Integration Platform for Biomedical Image Analysis

Ville Rantanen

Genome-Scale Biology research program, Faculty of Medicine and
Institute of Biomedicine, University of Helsinki

Helsinki, Finland

Academic dissertation

To be publicly discussed, with the permission of
the Faculty of Medicine of the University of Helsinki,
in Biomedicum Helsinki, Lecture Hall 2, Haartmaninkatu 8,
on April 24^{th} at 12 noon.

Helsinki 2015

Supervisor

Sampsa Hautaniemi, DTech, Professor
Genome-Scale Biology research program, Faculty of Medicine and
Institute of Biomedicine, University of Helsinki
Helsinki, Finland

Reviewers

Elina Ikonen, MD PhD, Professor
Department of Anatomy, Faculty of Medicine, University of Helsinki
Helsinki, Finland

Jaakko Hollmén, D.Sc. Tech
Department of Information and Computer Science, Aalto University
Espoo, Finland

Official opponent

Peter Horvath, PhD,
Institute of Biochemistry,
Biological Research Centre of the Hungarian Academy of Sciences
Szeged, Hungary

ISBN 978-951-51-0981-1 (paperback)
ISBN 978-951-51-0982-8 (PDF)
http://ethesis.helsinki.fi
Unigrafia Oy
Helsinki 2015

Contents

List of original publications

Publication I **Ville Rantanen**, Miko Valori, Sampsa Hautaniemi.
Anima: Modular workflow system for comprehensive image data analysis.
Frontiers in Bioengineering and Biotechnology, 2014, 2:25.

Publication II Olli S. Mattila*, **Ville Rantanen***, Jani Saksi, Daniel Strbian, Tero Pikkarainen, Sampsa Hautaniemi, Perttu J. Lindsberg.
Workflow for automated quantification of cerebromicrovascular gelatinase activity.
Microvascular Research, 2015, 97, 19–24.

* equal contribution to work

Publication III Minna Taskinen, Riku Louhimo, Satu Koivula, Ping Chen, **Ville Rantanen**, Harald Holte, Jan Delabie, Marja-Liisa Karjalainen-Lindsberg, Magnus Björkholm, Øystein Fluge, Lars Møller Pedersen, Karin Fjordén, Mats Jerkeman, Mikael Eriksson, Sampsa Hautaniemi, Sirpa Leppä.
Deregulation of COMMD1 is associated with poor prognosis in diffuse large B-cell lymphoma.
PLoS ONE, 2014, 9, 3:e91031.

Publication IV Emma M. Savilahti, **Ville Rantanen**, Jing Lin, Sirkku Karinen, Kristiina M. Saarinen, Marina Goldis, Mika Mäkelä, Sampsa Hautaniemi, Erkki Savilahti, Hugh Sampson.
Early recovery from cow's milk allergy is associated with decreasing IgE and increasing IgG4 binding to cow's milk epitopes.
Journal of Allergy and Clinical Immunology, 2010, 125, 6:1315–1321.

Publications included in other thesis

Publication IV was included in the thesis of Emma Savilahti (Cow's milk allergy and the development of tolerance, Helsinki 2010).

Author's contributions

Publication I: "Anima: Modular workflow system for comprehensive image data analysis"

The author of this thesis designed and implemented the Anima workflow system on top of the Anduril platform. All components of the system except the Fiji interoperability were written by the author. The author executed the analysis and interpreted the results. The article was written by the author.

Publication II: "Workflow for automated quantification of cerebromicrovascular gelatinase activity"

The author designed and implemented the computational part of the workflow and measurement metrics. The part is displayed as "B" in the Figure 2 of the article. The author analyzed images and performed data analysis. The parts describing the image analysis were written by the author.

Publication III: "Deregulation of COMMD1 is associated with poor prognosis in diffuse large B-cell lymphoma"

The author of this thesis designed and implemented the computer aided k-NN color segmentation method used in this article. The author also analyzed the images using the method. The part explaining the quantitative image analysis was written by the author.

Publication IV: "Early recovery from cow's milk allergy is associated with decreasing IgE and increasing IgG4 binding to cow's milk epitopes"

The author designed, implemented and performed the data import, quality assessment and image analysis. The shape filtered segmentation method for finding regularly shaped objects was developed by the author. The section "Bioinformatic analysis" excluding the decision tree part was written by the author.

Thesis contributions

The unpublished segmentation methods: computer aided k-NN color segmentation and shape filtered segmentation were both developed further by the author.

The author designed and implemented two visualization tools to be used as standalone programs and to be integrated in Anima. Qalbum: the tool to create easy-to-browse image galleries and NiceCSV: a tabular data browser and formatter.

Abbreviations

API	Application Programming Interface
CAS	Computer Aided Segmentation
CMA	Cow's Milk Allergy
COMMD1	Copper Metabolism (Murr1) Domain Containing 1
DLBCL	Diffuse Large B-Cell Lymphoma
FFPE	Formalin-Fixed Paraffin-Embedded
GFAP	Glial Fibrillary Acidic Protein
GFP	Green Fluorescence Protein
GUI	Graphical User Interface
H&E	Hematoxylin and Eosin
HPF	High-Power Field
IHC	Immunohistochemistry
k-NN	k-Nearest Neighbors
NeuN	Neuronal Nucleus
RGB	Red-Green-Blue
ROI	Region of interest
SI	System Integration
STED	Stimulated-Emission-Depletion
TMA	Tissue Microarray
vWF	von Willebrand Factor
XP	Extreme Programming

Abstract

Images provide invaluable information to Biomedicine. Especially, microscopy as an information source has been providing knowledge for research and clinical diagnostics. We have moved away from simply looking at the images to quantifiable computerized image analysis. Over the last decades, image analysis developers have prepared algorithms and software to address various scientific enquiries using images. These software are often created for a single purpose. Naturally, not even the most generic software can include all the algorithms ever created. From an image analysis developer point of view, the choice of software creates limitations. It limits the developer to the algorithms included and to the language it was developed in. Even if the software is modular and extendable, a specific language is required and the earlier algorithm implementations would have to be ported.

This thesis presents an integration platform for image analysis: Anima. It is capable of using existing software and including them in analysis workflows. Since image analysis is very case specific, custom processing commands are frequently needed. Anima comes with a large number of data and image analysis components developed directly for the platform, as well as components that send custom commands to the integrated software. All of the components can be executed in a single analysis pipeline.

Anima itself is built on top of Anduril, another software, inheriting its software architecture. Anduril gives Anima the power of parallel processing and rerun prevention mechanism, speeding up the development cycle of new algorithms. The usability of Anima for method development is shown by implementing new segmentation algorithms and visualization tools. The tools and methods are all suited to large data sets. To display the modularity, the tools are published as separate programs that are then integrated in Anima.

The usefulness of the platform is shown by applying it in different biomedical research settings. The settings include different organisms: human, rat and nematode; different sample material: brain tissue, lymphatic nodes and serum; and different medical interests: cerebral ischemia, cancer and allergy.

Anima is a versatile open-source image analysis platform, that encourages the use of best practices of programming habits. It makes the development of analysis workflows and individual algorithms more efficient.

Tiivistelmä

Kuvantaminen on tärkeä tiedon lähde lääketieteelle. Erityisesti mikroskopia on tärkeä kuvapohjaisen tiedon tuottaja biolääketieteellisessä tutkimuksessa. Tietokoneteknologian ansiosta emme ole enää riippuvaisia ihmissilmistä kuvien tulkitsijana. Viime vuosikymmeninä kuva-analyysien kehittäjät ovat luoneet algoritmeja ja kokonaisia ohjelmistoja kuvien hyödyntämiseksi tieteellisiin tarkoituksiin. Useimmat näistä ohjelmista tehdään yhtä tutkimuskysymystä varten. Edes yleisluontoiset ohjelmistopaketit eivät voi sisältää menetelmiä kaikkiin tarkoituksiin. Kuva-analyysikehittäjälle ohjelman valinta luo myös rajoituksia. Ohjelmistoalustan valinta rajoittaa uusien algoritmien kehittäjän käyttämään vain alustan omaa ohjelmointikieltä. Modulaariset ohjelmistot eivät aina vapauta kehittäjää kielivalinnasta. Aikaisemmin julkaistujen algoritmikirjastojen käyttö vaikeutuu, koska todennäköisesti ne pitäisi ohjelmoida uudelleen toisella kielellä.

Tässä väitöskirjassa esitellään sovelluksia yhdistävä kuva-analyysin kehitysalusta: Anima. Se osaa ajaa muita kuva-analyysiohjelmia ja sisällyttää ne yhteen sulavaan kokonaisuuteen. Animassa itsessään on suuri määrä tiedon ja kuvien analysointikomponentteja. Sen lisäksi Animaan voidaan yhdistää ulkopuolisia ohjelmia suorittamaan kuva-analyysin osuuksia, esimerkiksi silloin kun haluttu osuus on jo ohjelmoitu valmiiksi toiseen ohjelmaan.

Anima on rakennettu toisen alustan, Andurilin päälle. Siten Anima käyttää suoraan Andurilin ohjelmistoarkkitehtuuria. Anduril hyödyntää resursseja järkevästi: se osaa rinnakkaisprosessoida, eikä se suorita jo ajettuja analyysin osia uudestaan. Animan hyöty kuva-analyysin kehitystyössä näytetään esittelemällä kaksi suurten kuvamäärien segmentointiin soveltuvaa algoritmia ja kaksi visualisointityökalua, jotka ovat kaikki kehitetty joko Animan avulla tai sen suoritettavaksi.

Tämä väitöskirja esittää Animan edut lääketieteellisen kuva-analyysin työkaluna. Animaa on käytetty analysoimaan kuvia näytteistä, jotka ovat otettu eri eläinlajeista: ihmisestä, rotasta ja sukkulamadosta. Näytteistä on kuvattu eri kohteita: aivokudosta, imusolmuketta ja veriseerumia. Tämän lisäksi tutkimuskysymykset vaihtelevat eri lääketieteen alojen välillä: iskemian, syövän, sekä allergian.

Kaiken kaikkiaan Anima on moneen kykenevä avoimen lähdekoodin analyysialusta. Sillä on tehokasta kehittää uusia kuva-analyysialgoritmeja ja -työnkulkuja. Modulaarisuutensa ansiosta uudet algoritmien toteutukset ovat myös käytettävissä muualla kuin Animassa itsessään.

1 Introduction

Our vision is the sense we tend to trust the most [1]. Since all the senses are governed by the brain, they are all susceptible to illusions. Sounds can influence our sight [2] and *vice versa* [3]. While great scientific breakthroughs have been achieved with vision alone, for example, the cell described by Robert Hooke using his primitive microscope in 1665 [4], we now know our vision can be tricked easily. The Martian canals reported by Schiaparelli in 1877 and, more recently, the face on Mars (Viking 1 mission, 1976) are great examples of misinterpreting images with major influence on our scientific understanding.

It is easy to agree that using information provided by imaging in the sciences should not be influenced by illusions. The interpretation should be accurate, repeatable and not altered by human errors. These properties are exactly what computers can deliver us. Image analysis tries to mimic the *trained eye* with computer software. For example, a trained researcher may count cells in a microscope view. The task can be taught to a computer and repeated without ever tiring.

Since the birth of image processing, the biomedical field has been one of the fields of its application. Throughout the decades, many algorithms have been published, along with software implementing them. Today it is growingly difficult to choose the software to use, since all of them provide a selection of different algorithms [5]. In the worst case, analysis developer might resort to using several programs, transferring the data from one to another at each processing step. Each manual data transfer from one program to another is both delaying analysis and prone to creating errors. Programmers wishing to develop more algorithms are limited by the language of the platform they choose. Even if a modular and extensible platform is found, usually the modules need to be programmed with specific languages [6]. All the existing implementations of working methods would have to be ported to that language. Porting code often results in slightly different operation [7]. After all, scientific research must be reproducible – a published method is intended to be used as it was published.

As the number of scientific image analysis implementations increases, there is a growing need to use them together efficiently. The implementations themselves should be developed with better interoperability [8], but the problem can be addressed from above too. Software integration platforms are a class of software that join existing software together, providing them with standard file formats and other mechanisms to communicate with each other [9].

The goal of this thesis is to develop an image analysis platform that can use existing software, picking the most suitable algorithms from each, independent of the programming language. The platform provides a development environment, where the development of new algorithms is convenient and efficient.

The publications for this thesis were selected to display the different uses of the image analysis platform introduced here. The sample material varies from synthesized data, rat and human tissue to serum and even whole nematodes. The biological interests are selected with a broad scope too. The platform shows its capabilities with cerebral ischemia, cancer detection and allergy testing.

The structure of the thesis is as follows: First, to explain the data sources, microscopy is reviewed. Then, to understand the function of image analysis programs, basic image processing is introduced. A chapter on software development finishes the literary review.

The materials and methods used in the publications are explained, and further, an example of each type of image data used in the studies is presented. The results of the thesis are viewed from the platform development point of view. Thus, the results shown in this thesis are not necessarily the main results of the publications. Finally, the implications of the results are discussed.

2

Microscopy is a generic term for any optical system with the intention to see something too small for the naked eye. While a large number of innovations in microscopy exist, this thesis concentrates on one of the most common techniques: visible light microscopy and immunochemical labeling.

2.1 Microscopy in the early years

The advancement of modern medicine and biology was accelerated by the invention of microscopes. In 1665, Robert Hooke extensively explored the microscope in his book Micrographia [4]. Even today, a standard light microscope follows the same principle as Hooke's: A light is beamed through a sample and the transmitted light is viewed via magnifying optics, as presented in Figure 2.1. The tools to build the optics have improved since, increasing the resolving power up to a limit. The theoretical maximum resolution was reached already in the late nineteenth century.

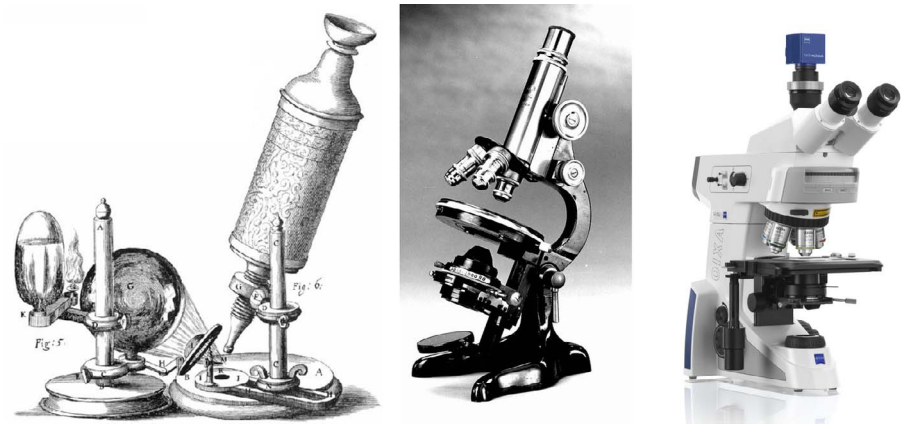


Figure 2.1: A Montage of microscope development. The seventeenth, the nineteenth and the twenty-first century light microscopes work with the same principle. Image credits: Cropped from *Robert Hooke, Public Domain. NIH/DeWitt Stetten Jr., Museum of Medical Research, Public Domain. ZEISS Microscopy, Creative Commons Attribution 2.0 license*.

Using his microscope, Hooke discovered plants were made of cells. Hooke's and the modern microscopes are, however, unable to resolve the structure of mammalian cells. Mammalian cells are small, but they vary greatly in size. For example, the human cell diameter is typically between 2 and 120 μm and on average 40 μm [10, 11]. The cells found in organisms are typically spherical or cylindrical (*in vivo*), but when grown on a dish (*ex vivo*), they tend to grow flat – their height is not much larger than the nucleus with a diameter of few micrometers [12]. The density of the contents of the cells is close to the density of water [13]. The small size and the density makes light impervious to single cells. If a beam of light goes through a thin layer of water, it is almost unchanged when it hits the eye of the observer. Similarly, a beam of light hardly interacts with a single layer of cells.

In addition, Abbe's law of diffraction states that a microscope can not resolve details smaller in length than $d = 0.5\lambda/NA$, where λ is the wavelength of light and NA is the numerical aperture of the microscope objective [14]. For visible light wavelengths, the limit is approximately 0.2 μm, which is enough to recognize cell nuclei. However, without any contrasting methods or labeling, a single mitochondrion of diameter 0.5 μm is difficult to observe even in theory (BNID 110892[15]).

2.2 Chemical staining of tissue

Hooke's microscope was a bright-field microscope. It means that white light is transmitted through and absorbed by the sample before viewed by the eye. Staining of the sample will increase the absorption of light increasing the perceived contrast. To view the elusive cells, the staining of cell compartments was introduced. An especially popular early staining method, the hematoxylin and eosin (H&E) staining, is still widely used. These two staining chemicals color the nuclei blue, and the cytoplasm and connective tissue pink (see Figure 2.2). This staining combination is popular in histopathology. For example, pathologists have learned to differentiate the morphology of healthy or diseased tissues by using H&E. The H&E-type of staining can be called unspecific staining, since it stains a collection of cell organelles at the same time, with the same color.

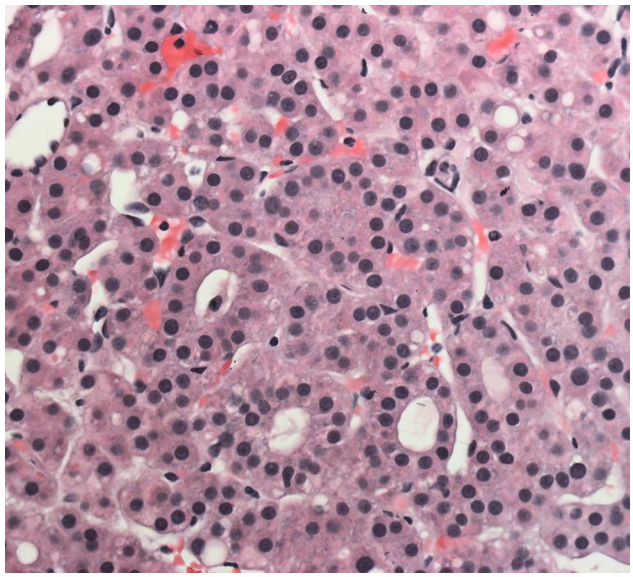


Figure 2.2: H&E staining of Hepatocellular carcinoma. H&E colors the nuclei blue, and the cytoplasm and connective tissue pink. Image credits: *Dr. Mitchell Wachtel, University Medical Center, Lubbock, TX. Creative Commons Attribution-Share Alike 4.0 Unported license.*

To learn more in detail the causes of diseases, it is not enough to stain nuclei and the rest of the organelles with two different colors. The specific cellular compartments or even single proteins need to be stained specifically and separately. The specificity gives the researchers for example the key to finding differences between patients and healthy controls in the distribution of certain proteins in their tissues.

The staining of specific targets through antibodies is called immunohistochemistry (IHC). The first IHC study was reported in 1941 by Coons and his colleagues [16]. An antibody is a protein secreted by B-cells. The immune system uses antibodies to identify proteins or other structures, for example, to recognize potentially harmful foreign agents. The specific antibody targets are called antigens. Antibodies can be engineered to bind to an antigen of choice, thus allowing researchers to choose a specific target to measure and study.

A common way of performing IHC uses a secondary antibody to attach the label, as shown in Figure 2.3. First, the protein of interest, the antigen (A) is selected and a primary antibody (B) specific for the protein is selected or developed. The secondary antibody (C) carrying a dye (D) binds to the primary antibody (B). The primary-secondary antibody structure allows flexible labeling of multiple targets. The same secondary antibody and dye can be used to bind to various targets without the costly research of finding a working dye-antibody combination for each antigen.

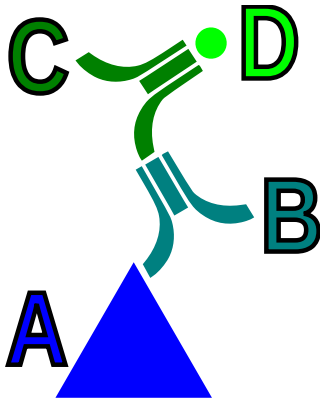


Figure 2.3: Labeling a protein of interest. A secondary antibody (C) carrying a dye (D) binds to the primary antibody (B). The primary antibody (B) binds to the antigen, or the protein of interest (A).

It is important to understand that labeling techniques do not make the proteins of interest visible directly. We can only perceive them via multi-layered indirect methods. Each of the layers require a complex protocol of attaching antibodies and the dyes. The antibodies must find their antigens and survive several washing steps. IHC staining can provide us with a great deal of specific information, but the dye is not the same thing as its target.

2.3 Fluorescence microscopy

In the IHC study by Coons *et al.* they already used a fluorescent microscope instead of a traditional transmitted light microscope. Fluorescence adds important properties to imaging. Although imaging with fluorescent dyes adds complexity to the experiment, the added values are greater than the possible error sources. In a fluorescence microscope, the wavelengths of the light source are completely filtered out, leaving only the faint light emitted by the dye attached to a protein of interest. This way a fluorescence microscope can show only the labeled target removing everything else from the picture.

2.3.1 Fluorescence as a phenomenon

Fluorescence phenomenon starts with a photon hitting an electron in an atom. The electron receives energy and is excited to a higher energy level. The energy is then released a few

microseconds later. As the energy is converted to other forms in the process, the emitted photon is of lower energy, or in other terms, of longer wavelength than the original. The effect of energy loss is called Stokes shift [17]. In a simplified example, a short wavelength blue excitation photon turns in to a longer wavelength green emission photon. The process is displayed in Figure 2.4.

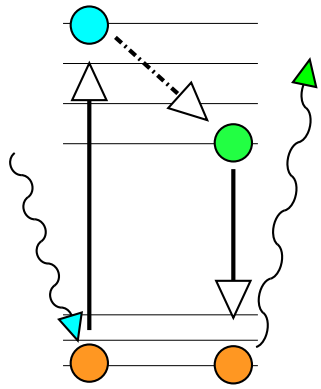


Figure 2.4: A Jablonski diagram: A high energy photon excites an electron at a low energy level to a higher level. Energy is lost in the process. Later, the electron returns to lower state, emitting a lower energy photon. The horizontal lines represent possible quantum energy states.

2.3.2 Imaging fluorescence

To image proteins or other targets found within the cell, the targets must be first stained with a fluorescent dye. The staining procedure follows the IHC mechanism introduced earlier. The most famous fluorescent protein is the green fluorescent protein (GFP). It was found in nature in the bioluminescent jellyfish by Osamu Shimomura [18]. Martin Chalfie later managed to insert the GFP producing gene in *E. coli* bacteria [19]. Finally, Roger Tsien studied the structure of the protein and engineered it to better suit microscopy and imaging [20]. For these accomplishments, the three were awarded the 2008 Nobel Prize in chemistry "for the discovery and development of green fluorescent protein".

Fluorescence phenomenon changes the wavelength of the light, but in practice, the emitted light is also much lower in intensity than the excitation light. Partly, it is due to the small size of the electron – huge amount of photons are needed for even one of them to hit a suitable electron. With the naked eye, it is impossible to see the emission amidst the excitation light. Fluorescence microscopy copes with it by presenting emission filters. The light emitted from the sample is directed through a filter that fully blocks the excitation wavelengths, but passes on the emission wavelengths. Usually the excitation light itself contains the wavelengths of the emission and they must be first filtered out not to be mixed with the emission. For this purpose, the excitation light is filtered first with an excitation filter that removes the emission wavelengths. The flow of light in a microscope is often presented with a light path diagram, as seen in Figure 2.5. A micrograph imaged with a typical fluorescence microscope is shown in Figure 2.6.

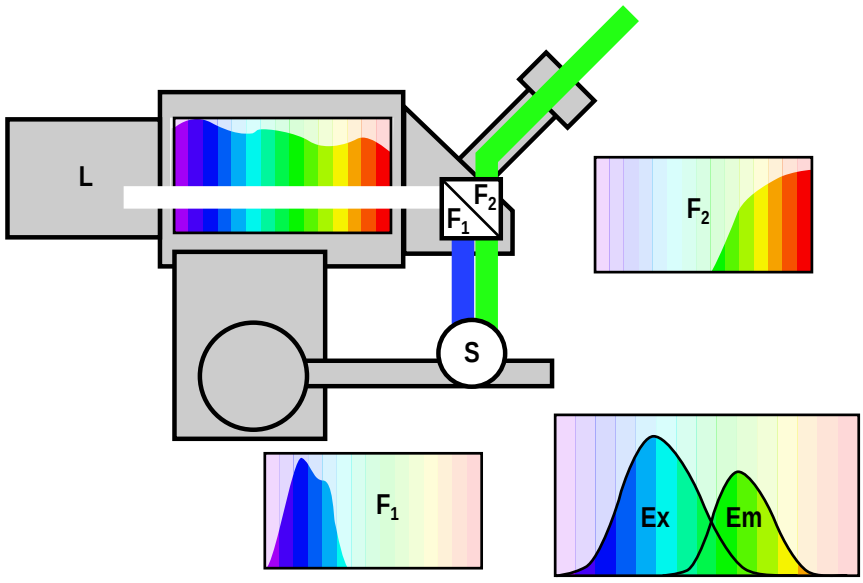


Figure 2.5: The light path of a typical fluorescence microscope. White light from the light source (L) is directed to a filter bank (F_1\F_2). Excitation filter (F_1) passes on only the short wavelengths needed for excitation. Long wavelength emission light returns from the sample (S) after experiencing Stokes shift. Any reflected excitation light is removed with the emission filter (F_2). The dye in the sample (S) prefers to be excited with specific wavelengths (Ex), and it emits another range of wavelengths (Em). Knowing these spectra is required to select the correct filters for a given dye.

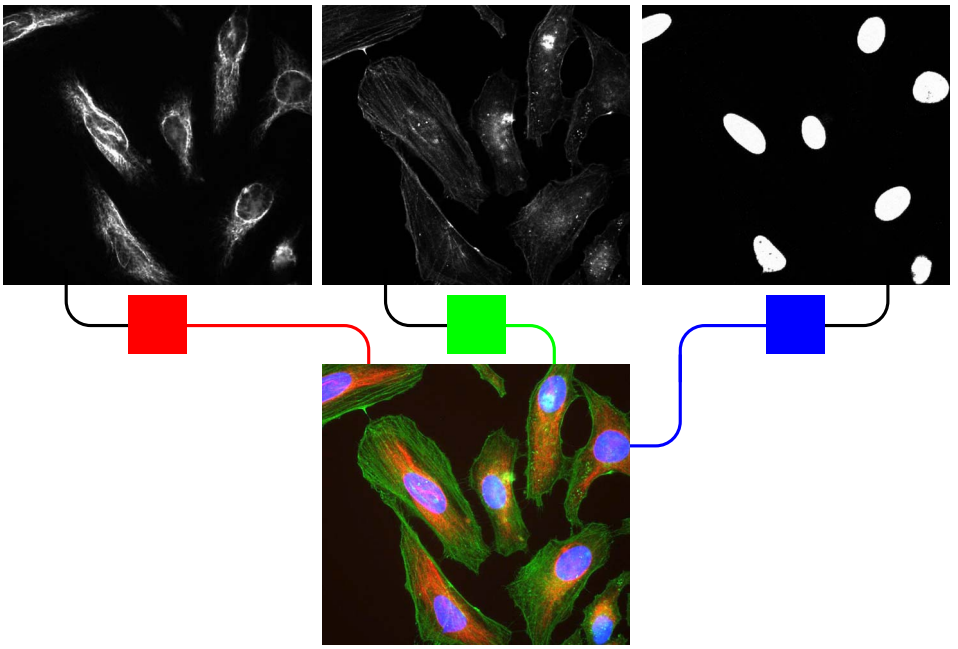


Figure 2.6: HeLa cells grown in tissue culture and stained with antibody to actin (green), vimentin (red) and DNA (blue). Each channel has been imaged one at a time with different filter set. The three images can be colored and merged together with image processing. Image credits: *Gerry Shaw, EnCor Biotechnology Inc., Creative Commons Attribution-Share Alike 3.0 Unported license.*

3

Image processing emerged with digital images – as early as with the Bartlane cable picture transmission system in the 1920s. The early image processing was mainly concerned with signal processing and mostly involved preserving contrast during the transfer of the image. Computer based image processing was introduced to correct the distortions of the television camera on-board the Ranger 7 moon space probe, launched in 1964. It sent over 4000 photographs before impact on the moon. The first of these photographs is shown in Figure 3.1.

Figure 3.1: The moon as seen by Ranger 7 space probe in 1964. Image credits: *NASA, not copyrighted.*

A digital photograph or bitmap, is a matrix of numbers. The elements of the matrix have a special name in the imaging context: picture element or pixel in short. In a simple case, the image is a two-dimensional matrix where each pixel has eight neighbors around it, as in Figure 3.2.

7	8	1
6	0	2
5	4	3

Figure 3.2: Pixel "0" and its 8 neighbors.

3.1 Image sources

In this thesis, the microscope is the source for all images. However, it is important to notice that the actual source of the image does not change the tools used to process and analyze the images. Image processing is a very generic and widely applicable field of research. Figure 3.3 illustrates the genericity. The images in the figure are of very different scales,

from kilometers to micrometers. Even so, we could ask the same interesting question of all of the images, "how many branches are there?" To answer the question, we might even use the exact same algorithms in the analysis.

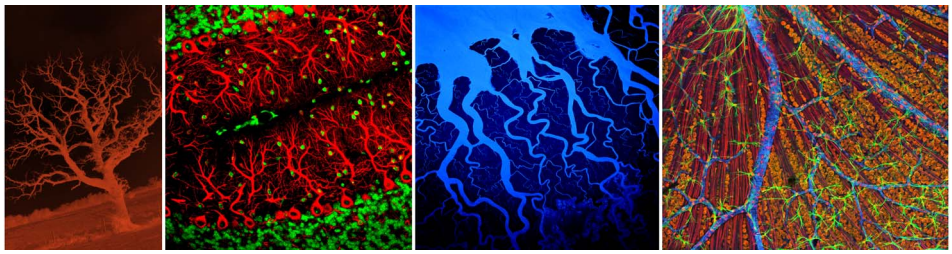


Figure 3.3: Images contain similar properties independent of their source. The question posed to the images of different scales may the same, for example, "what is the number of branches?" Image contents and credits from left to right: A dead tree (processed red tint) by *R Neil Marshman, published in the Creative Commons license.* Purkinje cells in red by *Yinghua Ma and Timothy Vartanian, Cornell University, Ithaca, N.Y.* Ganges River Delta, Bangladesh and India (processed reverse color) by *NASA, released in the public domain.* Mouse retina: glial cells in green, blood vessels in blue, by *Tom Deerinck, National Center for Microscopy and Imaging Research.*

Image processing in practice is very case specific [21]. There is a finite number of processing operations, but they can be combined in an infinite number of ways. Therefore, the processing and analysis starts from the semantic view of the image. We want to understand which qualities of the image are important for a specific question. For meaningful research, the question is thought of first, before setting up the imaging experiment.

Automatic image analysis refers to methods where the decisions are partly made by a machine. Even if the analysis is automated, the original question is set by the human. An example of this is counting cells. Earlier, researchers peered into the microscope, and counted cells by eye. Now, we can count them automatically with image analysis. The analyst must, however, learn how the expert human counts cells, and mimic the process with a computer. This approach is applicable to a wider range of image analysis applications. The analysis is developed to mimic the expert.

3.2 Processing basics

Simple image processing typically consists of one or many of these three types of operations:

1. A single value is calculated from the whole image: Image statistics.
2. Each pixel value is calculated individually: Point processing.
3. Each pixel value is calculated based on one or more of its neighbors: Local processing.

To give an overview on how image processing works, the three image operation types are examined.

Image statistics

Image statistics is a type of image operation where the pixels of the whole image are used to calculate a single value [22]. The value can be, for example, the mean, median or sample variance of the values of the pixels. Image statistics is not image processing specifically, since processing typically results to another image as an output. Operations that produce numerical values interpreted as other than image are called image analysis operations. Image statistics is used, for example, to normalize a set of images to standard mean, or to measure values to be used as parameters in image processing.

Point processing

The group of operations modifying the individual pixels of an image are called point processing [23]. The most familiar of these are the brightness and contrast changes. They are performed by changing the pixel values by addition and multiplication. Point processing is also the key to basic segmentation – a wide topic to be divulged in Chapter 4.

Local processing

Local processing differs from point processing in that it takes the neighborhood in to account [23]. A classic example is an edge detector. To define an edge, we must always consider more than one pixel. We can calculate the mean values of the right side neighbors (Figure 3.2: 1-3) and the left side neighbors (Figure 3.2: 5-7) of a center pixel. If the difference of these means is high, the center pixel sits at an edge. The neighborhood in Figure 3.2 is defined as a three by three square matrix. The neighborhood shape is not limited to squares. It may be of any size and shape, and the origo (the zero pixel) can be placed at any location. Even by considering just the nearest eight neighbors, image analysts can create complex and useful processing tools: shape detection, noise canceling, sharpness enhancement, for example.

3.3 Modeling principle

A common misconception regarding image analysis is that a human should have to see all the images being analyzed. In many cases it is enough that the analysis question is proposed based on a subset of images. The answer can be sought in a larger set of images without viewing each one of them. In mathematics, the equivalent term for the question is a model. If we can create a good model of what a cell looks like, the model can be used to finding cells in any number of images.

Working with models always requires good mechanisms to confirm the validity of the model. Especially in imaging, different kinds of visualizations are a fast way to confirm model behavior, for example the success of a cell finding operation.

4 Segmentation

If the purpose of an image analysis project is to count cells, it is easy to see that the most crucial part of that analysis is to detect the cells. Segmentation is generally the separation of the image pixels into those within objects and those outside objects. In the cell counting case, segmentation would separate the cells and the background. Segmentation is the most crucial part in any image based analysis, since it determines the success or failure of the analysis [23].

Segmentation is often a two-class problem. The pixels are either object or background pixels. Sometimes, however, more classes are needed. Multiclass segmentation can be used to label each pixel to background or multiple types of objects. In addition, multiclass segmentation methods can be used to simplify an image. In some cases, simplification can be used as an intermediate step before the final segmentation [24]. As an example, the pixels may be labeled with numbers from one to four in brightness order, and later it can be specified that the first label is background, and labels from two to four are object pixels.

4.1 Thresholding

The most simple form of segmentation is thresholding. A threshold level is selected and the pixels are labeled object or background pixels, depending on whether the value is greater or lower than the threshold. An example of simple thresholding is presented in Figure 4.1. In manual thresholding, the threshold level is selected by trial and error by viewing the resulting labeled image and adjusting the value.

4.2 Machine learning

Machine learning is an approach to data analysis that helps to categorize data. The approach is divided into two distinct domains: unsupervised and supervised machine learning.

Figure 4.1: Manual segmentation through thresholding. Each pixel is labeled as object or background. Object pixels are white. Image contents and credits: The Endangered Karner Blue, *Lycaeides melissa samuelis. USGS Native Bee Inventory and Monitoring Laboratory from Beltsville, USA, Creative Commons Attribution 2.0 Generic license*

Unsupervised machine learning

In unsupervised machine learning techniques, the data itself is used to define in which category each data point belongs to, although the total number of expected categories may have to be presented *a priori* [25]. Such an approach is also called clustering, and the categories are called clusters. Clustering is the process of replacing data values with their cluster labels.

Supervised machine learning

In supervised machine learning, categorized training data is presented to an algorithm that produces a set of mathematical rules on how the data points were categorized [25]. The categories are usually called classes and the rules a classifier. The training data may be categorized by a human observer, or the data itself can be from a known source. For example, in patient data it is often known whether the sample came from a healthy or diseased source. Uncategorized data points can be classified using the classifier.

Both of the machine learning domains can be used to segment images, since they label data points that somehow belong together. Pixels may belong together because their values resemble each other by intensity, texture or localization. They may also belong together because a training set creates a match for them.

4.3 Automated segmentation

To automate the segmentation process, the decision of selecting the parameters for the segmentation method is given to the pixel data. For example, we could calculate image statistics and use the mean of the pixel values as a threshold level. The values above the mean would be labeled as object and below the mean as background pixels – a very crude and simple unsupervised clustering.

A better solution is to establish a few expectations (i.e. to model) on the pixel value data. To segment an image, we might assume the object pixels are more similar to each other than to the background pixels, and *vice versa*. In mathematical terms: intra-class variance minimization. If the values within a class are similar, their variance is small. As a minimization problem it can be presented as pseudocode:

$$\text{minimize } \mathit{variance}(I \geq t) + \mathit{variance}(I < t),$$

where I is the image matrix and t is the threshold level to be found. Since the image data can not be formally derived, the problem turns out to be a numerical optimization problem. A common solution for the optimization is the Otsu method, which proves that minimum intra-class variance leads to maximum inter-class variance, which can be quickly calculated by using the histogram of the image [26]. The results from the two automatic thresholding methods, above-the-mean and Otsu, are displayed in Figure 4.2.

The simple intra-class variation minimization approach has many variations to improve the accuracy. Figure 4.2 displays the standard global variation, *i.e.* it calculates a single value over the image. Typically, the global thresholding methods have their local and seeded variants. In local thresholding the image is cropped into small fields and the value is

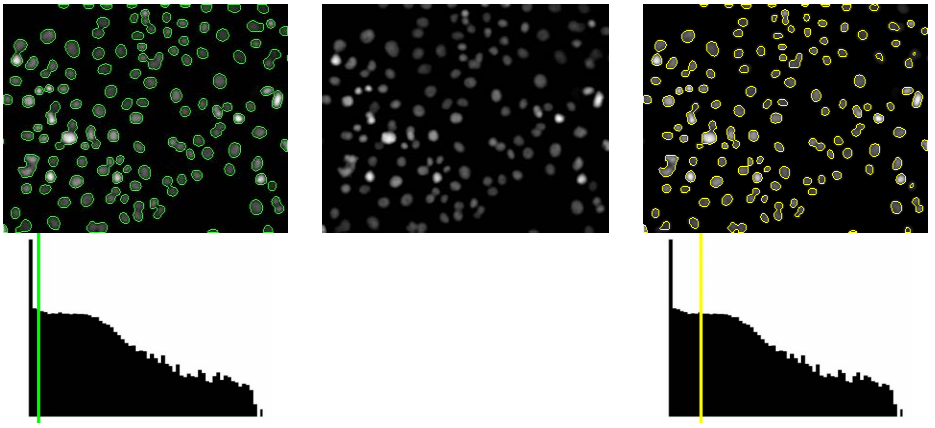


Figure 4.2: An example thresholding of DNA stained cells. The histograms are in logarithmic scale to better display the intensities. The green line is at the mean of values. The yellow line is the level of minimum intra-class variance. Above the histograms are the corresponding thresholded images. The mean value threshold is worse than the minimum intra-class variance, since it fuses more cells together. Image contents and credits: The center image contains Human HT29 colon-cancer cells, from the benchmark set BBBC001v1 [27]

calculated separately in each, helping the segmentation if a part of the image is darker than another. The seeded variations work by first making a rough estimate on the segmentation and then fine-tuning the result with another round of segmentations for each object. In addition, there are iterative methods for segmentation. For example, active contours are a group of methods that make the segmentation more accurate by applying a model for each separate object. The models are set to modify themselves iteratively, until the best fit to the object is found [28, 29].

4.4 Computer aided segmentation

No computer model can fit to the nature with 100% accuracy. If attempted, the model should contain all the information in the universe, which is impossible. The same issue appears with automated segmentation. With some work, an algorithm can be tuned to find 90 – 99% of the objects, but there will always be some inaccuracy. The final one percent may be years of work, in which time a completely manual method would have solved the problem.

Computer aided methods refer to semi automation. Specifically, Computer Aided Segmentation (CAS) is automated segmentation that uses a human expert to help the algorithms. Traditionally, there are two ways of performing CAS: The user is given an automatically segmented image, which is corrected manually, or the user gives clues on the segmentation, which are then used to automatically segment the images. CAS can be more accurate than fully automated methods, but it relies on the objectivity of the human. Figure 4.3 displays an example of automated and aided segmentation. The human factor provides semantic information to the picture. As two colors may be close to each other in mathematical terms, they may have a very different semantic meaning. For example, foliage and frogs are both green, sunflower petals are yellow. Semantically, sunflower petals and its green leaves are closely related – closer than the frog to either.

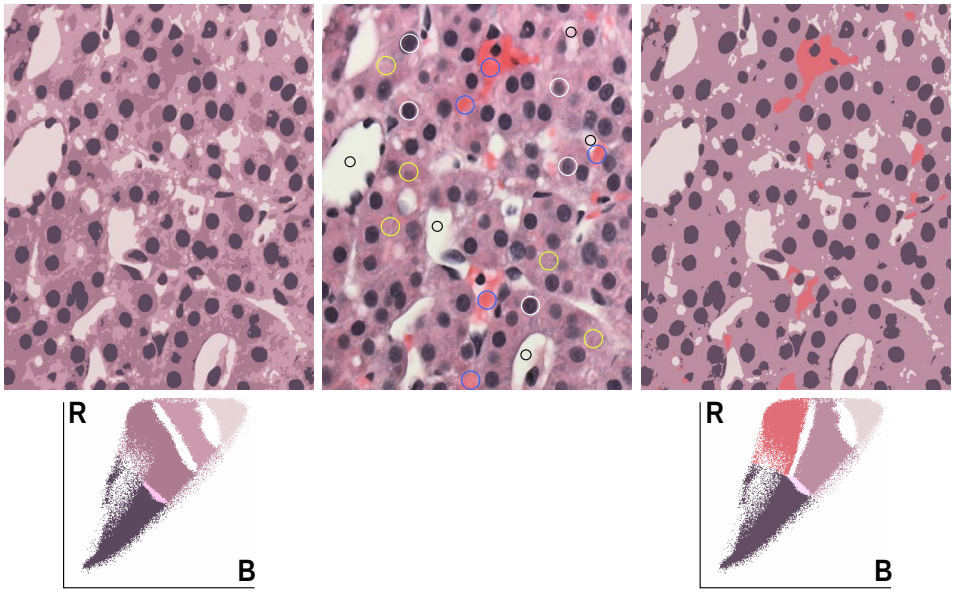


Figure 4.3: Unsupervised and supervised segmentation of an image. The center image contains an H&E staining of Hepatocellular carcinoma imaged with a microscope. A human observer has indicated example colors in the image: background with black, nuclei with white and different types of tissue with yellow and blue circles. The four-color versions at the sides are the segmented results. The scatter plots underneath display the pixel values in blue-red axis, omitting the third, green, axis. The version on the left is segmented with unsupervised k-means, in which cluster centers tend to evenly space out in lack of obvious clusters. On the right, the spots of the image at the center are used for training, and the rest of the pixels are classified with nearest neighbor matching. The supervised segmentation matches more closely to our semantic view of the image: the colors that differentiate nuclei and cytoplasm are separated and the two shades of pink are distinct. Center image credits: *Dr. Mitchell Wachtel, University Medical Center, Lubbock, TX. Creative Commons Attribution-Share Alike 4.0 Unported license.*

4.5 Features

Once the objects of interest are segmented, they are measured for features. Features are a generic term for anything measured from any object, or even the whole image. Features can range from simple mean intensity measurement to multi-valued texture features, model fitting parameters, and so on. To be specific about which kind of features are discussed, the kind should be declared. These may be for example object features, image features or texture features.

By far the most commonly measured feature in fluorescence microscopy is the mean intensity within an object. Since each fluorescent molecule binds to a specific protein, the more protein at any location, the brighter it will shine under excitation light. Therefore, the intensity value represents the relative amount of a specific protein. An example of intensity feature measurement is shown in Figure 4.4. If two or more fluorescent labels are imaged at the same time, the intensities of the different colors can be used for colocalization analysis. Colocalization means that two or more targets are expressed in the same place. Often, colocalization is used as a prerequisite to interaction between two proteins, since proteins need to be very close to each other to interact.

Common morphological (shape related) features are the area of an object, its eccentricity (how elliptical an object is) and roundness (perimeter smoothness). Eccentricity of an ellipse is commonly denoted as $e = \sqrt{1 - b^2/a^2}$, where a is the semi-major axis length and b is the semi-minor axis length. If $a = b$, the shape is a circle and its eccentricity is zero. For an infinitely long major axis, or a line, eccentricity is one. Since objects are not typically perfect ellipses, the best fitting ellipse is used. Roundness is defined as $r = \sqrt{4\pi A/p^2}$, where A is the area of the object and p is the perimeter length. For a perfect circle, the radius and the area are linked by equation $A = \pi r^2$. Further, the radius and perimeter length (circumference) are related by $p = 2\pi r$. Joining the previous gives the area of circle by perimeter alone: $A = \pi(p/2\pi)^2 = p^2/(4\pi)$. The roundness of a circle is therefore $r = \sqrt{4\pi A/p^2} = \sqrt{\frac{4\pi p^2}{4\pi p^2}} = 1$. When the perimeter length increases in comparison to the area, roundness decreases towards zero.

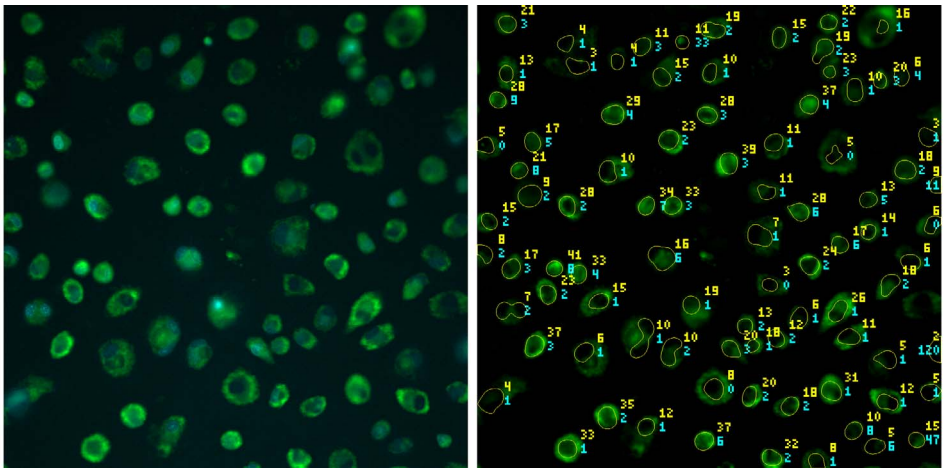
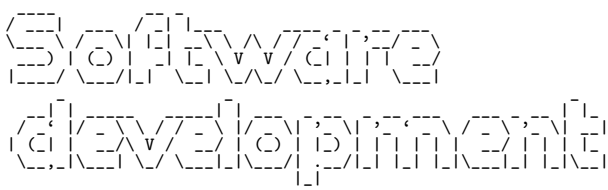


Figure 4.4: Cells stained to display the nucleus in blue and a protein of interest in green. On the right, the nuclei have been marked with a perimeter line. The yellow value represents the intensity within the nucleus. The cyan value is the multiplier of intensity at the immediate outside of the nucleus. A nucleus with high amount of protein inside the nucleus and no protein outside has a high multiplier.

The more complex features measured from images, such as texture features, may be extremely helpful for machine learning approaches. They can be used, for example, to classify the cycle of individual cells. The texture feature values themselves are often very difficult to interpret biologically. A high value in a texture feature vector does not correspond to any biological function but, rather, describes what the object looks like as an abstract vector of numbers. In this thesis, the focus is on the kind of features that have the potential to explain a biological or semantic property.

5



Software has been developed for as long as there have been computers. This statement is a natural one, since computer hardware does not do anything without the software. The decades of software development have taught us a great deal about the best practices of development. Especially the business world has advanced the development of programming methods, since the minimization of resources spent for a software project is crucial for business value. In addition, code quality is an important issue, since returning to the code to repair errors later on is a costly operation. Development practices matter in the academic world too. Ensuring high quality of code is important in preventing incorrect implementations of algorithms that produce erroneous results [30, 31]. In addition, good development practices make programs easier to maintain and to add more functionality later, even by people other than the original developer [8].

5.1 Development of data analysis software

Scientific software are developed to improve domain understanding [32]. The specifications of the software are not often received from clients, but from the developers themselves. Depending on the specification, the developer may have to address problems at very different levels of computing, each requiring their own specialized expertise [33].

Here, the algorithm, platform, user interface and analysis development are defined as separate levels of development. Each of the levels include design and implementation. Often in small projects, the algorithm, platform and user interface development are all done by a single person or a small group. The analysis developer is considered to be the end-user of the platform. In a modular environment, each of these tasks can be developed by different people. For instance, research group *A* creates an algorithm, which is then implemented in a platform developed in group *B*, to which person *C* creates a user interface. Finally, data analyst *D* uses the platform to produce results for a study.

The implementation of any level should also contain the testing of the software. The testing of scientific software differs from the commercial. Commercial software are often implementations of a known model, and they can be validated against real data. When developing something completely novel, there are no *oracles* who know the absolutely correct outcome [34].

Algorithm development

Algorithms are at the lowest level of analysis software development. Low level refers to being closest to the actual data. As an example, an implementation could include the reading of a tabular data file, clustering using certain columns of the data, and writing an output. Algorithms are often published as libraries or executable binaries, without publishing the platform used to call them.

Platform development

A platform facilitates analysis by providing an engine that calls and runs the algorithms. A platform developer creates ways the computer resources are allocated and when to run individual parts of the analysis [35]. The platform may be a generic programming language, such as the popular scientific calculation language R [36]. Alternatively, it may be a platform with a specific purpose. For example, CellProfiler is an image analysis specific platform with a menu driven user interface and a selection of analysis modules addressing a wide range of situations [37].

User interface development

Typically, the platform includes a user interface internally. However, if the platform itself is a library, the user interface development can be externalized as a separate project.

A user interface is developed to allow analysis developer to communicate the chain of tasks to the platform. The interface also conveys success or error messages back to the analysis developer. The user interface may be a graphical point-and-click type (*e.g.* CellProfiler) or a simple configuration text file (*e.g.* R).

Typically, user interface development is about making each of the platform's capabilities available. This is often the difficulty with graphical user interfaces (GUI) – as new capabilities are added, they have to fit within the menus or configuration screens, gradually filling the view until a menu hierarchy has to be added. Scriptable platforms do not suffer from such a dilemma: the user interface for a scripted platform is often automatically generated. From a developer point of view, scripted environments are more desirable than graphical user interfaces [38].

Analysis development

To develop analysis, one needs working algorithms, the knowledge on how to use them and how to communicate them to the platform through the user interface. The analysis pipeline itself is a chain of algorithms and tools to find answers using data. An example could be a chain of:

- defining data source,
- transposing the data,
- selecting columns for clustering,
- calling the clustering algorithm,
- joining the original values and the clusters,
- plotting data values using separate colors for each cluster.

The developer needs to find appropriate tools and make them work to accomplish the chain. A good user interface is the key to make the task easy. The platform decides how the computer infrastructure is used. Finally, it is up to the performance of the algorithm implementations to complete the analysis quickly and accurately.

5.2 Software system integration

As a common programming practice, libraries created by developers are used to make programming less tedious. Libraries typically contain procedures that are often used and are standard enough to be used in many other projects. With libraries, the developer does not have to reinvent all of the functions of the program for each project. Using libraries is part of a greater programming paradigm: software reuse [9].

More generally, the library approach can be broadened to full software. System integration (SI) is a process that joins different subsystems or components as one large system. It ensures that each integrated subsystem functions as required.

In this thesis, the subsystems for SI are other full software packages. Here, software system integration is defined as a term for building a commander software that uses worker software like a common program uses libraries. Typically the integrated worker software do not have a proper application programming interface (API) for the commanding software. Therefore the developer may have to use code generation or reverse engineering to access the worker software.

To enable integration, the worker software needs to be either callable or scriptable. A callable worker software means that it can be configured from the command line using switches or it has an API library which can be used directly within the commander software. A scriptable worker software has parameters that need to be set by writing a configuration file. The file can be created by using source code generation.

Source code generation means that a piece of program generates code to be run in another programming environment [39]. For example, a Bash code can generate Python code. A line printed in Bash is executed in Python to produce the "Hello World!" text:

```
# WHAT="World"
# echo 'print("Hello '$WHAT'!")' | python
Hello World!
```

5.3 Example platforms and programs

There are many software platforms that can be used for image analysis, and some that are specifically developed for it. Further, there are programs built for single purpose image analysis, such as detecting the blood vessels in a retina [40], or live cell phenotype profiling [41]. Single purpose software may have multiple algorithms implemented, but they are tuned to solve only one type of a problem and their user interfaces are focused on showing only the relevant parameters for the problem. Generic platforms can be used to solve many different kinds of problems, but the user needs to learn which algorithms can be used for the specific problem and how to interpret the question to the platform through the user interface.

Image analysis platforms

Single purpose image analysis programs are often controlled with a graphical user interface and they are rarely scriptable, or controllable in other ways. It is often hard to include them

in automated analysis. Therefore, when integrating image analysis platforms the focus is on the few scriptable platforms.

CellProfiler is a high content screening oriented image analysis pipeline engine with a graphical user interface [37]. The user can select modules from a list to perform various image processing and analysis tasks. By default, the user is shown each step performed in the analysis, removing the need to separately visualize the steps. The pipeline engine can be started without the GUI, making it suitable for integration, but on the other hand the pipeline configuration files are hard to generate anywhere else than in the user interface of the platform itself.

Fiji [42] is a modified version of ImageJ [21] that is a very generic image processing and analysis toolbox with a graphical menu driven user interface. It is aimed at processing single images, although it can be used for larger image sets through macros. In addition, Fiji adds to ImageJ a scripting environment, which makes it possible to control the platform for example with Python language. The programming language interface makes Fiji a good program for integration.

ImageMagick [43] is solely an image processing platform lacking analysis features. It is mainly a command line tool, making it exceptionally easy to integrate. The lack of higher level object detection tools and feature measurement makes it only usable in preprocessing of images.

Generic platforms

Any programming language with bindings to image reading libraries are eligible for integration. Probably the most used languages in scientific image analysis are R, Python, Perl and the commercial Mathworks MATLAB. R language [36] has the EBImage [44] library which can be used in conjunction with the wide range of machine learning libraries of R. Python has a selection of basic image handling libraries, like the Python Imaging Library. In addition, there are even high level analysis libraries for machine vision applications, such as the OpenCV library. Perl is often used for its bindings to the ImageMagick library. Mathworks MATLAB can be appended with an image processing toolbox. It has low and high level functions for image analysis and it is very popular for developing new algorithms.

Integration platforms

An integration platform can be any programming language that is capable of launching processes. However, there are some programming languages that are specifically designed to launch processes making them better choices as the integration platform. Such languages are for example Python and Perl. In addition, environments that are almost solely created for launching processes, such as Bash and the Make Utility, can be used as an integration platform. In addition, there are tools for programming language interoperability, such as the Babel project [45]. These platforms are low level and provide only the means of starting a process or calling source code. Any higher level data handling capabilities and user friendliness have to be built later on.

There have been a few attempts to create a more intelligent way of handling the integration, with better handling of computing resources and code reuse. Some of them are built with a

GUI, like Chipster [46] and Galaxy [47], and some rely on text file scripts, as do Swift [48] and Anduril [49].

Anduril is a rapid development environment that minimizes porting errors by using original implementations. It uses smart resource management to prevent rerunning of analysis steps that have already been executed, and it can parallelize the analysis steps to speed up processing. Anduril is equipped with hundreds of data processing and analysis tools to establish generic scientific computation workflows.

5.4 Extreme programming

Agile software development methods are a group of methods in which continuous evolution of the requirements and solutions are applied in an iterative development cycle. Extreme Programming (XP) is one of the agile methods. Agile methods are recent – the first XP project was started in 1996 by Kent Beck [50]. It is a programming paradigm that provides a number of strong tools to cope with complex software projects. The name refers to using the best practices in an extreme fashion.

The core of XP is the customer-developer relationship. It requires the customer to closely work with the developers, preferably sitting in the same room. The customer and the developer have their roles, well defined responsibilities, and they should communicate on a daily basis. The customer informs the developer on the business value, while the developer educates the customer on the effects and costs of adding software features.

The academic world does not often compete with business value or rapid development, but rather with best solution. Even so, better programming practices in academia would help to create better software [51, 8, 52]. Scientific software developers themselves admit they do not know enough about practices such as software testing and validation [53]. However, according to another study, the scientific community seems to accept agile (e.g. XP) methods more readily than the business world [38].

As an example, XP can be converted to biomedical research environment by replacing the customer with a biology researcher and the developer with an image analysis researcher. Here, the biologist specifies the problem and informs the image analyst on the biomedical value of the biological findings. The image analyst has the knowledge of algorithms and the kinds of analysis inquiries the data allow. Even when developing programs for internal use in a small computational laboratory, the customer-developer roles can be utilized [54]. For instance, when a researcher is developing a library and another one is using it.

In addition to the customer-developer relationship, XP has many valuable lessons for any program development. The full list of best practices touches the fields of planning, managing, designing, testing and of course programming itself – all the phases of a good software development cycle [52, 8].

6

The main goal of the thesis was to develop a rapid and robust image analysis development platform that supports software integration. In the spirit of code reuse, the platform was not to be built from scratch, but to find an existing platform and adapt it for the main goal. The usability of the platform was to be tested by applying it in various image analysis tasks. The research aim can be divided to these steps:

1. To find a suitable system integration platform, and append it with pipeline based image analysis capabilities.
2. To develop new robust segmentation methods suitable for pipeline based analysis.
3. To develop new visualization tools for confirmation of results.
4. To integrate the new tools and methods in the platform and apply them in meaningful biomedical contexts.

7

Each of the publications selected for this thesis is based on different types of image data. All of them are from a microscope source, but the sample species and labeling techniques vary. Table 7.1 displays a summary of the different image sources. Since the research question in each publication is different, the steps needed to analyze the images differ too.

Table 7.1: Summary of image sources used in the thesis

Source type	Used in publication
Benchmark image sets: nuclei and whole animals	I
Tissue slides of focal cerebral ischemia	II
Tissue microarrays of diffuse large B-cell lymphoma	III
Peptide microarrays of cow's milk allergy	IV

7.1 Benchmark image sets

Two benchmark image sets were used to display the performance of the image analysis platform. The use of benchmark data is important in software and method development, since they provide the means to compare and evaluate performance. With a standard set of benchmark data, the method developers have a quantifiable metric to use to improve their algorithms.

The first image set contains 9600 synthetic cell nucleus images (600 images $\times$ 16 different simulated levels of out-of-focus effect). The images look roughly like a common DNA staining would look like. The second image set consists of 97 unstained images of *C.elegans*. Experts have classified the images into classes of either living or dead worms. Examples of both of these sets are shown in Figure 7.1.

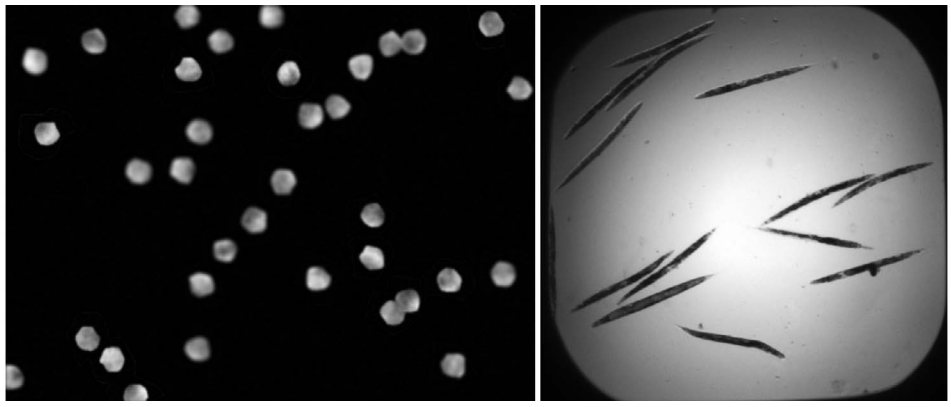


Figure 7.1: A montage of example benchmark images. The left image is from the BBBC005v1 [27] set of synthetic cell images. The right image is from BBBC010v1 [27] set of live/dead *C.elegans*.

7.2 Tissue slides of focal cerebral ischemia

To study cerebral ischemia, anesthetized adult male Wistar rats were used. Focal ipsilateral cerebral ischemia was induced, the animals were sacrificed and frozen. 8 μm thick sections of the brains were prepared for staining. The sections underwent in situ zymography, and then were stained for fluorescent detection of neurons (Neuronal Nucleus, NeuN), astrocytes (Glial Fibrillary Acidic Protein, GFAP) or endothelial cells (von Willebrand Factor, vWF).

Imaging was performed using an Axioplan 2 epifluorescent microscope (Carl Zeiss, Hallbergmoos, Germany) with a 20×-objective. Five regions of interest (ROI) were acquired from predefined sites (three cortical and two subcortical) from both hemispheres with an AxioCam camera, (1300 × 1030 pixels) and Axiovision software (v 3.0.6, Carl Zeiss). Image sets were acquired using constant exposure times for all samples. An example of these images is shown in Figure 7.2. The full image set is visualized at `http://anduril.org/pub/anima/ISZ_activity/`.

All experiments were approved by local authorities (ELLA animal experiment board, Finland), and conducted in accordance with The Finnish Act on Animal Experimentation (62/2006).

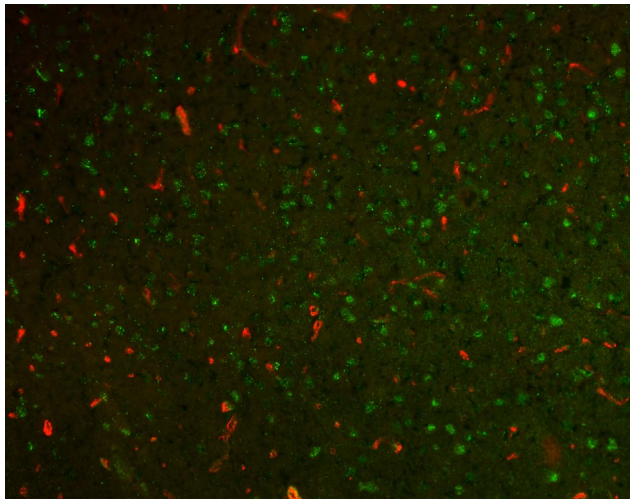


Figure 7.2: An example image from cerebral ischemia study. The neuronal nuclei are colored green, and the endothelial cells (von Willebrand Factor) are colored red.

7.3 Tissue microarrays of diffuse large B-cell lymphoma

The prospectively collected tissue microarray (TMA) cohorts consisted of diffuse large B-cell lymphoma (DLBCL) patients who were less than 65 years old and had primary high-risk disease. They were treated in the Nordic Lymphoma Group Large B-Cell phase II (NLG-LBC-04) study. The original clinical study had 156 patients. Histological diagnosis was established from surgical or needle biopsy of the pretreatment tumor tissue by local pathologists according to current criteria of the World Health Organization classification, and subsequently reviewed by expert hematopathologists on a national basis.

The infiltration of lymphoma cells in the tissue was assessed from frozen tissue section using H&E and toluidine blue staining, marking the COMM domain-containing protein 1 (COMMD1) positivity. Formalin-fixed paraffin-embedded (FFPE) tissue containing adequate material was used for the preparation of TMAs (n=70). The tissue sections on TMA slides contained 2–4 tissue cores/patient, with a core diameter of 1 *mm*.

To validate the findings, an independent series of 146 primary DLBCL patients treated with chemoimmunotherapy at the Helsinki University Central Hospital between 2001 and 2010 was used. The cases were selected based on the availability of FFPE tissue and clinical information. The validation samples were whole tissue sections.

To score the tissues, COMMD1-positivity was evaluated from one to three high-power fields (HPF) with a bright-field microscope using 63 $\times$ magnification (Leica DM LB, Leica Microsystems GmbH) and a camera attached to it (Olympus DP50, InStudio 1.0.1 Software). The most representative areas with intense staining pattern were first selected with low magnification and further digitized with HPF, resulting in microscopic images with area size of 0.02 mm^2. An example image is shown in Figure 7.3.

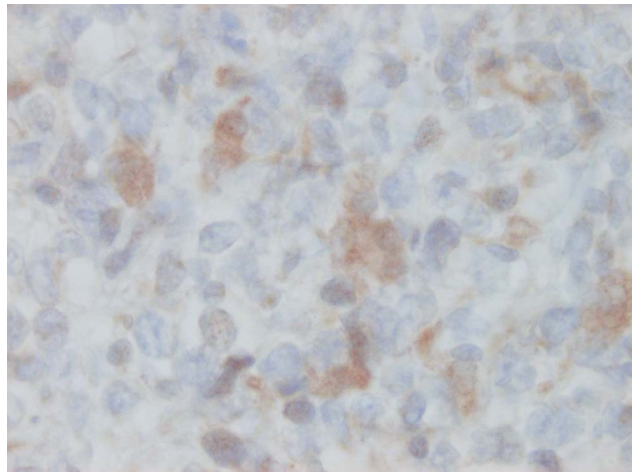


Figure 7.3: An example of tissue stained for COMMD1-positivity evaluation.

7.4 Peptide microarrays of cow's milk allergy

For the cow's milk allergy (CMA) study, serum from 23 children with CMA was collected at three time points. Additionally, the serum of six nonatopic control subjects was collected for follow-up (mean age, 8.6 years; range, 8.1-9.3 years). The clinical data of the patients were available from previous studies. Serum samples were stored at -80 °C until measured with a peptide microarray–based immunoassay.

A library of peptides consisting of 20 amino acids overlapping by 17 amino acids (3-offset) corresponding to the primary sequences of α s1-, α s2-, β-, and κ-caseins and β -lactoglobulin was commercially synthesized. Peptides were printed in two sets of triplicates on epoxy-derivatized glass slides by using the NanoPrint Microarrayer 60 (TeleChem International, Inc). Protein Printing Buffer alone was used as a negative control and for background normalization. The slides were incubated with each patient's serum, and then labeled for Immunoglobulins A, E and G4 detection with fluorescent stains. The slides were scanned with a ScanArray Gx (PerkinElmer, Waltham, Mass). Example slide is shown in Figure 7.4.

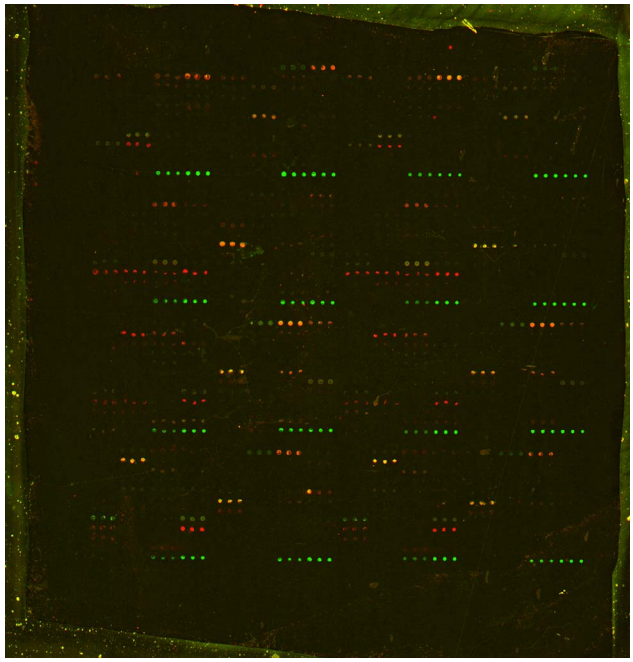


Figure 7.4: An example of a scanned peptide microarray slide. The IgE sensitive library is colored green, and the IgG4 sensitive red.

8

The results presented in this thesis are the tools and methods that have been developed to produce the results in the publications. First, the image analysis platform is explained. After that, the specific methods and tools used by the platform are described. A summary of the results is presented in Table 8.1.

Table 8.1: Summary of results in the thesis

Name	Category	Used in publication
ANIMA: Image analysis integration platform	Platform	I,II,III
Shape filtered segmentation	Method	IV
k-NN color segmentation	Method	III
Qalbum: Quick image browser	Tool	I,II,III
NiceCSV: A delimited text browser and printer	Tool	I,II,III

8.1 Image analysis integration platform

The corner stone of this thesis is the development of an image analysis platform that enabled the development of the algorithms, and the numerous analysis pipelines using existing algorithms. The name of the platform is Anima, and it is built as an extension to Anduril [49], and therefore shares its software infrastructure. The platform is specified and explained thoroughly in Publication I. The software architecture is shown in Figure 8.1. Anima was designed to comply with the concepts of usability as suggested by Carpenter *et al* [55]. The source code can be downloaded from Anima web site `http://anduril.org/anima`.

Anima has several unique software features compared to existing analysis platforms. One of the striking features is that Anima is strictly an integration platform and it avoids overlapping and competing with the existing tools. It uses a simple scripting language, allowing itself to be integrated easily. To use Anima, the user writes a script that joins the inputs and outputs of components to create a flow of data. The components contain program code and may be of different languages, since Anima launches them as separate processes. The components may call existing libraries or integrated software. At each run of a component, the output is saved. The existing outputs can be reused at consecutive runs of the script. See Publications I and II with their supplementary materials for example scripts.

Anima is highly modular, and it doesn't come installed with dependencies: it comes with installers for the dependencies, but a user may decide to use the software and libraries already present in their system. These dependencies include, for example, programming languages: Java, R, Bash, Python and Mathworks MATLAB. Further, several scriptable imaging platforms are supported: ImageMagick, ImageJ/Fiji and CellProfiler.

Using Anduril as a backbone, Anima has an existing code base for data analysis, including machine learning libraries, such as WEKA library [56]. A new user to Anima can use any of the hundreds of analysis components of Anduril, which are well tested and documented,

as well as the integrated platforms without having to port the basic analysis tools to a new system. Included in these components are, for example, reporting tools that enable clear communication of tabular and image data to collaborators or customers.

As a pipeline engine, Anduril provides automatic process parallelization and efficient resource use for rapid analysis development. It separates algorithm and analysis development by abstracting the analysis language and the processing (algorithm) language. All of the components in Anima were developed with test cases, in a modular fashion, and also applying other XP methods. The structure of the component model in Anduril encourages the use of XP methodologies, such as test driven development and developer/customer role switching.

It is imperative to notice Anima is aimed at developers of image analysis. While the scripting language is simple enough for a majority of users to run established analysis, the main purpose of the platform is to make development rapid.

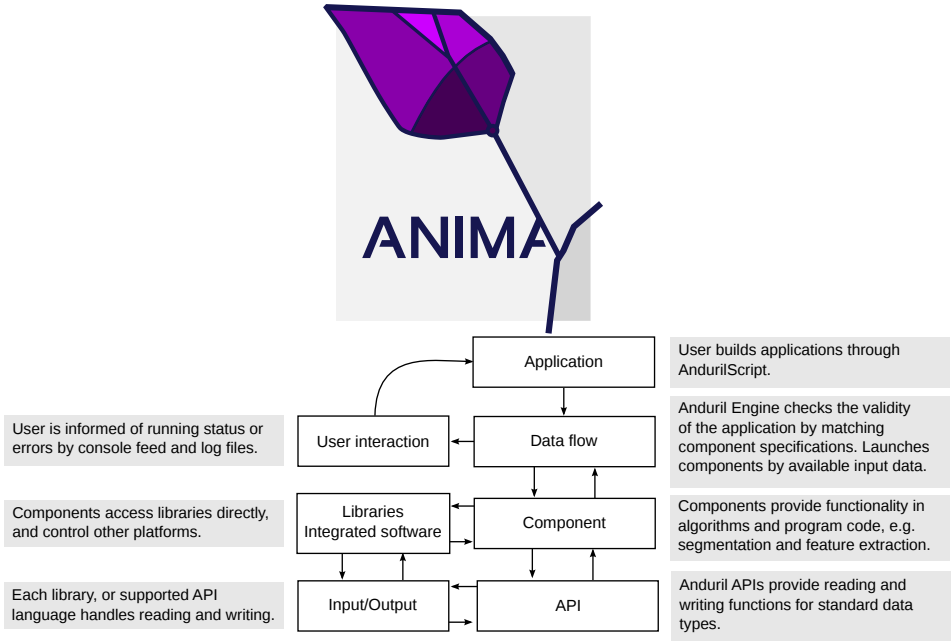


Figure 8.1: Software architecture of Anima and Anduril. Analysis development is done at the Application level. Algorithm development is done at the component level. Data flow and API levels are within the (Anduril) platform development domain. Anima extends Anduril within the component and libraries levels.

8.2 Shape filtered segmentation

Shape filtered segmentation is an algorithm based on modeling approach. The current version of the algorithm assumes that objects:

- are brighter than their immediate neighborhood,
- are separated by uninterrupted darker region,
- fit between expected minimum and maximum area, roundness and eccentricity.

The algorithm iterates over each possible intensity level and thresholds the image. Any object that fits the expected criteria are united with the previous accepted objects at each iteration. The operation of the algorithm is visualized in Figure 8.2. The algorithm is freely available as Mathworks MATLAB function included in the Anima platform [57].

The algorithm was used to segment the peptide microarrays in Publication IV. The peptide microarray printing resulted in a number of malformed dots on the slide, which were considered unusable data for the analysis. The slides had six replicates of each peptide allowing a few to be removed and still calculate a reliable median value.

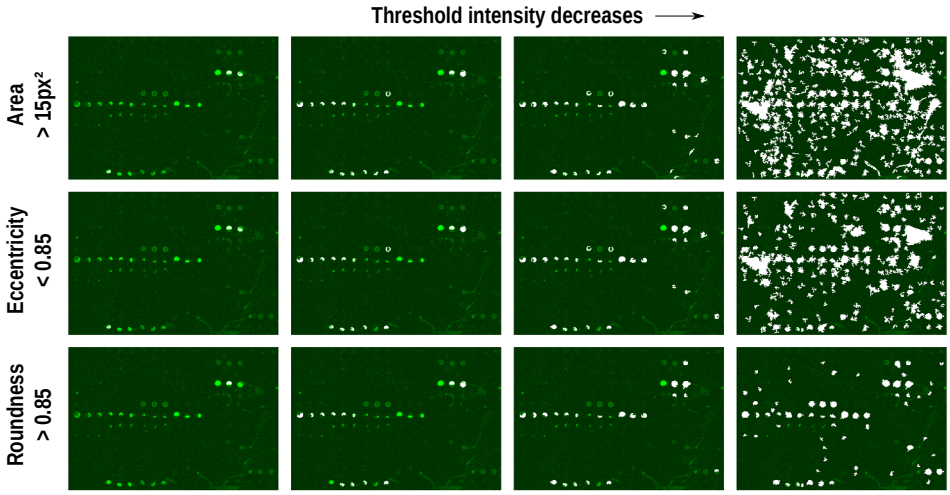


Figure 8.2: Shape based segmentation example. The intensities shown in green are thresholded at each possible intensity level. At each iteration, only the objects matching the shape expectations are kept for the next iteration. Valid objects are marked white. On the first row objects of area greater than 15 square pixels are kept. On the second row, in addition to size limit, only circular objects are kept and highly elliptical ones are discarded. On the third row, only smooth, round objects are kept, in addition to the earlier limits. The final accepted objects are shown at the lower right corner.

8.3 k-NN color segmentation

Great majority of segmentation methods expect image data to be gray scale. In a gray scale image, each pixel value is denoted with a single value. For example, an RGB color image consists of three values for each pixel. In the case of a color image, predefined segmentation

strategies are more difficult to implement. Therefore, a color image segmentation is a good environment to exercise computer assisted segmentation.

H&E stained tissue sections, such as found in Publication III, are difficult to segment without prior knowledge of the colors of the different object types. For example, there may be two similar brown colors that mark two different types of object. To tackle the problem, we used a graphical user interface to get a human opinion of the colors of the object classes. Then, we used the k-Nearest Neighbors algorithm (k-NN) to classify each pixel in an image to the nearest class given by the human. k-NN is a fast algorithm that finds the nearest neighbor (when k=1) of a point in a given N-dimensional set of data points [58]. An example of this kind of segmentation is seen in Figure 4.3.

The images in Publication III required normalization before the segmentation functioned properly. A simple white balancing was enough to ensure color similarity. Ten example points for each of the four expected color classes were enough to provide the training set to analyze more than 200 images. The implementation of the segmentation was developed in Mathworks MATLAB, and is available for download [59].

8.4 Visualization of image data

Advanced analysis of data must have good ways of displaying the results in order to be useful. Naturally, image data can be saved as image files, and there are plenty of good image viewers around, but only few of them are suitable for large image sets or are operating system independent. Every modern operating system comes with a web browser that is capable of producing a user interface as well as display images. Therefore, we created a tool, Qalbum, that generates web pages for image browsing. Static web pages are easy to distribute and they are operating system independent. They work the same way on every standard web browser. Qalbum was developed in Python language, for maximum portability. It depends only on Python standard libraries and ImageMagick for thumbnail generation. The output of the tool is a set of HTML and JavaScript files, and image thumbnails.

The highlights of the Qalbum image gallery generator include the following features:

- It is configured via command line switches and configuration files. In addition, it can be called as a Python library.
- It supports image annotation and insertion of custom HTML snippets.
- The generated image gallery is a client side program; the gallery can be copied on a local disk, or hosted on a simple HTTP server, since all the code is executed in the web browser.
- The gallery has active elements implemented in JavaScript, for quick navigation and for touch screen support.

Galleries generated with Qalbum are shown in Figure 8.3. Qalbum is an internal part of an Anima component for creating image galleries, and it is used in nearly all of the analysis projects performed with it. The source code of Qalbum is accessible at `https://bitbucket.org/MoonQ/qalbum` and a demo gallery at `http://moonq.org/qalbum` [60].

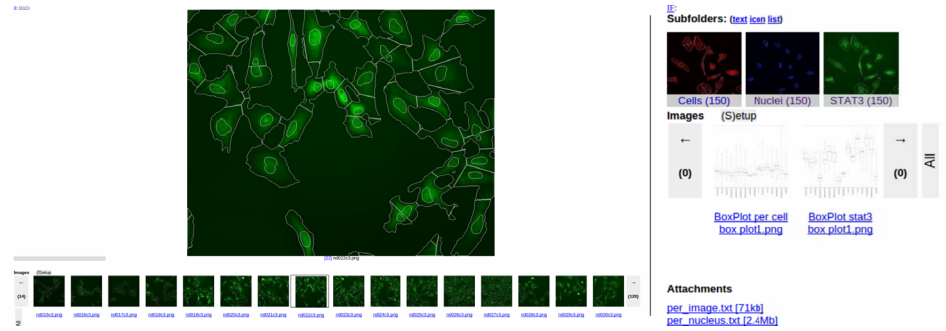


Figure 8.3: Screen shots of a gallery generated with Qalbum. On the left is the browsing mode, where the user can move to the next image by clicking the thumbnail, using the keyboard arrow keys or a swipe gesture on a touch screen. Images are preloaded for quick interaction. On the right is the view with subdirectories with a thumbnail, as well as a list of images in the current folder. Any additional files are considered attachments.

8.5 Tabular data browser

Anduril and Anima use delimited text files for conveying tabular data from one component to another. The fundamental reason is the openness and simplicity of the format, as well as the freedom to choose from a vast range of readers and writers. A user can read the data with the simplest of tools. If the data file is large, however, the simple tools become inadequate to provide a good overview of the data.

For easy browsing of tabular data, we developed NiceCSV, which is an interactive table browser, as well as a text based tabular data formatter. An application of the formatter may be, for example, including tabular results in an email. In the browsing mode, the user can quickly browse through the data using keyboard, view basic statistics for each column, sort by column, format the table in various ways, change alignment, or search for a keyword. See Figures 8.4 and 8.5 for usage examples. NiceCSV works in unison with other Unix tools, such as *grep*, *cut* and *sed*, reusing their capabilities for tabular data processing.

NiceCSV is freely downloadable and the source code is available at `https://bitbucket.org/MoonQ/ncsv` [61]. It has no dependencies other than the standard Python libraries. The program is fully portable to all operating systems that run Python. NiceCSV is used by Anduril platform in a component that provides generic file format conversions.

```
      RowId        +        File        |Object|Area|Perimeter|Roundness|Eccentricity|   Ly   |   Lx   |  Alpha
c ┌Info:──────────────────────────────────────────────────────────────────────────────────────────┐ 51 41.3802  0.964793
c │File name: testdata.csv                                                                       │ 68 29.9521  0.394987
c │Cols,Rows: 20,70                                                                              │ 48 17.2082 -0.531034
c │Column names:   Mean:     Max:      Min:      Sum:      StDev:    NAs:  Uniq:               │ 93 17.5314  -0.65147
c │RowId                                                             0     70                    │ 15  13.351  0.639129
c │File                                                              4     5                     │ 21 19.7818 -0.704034
c │Object          18        35        1         1260      10.1      0     35                    │ 06  23.484  -1.51497
c │Area            1161      4252      324       8.13e+04  1006      0     36                    │ 89 15.1124 -0.120869
c │Perimeter       122.3     270.1     67.46     8561      54.24     0     36                    │ 68  13.321  -1.50277
c │Roundness       0.9269    0.9935    0.7951    64.88     0.0566    0     36                    │ 24 16.6583  -1.38291
c │Eccentricity    0.7109    0.9456    0.3292    49.76     0.1664    0     36                    │ 04  24.803  0.383957
c │Ly              14.1      28.86     7.64      986.8     5.532     0     36                    │ 44 37.0271  0.956838
c │Lx              23.01     54.47     11.85     1611      11.17     0     36                    │ 85 21.0051 -0.362258
c │Alpha           1.75e-05  1.515     -1.515    0.001228  0.7284    0     70                    │ 23 12.7996 -0.317504
c │Sk width        24.33     51.37     11.46     1703      10.46     0     65                    │ 27 26.9839 -0.266479
c │Sk width std    1.726     5.114     0          120.9     1.112     0     65                    │ 84 40.7186   0.19682
c │Skeleton length 18.4      63        2         1288      16.05     0     25                    │ 31 11.9275 -0.711779
c │Branch points   0         0         0         0         0         0     1                     │ 98 12.2827 -0.404409
c │End points      2         2         2         140       0         0     1                     │ 76 42.7326 -0.964522
c │Maximum length  45.55     108.9     23.41     3189      22.42     0     34                    │ 71 22.0597   1.40237
c │Length estimate 42.57     99.5      22.16     2980      20.51     0     36                    │ 76 19.1305  0.283288
c │Distance(1)     39.84     67.5      21.94     2789      12.76     0     29                    │ 05 23.9224 -0.552105
c │X               224.5     437.5     11.47     1.57e+04  124.2     0     70                    │ 88 11.9552 0.0388738
c │Y               167.9     348.2     22.3      1.18e+04  98.81     0     36                    │ 98 45.7427 -0.883844
c └────────────────────────────────────────────────────────────────────────────────────────────┘ 59 21.2628 -0.399733
cells2.png_29          cells2.png          29  699   97.2548  0.963679    0.651043 13.0054  17.134 -0.0855953
cells2.png_30          cells2.png          30  972   114.912  0.961776    0.583359 15.8618  19.529 -0.0192607
cells2.png_31          cells2.png          31  539   84.1838  0.977622    0.552223 11.9717 14.3598  -0.324327
cells2.png_32          cells2.png          32 1026   118.569  0.957655    0.626861 15.9604 20.4849  -0.457944
cells2.png_33          cells2.png          33  507   92.1838  0.865874    0.897245 8.45366 19.1461  0.133006
cells2.png_34          cells2.png          34  444   77.3553   0.96562     0.68234 10.1748 13.9184   0.34725
cells2.png_35          cells2.png          35  358   68.2843  0.982258    0.581356 9.64306 11.8516  0.535459
cells.tif_DAPI.png_1   cells.tif_DAPI.png   1  358   68.2843  0.982258    0.581356 9.64306 11.8516 -0.535459
cells.tif_DAPI.png_2   cells.tif_DAPI.png   2  444   77.3553   0.96562     0.68234 10.1748 13.9184  -0.34725
cells.tif_DAPI.png_3   cells tif_DAPI.png   3  507   92.1838  0.865874    0.897245 8.45366 19.1461 -0.133006
cells.tif_DAPI.png_4   cells.tif_DAPI.png   4 1026   118.569  0.957655    0.626861 15.9604 20.4849  0.457944
```

Figure 8.4: Screenshot of NiceCSV displaying a table in the browsing mode. On top, there is a window showing brief summary of statistics of the table.

```
#:~$ cut -f1,5,6,8,9  testdata.csv | head
"RowId" "Perimeter"     "Roundness"     "Ly"    "Lx"
"cells.tif_DAPI.png_1"  "68.2843"       "0.982258"      "9.64306"       "11.8516"
"cells.tif_DAPI.png_2"  "77.3553"       "0.96562"       "10.1748"       "13.9184"
"cells.tif_DAPI.png_3"  "92.1838"       "0.865874"      "8.45366"       "19.1461"
"cells.tif_DAPI.png_4"  "118.569"       "0.957655"      "15.9604"       "20.4849"
"cells.tif_DAPI.png_5"  "84.1838"       "0.977622"      "11.9717"       "14.3598"
"cells.tif_DAPI.png_6"  "114.912"       "0.961776"      "15.8618"       "19.529"
"cells.tif_DAPI.png_7"  "219.279"       "0.828743"      "18.2998"       "45.7427"
"cells.tif_DAPI.png_8"  "110.326"       "0.929587"      "12.5459"       "21.2628"
"cells.tif_DAPI.png_9"  "97.2548"       "0.963679"      "13.0054"       "17.134"
#:~$ cut -f1,5,6,8,9  testdata.csv | head | ncsv --cf a -a auto -s %.2f -W 6
.-----------------------------------------------------.
|RowId               |Perimeter|Roundness|    Ly|    Lx|
|--------------------+---------+---------+------+------|
|cells.tif_DAPI.png_1|    68.28|     0.98|  9.64| 11.85|
|cells.tif_DAPI.png_2|    77.36|     0.97| 10.17| 13.92|
|cells.tif_DAPI.png_3|    92.18|     0.87|  8.45| 19.15|
|cells.tif_DAPI.png_4|   118.57|     0.96| 15.96| 20.48|
|cells.tif_DAPI.png_5|    84.18|     0.98| 11.97| 14.36|
|cells.tif_DAPI.png_6|   114.91|     0.96| 15.86| 19.53|
|cells.tif_DAPI.png_7|   219.28|     0.83| 18.30| 45.74|
|cells.tif_DAPI.png_8|   110.33|     0.93| 12.55| 21.26|
|cells.tif_DAPI.png_9|    97.25|     0.96| 13.01| 17.13|
'-----------------------------------------------------'
```

Figure 8.5: Screenshot of NiceCSV being used in combination with standard Unix tools that select columns and limit the number of rows. The upper half shows the input without NiceCSV formatting, where the header and value positions do not match. By using NiceCSV, the table is easy to read: elements are separated with line characters, numbers are formatted to two decimal points and minimum column width is set to six characters. The output is ready to be inserted for example in an email.

9

Over the years bioinformaticians and image analysts have developed a great number of tools that solve specific analysis problems. To prevent *reinventing the wheel*, it is wise to concentrate on using those tools rather than build new ones from scratch. Also, when building new methods, they should be made easily usable by other researchers in the field.

This thesis presents an image analysis platform, Anima, that is built on top of a generic analysis platform, Anduril. Anduril is a developer oriented integration platform – a software that runs other software to benefit from their functionality. Anima extends Anduril with image analysis possibilities. It is the result of applying generic platforms, code reuse, software integration and XP methodologies in biomedical image analysis.

Automated analysis, code reuse and software integration do not exist just for better imaging software development, they are all linked to good conduct of scientific research. An automated analysis pipeline is unchanged by the data. For the same image set, automated image analysis repetitively produces the same, unbiased, result. Humans tend to tire, decide based on a feeling and unconsciously create errors. We change our decision making strategies with new data. Since automated analysis is also built by a human, the automation may contain errors. However, programs can be systematically tested for errors with other data following the XP methodologies. The topic of result consistency and speed of analysis between automatic and manual analysis is further discussed in Publication II.

Code reuse along software integration speak for reproducible research. Publishing the implementation with the theoretic algorithm makes the research of the method reproducible. Porting implementations to other programming languages deteriorates reproducibility, since the ported code does not function exactly like the original. In addition, publishing image analysis pipelines establishes an exact record on how, for example, blood vessels to be analyzed were selected from an image, and which level of fluorescence expression is selected to be a positive one. Too often such important selections are not explained in research articles, as noted in Publication II. Even if the platform presented here is aimed at image analysis developers, the ideology behind its development touches all disciplines of science.

The analysis platform, Anima, was chosen to be built without a GUI. Scripting (i.e. no GUI) increases its usability overall but decreases user friendliness. For any user, starting new analysis development with an empty script file is harder than starting one with a menu driven interface. On the other hand, scripting makes Anima available for further system integration – it can be used by other software, as Anima itself calls integrated software. In addition, developing new methods is more efficient in a scripted environment than in a graphical one.

Every time a conventional computer program is run, the whole program runs from scratch to finish. When an Anima pipeline is run, each of the separate components with integrated software or libraries are used to process data and save the outputs as files. The files may contain processed images or tabular data. Since the intermediate outputs are saved as files, unchanged parts of the pipeline are skipped, and the results from a previous run are reused instantly. Further, the components are run as separate processes. It means they can be of different programming languages, they are easily parallelized and even sent to a different computer in a cluster environment. While these properties are good for any image analysis

program, they save a lot of precious time, especially for a developer. With Anima, an image analysis algorithm developer may concentrate on creating new algorithms for a specific step in an analysis pipeline, while being confident the other steps function as they should without being affected by the work at hand.

One of the aims of this thesis was to use the platform and its tools in meaningful biomedical research contexts. The publications selected for this thesis show the versatility of the Anima platform with different types of image data. The platform has been utilized in several other studies too. Including the publications in this thesis, as of March 2015, Anima has been used to provide image analysis results in 21 published articles. The full list is available at `https://bitbucket.org/anduril-dev/anima/wiki/Publications`.

The segmentation methods explained in this thesis were both born with the idea of creating robust methods for large image sets. One of them utilized the modeling principle and the other interactively uses the human expert as a teacher. The methods are based on simple ideas, but they are fast and they use intuitive parametrization. The shape filtered segmentation is best at finding fluorescent labeled objects with a clearly defined shape. It tends to leave out fuzzy or partly obfuscated objects, but there are rarely false negatives. Its reliability in finding objects makes it a good method for larger image sets where the number of images makes up for the objects left out. The k-NN color segmentation is a powerful but simple classifier that relies on user input. However, it is only capable of segmenting images when they have been acquired very similarly or they have been normalized well. Unfortunately, this is not always the case, especially with transmitted light labeling.

Finally, this thesis presents two visualization tools. They are both quick browsers of data and portable to variety of operating systems, even mobile devices. Both of them are published with an open-source license and they are integrated in Anduril and Anima. Qalbum is a generic image data visualization tool. The tool can also display other than microscopy images – any plot or graph viewed on a screen can be saved as an image file. Qalbum has been used to browse through a large array of scatter plots, among other types of images in the data analysis for Publication III. In addition, many other projects are benefiting from quick and shareable image browsing.

Image analysis often results in some visualizations saved as images, but some of the results are always tabular data. While developing analysis tools, the programmer wants to see that the output is correct. NiceCSV was developed to confirm the correct formatting of tabular data. Over time, it has evolved in to a powerful data browser. NiceCSV can quickly show the contents of tabular data files, as well as calculate statistics and reformat the table for display purposes.

The future of image analysis is dependent on the development of imaging technologies and on how we understand images on the whole. Imaging technology as a scientific data collection tool is developing at two fronts; the images will have more detail and more images will be acquired from the same sample.

The image detail is increased with more accurate microscopes. For example, the diffraction barrier discovered by Abbe has been broken with fluorescence microscopy using stimulated-emission-depletion (STED) [14, 62] – an invention the 2014 Nobel Prize in chemistry was awarded for. With STED, fluorescence can be imaged at a single molecule detail. As single molecules get imaged, solid objects are seen as strings of points, rather than solid surfaces. It will change the strategies of segmentation from thresholding to structure reconstruction. Also, classic colocalization studies do not apply anymore, since two molecules never

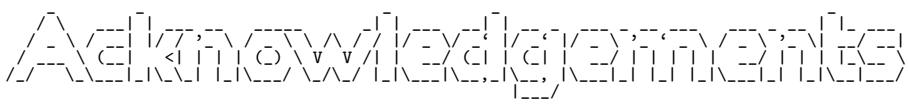
occupy the same space. Neighborhood type of analysis, like we have already presented, needs to be examined further [63]. As more imaging technologies emerge, more image analysis methods are developed. Following good programming practices, the developers will make these methods available for the research community.

The sheer number of images per study is also increasing. To give a more representative view, whole samples and more samples are being imaged with robotics. Also, more markers are used to measure more targets simultaneously. To cope with the ever increasing amount of multidimensional data, platforms need to be developed to utilize parallelization and computer cluster environments. In this aspect, Anima is already well equipped.

In this thesis, image analysis has been performed for biologists with a specific research question. As more biologists get experienced with the capabilities of image analysis, also the research questions evolve to more complex ones. Similarly to the other data rich science fields, image analysis can explore data mining techniques. Data mining of large, but well annotated image atlases may, for example, find unexpected similarities between tumor cells. Data mining approaches already exist [64], but the size and genericity of the atlases will grow tremendously in the future. As different fields of science borrow methods from each other and have their *de facto* standard tools that do not readily communicate with each other, software system integration will play a greater role in data analysis.

This thesis presents the benefits of system integration in the biomedical image analysis domain. The platform developed for the thesis, Anima, has shown its capabilities in diverse research settings. It has been used to develop several tools and methods. Analysis workflows built with Anima have provided valuable information to several biomedical studies.

10



This research was carried out in the Systems Biology Laboratory at the Medical Faculty of the University of Helsinki during 2008-2015. I thank professor Sampsa Hautaniemi for leading the laboratory in a relaxed, but inspiring spirit.

The research leading to this thesis has received funding from the European Community's Seventh Framework Programme FP7/2007-2011 under grant agreement no. 201837. Additionally, funding was provided by Biocentrum Helsinki and Biocenter Finland.

I thank the reviewers professor Elina Ikonen and docent Jaakko Hollmén. Your expertise gave this thesis credibility and precision.

I have worked with several collaborators during the research for this thesis. The analysis tasks given by the groups of professors Stefan Hell, Jens Habermann, Claus Seidel and Jerker Widengren have been the acid test of Anima. Especially, thanks to Peter Zentis for patience with the many repeats and multiple ways of looking at classification results. Emma Savilahti gave me a challenge with a fairly difficult project in the beginning. Minna Taskinen has been a user of Anima, and she has provided an important user point of view. Olli Mattila has completed several projects with me and he has been a relentless guinea pig in running image analysis pipelines at various stages of development. Thank you all for helping me.

I want to thank Kristian Ovaska for starting up the Anduril project, which established the basis of this thesis. Big thanks to Javier Núñez-Fontarnau for helping me with the thesis and for developing Anduril further. Miko Valori: thank you for your involvement in building Anima components. Riku Louhimo is a cornucopia of ideas. Thank you for the enlightened discussions, independent of the topic. Thank you Anna-Maria Lahesmaa-Korpinen for your deep interest in my work. I enjoyed your relaxing company as my peer/beer support. Tiia Pelkonen is thanked for polishing up the thesis and of course for running just about everything in our lab.

Thanks to all the magnificent people at Biomedicum I have spent time with. A great big thanks goes out to everybody in our lab. It's always fun to come at work and get things done with you! Thank you alumni members Sirkku, Marko, Erkka, Mikko, Mani, Vladimir, Elena, Ali, Kari, Janne & Jimmy. The current lab members, thanks in order of distance from my seat: Julia, Rainer, Katherine, Alejandra, Amjad, Chiara, Lilli, Chengyu, Emilia, Ping, Lauri, Markku and Mikko.

My in-laws Jukka and Kirsti have given me the time to work on the thesis by taking care of my son. Thank you for that. Hessu and Inkku, my parents, thank you for setting me on the academic path and supporting me all through my life. Sussu, thanks for being such an inspiration. You strive for scientific breakthroughs and celebrate every small success. You motivated me in finishing this thesis. Finally, I want to thank my son Rasmus, who keeps reminding me of the truly important things in this world: eat, drink, sleep and play.

Ville Rantanen
Helsinki, April 24th 2015

Page(s) # References

[1] Robinson, C. W & Sloutsky, V. M. (2004) Auditory dominance and its change in the course of development. *Child Dev.* **75**, 1387–1401. 1

[2] Noesselt, T, Tyll, S, Boehler, C. N, Budinger, E, Heinze, H. J, & Driver, J. (2010) Sound-induced enhancement of low-intensity vision: multisensory influences on human sensory-specific cortices and thalamic bodies relate to perceptual enhancement of visual detection sensitivity. *J. Neurosci.* **30**, 13609–13623. 1

[3] King, A. J. (2009) Visual influences on auditory spatial learning. *Philos. Trans. R. Soc. Lond., B, Biol. Sci.* **364**, 331–339. 1

[4] Hooke, R, Martyn, J, & Allestry., J. (1665) *Micrographia: or some physiological descriptions of minute bodies made by magnifying glasses with observations and inquiries thereupon*. (the Royal Society, England), p. 323. 1, 3

[5] Wiesmann, V, Franz, D, Held, C, Münzenmayer, C, Palmisano, R, & Wittenberg, T. (2014) Review of free software tools for image analysis of fluorescence cell micrographs. *J. Microsc.* 1

[6] Beazley, D. (2003) Automated scientific software scripting with SWIG. *Future Generation Computer Systems* **19**, 599 – 609. 1

[7] Ray, B, Kim, M, Person, S, & Rungta, N. (2013) Detecting and characterizing semantic inconsistencies in ported code. *2013 28th IEEE/ACM International Conference on Automated Software Engineering*. 1

[8] Prlić, A & Procter, J. B. (2012) Ten simple rules for the open development of scientific software. *PLoS Comput Biol* **8**, e1002802. 1, 17, 21

[9] Sametinger, J. (1997) *Software Engineering with Reusable Components*. (Springer-Verlag New York, Inc., USA). 1, 19

[10] Vander, A. J, Sherman, J. H, & Luciano, D. S. (2001) *Human Physiology*. (McGraw Hill), 8th edition. 3

[11] Becker, W. M, Kleinsmith, L. J, Hardin, J, & Bertoni, G. P. (2008) *The World of the Cell, 7th Edition*. (Benjamin Cummings), 7 edition. 3

[12] Maul, G. G & Deaven, L. (1977) Quantitative determination of nuclear pore complexes in cycling cells with differing dna content. *J Cell Biol* **73**, 748–760. 3

[13] Bryan, A. K, Goranov, A, Amon, A, & Manalis, S. R. (2010) Measurement of mass, density, and volume during the cell cycle of yeast. *Proc. Natl. Acad. Sci. U. S. A.* **107**, 999–1004. 3

[14] Abbe, E. (1873) Ueber einen neuen beleuchtungsapparat am mikroskop. *Archiv f. mikrosk. Anatomie* **9**, 469–480. 4, 35

[15] Milo, R, Jorgensen, P, Moran, U, Weber, G, & Springer, M. (2010) Bionumbers–the database of key numbers in molecular and cell biology. *Nucleic Acids Res* **38**, D750–D753. 4

[16] Coons, A. H, Creech, H. J, & Jones, R. N. (1941) Immunological properties of an antibody containing a fluorescent group. *Proc. Soc. Exp. Biol. Med.* **47**, 200–202. 5

[17] Lakowicz, J. R, ed. (2006) *Principles of Fluorescence Spectroscopy*. (Springer, USA). 6

[18] Shimomura, O, Johnson, F. H, & Saiga, Y. (1962) Extraction, purification and properties of aequorin, a bioluminescent protein from the luminous hydromedusan, aequorea. *J. Cell. Comp. Physiol.* **59**, 223–239. 6

[19] Chalfie, M, Tu, Y, Euskirchen, G, Ward, W. W, & Prasher, D. C. (1994) Green fluorescent protein as a marker for gene expression. *Science* **263**, 802–805. 6

[20] Heim, R & Tsien, R. Y. (1996) Engineering green fluorescent protein for improved brightness, longer wavelengths and fluorescence resonance energy transfer. *Curr. Biol.* **6**, 178–182. 6

[21] Schneider, C. A, Rasband, W. S, & Eliceiri, K. W. (2012) NIH Image to ImageJ: 25 years of image analysis. *Nat. Methods* **9**, 671–675. 10, 20

[22] Hyvärinen, A, Hurri, J, & Hoyer, P. O. (2009) *Natural Image Statistics*. (Springer, England). 11

[23] Gonzalez, R. C & Woods, R. E. (1992) *Digital Image Processing*. (Addison-Wesley Longman Publishing Co., USA), 2nd edition. 11, 12

[24] Korzynska, A & Zdunczuk, M. (2008) in *Information Technologies in Biomedicine*, Advances in Soft Computing, eds. Pietka, E & Kawa, J. (Springer Berlin Heidelberg) Vol. 47, pp. 345–356. 12

[25] Bishop, C. M. (2006) *Pattern Recognition and Machine Learning*, Information Science and Statistics. (Springer-Verlag, USA). 13

[26] Otsu, N. (1979) A threshold selection method from gray-level histograms. *IEEE Trans. Syst. Man, Cybern. B, Cybern.* **9**, 62–66. 13

[27] Ljosa, V, Sokolnicki, K. L, & Carpenter, A. E. (2012) Annotated high-throughput microscopy image sets for validation. *Nat. Methods* **9**, 637. 14, 23

[28] Fok, Y. L, Chan, J. K, & Chin, R. T. (1996) Automated analysis of nerve-cell images using active contour models. *IEEE Trans. Med. Imag.* **15**, 353–368. 14

[29] Klemencic, A, Kovacic, S, & Pernus, F. (1998) Automated segmentation of muscle fiber images using active contour models. *Cytometry* **32**, 317–326. 14

[30] Merali, Z. (2010) Computational science: ...error. *Nature* **467**, 775–777. 17

[31] Hatton, L. (1997) The T Experiments: Errors in scientific software. *Comput. Sci. Eng.* **4**, 27–38. 17

[32] Segal, J & Morris, C. (2008) Developing scientific software. *Software, IEEE* **25**, 18–20. 17

[33] Kelly, D. (2007) A software chasm: Software engineering and scientific computing. *Software, IEEE* **24**, 120–119. 17

[34] Hook, D & Kelly, D. (2009) Testing for trustworthiness in scientific software. *ICSE Workshop on SECSE '09* pp. 59–64. 17

[35] Woollard, D, Medvidovic, N, Gil, Y, & Mattmann, C. (2008) Scientific software as workflows: From discovery to distribution. *Software, IEEE* **25**, 37–43. 18

[36] R Core Team. (2014) *R: A Language and Environment for Statistical Computing* (R Foundation for Statistical Computing, Vienna, Austria). 18, 20

[37] Carpenter, A. E, Jones, T. R, Lamprecht, M. R, Clarke, C, Kang, I. H, Friman, O, Guertin, D. A, Chang, J. H, Lindquist, R. A, Moffat, J, Golland, P, & Sabatini, D. M. (2006) CellProfiler: 18, 20 image analysis software for identifying and quantifying cell phenotypes. *Genome Biol.* **7**.

[38] Carver, J. C, Kendall, R. P, Squires, S. E, & Post, D. E. (2007) *Software Development Environments for Scientific and Engineering Software: A Series of Case Studies*, ICSE '07. 18, 21 (IEEE Computer Society, Washington, DC, USA), pp. 550–559.

[39] Gomez, C & Scott, T. (1998) Maple programs for generating efficient fortran code for serial 19 and vectorised machines. *Comput. Phys. Commun.* **115**, 548–562.

[40] Bankhead, P, Scholfield, C. N, McGeown, J. G, & Curtis, T. M. (2012) Fast retinal vessel 19 detection and measurement using wavelets and edge location refinement. *PLoS ONE* **7**.

[41] Held, M, Schmitz, M. H, Fischer, B, Walter, T, Neumann, B, Olma, M. H, Peter, M, Ellenberg, J, & Gerlich, D. W. (2010) CellCognition: time-resolved phenotype annotation in high-throughput 19 live cell imaging. *Nat. Methods* **7**, 747–754.

[42] Schindelin, J, Arganda-Carreras, I, Frise, E, Kaynig, V, Longair, M, Pietzsch, T, Preibisch, S, Rueden, C, Saalfeld, S, Schmid, B, Tinevez, J. Y, White, D. J, Hartenstein, V, Eliceiri, K, Tomancak, P, & Cardona, A. (2012) Fiji: an open-source platform for biological-image 20 analysis. *Nat. Methods* **9**, 676–682.

20 [43] ImageMagick. (2014) ImageMagick: Convert, Edit, or Compose Bitmap Images.

[44] Pau, G, Fuchs, F, Sklyar, O, Boutros, M, & Huber, W. (2010) EBImage—an R package for 20 image processing with applications to cellular phenotypes. *Bioinformatics* **26**, 979–981.

[45] Kohn, S, Kumfert, G, Painter, J, & Ribbens, C. J. (2001) *Divorcing language dependencies* 20 *from a scientific software library*. (C. Koelbel and J. Meza (eds.), SIAM, Philadelphia, PA).

[46] Kallio, M. A, Tuimala, J. T, Hupponen, T, Klemela, P, Gentile, M, Scheinin, I, Koski, M, Kaki, J, & Korpelainen, E. I. (2011) Chipster: user-friendly analysis software for microarray and 21 other high-throughput data. *BMC Genomics* **12**, 507.

[47] Giardine, B, Riemer, C, Hardison, R. C, Burhans, R, Elnitski, L, Shah, P, Zhang, Y, Blankenberg, D, Albert, I, Taylor, J, Miller, W. C, Kent, W. J, & Nekrutenko, A. (2005) Galaxy: a platform 21 for interactive large-scale genome analysis. *Genome research* **15**, 1451–1455.

[48] Stef-Praun, T, Clifford, B, Foster, I, Hasson, U, Hategan, M, Small, S. L, Wilde, M, & Zhao, Y. (2007) Accelerating medical research using the swift workflow system. *Stud. Health Technol.* 21 *Inform.* **126**, 207–216.

[49] Ovaska, K, Laakso, M, Haapa-Paananen, S, Louhimo, R, Chen, P, Aittomäki, V, Valo, E, Núñez-Fontarnau, J, Rantanen, V, Karinen, S, Nousiainen, K, Lahesmaa-Korpinen, A.-M, Miettinen, M, Saarinen, L, Kohonen, P, Wu, J, Westermarck, J, & Hautaniemi, S. (2010) Large-scale data integration framework provides a comprehensive view on glioblastoma multiforme. *Genome* 21, 27 *Med.* **2**, 65.

[50] Beck, K & Andres, C. (2004) *Extreme Programming Explained: Embrace Change*. (Addison-21 Wesley Professional, USA), 2nd edition.

[51] Brown, J, Knepley, M, & Smith, B. (2014) Run-time extensibility and librarization of simulation 21 software. *Comput. Sci. Eng.* **PP**, 1–1.

[52] Dubois, P, Epperly, T, & Kumfert, G. (2003) Why johnny can't build [portable scientific software]. *Comput. Sci. Eng.* **5**, 83–88. 21

[53] Hannay, J. E, MacLeod, C, Singer, J, Langtangen, H. P, Pfahl, D, & Wilson, G. (2009) *How Do Scientists Develop and Use Scientific Software?*, SECSE '09. (IEEE Computer Society, Washington, DC, USA), pp. 1–8. 21

[54] Wood, W. A & Kleb, W. L. (2002) in *Extreme Programming and Agile Methods — XP/Agile Universe 2002*, Lecture Notes in Computer Science. (Springer Science + Business Media), p. 89–99. 21

[55] Carpenter, A. E, Kamentsky, L, & Eliceiri, K. W. (2012) A call for bioimaging software usability. *Nat. Methods* **9**, 666–670. 27

[56] Hall, M, Frank, E, Holmes, G, Pfahringer, B, Reutemann, P, & Witten, I. H. (2009) The WEKA data mining software: an update. *SIGKDD Explorations Newsletter* **11**, 10–18. 27

[57] Rantanen, V. (2015) Shape filtered segmentation. `https://bitbucket.org/anduril-dev/anima1/src/tip/components/ImageSegment/shapesegment.m?at=default`. 29

[58] Altman, N. S. (1992) An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician* **46**, 175–185. 30

[59] Rantanen, V. (2015) k-NN color segmentation. `https://bitbucket.org/anduril-dev/anima1/src/tip/components/ImageSeedSegment/knnsegment.m?at=default`. 30

[60] Rantanen, V. (2015) Qalbum. `https://bitbucket.org/MoonQ/qalbum`. 30

[61] Rantanen, V. (2015) NiceCSV. `https://bitbucket.org/MoonQ/ncsv`. 32

[62] Hell, S. W & Wichmann, J. (1994) Breaking the diffraction resolution limit by stimulated emission: stimulated-emission-depletion fluorescence microscopy. *Opt. Lett.* **19**, 780–782. 35

[63] Blom, H, Ronnlund, D, Scott, L, Spicarova, Z, Rantanen, V, Widengren, J, Aperia, A, & Brismar, H. (2012) Nearest neighbor analysis of dopamine D1 receptors and Na(+)-K(+)-ATPases in dendritic spines dissected by STED microscopy. *Microsc. Res. Tech.* **75**. 36

[64] Zhang, A, Gertych, A, Liu, B. J, & Huang, H. K. (2005) Data mining and visualization of average images in a digital hand atlas. *Proc. SPIE, Medical Imaging 2005: PACS and Imaging Informatics* **5748**, 65–72. 36