37	NA93-52	ANE
neu38	NA93-68	PA
neu41	NA93-104	DU
neu42	NA93-120	ANE
neu43	NA93-127	DU
neu47	NA95-10	PA
s1	NA91-353	AE
s10	NA93-02	DU
s12	NA93-89	PA
s13	NA93-94	PA
s16	NA93-135	AE
s17	NA93-358	AE
s18	NA94-284	DU
s19	NA95-66	PA
s2	NA92-115	ANE
s4	NA92-278	PA
s7	NA92-345	PA

Figure A.1: Image name, slide code and case group.

Appendix B

SGmodel system

System description

SGmodel is an iterative system that uses a simple graphical user interface *GUI*, developed using C language, X Window System libraries *Xlib*, and the X Window Manager libraries *Motif*. SGmodel system was implemented and tested in a Sun Sparc Station 10, under Unix operating system SunOS5. When the program is executed by typing in the Unix environment gmodel, the SGmodel main window is displayed. On the top of this window the main action bar show three options; "File", "Model" and "Analysis". The selection of an option is made by clicking on it with the left button on the mouse.

- If the option "File" is selected, a menu to handle files is scrolled down, and in it several actions can be activated, see figure B.1.

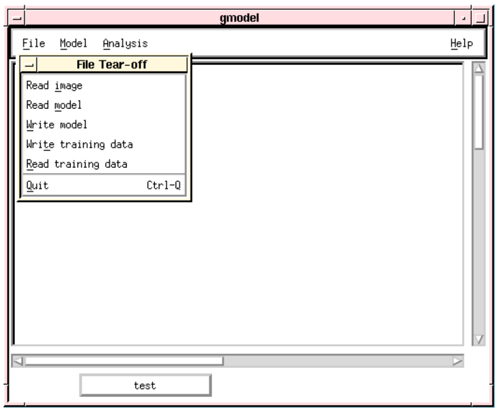

Figure B.1: File actions menu.

  - **Read Image**: With this option a file selection window is displayed with the images available in the images directory specified. The image selected is then displayed in the main SGmodel window, see figure B.2.
  - **Write training data**: Write in file the data obtained during the model training action which is going to be described below.

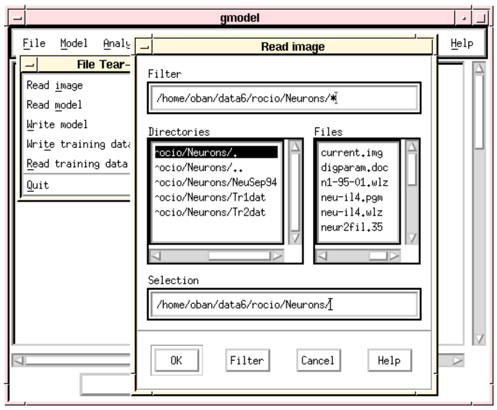

Figure B.2: File selection menu to display an image.

- **Read training data**: Read training data written in a previous session and put them available to be processed by the system. The file selection menu is similar to the one displayed for an image selection, but the list of files is from the data directory specified.
- **Back to image from data**: When this option is selected a list of the image name included in the current training data is displayed, then when a name is selected, the image is displayed in the main window and the instances sampled in this image are labelled with the model name and its image coordinates.

• If the option "Model" from the main actions bar is selected, a menu with the following model action is displayed, see figure B.3:

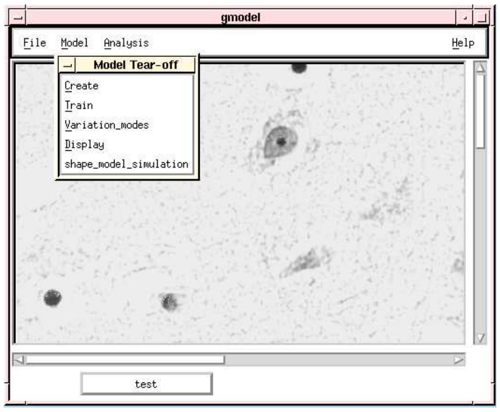

Figure B.3: Model actions menu.

- **Create**: When this action is selected, a window to establish a model set of parameters is displayed. These parameters determine the default points distribution form that is going to be used while sampling data from images for a particular model. In other words defining for the radial template, the number of radii and their angles, and the number of points to be sampled per radius, see figure B.4.
- **Train**: If this option is selected a model selection window is displayed. One model name should be selected and the default parameters for that model are made available for the sampling template. The models are trained by sampling the grey values

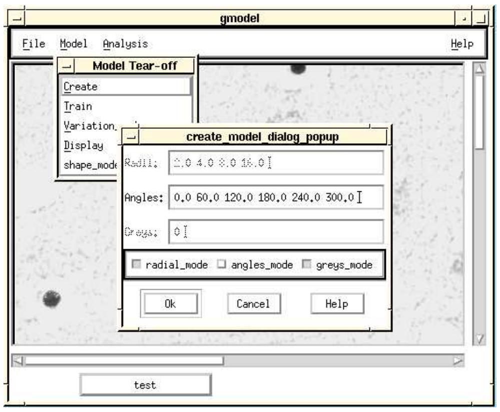

Figure B.4: Model parameter selection.

using the the distribution of this template on the objects of interest. First the left button of the mouse should be clicked the object of interest selected (see figure B.5), a magnification window is then displayed to ease in the selection of the points where the grey values are going to be sampled, see figure B.6, then the angle of the first radius should be chosen using a fix predefined criterion. The number of points defined for the model chosen, are then sampled using the default distances defined when the model was created or using criteria according with the objects features. For example if a model neuron is chosen, first the criterion to chose the first angle could be the direction where the shortest distance between the nucleolus and the edge of the nucleus is found, if five points per radius are going to be sampled, then the criteria to place the sampling points could be, for example, the edge of the nucleolus, a point between the edge of the nucleolus and the edge of the nucleus, a point on the edge of the nucleus and the last two points with fixed distances, this is repeated for all the radii on each of the objects sampled. The number of sampled object per model should be big enough to be able to capture all the possible variation modes for that particular object class, obtaining a representative data set of its grey values distribution. The objects sampled are labelled with the model name and their image coordinates, this avoids the possibility of multiple sampling and allows going back to image to review sampled objects.

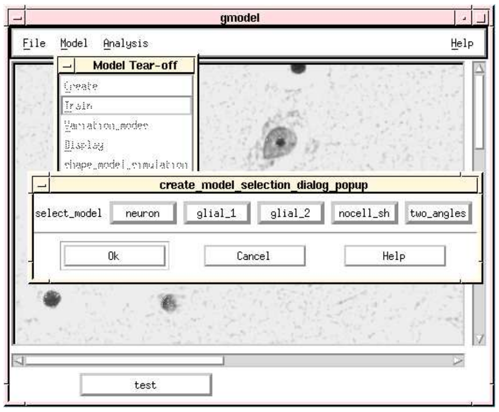

Figure B.5: Selection of a model to be trained.

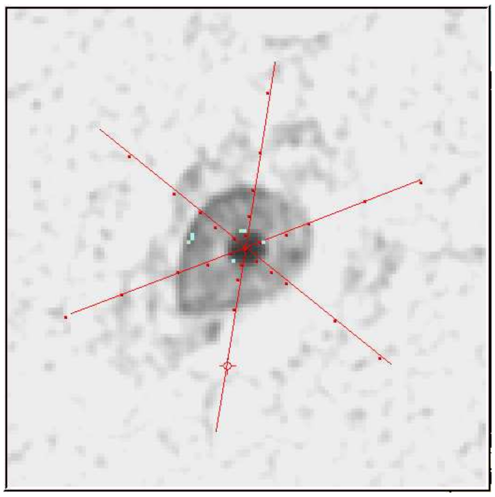

Figure B.6: Sampling grey values in a class object example.

– **Variation modes**: The data set obtained from the training model procedure are stored, for each object in a relative positions vector and associated grey values. Principal component analysis (PCA) is applied to these vectors, obtaining the variation modes within the training set.

– **Display**: A mean shape obtained from the PCA is display in another window. In the bottom part of the window, a set of slide bars, one per mode of variation obtained as well during the PCA, is displayed. When one of the bars is moved to the minimum of maximum value, it is possible to see how this variation mode affects the mean shape and its grey values distribution, see figure B.7.

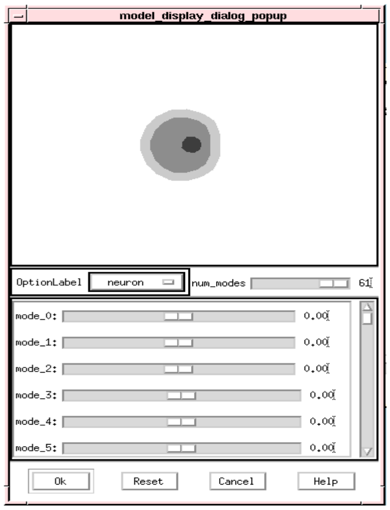

Figure B.7: Display of the mean object obtained from the training set.

– **Shape model simulation**: With this option a set of artificial model cell is generated with a restricted model parameters in order to analyse the behaviour of the modes of variation, as described in chapter 6.

- Finally if the option "Analysis" is chosen it is performed a comparison of the selected model with the object candidates located in the displayed image. The template matching is performed using the trained model and its main modes of variation, deforming iteratively the model template including center position, orientation and weight parameters. Each time that the model instance is adjusted, the vector of weights and its probability density were recalculated, this process is repeated until the maximum probability is reached. A genetic algorithm is used here as iterative optimisation method in order to find the best possible solution.

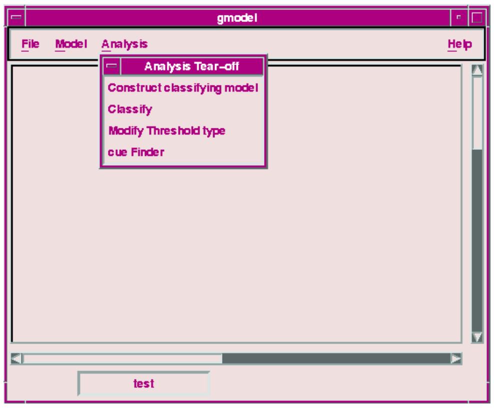

Figure B.8: Analysis Menu, Threshold modifier, cue Finder, calssification model construction and classification process.

The analysis menu actions are described below, the order corresponds to a normal execution sequence:

- bf cue Finder: By selecting this option the system locate all the possible object candidates in the current image. If there is not image displayed in the main window an error message is displayed. This option is activated individually by pressing the cue Finder option in the menu in order to check the system performance for the current conditions. This action is also performed automatically when the option Classify is chosen.
- **Modify Threshold type**: This action was designed for calibration. If the cue Finder performance is poor for the current conditions, a different thresholding type can be selected.
- **Construct classifying model**: This option has to be executed before the classification process (Classify) is activated. The training data have to be read and the modes of variation calculated previously.
- **Classify**: An image should be read before choosing this action. When selected the model class for the classification process has to be chosen. The candidates in the current image are located, the best possible match for each model class is calculated and concatenated in a single vector to compare with the classifying model and the probability of the object belonging to the model class selected is estimated.

Appendix C

Microglia and astrocytes data

The table below shows microglia and astrocytes counts performed as part of the comparative assessment for the *HIV-CNS* project[1] (Roberts et al. 1997). The groups studied, as described in chapter 2, were: *AIDS* Encephalitis drug users (AE), *AIDS* Non-Encephalitis drug users (ANE), HIV^+ Pre-*AIDS* drug users (PA), HIV^- drug users (DU) and Control HIV^-, non-drug users (C). A total of 48 cases were used (between 7 and 12 cases per group, in the table s1-s12). Astrocytes and microglia cells were semi-automatically counted of brain tissue with no *CNS* opportunistic infections or lymphomas. The counts were performed in white and grey matter.

Astrocytes White matter	s1	s2	s3	s4	s5	s6	s7	s8	s9	s10	s11	s12
AE	1252	1209	333	222	1360	1326	80	761	548			
ANE	45	503	176	116	387	241	591					
PA	182	346	921	0	781	20	72	45	1071	115	492	240
DU	15	776	284	0	76	416	896	74	67	30		
C	0	366	414	177	58	276	34	524	139	602		

Astrocytes Grey matter	s1	s2	s3	s4	s5	s6	s7	s8	s9	s10	s11	s12
AE	533	284	229	118	147	806	14	118	52			
ANE	94	5	4	2	46	8	3					
PA	5	8	13	2	1	5	6	5	4	6	10	11
DU	1	3	29	9	16	4	29	7	6	3		
C	8	1	9	1	16	2	2	37	29	31		

Microglia White matter	s1	s2	s3	s4	s5	s6	s7	s8	s9	s10	s11	s12
AE	11	7	15	8	7	7	2	13	7			
ANE	2	2	2	4	4	5	4					
PA	5	3	5	5	3	2	6	4	5	4	2	2
DU	3	3	3	4	2	3	2	2	2	2		
C	2	2	1	2	0	2	2	1	0	1		

Microglia Grey matter	s1	s2	s3	s4	s5	s6	s7	s8	s9	s10	s11	s12
AE	6	4	5	3	4	15	1	3	5			
ANE	2	0	1	2	2	3	4					
PA	2	1	2	3	2	2	3	3	2	2	2	1
DU	2	2	1	3	0	2	1	0	2	1		
C	0	0	0	0	0	0	1	0	0	0		

Figure C.1: Cell counts for groups AE, ANE, PA, DU and C, see Chapter 2.

[1] At Neuropathology Laboratory, University of Edinburgh.

Appendix D

Case-by-case comparison

The table below was presented in chapter 7 (figure 7.8). In a case-by-case comparison it was observed in some images higher variation between the automatic neuron estimate and any of the manual estimates. For example in image 29 (neu37-15) the automatic neuron estimate counted considerably more objects than any of the manual counts. In the other hand the automatic estimate in image 42 (s19-15) was lower than in the manuals counts.

	Image File	*Slide ID*	*Cues*	*Manual1*	*Predicted1*	*Manual2*
1	neu01-15	NA89-329	56	11	13	13
2	neu02-15	NA90-260	39	15	9	16
3	neu04-15	NA91-102	36	11	8	11
4	neu05-15	NA91-115	38	7	9	6
5	neu06-15	NA91-127	47	8	12	8
6	neu07-15	NA91-155	35	6	8	6
7	neu10-15	NA91-399	31	7	7	15
8	neu11-15	NA91-401	54	9	13	9
9	neu12-15	NA91-402	30	15	7	16
10	neu13-15	NA91-417	21	5	5	6
11	neu14-15	NA91-458	41	14	9	13
12	neu15-15	NA91-472	40	8	9	8
13	neu16-15	NA92-58	43	14	10	13
14	neu19-15	NA92-161	50	11	12	11
15	neu20-15	NA92-222	41	8	9	7
16	neu21-15	NA92-241	76	11	18	11
17	neu22-15	NA92-259	28	6	6	3
18	neu24-15	NA92-290	46	8	11	8
19	neu25-15	NA92-319	32	11	7	13
20	neu26-15	NA92-337	34	8	8	6
21	neu27-15	NA92-338	46	7	12	6
22	neu29-15	NA92-353	62	17	15	19
23	neu30-15	NA92-365	46	5	11	4
24	neu31-15	NA92-387	25	4	6	4
25	neu32-15	NA92-390	29	6	7	9
26	neu33-15	NA92-451	59	12	15	9
27	neu35-15	NA93-07	57	5	14	6
28	neu36-15	NA93-30	26	3	6	3
29	neu37-15	NA93-52	67	6	16	4
30	neu38-15	NA93-68	39	7	9	7
31	neu41-15	NA93-104	42	10	10	9
32	neu42-15	NA93-120	66	13	16	6
33	neu43-15	NA93-127	29	12	7	10
34	neu47-15	NA95-10	58	14	13	14
35	s1-15	NA91-353	44	8	11	6
36	s10-15	NA93-02	34	15	8	11
37	s12-15	NA93-89	40	15	9	13
38	s13-15	NA93-94	55	12	12	11
39	s16-15	NA93-135	42	16	10	13
40	s17-15	NA93-358	68	15	17	11
41	s18-15	NA94-284	53	12	12	10
42	s19-15	NA95-66	30	12	7	12
43	s2-15	NA92-115	49	11	12	11
44	s4-15	NA92-278	41	14	9	10
45	s7-15	NA92-345	65	15	16	11
	total counts		*1990*	*459*	*470*	*428*
	percentage			*100%*	*102.39%*	*93.25%*
				man1-auto	*man2-auto*	*man1-man2*
			T-Test:	*0.695517*	*0.190757*	*0.053962*

Figure D.1: Evaluation results, comparing manual vs automatic estimates of neurons, total counts, averages and *T-test* results.

This higher variation examples are clearer in graph D.2 where automatic and manual estimates are plotted.

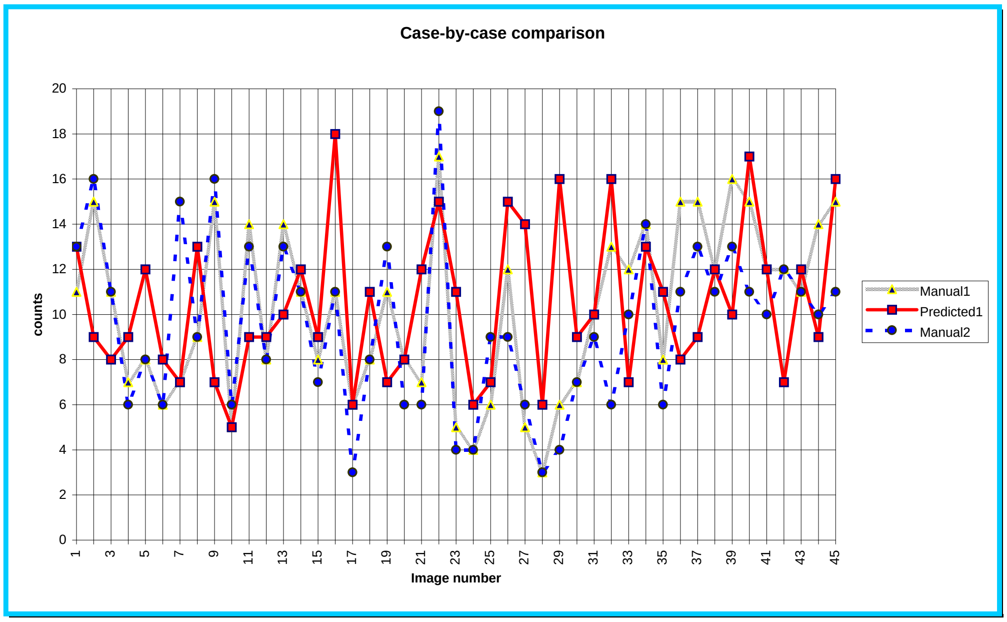

Figure D.2: Case-by-case comparison between Manual1, Predicted and Manual2 estimates.

As discussed earlier this higher variation would be important if more accuracy is needed in individual cases, this depends on the application requirements. Descriptive statistical analysis are included in table D.3. However for a good degree of accuracy a bigger sample size of human experts would be needed, and collaboration with different related research centres would be required. With the given sample the variation between the manual estimates is smaller than between automatic and either of the manual counts.

As a measure of variability, table D.3 presents the following computations:

- in column A the mean value from the Manual1, Predicted (automatic) and Manual2 estimates per image,
- columns B, C and D are the substracted mean from each estimate and divided by the mean to get a "normalised deviation" from the mean,
- the squared deviation from the mean are presented in columns E, F and G,
- at the bottom of this columns n (45 images) is included, and computations such as

  the sum of the squares of the deviations from the mean ($Sum\ d\hat{}2$), the *variance* (*Sum d^2 / n-1*), *i.e.,* the sum of the above squares divided by one less than the number of values in the set of data,

the standard deviation $\sigma = \sqrt{variance}$, and

and the standard deviation of the mean $= \frac{\sigma}{\sqrt{n}}$ (*s.d.*), which is often called **the standard error** (Elston and Johnson 1994).

	A	B	C	D	E	F	G
	Mean	D(M1)/Mean	D(P)/Mean	D(M2)/Mean	d(M1)^2	d(P)^2	d(M2)^2
1	12.3333	-0.1081	0.0541	0.0541	0.0117	0.0029	0.0029
2	13.3333	0.1250	-0.3250	0.2000	0.0156	0.1056	0.0400
3	10.0000	0.1000	-0.2000	0.1000	0.0100	0.0400	0.0100
4	7.3333	-0.0455	0.2273	-0.1818	0.0021	0.0517	0.0331
5	9.3333	-0.1429	0.2857	-0.1429	0.0204	0.0816	0.0204
6	6.6667	-0.1000	0.2000	-0.1000	0.0100	0.0400	0.0100
7	9.6667	-0.2759	-0.2759	0.5517	0.0761	0.0761	0.3044
8	10.3333	-0.1290	0.2581	-0.1290	0.0166	0.0666	0.0166
9	12.6667	0.1842	-0.4474	0.2632	0.0339	0.2001	0.0693
10	5.3333	-0.0625	-0.0625	0.1250	0.0039	0.0039	0.0156
11	12.0000	0.1667	-0.2500	0.0833	0.0278	0.0625	0.0069
12	8.3333	-0.0400	0.0800	-0.0400	0.0016	0.0064	0.0016
13	12.3333	0.1351	-0.1892	0.0541	0.0183	0.0358	0.0029
14	11.3333	-0.0294	0.0588	-0.0294	0.0009	0.0035	0.0009
15	8.0000	0.0000	0.1250	-0.1250	0.0000	0.0156	0.0156
16	13.3333	-0.1750	0.3500	-0.1750	0.0306	0.1225	0.0306
17	5.0000	0.2000	0.2000	-0.4000	0.0400	0.0400	0.1600
18	9.0000	-0.1111	0.2222	-0.1111	0.0123	0.0494	0.0123
19	10.3333	0.0645	-0.3226	0.2581	0.0042	0.1041	0.0666
20	7.3333	0.0909	0.0909	-0.1818	0.0083	0.0083	0.0331
21	8.3333	-0.1600	0.4400	-0.2800	0.0256	0.1936	0.0784
22	17.0000	0.0000	-0.1176	0.1176	0.0000	0.0138	0.0138
23	6.6667	-0.2500	0.6500	-0.4000	0.0625	0.4225	0.1600
24	4.6667	-0.1429	0.2857	-0.1429	0.0204	0.0816	0.0204
25	7.3333	-0.1818	-0.0455	0.2273	0.0331	0.0021	0.0517
26	12.0000	0.0000	0.2500	-0.2500	0.0000	0.0625	0.0625
27	8.3333	-0.4000	0.6800	-0.2800	0.1600	0.4624	0.0784
28	4.0000	-0.2500	0.5000	-0.2500	0.0625	0.2500	0.0625
29	8.6667	-0.3077	0.8462	-0.5385	0.0947	0.7160	0.2899
30	7.6667	-0.0870	0.1739	-0.0870	0.0076	0.0302	0.0076
31	9.6667	0.0345	0.0345	-0.0690	0.0012	0.0012	0.0048
32	11.6667	0.1143	0.3714	-0.4857	0.0131	0.1380	0.2359
33	9.6667	0.2414	-0.2759	0.0345	0.0583	0.0761	0.0012
34	13.6667	0.0244	-0.0488	0.0244	0.0006	0.0024	0.0006
35	8.3333	-0.0400	0.3200	-0.2800	0.0016	0.1024	0.0784
36	11.3333	0.3235	-0.2941	-0.0294	0.1047	0.0865	0.0009
37	12.3333	0.2162	-0.2703	0.0541	0.0467	0.0730	0.0029
38	11.6667	0.0286	0.0286	-0.0571	0.0008	0.0008	0.0033
39	13.0000	0.2308	-0.2308	0.0000	0.0533	0.0533	0.0000
40	14.3333	0.0465	0.1860	-0.2326	0.0022	0.0346	0.0541
41	11.3333	0.0588	0.0588	-0.1176	0.0035	0.0035	0.0138
42	10.3333	0.1613	-0.3226	0.1613	0.0260	0.1041	0.0260
43	11.3333	-0.0294	0.0588	-0.0294	0.0009	0.0035	0.0009
44	11.0000	0.2727	-0.1818	-0.0909	0.0744	0.0331	0.0083
45	14.0000	0.0714	0.1429	-0.2143	0.0051	0.0204	0.0459
				n	*45*	*45*	*45*
				Sum d^2	*1.2028*	*4.0840*	*2.1550*
				Sum d^2/n-1	*0.0273*	*0.0928*	*0.0490*
				sqrt	*0.1653*	*0.3047*	*0.2213*
				s.d.	*0.0246*	*0.0454*	*0.0330*

Figure D.3: Statistics comparing Manual1, Predicted and Manual2 in case-by-case image.

Appendix E

SGmodel code listings

The following figure includes the file names of the system code within the SGmodel structure. The code listings from the main modules are also included below, those pointed with a triangle on the right hand side of the file name.

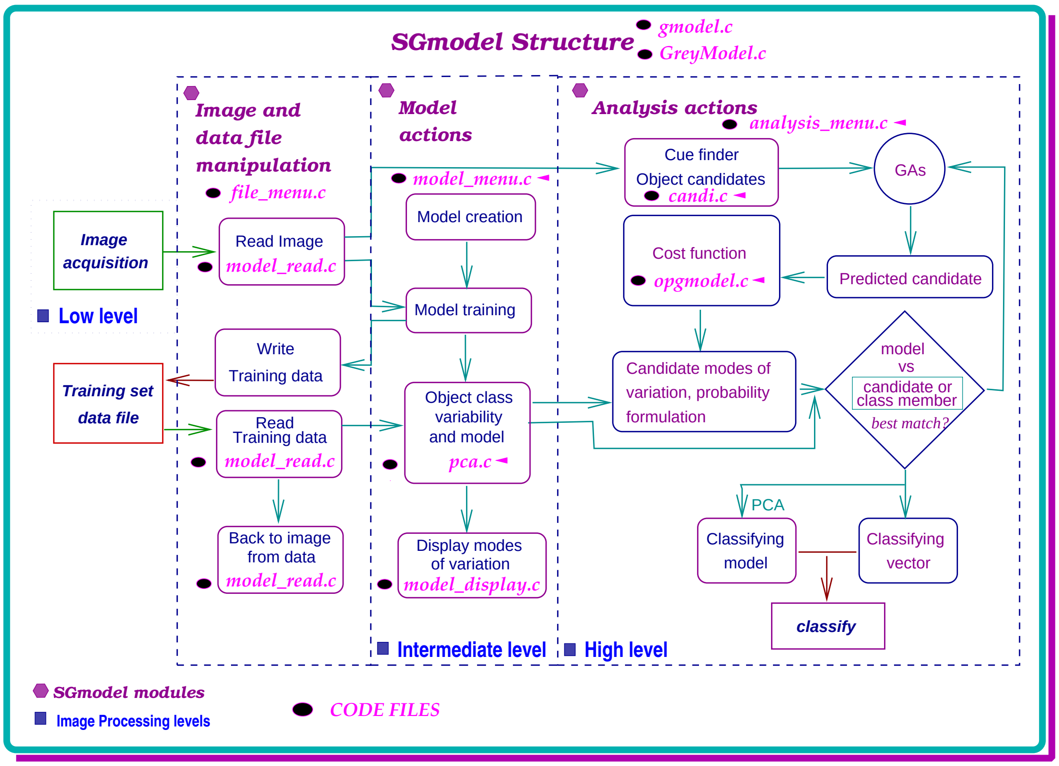

Figure E.1: System code within the SGmodel structure.

E.1 Model menu

```
/************************************************************************
*   Function      : model_menu.c *
*   Last Update   : Thu Mar 20 15:41:08 1997 *
*************************************************************************
*   Synopsis      : create_model, train_model, variation_modes/display *
*************************************************************************
*   Project       : Model Based System for Biomedical Image Analysis     *
*   System Title  : Gmodel  *
*   Authors       : Rocio Aguilar Chongtay and Richard Baldock *
************************************************************************/
static char SccsId[] = "%Z%%M% %I%(%G%) - RACH";
#include <stdio.h>
#include <math.h>

#include <Xm/FileSB.h>
#include <Xm/Form.h>
#include <Xm/Frame.h>
#include <Xm/Label.h>
#include <Xm/PushBG.h>
#include <Xm/RowColumn.h>
#include <Xm/SeparatoG.h>
#include <Xm/Text.h>
#include <Xm/TextF.h>
#include <Xm/ToggleBG.h>

#include <HGU_XmUtils.h>

#include <wstruct.h>
#include <woolz_extern.h>
#include <HGU_XInteract.h>


#include <gmodel.h>
#include <model_menu.h>
#include "error.h"
#include "alloc.h"

/* menu items structures */
static MenuItem model_menu_itemsP[] = { /* model_menu items */
{"create", &xmPushButtonGadgetClass, 0, XmTEAR_OFF_DISABLED,
     NULL, NULL, create_model_cb, NULL},
{"train", &xmPushButtonGadgetClass, 0, XmTEAR_OFF_DISABLED,
     NULL, NULL, train_model_cb, NULL},
{"variation_modes", &xmPushButtonGadgetClass, 0, XmTEAR_OFF_DISABLED,
     NULL, NULL, variation_modes_cb, NULL},
{"display", &xmPushButtonGadgetClass, 0, XmTEAR_OFF_ENABLED,
     NULL, NULL, display_model_cb, NULL},
{"shape_model_simulation", &xmPushButtonGadgetClass, 0, XmTEAR_OFF_DISABLED,
     NULL, NULL, shape_model_simulation_cb , NULL},
    NULL,
};

MenuItem *model_menu_items = &(model_menu_itemsP[0]);

static int def_num_radii=4;
static float def_radii[] = {2.0, 4.0, 8.0, 16.0, 32.0};
static int def_num_angles=6;
static float def_angles[] = {0.0, 60.0, 120.0, 180.0, 240.0, 300.0};
static int def_num_greys=1;
static int def_greys[] = {0};

int create_default_models()
{
    create_neuron6_model();
    create_NOneuron_model();
    create_nocell_sh_model();
}

int create_neuron6_model()
{
    GreyModel *model;
    GREYP greyp;
    int i;
    static float radii[] = {2.0, 4.0, 8.0, 16.0, 32.0, 48.0};
    static float angles[] = {0.0, 60.0, 120.0, 180.0, 240.0, 300.0};
    static int greys[] = {0};

    /* check existing */
    for(i=0, model=NULL; i < globals.num_models; i++){
if( strcmp("neuron6", globals.models[i]->name) == 0 )
    return;
    }

    greyp.inp = def_greys;
    model = GreyModel_make(GMODEL_RADIAL, GMODEL_MODE_G|GMODEL_MODE_R,
   "neuron6", radii, 6, angles, 6, INT_GREY, greyp, 1);

    if( i >= globals.max_num_models ){
globals.max_num_models += 16;
globals.models = (GreyModel **)
    realloc(globals.models, sizeof(GreyModel *) *
    globals.max_num_models);
    }
    globals.models[i] = model;
    if( globals.num_models <= i ) globals.num_models = i+1;

    return;
}

int create_NOneuron_model()
{
    GreyModel *model;
    GREYP greyp;
    int i;
    static float radii[] = {2.0, 4.0, 6.0, 8.0, 14.0, 20.0};
    static float angles[] = {0.0, 60.0, 120.0, 180.0, 240.0, 300.0};
    static int greys[] = {0};

    /* check existing */
    for(i=0, model=NULL; i < globals.num_models; i++){
if( strcmp("NOneuron", globals.models[i]->name) == 0 )
    return;
    }

    greyp.inp = def_greys;
```

```
    model = GreyModel_make(GMODEL_RADIAL, GMODEL_MODE_G|GMODEL_MODE_R,
   "NOneuron", radii, 6, angles, 6, INT_GREY, greyp, 1);

    if( i >= globals.max_num_models ){
globals.max_num_models += 16;
globals.models = (GreyModel **)
    realloc(globals.models, sizeof(GreyModel *) *
    globals.max_num_models);
    }
    globals.models[i] = model;
    if( globals.num_models <= i ) globals.num_models = i+1;

    return;
}

 int create_nocell_sh_model()
{
    GreyModel *model;
    GREYP greyp;
    int i;
    static float radii[] = {4.0, 8.0, 16.0};
    static float angles[] = {0.0, 20.0, 40.0, 60.0, 80.0, 100.0,
                                    120.0, 140.0, 160.0, 180.0, 200.0,
                                    220.0, 240.0, 260.0, 280.0, 300.0,
                                    320.0, 340.0};
    static int greys[] = {0};

    /* check existing */
    for(i=0, model=NULL; i < globals.num_models; i++){
if( strcmp("nocell_sh", globals.models[i]->name) == 0 )
    return;
    }

    greyp.inp = def_greys;
    model = GreyModel_make(GMODEL_RADIAL, GMODEL_MODE_G|GMODEL_MODE_R,
   "nocell_sh", radii, 3, angles, 18, INT_GREY, greyp, 1);

    if( i >= globals.max_num_models ){
globals.max_num_models += 16;
globals.models = (GreyModel **)
    realloc(globals.models, sizeof(GreyModel *) *
    globals.max_num_models);
    }
    globals.models[i] = model;
    if( globals.num_models <= i ) globals.num_models = i+1;

    return;
}

static void set_radii( text, model, cbs )
Widget text;
GreyModel *model;
XmAnyCallbackStruct *cbs;
{
    String str=NULL, str1;
    int i, sscanf_ret;
    float test_radius;

    if( !XtIsSubclass(text, xmTextWidgetClass) &&
       !XtIsSubclass(text, xmTextFieldWidgetClass) )
return;

    XtVaGetValues(text, XmNvalue, &str, NULL);
    if( str == NULL )
return;

    str1 = (String) malloc(sizeof(char) * (strlen(str) + 2));
    strcpy( str1, str );

    if( model->radii == NULL ){
model->num_radii = 10;
model->radii = (float *) malloc(sizeof(float) * model->num_radii);
    }

    i = 0;
    while( (sscanf_ret = sscanf(str1, "%f %[^,\n]", &test_radius, str1)) > 0 ){
if( i >= model->num_radii ){
    model->num_radii += 10;
    model->radii = (float *) realloc(model->radii,
     sizeof(float)*model->num_radii);
}
model->radii[i] = test_radius;
i++;
if( sscanf_ret == 1 )
    break;
    }
    model->num_radii = i;

    free( str1 );
}

static void set_angles( text, model, cbs )
Widget text;
GreyModel *model;
XmAnyCallbackStruct *cbs;
{
    String str=NULL, str1;
    int i, sscanf_ret;
    float test_angle;

    if( !XtIsSubclass(text, xmTextWidgetClass) &&
       !XtIsSubclass(text, xmTextFieldWidgetClass) )
return;

    XtVaGetValues(text, XmNvalue, &str, NULL);
    if( str == NULL )
return;

    str1 = (String) malloc(sizeof(char) * (strlen(str) + 2));
    strcpy( str1, str );

    if( model->angles == NULL ){
model->num_angles = 10;
model->angles = (float *) malloc(sizeof(float) * model->num_angles);
    }

    i = 0;
    while( (sscanf_ret = sscanf(str1, "%f %[^,\n]", &test_angle, str1)) > 0 ){
```

```
if( i >= model->num_angles ){
    model->num_angles += 10;
    model->angles = (float *) realloc(model->angles,
      sizeof(float)*model->num_angles);
}
model->angles[i] = test_angle;
i++;
if( sscanf_ret == 1 )
    break;
    }
    model->num_angles = i;

    free( str1 );
}

static void set_greys( text, model, cbs )
Widget text;
GreyModel *model;
XmAnyCallbackStruct *cbs;
{
    String str=NULL, str1;
    int i, sscanf_ret, test_grey;

    if( !XtIsSubclass(text, xmTextWidgetClass) &&
       !XtIsSubclass(text, xmTextFieldWidgetClass) )
return;

    XtVaGetValues(text, XmNvalue, &str, NULL);
    if( str == NULL )
return;

    str1 = (String) malloc(sizeof(char) * (strlen(str) + 2));
    strcpy( str1, str );

    if( model->greys.inp == NULL ){
model->num_greys = 10;
model->greys.inp = (int *) malloc(sizeof(int) * model->num_greys);
    }

    i = 0;
    while( (sscanf_ret = sscanf(str1, "%d %[^,\n]", &test_grey, str1)) > 0 ){
if( i >= model->num_greys ){
    model->num_greys += 10;
    model->greys.inp = (int *) realloc(model->greys.inp,
       sizeof(int)*model->num_greys);
}
model->greys.inp[i] = test_grey;
i++;
if( sscanf_ret == 1 )
    break;
    }
    model->num_greys = i;
    free( str1 );
}

static void set_radial_mode_cb( w, model, cbs )
Widget w;
GreyModel *model;
XmToggleButtonCallbackStruct *cbs;
{
    if( cbs->set )
model->mode |= GMODEL_MODE_R;
    else
model->mode &= ~GMODEL_MODE_R;
}

static void set_angles_mode_cb( w, model, cbs )
Widget w;
GreyModel *model;
XmToggleButtonCallbackStruct *cbs;
{
    if( cbs->set )
model->mode |= GMODEL_MODE_A;
    else
model->mode &= ~GMODEL_MODE_A;
}

static void set_greys_mode_cb( w, model, cbs )
Widget w;
GreyModel *model;
XmToggleButtonCallbackStruct *cbs;
{
    if( cbs->set )
model->mode |= GMODEL_MODE_G;
    else
model->mode &= ~GMODEL_MODE_G;
}

static void set_sensitive_cb( w, widget, cbs )
Widget w;
Widget widget;
XmToggleButtonCallbackStruct *cbs;
{
    if( cbs->set )
XtSetSensitive(widget, False);
    else
XtSetSensitive(widget, True);
}

static Widget create_model_dialog( topl, model )
Widget topl;
GreyModel *model;
{
    Widget dialog, control, widget, widg, frame, checkbar;
    char str[256];
    int i;

    dialog = HGU_XmCreateStdDialog(topl, "create_model_dialog",
                                   xmFormWidgetClass, NULL, 0);

    control = XtNameToWidget( dialog, "*control" );

   frame = XtVaCreateManagedWidget("frame", xmFrameWidgetClass, control,
                                    XmNshadowType,      XmSHADOW_ETCHED_IN,
                                    NULL);

    checkbar = XtVaCreateManagedWidget("checkbar", xmFormWidgetClass, frame,
                                       XmNfractionBase, 3,
```

```
                                        XmNborderWidth, 0,
                                        NULL);
    str[0] = '\0';
    for(i=0; i < model->num_radii; i++)
sprintf( str, "%s%.1f ", str, model->radii[i] );
    widget = HGU_XmCreateTextLine( "radii", control, str,
  set_radii, (XtPointer) model );
    (void) XtVaSetValues(widget,
 XmNtopAttachment, XmATTACH_FORM,
 XmNleftAttachment, XmATTACH_FORM,
 XmNrightAttachment, XmATTACH_FORM,
 NULL);
    widg = widget;
    if( model->mode&GMODEL_MODE_R ){
XtSetSensitive( widg, False );
    }

    widget = XtVaCreateManagedWidget("radial_mode", xmToggleButtonGadgetClass,
     checkbar,
     XmNtopAttachment, XmATTACH_FORM,
     XmNbottomAttachment,XmATTACH_FORM,
     XmNleftAttachment, XmATTACH_POSITION,
     XmNrightAttachment,XmATTACH_POSITION,
     XmNleftPosition, 0,
     XmNrightPosition, 1,
     NULL);
    XtAddCallback(widget, XmNvalueChangedCallback, set_radial_mode_cb,
  (XtPointer) model );
    XtAddCallback(widget, XmNvalueChangedCallback, set_sensitive_cb,
  (XtPointer) widg );
    if( model->mode&GMODEL_MODE_R ){
XmToggleButtonSetState( widget, True, False );
    }

    str[0] = '\0';
    for(i=0; i < model->num_angles; i++)
sprintf( str, "%s%.1f ", str, model->angles[i] );
    widget = HGU_XmCreateTextLine( "angles", control, str,
  set_angles, (XtPointer) model );
    (void) XtVaSetValues(widget,
 XmNtopAttachment, XmATTACH_WIDGET,
 XmNtopWidget, widg,
 XmNleftAttachment, XmATTACH_FORM,
 XmNrightAttachment, XmATTACH_FORM,
 NULL);
    widg = widget;
    if( model->mode&GMODEL_MODE_A ){
XtSetSensitive( widg, False );
    }

    widget = XtVaCreateManagedWidget("angles_mode", xmToggleButtonGadgetClass,
     checkbar,
     XmNtopAttachment, XmATTACH_FORM,
     XmNbottomAttachment,XmATTACH_FORM,
     XmNleftAttachment, XmATTACH_POSITION,
     XmNrightAttachment,XmATTACH_POSITION,
     XmNleftPosition, 1,
     XmNrightPosition, 2,
     NULL);
    XtAddCallback(widget, XmNvalueChangedCallback, set_angles_mode_cb,
  (XtPointer) model );
    XtAddCallback(widget, XmNvalueChangedCallback, set_sensitive_cb,
  (XtPointer) widg );
    if( model->mode&GMODEL_MODE_A ){
XmToggleButtonSetState( widget, True, False );
    }

    str[0] = '\0';
    for(i=0; i < model->num_greys; i++)
sprintf( str, "%s%d ", str, model->greys.inp[i] );
    widget = HGU_XmCreateTextLine( "greys", control, str,
  set_greys, (XtPointer) model );
    (void) XtVaSetValues(widget,
 XmNtopAttachment, XmATTACH_WIDGET,
 XmNtopWidget, widg,
 XmNleftAttachment, XmATTACH_FORM,
 XmNrightAttachment, XmATTACH_FORM,
 NULL);
    widg = widget;
    if( model->mode&GMODEL_MODE_G ){
XtSetSensitive( widg, False );
    }

    widget = XtVaCreateManagedWidget("greys_mode", xmToggleButtonGadgetClass,
     checkbar,
     XmNtopAttachment, XmATTACH_FORM,
     XmNbottomAttachment,XmATTACH_FORM,
     XmNleftAttachment, XmATTACH_POSITION,
     XmNrightAttachment,XmATTACH_POSITION,
     XmNleftPosition, 2,
     XmNrightPosition, 3,
     NULL);
    XtAddCallback(widget, XmNvalueChangedCallback, set_greys_mode_cb,
  (XtPointer) model );
    XtAddCallback(widget, XmNvalueChangedCallback, set_sensitive_cb,
  (XtPointer) widg );
    if( model->mode&GMODEL_MODE_G ){
XmToggleButtonSetState( widget, True, False );
    }

    (void) XtVaSetValues(frame,
 XmNtopAttachment, XmATTACH_WIDGET,
 XmNtopWidget, widg,
 XmNleftAttachment, XmATTACH_FORM,
 XmNrightAttachment, XmATTACH_FORM,
 NULL);

    XtManageChild( dialog );
    return( dialog );
}

/* action and callback procedures */
void create_model_cb(w, client_data, cbs)
```

```
Widget w;
XtPointer client_data;
XmAnyCallbackStruct *cbs;
{
    Widget dialog;
    String name, title;
    GreyModel *model;
    GREYP greyp;
    int i;

    /* get the model name */
    if( (name = HGU_XmUserGetstr(globals.topl,
 "Type in the model name:", "Ok", "Cancel",
 NULL)) == NULL )
return;
    for(i=0; i < strlen(name); i++)
name[i] = (name[i] == ' ') ? '_' : name[i];

    /* check existing */
    for(i=0, model=NULL; i < globals.num_models; i++){
if( strcmp(name, globals.models[i]->name) == 0 )
    model = globals.models[i];
    }
    if( model == NULL ){
greyp.inp = def_greys;
model = GreyModel_make(GMODEL_RADIAL, GMODEL_MODE_G|GMODEL_MODE_R,
       name,
       def_radii, def_num_radii,
       def_angles, def_num_angles,
       INT_GREY, greyp, def_num_greys);
    }
    if( i >= globals.max_num_models ){
globals.max_num_models += 16;
globals.models = (GreyModel **)
    realloc(globals.models, sizeof(GreyModel *) *
    globals.max_num_models);
    }
    globals.models[i] = model;
    if( globals.num_models <= i ) globals.num_models = i+1;

    /* get structural parameters */
    dialog = create_model_dialog( globals.topl, model );

    title = (String) malloc(sizeof(char) * (strlen(name) + 32));
    sprintf( title, "%s model parameters", name );
    XtVaSetValues( dialog, XmNtitle, title, NULL );

    (void) HGU_XmDialogConfirm(dialog, NULL, NULL,
       XmDIALOG_FULL_APPLICATION_MODAL );
    XtDestroyWidget( dialog );

    free( title );
    XFree( name );

    return;
}

float vtx_dist(type, v1, v2)
int type;
struct ivertex *v1;
struct ivertex *v2;
{
    float dist;
    double dx, dy;
    struct fvertex *fv1, *fv2;

    switch( type ){
    default:
    case 1:
dx = v1->vtX - v2->vtX;
dy = v1->vtY - v2->vtY;
break;
    case 2:
fv1 = (struct fvertex *) v1;
fv2 = (struct fvertex *) v2;
dx = fv1->vtX - fv2->vtX;
dy = fv1->vtY - fv2->vtY;
break;
    }

    dist = sqrt( dx*dx + dy*dy );
    return( dist );
}

int display_instance_marker( dpy, win, gc, gminst, intens, pf )
Display *dpy;
Window win;
GC gc;
GreyModelInstance *gminst;
int intens;
struct pframe *pf;
{
    struct ivertex ivtx;
    char str[32];

    ivtx.vtX = gminst->centre.vtX;
    ivtx.vtY = gminst->centre.vtY;
    sprintf(str, "(%.2f, %.2f)", gminst->centre.vtX, gminst->centre.vtY );

    (void) intens_GC( dpy, gc, intens );
    picvtx_X( dpy, win, gc, &ivtx, pf );
    picstr_X( dpy, win, gc, gminst->model->name, &ivtx, pf );
    pf->dy -= 96;
    picstr_X( dpy, win, gc, str, &ivtx, pf );
    pf->dy += 96;
    (void) intens_GC( dpy, gc, 1 );

}

/* action procedure to create a model instance */
static void get_model_instance( w, event, params, num_params )
Widget w;
XEvent *event;
String *params;
Cardinal num_params;
{
    Display *dpy = XtDisplay( w );
    Window win = XtWindow( w );
```

```
    GreyModel *model;
    int i;
    struct object *start_obj, *obj;
    struct polygondomain *polydmn;
    struct fvertex vtx, *start_vtx, *new_vtx, *fvtx;
    struct ivertex ivtx;
    GreyModelInstance *gminst;
    int a, r, g, k;
    float **feats;
    GREYP greyp;
    TrainingSet *tr_set;
    Window mag_win;
    struct pframe mag_pf;

    fprintf(stderr,"get_model_instance: create %s instance\n",params[0]);
    /* get the model structure */
    for(i=0, model=NULL; i < globals.num_models; i++){
if( strcmp(params[0], globals.models[i]->name) == 0 )
    model = globals.models[i];
    }
    if( model == NULL ){
fprintf(stderr, "get_model_instance: %s - no such model\n");
return;
    }

    /* get the training set */
    for(i=0, tr_set=NULL; i < globals.num_tr_sets; i++){
if( strcmp(model->name, globals.tr_sets[i].model_name) == 0 )
    tr_set = &(globals.tr_sets[i]);
    }

    /* get the centre vertex and check for existing instance */
    vtx.vtX = event->xbutton.x;
    vtx.vtY = event->xbutton.y;
    start_vtx = vtx_X_to_w( dpy, win, &vtx, globals.pf );
    gminst = NULL;
    if( tr_set != NULL ){
for(i=0; i < tr_set->num_data; i++){
    if( strcmp(tr_set->data[i]->filename, globals.file) )
continue;
    if( vtx_dist(2, start_vtx, tr_set->data[i]->centre) < 10.0 ){
int j;
ivtx.vtX = tr_set->data[i]->centre.vtX;
ivtx.vtY = tr_set->data[i]->centre.vtY;
gminst = tr_set->data[i];
display_instance_marker(dpy, win, globals.gc_ovly,
gminst, 0, globals.pf );
float2free( gminst->feats );

for(j=i+1; j < tr_set->num_data; j++)
    tr_set->data[j-1] = tr_set->data[j];
tr_set->num_data--;
break;
    }
}
    }

    /* get a magnified  window */
    mag_win = HGU_XCreateMagWin(dpy, win, event->xbutton.x,
 event->xbutton.y, 128, 128, 6, 0);
    mag_pf = *(globals.pf);
    mag_pf.scale *= 6;
    mag_pf.dx = 128*8*6/2;
    mag_pf.dy = 128*8*6/2;
    mag_pf.ox = start_vtx->vtX;
    mag_pf.oy = start_vtx->vtY;

    /* get actual centre and direction */
    polydmn = makepolydmn(2, NULL, 0, 2, 1);
    polydmn->nvertices = 1;
    ((struct fvertex *) polydmn->vtx)[0] = *start_vtx;
    start_obj = makemain(10, polydmn, NULL, NULL, NULL);
    obj = interact_poly_X(dpy, mag_win, 0, 0, &mag_pf, start_obj);
    freeobj( start_obj );
    free( start_vtx );

    if( obj == NULL ){
(void) HGU_XDestroyWindow( dpy, mag_win );
return;
    }

    if( ((struct polygondomain *) obj->idom)->nvertices < 2 ){
(void) HGU_XDestroyWindow( dpy, mag_win );
freeobj( obj );
return;
    }
    polydmn = (struct polygondomain *) obj->idom;

    /* create an instance record */
    if( gminst == NULL )
gminst = (GreyModelInstance *) malloc(sizeof(GreyModelInstance));
    gminst->type = model->type;
    gminst->linkcount = 0;
    if( globals.file != NULL ){
gminst->filename = (char *) malloc(sizeof(char) *
   (strlen(globals.file) + 2) );
strcpy(gminst->filename, globals.file);
    } else {
gminst->filename = NULL;
    }
    gminst->model = model;
    model->linkcount++;
    fvtx = (struct fvertex *) polydmn->vtx;
    gminst->centre = *fvtx;
    gminst->angle = (float) atan2((double) (fvtx[1].vtY - fvtx[0].vtY),
  (double) (fvtx[1].vtX - fvtx[0].vtX));
    if( globals.obj != NULL ){
gminst->grey_type = (globals.obj->vdom->type) % 10;
greyp.inp = greyval(globals.obj, (int) fvtx[0].vtY, (int) fvtx[0].vtX);
switch( gminst->grey_type ){
default:
```

```
case INT_GREY:
    gminst->centre_grey.inv = *greyp.inp; break;
case SHORT_GREY:
    gminst->centre_grey.shv = *greyp.shp; break;
case UBYTE_GREY:
    gminst->centre_grey.ubv = *greyp.ubp; break;
case FLOAT_GREY:
    gminst->centre_grey.flv = *greyp.flp; break;
}
    }
    if(model->num_greys != 0)
       float2alloc( &(gminst->feats),
                model->num_angles*model->num_radii*model->num_greys, 3 );
    else
       float2alloc( &(gminst->feats),
                model->num_angles*model->num_radii, 3 );

    feats = gminst->feats;

    /* display each radial line if required,
       put up a text message showing feature index and other info */
    ivtx.vtX = gminst->centre.vtX;
    ivtx.vtY = gminst->centre.vtY;
    picvtx_X( dpy, mag_win, globals.gc_ovly, &ivtx, &mag_pf );
    display_instance_marker(dpy, win, globals.gc_ovly,
    gminst, 1, globals.pf );

    for(a=0, k=0; a < model->num_angles; a++){
/* display the radial line */
float radius;
double theta;
if( !(model->mode&GMODEL_MODE_A) ){
    theta = gminst->angle + model->angles[a]*M_PI/180;
}
fvtx[1].vtX = fvtx[0].vtX + (50.0 * cos(theta));
fvtx[1].vtY = fvtx[0].vtY + (50.0 * sin(theta));
polydmn->nvertices = 2;
picframe_X(dpy, mag_win, globals.gc_ovly, obj, &mag_pf);

for(r=0, radius=0.0; r < model->num_radii; r++){
    /* display radial positions */
    radius += model->radii[r] - (r?model->radii[r-1]:0.0);

    if( model->mode&GMODEL_MODE_R )

      if( strcmp("NOneuron", model->name) == 0 ){
vtx.vtX = gminst->centre.vtX + radius*cos(theta);
                vtx.vtY = gminst->centre.vtY + radius*sin(theta);
        }
    else{
vtx.vtX = gminst->centre.vtX + radius*cos(theta);
vtx.vtY = gminst->centre.vtY + radius*sin(theta);
start_vtx = interact_fvtx_X(dpy, mag_win, 0, 0, &mag_pf, &vtx);
if( start_vtx == NULL ){
    /* should free space */
    (void) HGU_XDestroyWindow( dpy, mag_win );
    return;
}
radius = vtx_dist(2, &(gminst->centre), start_vtx);
                                vtx.vtX = start_vtx->vtX;
                                vtx.vtY = start_vtx->vtY;
free( start_vtx );
    }

    ivtx.vtX = gminst->centre.vtX + radius * cos( theta );
    ivtx.vtY = gminst->centre.vtY + radius * sin( theta );
    picvtx_X( dpy, mag_win, globals.gc_ovly, &ivtx, &mag_pf );

    for(g=0; g < model->num_greys; g++, k++){
/* get vertex and grey value */
feats[k][0] = theta - gminst->angle;
feats[k][1] = radius;
if( globals.obj ){
    int         g4[4];
    float dx;
                                  float      dy;
    ivtx.vtX = vtx.vtX;
    ivtx.vtY = vtx.vtY;
    grey4val(globals.obj, ivtx.vtY, ivtx.vtX, &(g4[0]));
    dx = vtx.vtX - ivtx.vtX;
    dy = vtx.vtY - ivtx.vtY;
    feats[k][2]  = ((1-dy) * ((1-dx)*g4[0] + dx*g4[1]) +
    dy * ((1-dx)*g4[2] + dx*g4[3]));

} else {
    feats[k][2] = 0.0;
}

    }
}
    }
    (void) HGU_XDestroyWindow(dpy, mag_win);

    /* add to training set */
    if( tr_set == NULL ){
if( i >= globals.max_num_tr_sets ){
    globals.max_num_tr_sets += 16;
    globals.tr_sets = (TrainingSet *)
realloc(globals.tr_sets, sizeof(TrainingSet) *
globals.max_num_tr_sets );
}
tr_set = &(globals.tr_sets[i]);
globals.num_tr_sets = i+1;
tr_set->type = model->type;
tr_set->linkcount = 0;
tr_set->model_name = model->name;
tr_set->num_data = 0;
tr_set->max_num_data = 64;
tr_set->data = (GreyModelInstance **)
    malloc(sizeof(GreyModelInstance *) * tr_set->max_num_data );
    }

    if( tr_set->num_data >= tr_set->max_num_data ){
tr_set->max_num_data += 64;
tr_set->data = (GreyModelInstance **)
```

```
    realloc(tr_set->data, sizeof(GreyModelInstance *) *
    tr_set->max_num_data );
    }
    tr_set->data[tr_set->num_data] = gminst;
    tr_set->num_data++;
}

static XtActionsRec get_model_actions[] = {
{"get_model_instance", get_model_instance},
};

static void install_get_model_translations( widget, model )
Widget widget;
GreyModel *model;
{
    XtTranslations translations;
    static int action_proc_added=0;
    String translations_table;

    /* install the get_model_instance action procedure */
    if( !action_proc_added ){
XtAppAddActions(globals.app_con, get_model_actions,
XtNumber(get_model_actions));
action_proc_added = 1;
    }

    /* install the translation table on the work_area */
    translations_table = (String) malloc((strlen(model->name) + 100) *
 sizeof(char) );
    sprintf(translations_table, "<Btn1Down>,<Btn1Up>: get_model_instance(%s)",
    model->name);
    translations = XtParseTranslationTable( translations_table );
    XtOverrideTranslations( globals.canvas, translations );
    free( translations_table );
}

void model_create_instance_cb(w, model, cbs)
Widget w;
GreyModel *model;
XmToggleButtonCallbackStruct *cbs;
{
    if( cbs->set != True )
return;

    install_get_model_translations( globals.canvas, model );
}

static Widget create_model_selection_panel( topl, models, num_models )
Widget topl;
GreyModel **models;
int num_models;
{
    Widget dialog, control, widget;
    MenuItem *items;
    int i;

    dialog = HGU_XmCreateStdDialog(topl, "create_model_selection_dialog",
                                   xmFormWidgetClass, NULL, 0);
    XtVaSetValues(dialog, XmNtitle, "Select model for training", NULL);

    control = XtNameToWidget( dialog, "*control" );

    items = (MenuItem *) malloc(sizeof(MenuItem) * (num_models+1) );
    for(i=0; i < num_models; i++){
items[i].name          = models[i]->name;
items[i].mnemonic      = 0;
items[i].accelerator   = NULL;
items[i].accel_text    = NULL;
items[i].callback      = model_create_instance_cb;
items[i].callback_data = (XtPointer) models[i];
    }
    items[i].name = NULL;

    HGU_XmCreatePB_Radio( "select_model", control, items, -1 );

    widget = XtNameToWidget(dialog, "*Ok");
    XtAddCallback( widget, XmNactivateCallback, UninstallTranslationsCallback,
  (XtPointer) globals.canvas);

    widget = XtNameToWidget(dialog, "*Cancel");
    XtAddCallback( widget, XmNactivateCallback, UninstallTranslationsCallback,
  (XtPointer) globals.canvas);

    XtManageChild( dialog );
    return( dialog );
}

void train_model_cb(w, client_data, cbs)
Widget w;
XtPointer client_data;
XmAnyCallbackStruct *cbs;
{
    Widget dialog, widget;

    /* set up selection panel */
    dialog = create_model_selection_panel(globals.topl, globals.models,
  globals.num_models );

    widget = XtNameToWidget(dialog, "*Ok");
    XtAddCallback(widget, XmNactivateCallback,
  SetSensitiveCallback, XtParent(w) );
    XtAddCallback(widget, XmNactivateCallback, DestroyWidgetCallback,
  XtParent(dialog) );

    widget = XtNameToWidget(dialog, "*Cancel");
    XtAddCallback(widget, XmNactivateCallback,
  SetSensitiveCallback, XtParent(w) );
    XtAddCallback(widget, XmNactivateCallback, DestroyWidgetCallback,
  XtParent(dialog) );

    XtSetSensitive(XtParent( w ), False );

    XtManageChild( XtParent(dialog) );

    return;
}

void shape_model_simulation_cb(w, client_data, cbs)
```

```
Widget w;
XtPointer client_data;
XmAnyCallbackStruct *cbs;
{
    char       str[128];
    String       filename, model_name;
    static String     oldfile = NULL;
    FILE       *fp;
    int       i, j,k,o, count;
    int              num_data, num_angles=18, num_rad=3, num_feats, greyt=3;
    float            p,q;
    float            coX, coY, cX, cY; /* offset coordinates */
                     /* nucleous radius (R), nucleolous radius (r) and */
             /* distance between offset and nucleolous centre point (d) */
    float            Rx, Ry, Rmin, Rmax, Rnoise, r, rmin, rmax, d;
    float            x, y, xmax, xmin;
    float            angle=0, *angles;
    float            gr1=100, gr2=0, gr3=0, no_pos;
    float            *feats, temp, tanthe, a, b, c, kons=100;
    double           theta;
    /* get the filename */
    model_name = "nocell_sh";
    sprintf(str, "Input filename for \"%s\"\n    training data", model_name);
    if( oldfile == NULL ){
oldfile = (String) malloc( sizeof(char)*16 );
(void) sprintf( oldfile, "%s.tr", model_name );
    }
    filename = HGU_XmUserGetstr( globals.topl, str, "Ok", "Cancel", oldfile );
    if( filename == NULL )
return;
    free( oldfile );
    oldfile = filename;

    /* write the data */
if( (fp = fopen(filename, "w")) == NULL ){
    sprintf(str, "Can't open file:\n    %s", filename);
    HGU_XmUserInfo(globals.topl,str,XmDIALOG_FULL_APPLICATION_MODAL);
    return;
}
    IfErr (angles = new_array_of(num_angles, FLOAT)){
        fprintf(stderr, "not enough core for angles array\n");
        exit(1);
    }
/* write_shape_set */
    fprintf(fp, "Training set for model: %s\n", model_name);
    fprintf(fp, "num_radii: %d\nradii:", num_rad);
    fprintf(fp, " variable");
    fprintf(fp, "\n" );
    fprintf(fp, "num_angles: %d\nangles:", num_angles);
    num_feats = 2*num_rad*num_angles+1;
    for (i=0; i< num_angles; i++){
       angles[i]= angle;
       fprintf(fp, " %f", angles[i] );
       angle+=360/num_angles;
     }
    fprintf(fp, "\n");
    fprintf(fp, "num_greys: 1\n" );
    fprintf(fp, "greys: variable\n" );
    coX = 500.0;
    coY = 500.0;
    Rmin = 5.0; Rmax = 20.0;
    rmin = 1.0; rmax = Rmin;
    d=0.0;
    num_data = 2*(Rmax-Rmin)*2*(rmax-rmin);
                      /* last Rmin is the range of distances
               between the offset and the nucleous centre point */
    fprintf(fp, "numData: %d\n", num_data );
    for(p=Rmin; p<Rmax; p+=0.5){
       for(q=rmin; q<rmax; q+=0.5){
             fprintf(fp, "filename: simulated_shapes\n");
             r=q; Ry=p; Rx=1.5*Ry;
/* adding some random noise to Rx and Ry */
             Rnoise = rand()/100000000;
             if(Rnoise<0) Rnoise= 0.0;
             if(Rnoise>(Ry*0.40)) Rnoise=Ry*0.40;
             Ry+=Rnoise;
        fprintf(stderr, "noise Ry = %f, Ry = %f\n", Rnoise, Ry);
             Rnoise = rand()/100000000;
             if(Rnoise<0) Rnoise= 0.0;
             if(Rnoise>(Rx*0.40)) Rnoise=Rx*0.40;
             Rx+=Rnoise;
        fprintf(stderr, "noise Rx = %f, Rx = %f\n", Rnoise, Rx);
/* end random to Rx and Ry */
             cX = coX;
             cY = coY;
             fprintf(fp, "centreXY: %f %f\n", cX, cY);
             fprintf(fp, "angle: 0.0\n");
             fprintf(fp, "grey type: %d\n", greyt);
             fprintf(fp, "begin features:\n" );
             gr3 = rand()/10000000; /* non-negative random value */
             if(gr3>256) gr3= 256.0;
             if(gr3<0) gr3 = 0.0;
             gr1 = rand()/10000000;
             if(gr1 > 256) gr1 = 256.0;
             if(gr1 < 100) gr1 = 100.0;
             gr2 = rand()/10000000;
             if(gr2 > 100) gr2 = 100.0;
             no_pos = 0;
             fprintf(fp, " %6.3f",gr1); /* nucleolous centre point */
             for(j=0; j < num_angles; j++){
                  theta = M_PI*angles[j]/180; /* convrt deg -> rad */
                  tanthe = tan(theta);
                  temp=(Rx*Rx)*tanthe*tanthe;
  a=Ry*Ry+temp;
                  b=-2*d*temp;
                  c=temp*d*d-(Rx*Rx*Ry*Ry);
                  temp=b*b-(4*a*c);
```

```
  if((angles[j] >90)&&(angles[j] < 270)){
                    xmin=(-b-sqrt(temp))/(2*a);
                    x=xmin;
   }
                  else{
                    xmax=(-b+sqrt(temp))/(2*a);
                    x=xmax;
   }
                 y= tanthe*(x-d);
                 temp=y*y+((x-d)*(x-d));
                 temp=sqrt(temp);
                 fprintf(fp, " %6.3f", r);
                 fprintf(fp, " %6.3f", gr1);
                 fprintf(fp, " %6.3f", temp);
                 fprintf(fp, " %6.3f", gr2);
                 fprintf(fp, " %6.3f", temp);
                 fprintf(fp, " %6.3f", gr3);
            }
            if(Rx-d > r) d += 0.5;
            else d = 0;
            fprintf(fp, "\n");
            fprintf(fp, "end features\n");
        }
      }
    fclose( fp );
}
```

E.2 PCA

```
/*************************************************************************
*   Function     : pca.c *
*   Last Update  : Thu Mar 20 15:41:08 1997 *
**************************************************************************
*   Synopsis     : Performs  Principal Components Analisis (PCA) *
*                  Based on the public Stolcke's PCA program *
*    Model instances generator (MIG model_display funct), *
*    new instances of the Grey_level model are generated  *
*    using the main modes of variation obtained with PCA *
**************************************************************************
*   Project      : Model Based System for Biomedical Image Analysis     *
*   System Title : Gmodel  *
*   Authors      : Rocio Aguilar Chongtay and Richard Baldock *
*************************************************************************/
static char SccsId[] = "%Z%%M% %I%(%G%) - RACH";
#include <stdio.h>
#include <math.h>
#include <string.h>

#include <gmodel.h>

#include <alloc.h>
#include <error.h>
#include <pca.h>



/*
 * computational subroutines for PCA
 */
static int
covariance(vecs, m, cov, n, meanv, stdev)
    FLOAT **vecs, **cov, *meanv, *stdev;
    int     m, n;
{
    FLOAT   *svar;
    int     i, j, k;

    IfErr (svar = new_array_of(n, FLOAT))
Erreturn("not enough core");

    /* compute means */


    for (i = 0; i < n; i++) {
        FLOAT sum = 0.0;
int   l = 0;

for (k = 0; k < m; k++) {
#ifndef NO_DONTCARES
    if ( !IS_DC(vecs[k][i]) )
#endif
    {
sum += vecs[k][i];
l += 1;
    }
}

meanv[i] = sum / (l != 0 ? l : 1);

#ifndef NO_DONTCARES
```

```
/* replace all D/C's my the mean on that dimension */
for (k = 0; k < m; k++)
    if ( IS_DC(vecs[k][i]) )
vecs[k][i] = meanv[i];
#endif

    }

/* calculating std */

    for (i = 0; i < n; i++) {
        FLOAT sumdev = 0.0;
        for (k = 0; k < m; k++) {
            sumdev = (vecs[k][i]-meanv[i]);
            svar[i] += sumdev*sumdev;
        }
        svar[i] /= (m-1);
        stdev[i] = sqrt(svar[i]);fflush(stdout);
    }


    /* compute covariance */
    for (i = 0; i < n; i++)
for (j = 0; j <= i; j++)
    cov[i][j] = 0.0;

    for (k = 0; k < m; k++)
for (i = 0; i < n; i++)
    for (j = 0; j <= i; j++)
cov[i][j] += (vecs[k][i] - meanv[i]) *
    (vecs[k][j] - meanv[j]);

    for (i = 0; i < n; i++) {
for (j = 0; j < i; j++) {
    cov[i][j] /= m;
    cov[j][i] = cov[i][j];


}
cov[i][i] /= m;
    }
    free(svar);
    return MY_OK;
}

static int
jacobi(a, n, d, v, nrot)
    FLOAT **a, d[];
    double **v;
    int     n, *nrot;
{
    int     j, iq, ip, i;
    FLOAT   *b, *z;

    IfErr (b = new_array_of(n, FLOAT))
Erreturn("not enough core");
    IfErr (z = new_array_of(n, FLOAT))
Erreturn("not enough core");

    for (ip = 0; ip < n; ip++) {
for (iq = 0; iq < n; iq++)
    v[ip][iq] = 0.0;
v[ip][ip] = 1.0;
    }

    for (ip = 0; ip < n; ip++) {
b[ip] = d[ip] = a[ip][ip];
z[ip] = 0.0;
    }

    *nrot = 0;
    for (i = 1; i <= 50; i++) {
FLOAT tresh;
FLOAT sm = 0.0;

for (ip = 0; ip < n - 1; ip++) {
    for (iq = ip + 1; iq < n; iq++)
sm += fabs(a[ip][iq]);
}

if (sm == 0.0) {
    free(z);
    free(b);
    return MY_OK;
}

if (i < 4)
    tresh = 0.2 * sm / (n * n);
else
    tresh = 0.0;

for (ip = 0; ip < n - 1; ip++) {
```

```
    for (iq = ip + 1; iq < n; iq++) {
FLOAT g = 100.0 * fabs(a[ip][iq]);

if (i > 4 &&
    fabs(d[ip]) + g == fabs(d[ip]) &&
    fabs(d[iq]) + g == fabs(d[iq]))
    a[ip][iq] = 0.0;

else if (fabs(a[ip][iq]) > tresh) {
    FLOAT tau, t, s, c;
    FLOAT h = d[iq] - d[ip];

    if (fabs(h) + g == fabs(h))
t = (a[ip][iq]) / h;
    else {
FLOAT theta = 0.5 * h / (a[ip][iq]);
t = 1.0 / (fabs(theta) + sqrt(1.0 + theta * theta));
if (theta < 0.0)
    t = -t;
    }

    c = 1.0 / sqrt(1 + t * t);
    s = t * c;
    tau = s / (1.0 + c);
    h = t * a[ip][iq];
    z[ip] -= h;
    z[iq] += h;
    d[ip] -= h;
    d[iq] += h;
    a[ip][iq] = 0.0;

#define rotate(a,i,j,k,l) \
g = a[i][j]; \
h = a[k][l]; \
a[i][j] = g - s *(h + g*tau); \
a[k][l] = h + s*(g - h*tau);

    for (j = 0; j < ip; j++) {
rotate(a, j, ip, j, iq)
    }
    for (j = ip + 1; j < iq; j++) {
rotate(a, ip, j, j, iq)
    }
    for (j = iq + 1; j < n; j++) {
rotate(a, ip, j, iq, j)
    }
    for (j = 0; j < n; j++) {
rotate(v, j, ip, j, iq)
    }

    *nrot += 1;
}
    }
}
for (ip = 0; ip < n; ip++) {
    b[ip] += z[ip];
    d[ip] = b[ip];
    z[ip] = 0.0;
}
    }

    Erreturn("too many Jacobi iterations");
}

static void
eigsrt(d, v, n)
    FLOAT   d[];
    double **v;
    int     n;
{
    int     k, j, i;

    for (i = 0; i < n - 1; i++) {
FLOAT p = d[k = i];

for (j = i + 1; j < n; j++)
    if (d[j] >= p)
p = d[k = j];

if (k != i) {
    d[k] = d[i];
    d[i] = p;

    for (j = 0; j < n; j++) {
p = v[j][i];
v[j][i] = v[j][k];
v[j][k] = p;
    }
}
     }
}
```

```
static int
gaussjt(a, n, b, m)
    FLOAT **a, **b;
    int     n, m;
{
    int    *indxc, *indxr, *ipiv;
    int     i, j, k, l;

    IfErr (indxc = new_array_of(n, int))
Erreturn("not enough core");
    IfErr (indxr = new_array_of(n, int))
Erreturn("not enough core");
    IfErr (ipiv = new_array_of(n, int))
Erreturn("not enough core");

    for (j = 0; j < n; j++)
ipiv[j] = 0;

    for (i = 0; i < n; i++) {
int icol, irow, ll;
FLOAT pivinv;
FLOAT big = 0.0;

for (j = 0; j < n; j++)
    if (ipiv[j] != 1)
for (k = 0; k < n; k++) {
    if (ipiv[k] == 0) {
if (fabs(a[j][k]) >= big) {
    big = fabs(a[j][k]);
    irow = j;
    icol = k;
}
    }
    else if (ipiv[k] > 1)
Erreturn("singular matrix");
}

ipiv[icol] += 1;

#define swap(a,b) { \
FLOAT temp = (a); \
(a) = (b); \
(b) = temp; \
    }

if (irow != icol) {
    for (l = 0; l < n; l++)
swap(a[irow][l], a[icol][l]);
    for (l = 0; l < m; l++)
swap(b[l][irow], b[l][icol]);
}

indxr[i] = irow;
indxc[i] = icol;

if (a[icol][icol] == 0.0)
    Erreturn("singular matrix");

pivinv = 1.0 / a[icol][icol];
a[icol][icol] = 1.0;

for (l = 0; l < n; l++)
    a[icol][l] *= pivinv;

for (l = 0; l < m; l++)
    b[l][icol] *= pivinv;

for (ll = 0; ll < n; ll++)
    if (ll != icol) {
FLOAT dum = a[ll][icol];
a[ll][icol] = 0.0;

for (l = 0; l < n; l++)
    a[ll][l] -= a[icol][l] * dum;

for (l = 0; l < m; l++)
    b[l][ll] -= b[l][icol] * dum;
    }
    }
    for (l = n - 1; l >= 0; l--) {
if (indxr[l] != indxc[l])
    for (k = 0; k < n; k++)
swap(a[k][indxr[l]], a[k][indxc[l]]);
    }
    free(ipiv);
    free(indxr);
    free(indxc);
    return MY_OK;
}

/* this was declared as static int before */
float **calculate_weights(eigvecs, n, ovecs, m, meanv)
```

```
    double **eigvecs;
    FLOAT **ovecs, *meanv;
    int     n, m;
{
    int i, k, j;
    FLOAT **rest, **evect, *newvec, sumpb, **w;


    IfErr (rest = new_2d_array_of(m, n, FLOAT)) /*allocating memory for rest */
        Erreturn("not enough core");

    IfErr (evect = new_2d_array_of(n, n, FLOAT)) /*allocating memory for evect */
        Erreturn("not enough core");

    IfErr (newvec = new_array_of(n, FLOAT)){
        fprintf(stderr, "not enough core for newvec\n");
        exit(1);
    }

    IfErr (w = new_2d_array_of(m, n, FLOAT))
        Erreturn("not enough core");

/* calculating diferences between training set and mean */
    for (k = 0; k < m; k++)
        for (i = 0; i < n; i++){
            rest[k][i] = ovecs[k][i]-meanv[i];
        }

/* transposition of eigenvector mat */

    for (k = 0; k < n; k++){
        for (i = 0; i < n; i++){
            evect[i][k] = eigvecs[k][i];
        }
    }

/* weights calculation */
    /*remove coments here if you want to print the original weights */
    for (k = 0; k < m; k++){
        for (i = 0; i < n; i++){
            w[k][i] = 0;
            for (j = 0; j < n; j++){
                w[k][i] += evect[i][j] * rest[k][j];
            }
       }
     }

   free(rest);
   free(evect);
   return(w);
}


/*
 * matrix I/O
 */
static int
write_matrix(fp, mat, n)
    FILE   *fp;
    FLOAT **mat;
    int     n;
{
    int     i, j;

    /* print column vectors line-by-line */
    for (j = 0; j < n; j++) {
for (i = 0; i < n; i++) {
    fprintf(fp, FLOAT_FORMAT, (double)mat[i][j]);
    fprintf(fp, " ");
}
fprintf(fp, "\n");
    }

    return MY_OK;
}

static int
read_matrix(fp, mat, n)
    FILE   *fp;
    FLOAT **mat;
    int     n;
{
    int     i, j;


    /* read column vectors line-by-line */
    for (j = 0; j < n; j++) {
for (i = 0; i < n; i++) {
    double f;
    if (fscanf(fp, "%lf", &f) != 1)
return MY_ERR;
    mat[i][j] = f;
```

```
}
    }

    return MY_OK;
}

static float *weights_std(weights, num_feats, num_patterns)
float      **weights;
int        num_feats;
int        num_patterns;
{
  int      i, j, k,numinst = 0;
  float    *w_svar=NULL;
  char     str[128];
  float      *wstd;

    IfErr(w_svar = new_array_of(num_feats, FLOAT)){
      fprintf(stderr, "%s: not enough core for variance of weights PCA\n");
      exit(1);
    }

  /* calculating w_std */
  for(i=0; i<num_feats; i++){
    FLOAT sumdev = 0.0;
    for(k=0; k<num_patterns; k++){   /*mean value excluded as it's = 0 */
      sumdev = (weights[k][i]);
      w_svar[i] += sumdev*sumdev;
    }
    w_svar[i] /= (num_patterns-1);
    wstd[i] = sqrt(w_svar[i]);
  }
  free(w_svar);
  return(wstd);
}


FT_Model *pca(pattern, lpat, npat, modname)
    FLOAT     **pattern; /* array of pattern vectors */
    int       lpat, npat;
    char      modname[70];
{
    FLOAT       **covar; /* covariance matrix */
    FLOAT       *eval; /* eigenvalues thereof */
    double      **evec; /* eigenvectors thereof */
    int         nrot, i, k, j; /* no of jacobi rots */
    FLOAT       **stdpattern;
    FLOAT       **stdcomps;
    FLOAT       *mean, *sdev, **weights, *wstd;
    FT_Model    *ft_model;
    FILE        *outfile;
    char        fnameout[150];

IfErr (covar = new_2d_array_of(lpat, lpat, FLOAT)) {
    fprintf(stderr, "not enough core for covar matrix\n");
    exit(1);
}

        IfErr (mean = new_array_of(lpat, FLOAT)){
    fprintf(stderr, "not enough core for mean\n");
    exit(1);
}

IfErr (eval = new_array_of(lpat, FLOAT)){
    fprintf(stderr, "not enough core\n");
    exit(1);
}

IfErr (evec = new_2d_array_of(lpat, lpat, double)) {
    fprintf(stderr, "not enough core for eigenbasis\n");
    exit(1);
}

        IfErr (sdev = new_array_of(lpat, FLOAT)){
    fprintf(stderr, "not enough core for sdev\n");
    exit(1);
}

IfErr (stdcomps = new_2d_array_of(npat, lpat, FLOAT)) {
    fprintf(stderr, "not enough core for stdcomps\n");
    exit(1);
}

IfErr (stdpattern = new_2d_array_of(npat, lpat, FLOAT)) {
    fprintf(stderr, "not enough core for stdpattern\n");
    exit(1);
}

       IfErr (weights = new_2d_array_of(npat, lpat, FLOAT))
        Erreturn("not enough core");

        IfErr (wstd = new_array_of(lpat, FLOAT)){
    fprintf(stderr, "not enough core for sdev\n");
```

```
    exit(1);
}

/* compute covariance matrix */
        fprintf(stderr, "computing covariance ...\n");
IfErr (covariance(pattern, npat, covar, lpat, mean, sdev)) {
    fprintf(stderr, "%s: covariance failed\n", ERR_MSG);
    exit(1);
}

/* compute eigenvectors and -values */

        fprintf(stderr, "computing eigenbasis ...\n");
IfErr (jacobi(covar, lpat, eval, evec, &nrot)) {
    fprintf(stderr, "%s: jacobi failed\n", ERR_MSG);
    exit(1);
}

/* order eigenvectors */
        fprintf(stderr, "sorting eigenbasis ...\n");
eigsrt(eval, evec, lpat);

        fprintf(stderr, "calculating weights ...\n");
        IfErr (weights=calculate_weights(evec, lpat, pattern, npat, mean)) {
            fprintf(stderr, "%s: inst failed\n", ERR_MSG);
            exit(1);
        }



    ft_model = (FT_Model *) malloc(sizeof(FT_Model));
    ft_model->type             = FT_MODEL;
    ft_model->linkcount        = 0;
    ft_model->num_feats        = lpat;
    ft_model->num_training_set = npat;
    ft_model->feats_mean       = mean;
    ft_model->feats_covariance = covar;
    ft_model->num_modes        = (npat < lpat) ? npat : lpat;
    ft_model->eigen_vec        = evec;
    ft_model->eigen_values     = eval;
    ft_model->weights          = weights;

    wstd = weights_std(weights, lpat, npat);

    /* writing PCA information to a file */

    strcpy(fnameout, "Dat/");
    strcat(fnameout, modname);
    strcat(fnameout, "PCA.dat");
    if( (outfile = fopen(fnameout, "w")) == NULL ){
      fprintf(stderr, "readfile: can't open file %s\n\007", modname);
      exit(1);
    }
    fprintf(outfile, "PCA information for model: %s\n", modname);
    fprintf(outfile, "number of features:        %d\n", lpat);
    fprintf(outfile, "number of examples used:   %d\n", npat);
    fprintf(outfile, "total modes:               %d\n", lpat);
    fprintf(outfile, "features mean:\n");
    for (i = 0; i< lpat; i++)
      fprintf(outfile, "%f ", mean[i]);
    fprintf(outfile, "\n\neigen_values:\n");
    for (i = 0; i< lpat; i++)
      fprintf(outfile, "%f ", eval[i]);

    fprintf(outfile, "\n\nweights STD:\n");
    for (i = 0; i< lpat; i++)
      fprintf(outfile, "%f ", wstd[i]);

    fprintf(outfile, "\n\neigen_vectors:\n");
    for (k = 0; k < lpat; k++){
      for (i = 0; i< lpat; i++){
fprintf(outfile, "%f ", evec[k][i]);
      }
      fprintf(outfile, "\n");
    }

    fprintf(outfile, "\n");
    fclose(outfile);
    /*end of writing file*/

    free(covar);
    free(mean);
    free(eval);
    free(evec);
    free(sdev);

    free(stdpattern);
    free(stdcomps);
    free(weights);
    free(wstd);
    return( ft_model );
}
```

E.3 Analysis menu

```
/***********************************************************************
*   Function     : analysis_menu.c *
*   Last Update  : Thu Mar 20 15:41:08 1997 *
************************************************************************
*   Synopsis     : matching and classification functions *
************************************************************************
*   Project      : Model Based System for Biomedical Image Analysis *
*   System Title : Gmodel  *
*   Authors      : Rocio Aguilar Chongtay *
***********************************************************************/
static char SccsId[] = "%Z%%M% %I%(%G%) - RACH";
#include <analysis_menu.h>
/* the following are for calculating processing times   */
#include <sys/times.h>
#include <limits.h>
#include <time.h>
#include <misc.h>
#include <Xm/PushB.h>
#include <X11/Shell.h>

int totparam;
int numodel;
int max_patterns = 2500;
int countcand;
int initnum;

/* menu items structures */
static MenuItem analysis_menu_itemsP[] = {
  {"Construct classifying model", &xmPushButtonGadgetClass, 0, XmTEAR_OFF_DISABLED,
   NULL, NULL, search_all_cb, NULL},
  {"Classify", &xmPushButtonGadgetClass, 0, XmTEAR_OFF_DISABLED,
   NULL, NULL, classify_cb, NULL},
  {"Modify Threshold type", &xmPushButtonGadgetClass, 0, XmTEAR_OFF_DISABLED,
   NULL, NULL, create_thstype_buttons, NULL},
  {"cue Finder", &xmPushButtonGadgetClass, 0, XmTEAR_OFF_DISABLED,
   NULL, NULL, cue_finder, NULL},
  NULL,
};

MenuItem *analysis_menu_items = &(analysis_menu_itemsP[0]);

/* action and callback procedures */

void new_instance(newinst, model, weights, nuModes, lpat, evec, meanvec)
float                   *newinst;
GreyModel               *model;
float                   *weights;
int                     nuModes;
int                     lpat;
float                   **evec;
float                   *meanvec;
{
   float      sumpb;
   int        i, j;

 for (i = 0; i < lpat; i++){
       sumpb = 0;
       for (j = 0; j < nuModes; j++){
           sumpb += evec[i][j] *  weights[j];
       }
       newinst[i]=meanvec[i]+sumpb;
   }
   return;
}

void read_off_line_greyV(cfeats, obj, cvtx, model, i_angle, mfeats, num_feats)
float              *cfeats;
struct object      *obj;
struct fvertex     cvtx;
GreyModel    *model;
float              i_angle;
float              *mfeats;
int                num_feats;
{
  struct object         *currentObj;
  struct ivertex ivtx, nvtx;
  char           str[32];
  struct fvertex fvtx, vtx;
  struct polygondomain *polydmn;
  double                theta;
  int                   a, i, k, g, r=1, c_inst;
  int                   rad_count=1, g4[4];
  float                 radius, dx, dy;


    currentObj = obj;        /* reading the object */

    currentObj->linkcount++;
    if(currentObj->linkcount == 1){
```

```
      currentObj->linkcount++;
    }

    ivtx.vtX = cvtx.vtX;
    ivtx.vtY = cvtx.vtY;
    /*take the centre grey value */
    for(g=0; g < model->num_greys; g++){
      grey4val(currentObj, ivtx.vtY, ivtx.vtX, &(g4[0]));
      dx = cvtx.vtX - ivtx.vtX;
      dy = cvtx.vtY - ivtx.vtY;
      cfeats[0] = ((1-dy) * ((1-dx)*g4[0] + dx*g4[1])
                     + dy * ((1-dx)*g4[2] + dx*g4[3]));
    }

    rad_count=1;
    for(a=0, k=0; a < model->num_angles; a++){
      float radius;
      int j;

      theta = i_angle + model->angles[a]*M_PI/180;

      for(j=0; j < model->num_radii; j++){
for(g=0; g < model->num_greys; g++, k++){
          cfeats[rad_count] = mfeats[rad_count];
  vtx.vtX = cvtx.vtX + (mfeats[rad_count]) * cos( theta );
  vtx.vtY = cvtx.vtY + (mfeats[rad_count]) * sin( theta );
  ivtx.vtX = vtx.vtX;
  ivtx.vtY = vtx.vtY;

  grey4val(currentObj, ivtx.vtY, ivtx.vtX, &(g4[0]));
  dx = vtx.vtX - ivtx.vtX;
  dy = vtx.vtY - ivtx.vtY;
  cfeats[rad_count+1] = ((1-dy) * ((1-dx)*g4[0] + dx*g4[1]) +
    dy * ((1-dx)*g4[2] + dx*g4[3]));
}
rad_count+=2;
      }
    }
    freeobj(currentObj);
  return;
}

float prob_gas(pred_candidate, numparam)
float *pred_candidate;
int numparam;
{
  struct fvertex      vtx;
  GreyModel       *model;
  TrainingSet         *tr_set;
  int        i, rest_modes=0;
  float      probModRest;
  float      *inst_feats= NULL;
  float      *prob_w=NULL,  rest_2, *prob_rest_w=NULL, *inst_W= NULL;
  float      *candidate=NULL;
  float      rest_avg=0, angle;
  int        n, xsize, ysize;
  float      *tot_prob_modes, *tot_prob_rest=NULL;
  float      probMod=0, restProb=0; /* probModRest=0; */
  float      *prob_modes=NULL, *prob_rest=NULL;
  float      dist_obs_mean=0, tot_b_modes=0;
  char      str[128];

  probModRest =  0.0;
  model = selectedModel;

  selectednum_feats = (2*model->num_radii*model->num_angles+1);
  selectedtotal_modes =  selectednum_feats;

  IfErr (candidate = new_array_of(selectednum_feats, FLOAT)) {
    fprintf(stderr, "not enough core for candidate\n");
    exit(1);
  }

  IfErr(prob_w = new_array_of(selectednum_feats, FLOAT)){
    fprintf(stderr, "%s: not enough core for variance of weights\n",
    model->name);
    exit(1);
  }
  IfErr(prob_rest_w = new_array_of(selectednum_feats, FLOAT)){
    fprintf(stderr, "%s: not enough core for variance of weights\n",
    model->name);
    exit(1);
  }

  IfErr (inst_W = new_array_of(selectednum_feats, FLOAT)) {
    fprintf(stderr, "not enough core for inst_W\n");
    exit(1);
  }

  IfErr (inst_feats = new_array_of(selectednum_feats, FLOAT)) {
    fprintf(stderr, "not enough core for inst_feats\n");
    exit(1);
```

```
  }

  IfErr (prob_modes = new_array_of(sizexy, FLOAT)) {
    fprintf(stderr, "not enough core for tot_prob_modes\n");
    exit(1);
  }

  IfErr (tot_prob_modes = new_array_of(sizexy, FLOAT)) {
    fprintf(stderr, "not enough core for tot_prob_modes\n");
    exit(1);
  }

  IfErr (tot_prob_rest = new_array_of(sizexy, FLOAT)) {
    fprintf(stderr, "not enough core for tot_prob_rest\n");
    exit(1);
  }


  for(i=3; i<numparam; i++){
    inst_W[i] = pred_candidate[i];
  }

  new_instance(candidate, selectedModel, inst_W, selectedtotal_modes, selectednum_feats, selectedeigen_vec, selectedfeats_mean);

  vtx.vtX=pred_candidate[0];
  vtx.vtY=pred_candidate[1];

  angle=pred_candidate[2];

  read_off_line_greyV(inst_feats, globals.obj, vtx, selectedModel, angle, candidate, selectednum_feats);

  /* Calculating Mahalanobis distance with the weights from GA's*/
  probMod =0;
  for (i = 0; i < selectednuModes; i++){
    float  lambda=0, w_i_2=0;
    lambda = selectedeigen_values[i];
    w_i_2 = inst_W[i]*inst_W[i];
    prob_w[i]=w_i_2/(lambda);
    probMod+=prob_w[i];
  }

  /* The unexplained data */
  rest_modes = selectedtotal_modes - selectednuModes;
  rest_avg = 0;

  /* calculating the average of the rest eigenvalues */
  for (i = selectednuModes; i < selectedtotal_modes; i++){
    rest_avg += selectedeigen_values[i];
  }
  rest_avg /= rest_modes;

 for (i = 0 ; i < selectednum_feats ; i++){
   dist_obs_mean+=(inst_feats[i] - selectedfeats_mean[i]) *
     (inst_feats[i] - selectedfeats_mean[i]);
 }
   for (i = 0 ; i < selectednuModes ; i++){
      tot_b_modes+=inst_W[i]*inst_W[i];
   }

   rest_2= dist_obs_mean-tot_b_modes;
   if(rest_2<0)rest_2=0;


  for (i = selectednuModes; i < selectedtotal_modes; i++){
    prob_rest_w[i]=rest_2/(2*rest_avg);
    restProb+=prob_rest_w[i];
  }
   probModRest =  probMod+restProb;

  if(totparam!=numparam)
    totparam = numparam;


  free(candidate);
  free(prob_w);
  free(prob_rest_w);
  free(inst_W);
  free(inst_feats);
  free(prob_modes);
  free(tot_prob_modes);
  free(tot_prob_rest);
  return(probModRest);
}

void get_model2_feats(inst_feats, models_parameters, totmodes, model, total_feats)
float     *inst_feats;
float *models_parameters;
int totmodes;
GreyModel *model;
int total_feats;
{
  Widget                w;
```

```
  struct fvertex                 vtx;
  TrainingSet                    *tr_set;
  int        i, j, k, count, rest_modes=0, model_modes=0;
  int        feats_sofar=0;
  float     *new_w = NULL;
  float     *join_feats = NULL;
  float     rest_avg=0, weight=0;
  float     *prob_w,  rest_2, *prob_rest_w, *inst_W= NULL;
  float     angle, *inst_data=NULL, *feats2models= NULL;
  int        feats,  n;
  float     *tot_prob_modes, *tot_prob_rest;
  float     probMod=0, restProb=0, probModRest=0;
  float     *prob_modes=NULL, *prob_rest, dist_obs_mean=0, tot_b_modes=0;
  char      str[128];


  IfErr (inst_data = new_array_of( selectednum_feats, FLOAT)) {
    fprintf(stderr, "not enough core for candidate\n");
    exit(1);
  }

  IfErr (inst_W = new_array_of((selectednum_feats+4), FLOAT)) {
    fprintf(stderr, "Not enough core for inst_W\n");
    exit(1);
  }

  printf("\n");
  count=0;
  for(i=3; i<totmodes+3; i++){   /*number of modes plus x, y and theta */
    weight = models_parameters[i];
    inst_W[count] = weight;
    count++;
  }

  new_instance(inst_data, model, inst_W, selectednuModes,
       selectednum_feats, selectedeigen_vec, selectedfeats_mean);

  vtx.vtX=models_parameters[0];
  vtx.vtY=models_parameters[1];
  angle=models_parameters[2];

  read_off_line_greyV(inst_feats, globals.obj, vtx, model, angle,
      inst_data, selectednum_feats);

  free(inst_data);
  free(inst_W);
  return;
}

static void weights_std(wstd, model, num_feats, num_patterns)
float      *wstd;
GreyModel  *model;
int        num_feats;
int        num_patterns;
{
  int      i, j, k,numinst = 0;
  float    *w_svar=NULL;
  char     str[128];


    IfErr(w_svar = new_array_of(num_feats, FLOAT)){
      fprintf(stderr, "%s: not enough core for variance of weights\n",
      model->name);
      exit(1);
    }

  /* calculating w_std */
  for(i=0; i<num_feats; i++){
    FLOAT sumdev = 0.0;
    for(k=0; k<num_patterns; k++){   /*mean value excluded as it's = 0 */
      sumdev = (model->ft_model->weights[k][i]);
      w_svar[i] += sumdev*sumdev;
    }
    w_svar[i] /= (num_patterns-1);
    wstd[i] = sqrt(w_svar[i]);
  }
  free(w_svar);
  return;
}



void read_PCAmodel(model_name, num_feats, num_pat, total_mod,
modelModes, feats_mean, eigen_values, wStd, eigen_vec)
String         model_name;
int           *num_feats;
int           *num_pat;
int           *total_mod;
int           *modelModes;
float         *feats_mean;
float         *wStd;
float         **eigen_vec;
```

```
float         *eigen_values;
{
  int           featsnumb = 0, patnumb = 0;
  int           modesnumb = 0, modmodesnumb = 0;
  int           namesnumb = 0;
  int           i,k,n;
  FILE          *pcafile;
  char         pcaname[150];
  char         str[32];


  /* reading PCA info for the chosen model*/
  strcpy(pcaname, "Dat/");
  strcat(pcaname, model_name);
  strcat(pcaname, "PCA.dat");
  if( (pcafile = fopen(pcaname, "r")) == NULL ){
    fprintf(stderr, "readfile: can't open file %s\n\007", pcaname);
    exit(1);
  }else{
    fscanf(pcafile, "%*s %*s %*s %*s %*s");
    fscanf(pcafile, "%*s %*s %*s%d", &featsnumb);
    fscanf(pcafile, "%*s %*s %*s %*s%d", &patnumb);
    fscanf(pcafile, "%*s %*s%d\n", &modesnumb);
    fscanf(pcafile, "%*s %*s%d\n", &modmodesnumb);
    fscanf(pcafile, "%*s %*s");

    *num_feats = featsnumb;
    *num_pat = patnumb;
    *total_mod = modesnumb;
    *modelModes = modmodesnumb;
    for (i = 0; i < featsnumb; i++)
      fscanf(pcafile, "%f ", &feats_mean[i]);
    fscanf(pcafile, "%*s");
    for (i = 0; i < featsnumb ; i++)
      fscanf(pcafile, "%f ", &eigen_values[i]);

    fscanf(pcafile, "%*s %*s");
    for (i = 0; i < featsnumb ; i++)
      fscanf(pcafile, "%f ", &wStd[i]);

    fscanf(pcafile, "%*s");
    for (k = 0; k < featsnumb; k++)
      for (i = 0; i< featsnumb; i++)
fscanf(pcafile, "%f ", &eigen_vec[k][i]);

    fclose(pcafile);
  }

  return;
}

void readInfoModel(model_name, num_feats, num_pat, modelModes, wStd)
String         model_name;
int           *num_feats;
int           *num_pat;
int           *modelModes;
float         *wStd;
{
  int           featsnumb = 0, patnumb = 0;
  int           modmodesnumb = 0;
  int           i,k,n;
  FILE          *pcafile;
  char         pcaname[150];
  char         str[32];


  /* reading PCA info for the chosen model*/
  strcpy(pcaname, "Dat/");
  strcat(pcaname, model_name);
  strcat(pcaname, "PCA.dat");
  if( (pcafile = fopen(pcaname, "r")) == NULL ){
    fprintf(stderr, "readfile: can't open file %s\n\007", pcaname);
    exit(1);
  }else{
    fscanf(pcafile, "%*s %*s %*s %*s %*s");
    fscanf(pcafile, "%*s %*s %*s%d", &featsnumb);
    fscanf(pcafile, "%*s %*s %*s %*s%d", &patnumb);
    fscanf(pcafile, "%*s %*s%*d");
    fscanf(pcafile, "%*s %*s%d\n", &modmodesnumb);
    fscanf(pcafile, "%*s %*s");

    *num_feats = featsnumb;
    *num_pat = patnumb;
    *modelModes = modmodesnumb;
    for (i = 0; i < featsnumb; i++)
      fscanf(pcafile, "%*f ");
    fscanf(pcafile, "%*s");
    for (i = 0; i < featsnumb ; i++)
      fscanf(pcafile, "%*f ");

    fscanf(pcafile, "%*s %*s");
    for (i = 0; i < featsnumb ; i++)
```

```
      fscanf(pcafile, "%f ", &wStd[i]);

    fscanf(pcafile, "%*s");
    for (k = 0; k < featsnumb; k++)
      for (i = 0; i< featsnumb; i++)
fscanf(pcafile, "%*f ");

    fclose(pcafile);
  }

  return;
}

void readmatchModel(model1_name, model2_name, joined_numfeats,
joined_numpat, joinedfeats, meanjoined)
String         model1_name;
String         model2_name;
int           *joined_numfeats;
int           *joined_numpat;
float         **joinedfeats;
float         *meanjoined;
{
  int           featsnumb = 0, patnumb = 0;
  int           i,k,n;
  FILE          *filematch;
  char         filename[150];
  char         str[32];


  strcpy(filename, "/net/jura/export/oban/rocio/Models/");
  strcat(filename, model1_name);
  strcat(filename, "-");
  strcat(filename, model2_name);
  strcat(filename, ".dat");
  if( (filematch = fopen(filename, "r")) == NULL ){
    fprintf(stderr, "readfile: can't open file %s\nmatching should be\nrun first\n\007", filename);
    exit(1);
  }else{

    fscanf(filematch, "%*s%d", &featsnumb);
    fscanf(filematch, "%*s%*s%d", &patnumb);

    *joined_numfeats = featsnumb;
    *joined_numpat = patnumb;

    for (k = 0; k < patnumb; k++)
      for (i = 0; i< featsnumb; i++)
       fscanf(filematch, "%f ", &joinedfeats[k][i]);

    fclose(filematch);
  }
  return;
}

void cue_finder(w, client_data, cbs)
Widget          w;
XtPointer       client_data;
XmAnyCallbackStruct     *cbs;
{
  int             numObjs = 0;
  float           startAngle = 0;
  static int      **cand_coord;
  char           str[32];
  FILE            *fp;

  if( globals.obj == NULL ){
    sprintf(str, "An image should be read first:\n");
    HGU_XmUserInfo( globals.topl, str, XmDIALOG_FULL_APPLICATION_MODAL );
    return;
  }
  else{
    if((fp = fopen(globals.file, "r")) == NULL ){
      sprintf(str, "FSB_ReadObject: Image file not found\n", globals.file);
      HGU_XmUserInfo( globals.topl, str, XmDIALOG_FULL_APPLICATION_MODAL );
      return;
    }

    IfErr(cand_coord = new_2d_array_of(100, 4, int)){
      fprintf(stderr, "%s: not enough core for candidates\n");
      exit(1);
    }

    lookfor_candidates(globals.obj, 5, &numObjs, cand_coord, &startAngle);

    free(globals.obj);
    globals.obj = readobj( fp );

    free(cand_coord);
    fclose( fp );
  }
  return;
}
```

```
void classifying_candidate_cb(w, model, cbs)
Widget  w;
GreyModel   *model;
XmToggleButtonCallbackStruct *cbs;
{
  static int                  action_proc_added=0;
  Widget          candialog;
  float           *new_w = NULL;
  int             all_numfeats = 0, all_numpat = 0;
  int             numpat = 0, count;
  int             numcand = 0;
  int             all_totalmodes, all_nuModes;
  float           **joined_feats = NULL;
  float           *mean_joinedfeats = NULL;
  float           *joined_instance = NULL;

  GreyModel   *model2, *currentModel, *currentModel2;

  static int      **cand_coord;
  struct fvertex  ivtx;
  double          MahDist=0;
  float           *prob_w = NULL;
  float           **modelprob;
  float           total_prob = 0, normalised_prob = 0;
  float           prob_cue_model = 0, total_estimate = 0;
  int             b,i,k,n,m, j=0, xsize, ysize;
  int             count_models=0, numObjs= 0;
  int             totmodes, nuModels = 2;
  int             current = 0, other = 0;
  float           *best = NULL;
  float           **featsVec = NULL;

  char           str[32], classifyname[70];

  int             num_feats = 0, nuModes = 0, swap = 0;
  int             num_patterns = 0, countW = 0;
  float           *wStd = NULL;

  float           *joined_wStd = NULL;
  float           **joined_eigen_vec = NULL;
  float           **transp_eigenvec = NULL;
  float           *joined_eigen_values = NULL;
  FILE            *fp;

  if( cbs->set != True )
    return;

  if( globals.obj == NULL ){
    sprintf(str, "An image should be read first:\n");
    HGU_XmUserInfo( globals.topl, str, XmDIALOG_FULL_APPLICATION_MODAL );
    return;
  }
  else{

    for(i = 0; i < nuModels; i++){
      if(strcmp(model->name, globals.models[j]->name) != 0){
swap = 1;
      }
    }

    selectedModel = model;
    maxnum_objs = 100;
    num_feats = (2*model->num_radii*model->num_angles+1);
    selectednum_feats = num_feats;
    all_numfeats = num_feats*2;

    IfErr(selectedfeats_mean = new_array_of(selectednum_feats, FLOAT)){
      fprintf(stderr, "%s: not enough core to read PCA info\n");
      exit(1);
    }
    IfErr(selectedeigen_values = new_array_of(selectednum_feats, FLOAT)){
      fprintf(stderr, "%s: not enough core to read PCA info\n");
      exit(1);
    }
    IfErr(selectedeigen_vec = new_2d_array_of(selectednum_feats,
      selectednum_feats, FLOAT)){
      fprintf(stderr, "%s: not enough core to read PCA info\n");
      exit(1);
    }

    IfErr(cand_coord = new_2d_array_of(maxnum_objs, 4, int)){
      fprintf(stderr, "%s: not enough core for candidates\n");
      exit(1);
    }

    IfErr(new_w = new_array_of(all_numfeats, FLOAT)){
      fprintf(stderr, "%s: not enough core for new_weights\n");
      exit(1);
    }

    IfErr(prob_w = new_array_of(all_numfeats, FLOAT)){
```

```
      fprintf(stderr, "%s: not enough core for variance of weights\n");
      exit(1);
    }

    IfErr(featsVec = new_2d_array_of(nuModels, num_feats, FLOAT)){
      fprintf(stderr, "%s: not enough core for feature vector\n");
      exit(1);
    }

    IfErr(best = new_array_of(num_feats+3, FLOAT)){
      fprintf(stderr, "%s: not enough core for Models compared\n");
      exit(1);
    }


    IfErr(joined_feats = new_2d_array_of(max_patterns,
 all_numfeats, FLOAT)){
      fprintf(stderr, "%s: not enough core for all objects features\n");
      exit(1);
    }

    IfErr(joined_instance = new_array_of(all_numfeats, FLOAT)){
      fprintf(stderr, "%s: not enough core to read PCA info\n");
      exit(1);
    }

    IfErr(mean_joinedfeats = new_array_of(all_numfeats, FLOAT)){
      fprintf(stderr, "%s: not enough core to read PCA info\n");
      exit(1);
    }

    IfErr(joined_wStd = new_array_of(all_numfeats, FLOAT)){
      fprintf(stderr, "%s: not enough core to read PCA info\n");
      exit(1);
    }

    IfErr(joined_eigen_values = new_array_of(all_numfeats, FLOAT)){
      fprintf(stderr, "%s: not enough core to read PCA info\n");
      exit(1);
    }
    IfErr(joined_eigen_vec = new_2d_array_of(all_numfeats, all_numfeats, FLOAT)){
      fprintf(stderr, "%s: not enough core to read PCA info\n");
      exit(1);
    }

    IfErr(transp_eigenvec = new_2d_array_of(all_numfeats, all_numfeats, FLOAT)){
      fprintf(stderr, "%s: not enough core to read PCA info\n");
      exit(1);
    }

    IfErr(selectedwStd = new_array_of(selectednum_feats, FLOAT)){
      fprintf(stderr, "%s: not enough core to read PCA info\n");
      exit(1);
    }

    IfErr(modelprob = new_2d_array_of(nuModels, maxnum_objs, FLOAT)){
      fprintf(stderr, "%s: not enough core for all objects features\n");
      exit(1);
    }

    for(m = 0; m < nuModels; m++){
      if(strcmp(selectedModel->name, globals.models[m]->name) != 0){
model2 = globals.models[m];
currentModel2 = model2;
prob_cue_model = 0.1595;
      }else{
        prob_cue_model = 1 - 0.1595;

      }
    }

    /* processing for each model */
    for(m = 0; m < nuModels; m++){

      readmatchModel(selectedModel->name, currentModel2->name,
     &all_numfeats, &all_numpat, joined_feats);

    strcpy(classifyname, selectedModel->name);
    strcat(classifyname, "Classif");

    read_PCAmodel(classifyname, &all_numfeats, &all_numpat,
   &all_totalmodes, &all_nuModes, mean_joinedfeats,
   joined_eigen_values, joined_wStd, joined_eigen_vec);

    /*reading image as it is modified with the cuefinder*/
    freeobj(globals.obj);
    if((fp = fopen(globals.file, "r")) == NULL ){
      sprintf(str, "FSB_ReadObject: Image file not found\n", globals.file);
      HGU_XmUserInfo( globals.topl, str, XmDIALOG_FULL_APPLICATION_MODAL );
      return;
    }
    if( (globals.obj = readobj( fp )) == NULL ){
```

```
      fprintf(stderr, "Failing while reading\nImage file %s\n");
      HGU_XmUserInfo( globals.topl, str, XmDIALOG_FULL_APPLICATION_MODAL );
      freeobj(globals.obj);
      exit(1);
    }
    fclose( fp );
    XClearWindow(globals.dpy, globals.win );

    picframe_X(globals.dpy, globals.win, globals.gc_greys,
       globals.obj, globals.pf);
    /* end of reading image */

    numObjs = 0;
    lookfor_candidates(globals.obj, 5, &numObjs, cand_coord, &startAngle);

    initnum = 0;
    numcand = numObjs;
    for(n=initnum; n < numcand; n++){

    xsize = cand_coord[n][2]-cand_coord[n][0];
    ysize = cand_coord[n][3]-cand_coord[n][1];
    sizexy = xsize*ysize;
    ivtx.vtX = cand_coord[n][0];
    ivtx.vtY = cand_coord[n][1];

    for(j = 0; j < nuModels; j++){
      if(strcmp(selectedModel->name, globals.models[j]->name) != 0){
currentModel = model2;
      }else{
currentModel = model;
      }

read_PCAmodel(currentModel->name, &selectednum_feats,
                      &selectednum_patterns, &selectedtotal_modes,
                      &selectednuModes, selectedfeats_mean,
                      selectedeigen_values, selectedwStd, selectedeigen_vec);

best = opgmodel(ivtx.vtX, ivtx.vtY, xsize, ysize,
        selectedwStd, selectednuModes, startAngle, currentModel->name);

        get_model2_feats(featsVec[j], best, selectednuModes,
                                currentModel, selectednum_feats);

      }

      if(swap == 0){
count=0;
for(j=0; j < nuModels; j++){
  for(i=0; i< selectednum_feats; i++){
    joined_instance[count] = featsVec[j][i];
    count++;
  }
}
      }else{
count=0;
        k = 1; /* this swaps the order */
for(j=0; j < nuModels; j++){
  for(i=0; i< selectednum_feats; i++){
    joined_instance[count] = featsVec[k][i];
    count++;
  }
          k = 0;
}
      }


      for (k = 0; k < all_numfeats; k++){
for (i = 0; i < all_numfeats; i++){
  transp_eigenvec[i][k] = joined_eigen_vec[k][i];
}
      }

      for (i = 0; i < all_numfeats; i++){
new_w[i] = 0;
for (j = 0; j < all_numfeats; j++){
  new_w[i] += transp_eigenvec[i][j] * (joined_instance[j]-mean_joinedfeats[j]);
}
      }

      MahDist =0;
      modelprob[m][n] = 0;
      for (i = 0; i < all_nuModes; i++){
float  lambda=0, w_i_2=0;
lambda = joined_eigen_values[i];
w_i_2 = new_w[i]*new_w[i];
if(lambda <= 0.0){
  continue;
}
prob_w[i]=w_i_2/lambda;
MahDist +=prob_w[i];
      }
      modelprob[m][n] = exp(-MahDist);
```

```
    }
    selectedModel = model2;
    currentModel2 = model;

   } /*end for each model*/

    currentModel = model;
    other = 1;

    printf("\nModel %s:\nInstance\tNormalised Prob\n", model->name);
    for(m = 0; m < nuModels; m++){ this is to calculate for two models
     total_prob = 0;
     normalised_prob = 0;
     for(n = initnum; n < numObjs; n++){
       total_prob = modelprob[m][n]/(modelprob[m][n]+modelprob[other][n]);

       normalised_prob = (total_prob * prob_cue_model)/
 ((total_prob * prob_cue_model)+
  ((1 - total_prob) * (1 - prob_cue_model)));

     printf("%d\t%f\n", n+1, normalised_prob);

       total_estimate += normalised_prob;
     }

     printf("Image file:\tEstimated %s:\n\n", model->name);
     printf("%s\t\t", globals.file);
     printf("%3.0f\n",  total_estimate);
     total_estimate = 0;
     other = 0;
     prob_cue_model = 1 - prob_cue_model;
     currentModel = model2;

   }

   free(modelprob);

  }

  free(transp_eigenvec);
  free(cand_coord);
  free(prob_w);
  free(new_w);
  free(selectedwStd);
  free(featsVec);
  free(joined_feats);
  free(mean_joinedfeats);
  free(joined_instance);
  free(selectedfeats_mean); /* these last three are globals for GA's */
  free(selectedeigen_vec);
  free(selectedeigen_values);
  free(joined_wStd);
  free(joined_eigen_vec);
  free(joined_eigen_values);
  return;
}



void select_all_candidates_cb(w, model, cbs)
Widget  w;
GreyModel  *model;
XmToggleButtonCallbackStruct *cbs;
{
  static  int               action_proc_added=0;
  char                        **image_files;
  int                         i, j, k,b,g,h;
  int                         xsize, ysize, index=0, nuModes;
  int                         count, image_instances;
  int                         imagescount, nuModels = 2;
  int                         all_numfeats, count_models=0;
  struct object               **obj;
  struct fvertex              initvtx, vtx;
  float                       *mean_feats=NULL, *wstd=NULL,angle;
  char                        modname[20], str[128];
  double                      init_angle;
  float                       *gminst_feats= NULL, *gminst2_feats= NULL;
  float                       *model2_feats=NULL;
  float                       *bestones=NULL, **new_w=NULL;
  float                       **joined_feats= NULL;
  float                       *mean_joinedfeats = NULL;
  TrainingSet                 *tr_set;
  FT_Model               *ft_classifmodel = NULL;

  GreyModelInstance           *gminst;
  GreyModel              *model2;
  FILE                        *outfile, *fp;
  char                        fnameout[150], doublename[150];
  char                        image_name[150], classifyname[70];

  int                         num_feats = 0, numFeats =0;
```

```
  int                           num_patterns = 0;
  int                           total_modes = 0;
  float                         *feats_mean = NULL;
  float                         *wStd = NULL;
  float                         **eigen_vec;
  float                         *eigen_values = NULL;
  int                           freed = 0, globalcount=0;
  int                           subset_inst = 0, sofar = 0;
  int                           globalsofar = 0, first = 0;


  if( cbs->set != True )
    return;

  if(globals.num_tr_sets==NULL){
    fprintf(stderr, "Training data should be read first\n");
    HGU_XmUserInfo( globals.topl, str, XmDIALOG_FULL_APPLICATION_MODAL );
    return;
  }
  else{
    for(i=0, tr_set=NULL; i < globals.num_tr_sets; i++){
      tr_set = &globals.tr_sets[i];
    }
  }

  if( tr_set == NULL ){
    sprintf(str, "No training set for model:\n    %s", model->name);
    HGU_XmUserInfo( globals.topl, str, XmDIALOG_FULL_APPLICATION_MODAL );
    return;
  }
  else{
    for(j = 0; j < nuModels; j++){
      if(strcmp(model->name, globals.models[j]->name) != 0){
model2 = globals.models[j];
      }else{
first = j;
      }
    }

    if( model2 == NULL ){
      fprintf(stderr, "At least TWO models should exist");
      HGU_XmUserInfo( globals.topl, str, XmDIALOG_FULL_APPLICATION_MODAL );
      return;
    }

    if(tr_set->num_data > 100){
      subset_inst = 100;
    }
    num_feats = 2*model->num_radii*model->num_angles+1;
    selectednum_feats = 2*model2->num_radii*model2->num_angles+1;


    IfErr(selectedfeats_mean = new_array_of(selectednum_feats, FLOAT)){
      fprintf(stderr, "%s: not enough core to read PCA info\n");
      exit(1);
    }
    IfErr(selectedeigen_values = new_array_of(selectednum_feats, FLOAT)){
      fprintf(stderr, "%s: not enough core to read PCA info\n");
      exit(1);
    }
    IfErr(selectedwStd = new_array_of(selectednum_feats, FLOAT)){
      fprintf(stderr, "%s: not enough core to read PCA info\n");
      exit(1);
    }
    IfErr(selectedeigen_vec = new_2d_array_of(selectednum_feats,selectednum_feats, FLOAT)){
      fprintf(stderr, "%s: not enough core to read PCA info\n");
      exit(1);
    }

    selectednum_feats = 0;
    selectednuModes = 0;
    selectednum_patterns = 0;
    selectedtotal_modes = 0;
    *selectedfeats_mean = NULL;
    *selectedwStd = NULL;
    **selectedeigen_vec = NULL;
    *selectedeigen_values = NULL;

    strcpy(fnameout, "/net/jura/export/oban/rocio/Models/");
    strcpy(doublename, model->name);
    strcat(doublename, "-");
    strcat(doublename, model2->name);
    strcat(fnameout, doublename);
    strcat(fnameout, ".dat");

    if( (outfile = fopen(fnameout, "w")) == NULL ){
      fprintf(stderr, "readfile: can't open file
      %s\nvariation modes over\n model training sets\n
                  should be calculated\n\007", fnameout);
      exit(1);
    }
```

```
    read_PCAmodel(model2->name, &selectednum_feats,
           &selectednum_patterns, &selectedtotal_modes,
           &selectednuModes, selectedfeats_mean, selectedeigen_values,
           selectedwStd, selectedeigen_vec);

    all_numfeats = num_feats+selectednum_feats;
    num_patterns = selectednum_patterns;
    fprintf(outfile, "NumFeats:\t\t%d\n", all_numfeats);
    fprintf(outfile, "Total instances:\t%d\n", tr_set->num_data);
    fclose(outfile);

    totparam = 0;
    numvar = selectednum_feats;
    image_instances = 0;
    numInst = 0;
    imagescount=0;

    printf("Creating model %s-%s\n", model->name, model2->name);

    for(k=0; k<tr_set->num_data; k++){
      if(freed == 0){
if( (outfile = fopen(fnameout, "a")) == NULL ){
  fprintf(stderr, "readfile: can't open file %s\n\007", fnameout);
  exit(1);
}

IfErr(bestones = new_array_of(all_numfeats, FLOAT)){
  fprintf(stderr, "%s: not enough core for Models compared\n");
  exit(1);
}

IfErr (model2_feats = new_array_of(selectednum_feats, FLOAT)) {
  fprintf(stderr, "Not enough core for model2_feats\n");
  exit(1);
}

IfErr(joined_feats = new_2d_array_of(tr_set->num_data+1, all_numfeats, FLOAT)){
  fprintf(stderr, "%s: not enough core for all objects features\n");
  exit(1);
}

IfErr(mean_joinedfeats = new_array_of(all_numfeats, FLOAT)){
  fprintf(stderr, "%s: not enough core to read PCA info\n");
  exit(1);
}
        freed = 1;
      }

      if(strcmp(image_name, tr_set->data[globalcount]->filename) != 0){
strcpy(image_name, tr_set->data[globalcount]->filename);
if(globalcount != 0)
  printf("Instances in image %d:\t%d\n\n", imagescount, image_instances);
imagescount++;
  printf("Processing image %d %s\n\n", imagescount, image_name);
if((fp = fopen(image_name, "r")) == NULL ){
  sprintf(str, "FSB_ReadObject: Image file not found\n",image_name);
  HGU_XmUserInfo( globals.topl, str, XmDIALOG_FULL_APPLICATION_MODAL );
  return;
}

image_instances = 0;
if( globals.obj != NULL ){
  freeobj(globals.obj);
}

if( (globals.obj = readobj( fp )) == NULL ){
  fprintf(stderr, "Failing while reading\nImage file %s\n",image_name);
  HGU_XmUserInfo( globals.topl, str, XmDIALOG_FULL_APPLICATION_MODAL );
  freeobj(globals.obj);
  exit(1);
}
fclose( fp );
      }

      gminst = tr_set->data[globalcount];
      gminst_feats = GreyModel_inst_to_feats(gminst);
      numInst++;
      image_instances++;
      printf("Image %d\t Current instance:\t%d/%d\n", imagescount,
     (globalcount+1),tr_set->num_data);
      xsize = 5;
      ysize = 5;
      sizexy = xsize * ysize;
      initvtx.vtX = gminst->centre.vtX;
      initvtx.vtY = gminst->centre.vtY;
      startAngle = gminst->angle;


      selectedModel = model2;

      bestones = opgmodel(initvtx.vtX, initvtx.vtY, xsize, ysize,
  selectedwStd, selectednuModes, startAngle, model2->name);
```

```
     get_model2_feats( model2_feats, bestones, selectednuModes, model2, all_numfeats);

     /*for the same order within the double vector */
     if(first == 0){
       for(i=0; i< num_feats; i++){
 joined_feats[globalcount][i] = gminst_feats[i];
       }
       count =0;
       for(i= num_feats; i< all_numfeats; i++){
 joined_feats[globalcount][i] = model2_feats[count];
 count++;
       }
     }else{
       for(i=0; i< num_feats; i++){
 joined_feats[globalcount][i] = model2_feats[i];
       }
       count =0;
       for(i= num_feats; i< all_numfeats; i++){
joined_feats[globalcount][i] = gminst_feats[count];
count++;
       }
     }/* end of the same order */

      globalcount++;
      sofar++;
      if((sofar == subset_inst) || (k == tr_set->num_data)){
printf("writing %d vectors to a file\n", sofar);
for(b = globalsofar; b < globalcount; b++){
  for(i = 0; i < all_numfeats; i++)
    fprintf(outfile, "%4.6f\t", joined_feats[b][i]);
  fprintf(outfile, "\n");
}

strcpy(classifyname, model->name);
strcat(classifyname, "Classif");
fclose(outfile);

ft_classifmodel = (FT_Model *) malloc(sizeof(FT_Model));
ft_classifmodel = pca(joined_feats, all_numfeats, globalcount, classifyname);
free(bestones);
free(model2_feats);
free(joined_feats);
free(ft_classifmodel);
sofar = 0;
freed = 0;
globalsofar+=subset_inst;
      }
    }

    free(selectedfeats_mean);
    free(selectedwStd);
    free(selectedeigen_vec);
    free(selectedeigen_values);
  }
  printf("\nMatching model has been generated (%d)\n", globalcount);
  return;
}

static Widget classify_model_selection(topl, models, num_models )
Widget topl;
GreyModel **models;
int num_models;
{
  Widget dialog, control, widget;
  MenuItem *items;
  int i;

   dialog = HGU_XmCreateStdDialog(topl, "Classifying_dialog",
                                  xmFormWidgetClass, NULL, 0);

   XtVaSetValues(dialog, XmNtitle, "Select classifying model", NULL);

  control = XtNameToWidget( dialog, "*control" );

  items = (MenuItem *) malloc(sizeof(MenuItem) * (num_models+1) );
  for(i=0; i < num_models; i++){
    items[i].name          = models[i]->name;
    items[i].mnemonic      = 0;
    items[i].accelerator   = NULL;
    items[i].accel_text    = NULL;
    items[i].callback      = classifying_candidate_cb;
    items[i].callback_data = (XtPointer) models[i];
  }
  items[i].name = NULL;

  HGU_XmCreatePB_Row("select_model", control, items );

  XtManageChild( dialog );
  return( dialog );
}
```

```
static Widget candidates_model_selection_panel( topl, models, num_models )
Widget topl;
GreyModel **models;
int num_models;
{
  Widget dialog, control, widget;
  MenuItem *items;
  int i;

   dialog = HGU_XmCreateStdDialog(topl, "matching_dialog",
                                xmFormWidgetClass, NULL, 0);

XtVaSetValues(dialog, XmNtitle, "Select matching model", NULL);

  control = XtNameToWidget( dialog, "*control" );

  items = (MenuItem *) malloc(sizeof(MenuItem) * (num_models+1) );
  for(i=0; i < num_models; i++){
    items[i].name          = models[i]->name;
    items[i].mnemonic      = 0;
    items[i].accelerator   = NULL;
    items[i].accel_text    = NULL;
    items[i].callback      = select_all_candidates_cb;
    items[i].callback_data = (XtPointer) models[i];
  }
  items[i].name = NULL;

  HGU_XmCreatePB_Row( "select_model", control, items );

  XtManageChild( dialog );
  return( dialog );

}


static void selecThreshnumCb(w, user_data, call_data)
Widget w;
XtPointer user_data;
XtPointer call_data;
{
  int i;
  int counthr = (int) user_data;
  fprintf(stderr, "Threshold type selected = compth%d\n", counthr);
  countype =counthr;
  return;
}

void create_thstype_buttons(w)
Widget          w;
{
    Widget list_w, dialog, control, button;
    int i, numtypes = 5;
    char thstypes[16];

    dialog = HGU_XmCreateStdDialog(globals.topl, "Select Threshold Type First",
                                xmFormWidgetClass, NULL, 0);

    control = XtNameToWidget( dialog, "*.control" );

    list_w = XtVaCreateManagedWidget("Compthr", xmRowColumnWidgetClass,
                                                              control,
      XmNleftAttachment, XmATTACH_FORM,
      XmNrightAttachment, XmATTACH_FORM,
      XmNbottomAttachment, XmATTACH_FORM,
      XmNtopAttachment, XmATTACH_FORM,
      XmNpacking, XmPACK_COLUMN,
      XmNorientation, XmVERTICAL,
      XmNnumColumns, 5,
      NULL);
    countype=1;
    for (i = 0; i < numtypes; i++) {
      sprintf(thstypes,"%d",countype);
      button = XtCreateManagedWidget (thstypes, xmPushButtonGadgetClass,
     list_w, (ArgList) NULL, 0);
      XtAddCallback(button, XmNactivateCallback, selecThreshnumCb, (XtPointer ) countype);

  countype++;
    }

    w = XtNameToWidget(dialog, "*Ok");
    XtAddCallback(w, XmNactivateCallback,
  SetSensitiveCallback, XtParent(w) );
    XtAddCallback(dialog, XmNokCallback,
  (void (*)()) XtUnmanageChild, NULL);

    w = XtNameToWidget(dialog, "*Cancel");
    XtAddCallback(w, XmNactivateCallback,
  SetSensitiveCallback, XtParent(w) );
    XtAddCallback(w, XmNactivateCallback, DestroyWidgetCallback,
  XtParent(dialog) );
```

```
    XtManageChild(list_w);
    XtManageChild(dialog);
    return;
}

void classify_cb(w, client_data, cbs)
Widget w;
XtPointer client_data;
XmAnyCallbackStruct *cbs;
{
  Widget dialog, widget;

  if(countype<=0||countype>5) {
    countype=1;
  }

  widget = w;
  dialog = classify_model_selection(globals.topl, globals.models,
       globals.num_models );

  widget = XtNameToWidget(dialog, "*Ok");
  XtAddCallback(widget, XmNactivateCallback,
SetSensitiveCallback, XtParent(w) );
  XtAddCallback(widget, XmNactivateCallback, DestroyWidgetCallback,
XtParent(dialog) );

  widget = XtNameToWidget(dialog, "*Cancel");
  XtAddCallback(widget, XmNactivateCallback,
SetSensitiveCallback, XtParent(w) );
  XtAddCallback(widget, XmNactivateCallback, DestroyWidgetCallback,
XtParent(dialog) );

  XtSetSensitive(XtParent( w ), False );

  XtManageChild( XtParent(dialog) );


  XtManageChild( dialog );

  return;
}

void search_all_cb(w, client_data, cbs)
Widget w;
XtPointer client_data;
XmAnyCallbackStruct *cbs;
{
  Widget dialog, widget;
  MenuItem *items;
  char       str[128];
  int           i;
  TrainingSet       *tr_set;
   struct object **obj;


    if(globals.num_tr_sets==NULL){
        sprintf(str, "Training data should be read first\n");
        HGU_XmUserInfo( globals.topl, str, XmDIALOG_FULL_APPLICATION_MODAL );
        return;
   }
   else{
     widget = w;
     if(countype<=0||countype>5) {
       countype=1;
     }

     dialog = candidates_model_selection_panel(globals.topl, globals.models,
       globals.num_models );


     widget = XtNameToWidget(dialog, "*Ok");
     XtAddCallback(widget, XmNactivateCallback,
   SetSensitiveCallback, XtParent(w) );
     XtAddCallback(widget, XmNactivateCallback, DestroyWidgetCallback,
   XtParent(dialog) );

     widget = XtNameToWidget(dialog, "*Cancel");
     XtAddCallback(widget, XmNactivateCallback,
   SetSensitiveCallback, XtParent(w) );
     XtAddCallback(widget, XmNactivateCallback, DestroyWidgetCallback,
   XtParent(dialog) );

     XtSetSensitive(XtParent( w ), False );

     XtManageChild( XtParent(dialog) );
    }

  return;
}
```

E.4 Candidates

```
/***********************************************************************
*   Function    : candi.c *
*   Last Update  : Thu Mar 20 15:41:08 1997 *
************************************************************************
*   Synopsis     : read woolz image, segment into object candidates *
************************************************************************
*   Project      : Model Based System for Biomedical Image Analysis     *
*   System Title : Gmodel  *
*   Authors      : Rocio Aguilar Chongtay *
***********************************************************************/
static char SccsId[] = "%Z%%M% %I%(%G%) - RACH";
#include<candi.h>
#include<wstruct.h>
#include<misc.h>

extern double principal();
int countype;

float lookfor_candidates(w, obj, mask, numobjs, candiCoord, init_angle)
Widget w;
struct object *obj;
int             mask;
int             *numobjs;
int             **candiCoord;
double          *init_angle;
{
 struct object  *thobj, *histobj, *dilobj, *dilerobj,
                *smoothobj, *erobj, *erdilobj, *discobj1, *discobj2;
 struct object  **objlist[MAXOBJS];
 char          str[32], c_num[15];
 struct ivertex ivtx, isvtx;
 int            thval, nobjs, i, j, brdsz, maxipeak;
 int            size = 1, hsize, median(), peakintens(), convolve();
 register GREYP g;
 WlzIwspace iwsp;
 WlzGwspace gwsp;
 register int k, current, maxigrey;
 double     dmaxigrey;
 int   *cx, *cy, q=0;

 if (mask == 0)
      size = mask;

 hsize = size / 2;  /* std size of the nbhd (1-15) */
       if (hsize <= 0)
                hsize = 1 ;        /* size should be bigger than zero */
       if (hsize > 7)
                hsize = 7 ;

/* object */
  if( globals.obj == NULL ){
    fprintf(stderr, "An image should be read first:\n");
    HGU_XmUserInfo( globals.topl, str, XmDIALOG_FULL_APPLICATION_MODAL );
    return;
  }
  else{

    obj = globals.obj;        /* reading the object */

    /* Segmentation */
    ASSIGN(histobj, histo(obj));  /* obtaining object's histogram */

    switch( countype ){
    default:
    case 1:
      thval = compthr1(histobj);
break;
    case 2:
      thval = compthr2(histobj);
break;
    case 3:
      thval = compthr3(histobj);
break;
    case 4:
      thval = compthr4(histobj);
break;
    case 5:
      thval = compthr5(histobj);
break;
    }

    ASSIGN(thobj, threshold(obj, thval));       /* applying the threshold */
    label(thobj, &nobjs, objlist, MAXOBJS, 25); /* the ASSIGN to objlist is
       made here */
    *numobjs = nobjs;
    if (nobjs > 0 ){
      /* Dilation and Erosion i.e. Closing*/
      q = 0;
      for (i = 0; i < nobjs; i++){
/* executing dilation */

ASSIGN(dilobj, structdilate(objlist[i], makesqr(9)));
ASSIGN(dilobj->vdom, obj->vdom);          /* attaching the value domain
     from the original obj to
     the dilated object */
                                          /* executing erosion over
     dilated object */
ASSIGN(dilerobj, structerose(dilobj, makesqr(9)));
ASSIGN(dilerobj->vdom, obj->vdom);        /* attaching the value domain
     from the original object to
     the eroded object */

/*  writting the dilated
    and eroded object */
/* smoothing object using opening i.e.
   erosion and dilation */
ASSIGN(erobj, erosion(dilerobj));
ASSIGN(erobj->vdom, obj->vdom);

ASSIGN(erdilobj, dilation(erobj));
ASSIGN(erdilobj->vdom, obj->vdom);
/* normalisation */

ASSIGN(smoothobj, normalise(erdilobj));
pixconvert(smoothobj, INT_GREY);
```

```
/* applying the 3 * 3 median filter */
brdsz = hsize == 0 ? 1 : hsize;
if (smoothobj->type == 1)
  smoothobj = seqpar(smoothobj,1,0,0,brdsz,edgegv(smoothobj),hsize,median);
else error ("median --- not type 1");
/*  obtaining the maximum peak for each obj */
maxipeak = peakintens(smoothobj);
/*              printf("object %d maximum peak\t%d\n", i,maxipeak); */
/* multiplying each operator by the maximum peak */
for (j = 0; j < 121; j++)
  op3[j] = (d3[j]*maxipeak);
for (j = 0; j < 81; j++)
  op5[j] = (d5[j]*maxipeak);
ASSIGN(discobj1, applyconvolution_new (smoothobj, &conv1));
ASSIGN(discobj2, applyconvolution_new (smoothobj, &conv2));
/* display the candidates */
/* storing in a vector the candidate obj coordinates */
candiCoord[q][0]=smoothobj->idom->kol1;
candiCoord[q][1]=smoothobj->idom->line1;
candiCoord[q][2]=smoothobj->idom->lastkl;
candiCoord[q][3]=smoothobj->idom->lastln;
/* calculating the initial angle,
           by calculating first the centre of mass and then
                                  the principal axis */
                centreofmass(smoothobj, 1, &cx, &cy);
                *init_angle = M_PI/180*(principal(smoothobj, cx, cy, 0));
                if((smoothobj->idom->kol1<(obj->idom->kol1+ 80))||
                   (smoothobj->idom->line1<(obj->idom->line1+20))||
                   (smoothobj->idom->lastkl>(obj->idom->lastkl-50))||
                   (smoothobj->idom->lastln>(obj->idom->lastln-20)))
   {
     q--;
     *numobjs--;
   }
                else{
                initgreyscan(smoothobj,&iwsp,&gwsp);
maxigrey = 0;
dmaxigrey = 0;
while (nextgreyinterval(&iwsp) == 0) {
  g = gwsp.u_grintptr;
  switch (gwsp.pixeltype) {
    case INT_GREY:
      for (k=iwsp.lftpos; k<=iwsp.rgtpos; k++)
if(*g.inp > maxigrey)
  maxigrey = *g.inp;
      break;
    case SHORT_GREY:
      for (k=iwsp.lftpos; k<=iwsp.rgtpos; k++)
if(*g.shp > maxigrey)
  maxigrey = *g.shp;
      break;
    case UBYTE_GREY:
      for (k=iwsp.lftpos; k<=iwsp.rgtpos; k++)
if(*g.ubp > maxigrey)
  maxigrey = *g.ubp;
      break;
    case FLOAT_GREY:
      for (k=iwsp.lftpos; k<=iwsp.rgtpos; k++)
   if(*g.flp> dmaxigrey)
  dmaxigrey = *g.flp;
      break;
    case DOUBLE_GREY:
      for (k=iwsp.lftpos; k<=iwsp.rgtpos; k++)
if(*g.dbp> dmaxigrey)
  dmaxigrey = *g.dbp;
      break;
  }
}

        }
freeobj(dilobj);
freeobj(dilerobj);
freeobj(erobj);
freeobj(erdilobj);
freeobj(smoothobj);
freeobj(discobj1);
freeobj(discobj2);
q++;   /* incrementing candidate counter */
   } /*else of discarding edges candidates */
    }
    else{
      fprintf(stderr, "No candidates found for model:\n");
      HGU_XmUserInfo( globals.topl, str, XmDIALOG_FULL_APPLICATION_MODAL );
      return;
    }
    freeobj(thobj);
    freeobj(histobj);
  }
}
```

E.5 Opgmodel, interacting with GA's

```
/************************************************************************
*   Function     : opgmodel.c, interacts with Levine's PGAPack *
```

```
*   Last Update  : Thu Mar 20 15:41:08 1997 *
*************************************************************************
*   Synopsis     : Optimise gmodel info to find best possible match *
*************************************************************************
*   Project      : Model Based System for Biomedical Image Analysis     *
*   System Title : Gmodel  *
*   Authors      : Rocio Aguilar Chongtay  *
*************************************************************************/
static char SccsId[] = "%Z%%M% %I%(%G%) - RACH";
#include <pgapack.h>
#include <opgmodel.h>
#include <gmodel.h>

double gmodeval(ctx, p, pop)
PGAContext *ctx;
int p;
int pop;
{
    int i, len;
    float *term, prob;

    len = PGAGetStringLength(ctx);

    term = (float *) malloc(sizeof(float)* len);
                         /*allocating memory for maxmodes */

    for (i = 0; i < len; i++) {
        term[i] = PGAGetRealAllele(ctx, p, pop, i);
                         /*candidate is produced here */
    }

   prob = prob_gas(term, len);
   return (prob);
}

/****************************************
*    PGA's user main program                      *
****************************************/

float *opgmodel(xo, yo, xsize, ysize, w_std, numgens, init_angle,modname)
float       xo;
float       yo;
int         xsize;
int         ysize;
float      *w_std;
int         numgens;
double  init_angle;
char   modname[20];
{
    PGAContext *ctx;     /* the context variable */
    double *l=NULL, *u=NULL;
    int i, best_p, len;
    double best_e;
    float *thebest;

    numgens+=3; /*ading xo,yx, angle*/
    l = (double *) malloc(sizeof(double)* numgens); /*allocating memory for l and u */
    u = (double *) malloc(sizeof(double)* numgens);

                        /* Number of Iterations to run Ga's */
    maxiter =100;
    numitera = 0;

    l[0] = xo;                            u[0] = xo+xsize;  /* x range */
    l[1] = yo;                            u[1] = yo+ysize;  /* y range */
    l[2] = 0;                              u[2] = 360;         /* angle range */

  /*initialising the weight params with 2 std values */
  /*    printf("%f,%f\t%f,%f\t%f,%f\n", l[0],u[0],l[1],u[1],l[2],u[2]); */
    for (i=3; i<numgens; i++) {
      l[i] = - 2*w_std[i-3];
      u[i] =   2*w_std[i-3];
    }

    ctx = PGACreate(&nargc, nargv, PGA_DATATYPE_REAL,
    numgens, PGA_MINIMIZE);

    PGASetRandomSeed(ctx, 1);

    PGASetRealInitRange(ctx, l, u);

    PGASetMaxGAIterValue(ctx, maxiter);

    PGASetSelectType(ctx, PGA_SELECT_PTOURNAMENT);
    PGASetMutationAndCrossoverFlag(ctx, PGA_TRUE);

    PGASetNumReplaceValue(ctx, 15);
    PGASetPrintFrequencyValue(ctx,100);

    PGASetUp(ctx);
    PGARun(ctx, gmodeval);

    /* producing the output for the best string to be returned */
    best_p = PGAGetBestIndex(ctx, PGA_OLDPOP);
    len = PGAGetStringLength(ctx);
    thebest = (float *) malloc(sizeof(float)* len);
      for (i = 0; i < len; i++) {
        thebest[i] = PGAGetRealAllele(ctx, best_p, PGA_OLDPOP,i);
    }

    PGADestroy(ctx);
    free(l);
    free(u);
   return(thebest);
}
```