RWTH AACHEN UNIVERSITY

The present work was submitted by Philips Electronics Nederland B.V. Research.

Master Thesis

Title: Image Segmentation and Semantic Description - Tools and Analytics

Presented by:
Name: Raluca-Maria Sandu
Student number: 350934

For the degree: **Master of Science, RWTH Aachen University (M.Sc. RWTH)**
in the field of study Biomedical Engineering

Evaluated by:
Institute: Department of Experimental Molecular Imaging, RWTH Aachen
Examiner: Univ.-Prof. Dr.-Ing. Volkmar Schulz
Company: Philips Electronics Nederland B.V., Research, Eindhoven
Examiner: Senior Scientist, Dr. Mounir Zeitouny

Town: Aachen, Germany
Date: March 31st, 2017

Acknowledgements

I would like to acknowledge here all those people who made the realization of this manuscript possible.

First of all, I would like to thank Dr. Jim Coombs for hosting me at Philips Research in the department of Personal Care and Wellness and the whole team for making my time there fruitful and memorable. Secondly, I warmly acknowledge my university Professor, Dr. Volkmar Schulz for accepting to guide both my internship and thesis project.

This work would not have existed and have had such a remarkable impact without the help of my supervisor, Dr. Mounir Zeitouny. I am grateful that he steadily believed in me from the beginning until the end, for his continuous teaching at any time of the day and night, and the constant professional and moral support he offered me. I would need to write another manuscript so there are enough words to express the immense contribution he brought to this work and to my overall development. Thank you and thank you. You made me believe that I can conquer mountains.

To my second Philips supervisor, Dr. Dmitry Znamenskiy, thank you for pushing me and this work forward with your brilliant ideas. This work was on an accelerated track due to your sound technical advice regarding algorithms and methods. Thanks to your vision and thorough understanding, I was able to grasp new concepts and move ten steps ahead with every discussion we had.

To Yicong Chen, thank you for your help in building the annotation website. You are an extremely knowledgeable person and I am glad you took time to share some of your experience with me. I can whole-heartedly admit that the website would not have become such a great tool without you. Together with Roman Nikitchenko we set up live this annotation tool that everyone is proud of.

Additionally, I would like to thank my friends, firstly for being my friends and staying with me through good and bad times. Firstly, to Ali, who saw something in me and never ceased to believe that I can succeed. You are a true friend and you had a great influence in making this manuscript see the light of the day. To Adriana and Andrei, who are always a source of knowledge about the world and supported me since the beginning of times. Finally, I would like to thank my family, because without them none of this would have been possible and I would not have gotten so far in life.

To come to an end, I would like to thank all the people who helped me and made my stay at Philips Research so pleasant. I loved all of it. It was one of the best experiences in my life. Thank you once again for teaching me so many things.

Contents

1 Introduction 1
2 Background 3
2.1 Skin structures 3
2.1.1 Pores 3
2.2 Related work 4
2.2.1 Software Tools 4
2.2.2 Literature review 4
3 Image Annotation Tool Software 6
3.1 User Story 7
3.2 Requirements 7
3.2.1 Functional Requirements 7
3.2.2 Non-Functional Requirements 8
3.3 Architecture 9
3.4 Implementation 10
3.4.1 Tools and technologies 10
3.4.2 Libraries 12
3.4.3 Zooming Annotations 14
3.4.4 Saving and Exporting Annotations 15
3.4.5 Exporting and Uploading Layers for Editing 16
3.5 User Interface 17
3.5.1 Use case 17
3.6 Conclusion 19
4 Feature Engineering for Image Segmentation 20
4.1 Image Pre-Processing 20
4.1.1 Morphological Operations 21
4.1.2 Contrast adjustment 22
4.2 Feature augmentation 23
4.2.1 Gaussian Filtering 24
4.2.2 Gaussian Pyramid 25
4.3 Feature Extraction 26
4.4 Conclusion 27
5 Classification Methods for Semantic Description 28
5.1 Classification Procedure 28
5.2 Dimensionality Reduction 31
5.3 Logistic Regression 33

5.4 Decision Trees 34
5.5 Random Forests 37
5.6 Conclusion 39
6 Validation Methods 40
6.1 Confusion Matrix 40
6.2 Receiver Operating Characteristic 41
6.3 Precision-Recall Curve 43
6.4 Out-of-bag error for Random Forests 45
6.5 Conclusion 45
7 Classification Results 46
7.1 Validation Procedure 46
7.2 RGB Images 47
7.3 RGB Images and Gaussian Pyramid 53
7.4 Grayscale Images and Gaussian Pyramid 58
7.5 Histogram Equalized Images and Gaussian Pyramid 60
7.6 Semantic Segmentation 62
8 Conclusion 65
List of Figures 67
List of Tables 70
Bibliography 71

Chapter 1

Introduction

Skin is a complex structure composed of various features that render its analysis challenging at a close-up level. In order to facilitate skin mapping for quantification and diagnosis purposes, tools that enable the detection and labelling of skin features are needed. It is useful to have methods that detect morphological skin structures and how these change over time due to application of skin care treatment devices or cosmetic substances. These skin structure could be anything from hairs, moles, pores or wrinkles. Moreover, these skin structures represent a unique identification tag of a specific body zone of a person. Consequently, their automatic localization could enable their use as natural skin markers in surgical navigation or image guided therapy.

Mapping the skin at pixel-level could also serve as an indicator of health status measured non-invasively at body-surface level. To exemplify, diagnosing and monitoring diseases such as psoriasis, eczema or skin carcinoma could become as facile as taking a picture in the comfort of ones' own home.

All in all, consumers want to track changes in their skin, changes that should cover both health and appearance aspects. A simple and automatic screening scheme would improve diagnosis and treatment outcomes in the case of skin health concerns. Skin mapping could empower people to take informed decisions regarding both their personal appearance and health aspects.

The focus of this work is to build a complete data analysis framework that enables skin characterization and detection of morphological structures through skin mapping. Thus, this study aims to provide semantic description of skin at pixel level by combining information related to skin texture, color and context. To achieve this goal, image annotation tools and data analytics methods for semantic segmentation are implemented and applied. Their purpose is to drive automatic classification of skin structures at pixel-level.

The structure of the skin and its elements of interest for our analysis task are discussed in Chapter 2. Additionally, we present related software tools that deal with image annotation and summarize a few publications that discuss image classification and segmentation. Chapter 3 describes the design of a software annotation tool from requirements formulation to tools and technologies used in its implementation. Several snapshots of the software's tool interface are shown at the end of Chapter 3. The annotation tool serves as a bridge between skin features and machine learning classifiers by providing labelled segmentation maps. These are in fact segmentation layers of several skin structures.

The segmentation maps facilitate feature extraction from the source images as discussed in Chapter 4. In order to detect key points of region of interest in the image, pre-processing morphological operations are implemented (Chapter 4.2.1). Contrast enhancement methods are also explored by modifying the histogram of an image. Features are the building blocks of good classification algorithms. Therefore they are augmented through building a Gaussian Image Pyramid (Chapter 4.3). Local pixel neighborhoods are extracted around key points and arranged into vector format. These vectors represent feature descriptors that, when compiled into tabular form, become the input data for machine learning classifiers.

Since labeled data is available through annotation, our classifiers will be part of the supervised machine learning models. The classification procedure and the algorithms of several machine learning classifiers are detailed in Chapter 5. Due to the high dimensionality aspect posed by image classification tasks,

dimensionality reduction approaches are explored in Chapter 5.2. This method is followed by a parametric classifier, i.e. Logistic Regression as described in Chapter 5.3. Non-parametric classifiers such as Decision Trees and Random Forest are introduced in Chapter 5.4 and Chapter 5.5, respectively.

After a machine learning classifier is fitted on input training data a model is outputted. Model validation measures the predictive power of a model in classifying future unseen images. The performance of a classifier is assessed through several methods from confusion matrix to cross-validation and their related metrics (Chapter 6).

We present and discuss our results for several combinations of input predictors in Chapter 7. These results consist of graphs for confusion matrix, receiver operating characteristic (ROC) and precision-recall curve. The plots and their derived metrics such as True Positive Rate and Area Under the Curve (AUC) indicate the expected predictive power of a classifier. Finally, we test the machine learning classifiers on an image with the aim of predicting specific skin structures. The result is semantic segmentation of an image as shown in Chapter 7.6.

Chapter 2

Background

2.1 Skin structures

Skin is the soft outer covering of vertebrates which serves as first line in the defense from external factors, insulation, temperature regulation, sensation, and the production of vitamin D [1]. Its structure is complex as it is composed of many layers with specific properties that reflect the ability of carrying out many functions [1].

Skin constitutes of higher level components such as freckles, moles, wrinkles and pores which are visible to the naked eye. These structures represent features that can be viewed as morphological variations of skin. For example, freckles and moles produce two-dimensional differences in color intensity. Wrinkles and pores cause deep furrows and flat planes and are inherently three-dimensional textures [2].

2.1.1 Pores

In this work we will focus solely on detecting pores, one of the reasons being that they can be visually quantifiable through counting. Pores can serve as starting point for skin structures mapping because they are visible to the naked eye. Our algorithms will attempt to detect and count pores automatically, and also to describe the enclosing area of a pore.

Pores, as shown in Figure 2.1, are dilated orifices of sebaceous and sweat glands that are found on the surface of skin. These glands are found within hair follicles. A high density of pores can appear in sebum-rich regions such as the face, since pores are connected to sebaceous glands. The function of the pore is to lubricate and protect the skin [2].

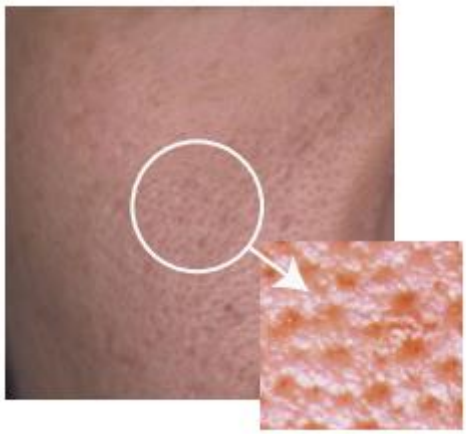

Figure 2.1 *Pores represent skin surface features or structures. Adopted from [2].*

Pores can be categorized into filled pore types where the content of pores is most visible, and enlarged pore types, where the pore and its dimensions appears enlarged and contrast visibly the surrounding tissue [3]. The enlarged pore types are distinguished by their enlarged visual appearance which contrasts visibly with the surrounding tissue [3]. Key physical characteristics of pores that deal specifically with visible pores are: ridges surrounding the pores or a barnacle-like structure that give rise to shadowing (Katsuta, Iida, & Inomata, 2004). In general pore visibility is described as a major concern for which individuals seek professional cosmetic and dermatological help [4].

2.2 Related work

In this chapter we present software tools related to image annotation and discuss publications concerned with image classification, in particular semantic segmentation.

2.2.1 Software Tools

There are several annotation tools openly available on the internet. These applications have a general scope of labelling landscape pictures in two major ways: through word labelling and outlining objects of interest. Word labelling refers to describing an image as belonging to a category. For example, an image that contains a single object such as an airplane will be labelled with the words “airplane”, “flight” and it might belong to the class “airplane”. Outlining objects refers to partitioning an image in several coherent parts and assigning a class label defined by a word. We present further some of the most popular available annotation tools.

LabelMe provides an online annotation tool to build labelled image datasets for computer vision research [5]. It was implemented using MATLAB and it has both annotation and classification abilities. An associated smartphone application allows taking pictures with the phone camera and labelling through bounding boxes and word tags that describe the contained objects.

LEAR Image Annotation Tool has been created by Alexander Kläser to use with IG02 dataset [6]. It has been implemented in Qt and it features pixel-wise object annotation, zoom in/out and different color types such background, object, occluded object among others. A mask file in raster format is created for each object separately.

ITK-SNAP is a software application used to segment structures in 3D medical images [7]. It provides semi-automatic segmentation using active contour methods, manual delineation and image navigation. This tool is built in Qt and supports many different 3D image formats, including NIfTI and DICOM.

ISIC (International Skin Imaging Collaboration) facilitates digital skin imaging technologies to help reduce melanoma mortality by advocating for international standards in dermatologic imaging [8]. It attempts to achieve this goal by creating a large open source public archive of annotated melanoma skin images. The individual lesions (moles) are annotated with pathology diagnosis information, symmetry attributes and color features. The focus of these annotations has been on dermoscopic images, since these are the most used for diagnosing melanoma.

More general applications, reminiscent of a drawing tool, are available through the web-browser, without any additional programs needed. One of these programs available through the GitHub platform is **SVG-Edit** [9]. This editor has many drawing capabilities, from zooming to drawing different shapes with different brush sizes and colors. However, it is more of a drawing tool than annotation one, as it the output is not linked with the source image.

All these applications have whether a broad scope (landscapes, objects) or a specific one (medical images). We deemed it necessary to design our own annotation tool tailored for semantic segmentation of skin.

2.2.2 Literature review

The goal of segmentation is to partition an image into multiple coherent parts. There are several types of segmentation. Foreground segmentation aims to separate the foreground objects from the background ones. Unsupervised segmentation (e.g. K-means clustering) attempts to group together pixels based on their similarity, generally described through their intensity ranges. Co-segmentation refers to identifying common objects in multiple different images. Bounding-boxes classify objects in an image by outlining them and assigning a class label. Instance segmentation assigns each pixel an object instance. Semantic

segmentation partitions an image into semantically meaningful parts and assigns a class label to each pixel in the image. It is generally synonym with pixel-wise labelling of images.

Next we summarize noteworthy publications that address image classification. Most of these studies address semantic segmentation through deep-learning methods.

Pixel-wise labelling is analyzed by Pinheiro and Collobert (2014) [10] where they attempt to infer pixels classification from weakly labelled images. These weakly labelled images consist of a bag of words that describe a picture, such as if a picture is contains a dog, its label will be class "dog". They assume that every training image has or not at least one pixel corresponding to the image class label so that the segmentation task can be rewritten as inferring the pixels belonging to the class of the object. As machine learning classification algorithm, they apply Convolutional Neural Networks (CNNs). This algorithm is not trained with segmentation labels, nor bounding box annotations. Instead, they consider a single object class label for a given image, and the model is constrained to assign more weight on important pixels for classification. The full RGB images is used as training input for the neural networks. They achieve a framework that can segment objects based on weakly supervision. They claim that their algorithm is able to distinguish, at a pixel level, the differences between different classes.

Deep convolutional neural networks are also described in the article "Semantic image segmentation with deep convolutional nets and fully connected CRFs" by Chen et. al (2014) [11]. They boost the model's ability to capture fine details by employing a fully-connected Conditional Random Field (CRF). This method is used combine class scores from classifiers with the low-level information provided by the local interactions of pixels and edges. Their model is applied directly on the raw pixels representation. The focus of their study is on fine-tuning the parameters of the neural network. The resulted model is tested on the PASCAL VOC 2012 segmentation benchmark (Everingham et al., 2014), consisting of 20 foreground object classes and one background class. They measure the performance of the model in terms of pixel intersection-over-union (IOU).

A publication related to classification of skin structures is "Dermatologist-level classification of skin cancer with deep neural networks" by Esteva, Krupel et al. (2017) [12]. They demonstrate classification of skin melanoma using Convolutional Neural Networks which they train directly on images using pixels and disease word labels as inputs. Their dataset consists of 129,450 clinical images comprising 2,032 different diseases labelled by dermatologists. The effectiveness of the neural networks algorithm is validated through nine-fold cross-validation and plotting the associated receiver operating characteristic curve (ROC). The gold-standard against which they test their labels is biopsy confirmed. They test whether the algorithm and dermatologists could distinguish malignant versus benign lesions of epidermal or melanocytic origin. Their conclusion is that the algorithms outperforms the dermatologist in most cases in classifying correctly the type of skin lesions. The area of the curve (AUC) for each classification case is minimum 0.91 which indicates almost perfect performance in classifying correctly future images.

Focusing solely on the topic of pores are Shaiek et al (2016) [13], which specifically developed an imaging device for their measuring and quantification. They measure in a small region of interest the facial skin pores and aim to quantify the impact of age and make-up upon their morphological features. A dedicated software, entitled "PoreTracker", has been developed to characterize the pores automatically. They calculate the size of pores in mm^2 and pore density per cm^2. For the *in* vivo validation part, they compute a correlation coefficient from linear regression between experts' scores the computed pore size. They define 6 categories of pores sizes and conclude based on a visual quantitative assessment that with age the shape of pores becomes more elongated, but their size and density remains constant.

Chapter 3

Image Annotation Tool Software

Based on a specific set parameters, image analytics algorithms can learn and automatically map thousands of future images, such as images uploaded through a smartphone application or from a hospital server. Being built on a set of images that have been labeled using a priori ‘’experts” knowledge, algorithms will able to propagate articulate labels that enable diagnosis and enhancement of therapy.

Towards the goal of identifying skin structures rapidly and accurately, learning material is needed in order to build powerful classification algorithms. Analyzing a digital skin image with classic image processing algorithms (e.g. thresholding) will describe the picture from a global point of view by dividing it into multiple coherent parts that visually stand out relative to their neighbors. Nonetheless, these algorithms have no sensitivity power, nor do they describe and understand images’ pixels. Thus they fail make distinctions between different skin conditions. To exemplify using a picture, Figure 3.1 shows at the top a black and white image which is the result of image thresholding. When provided with this image alone, both computer applications and people will have difficulty pinpointing whether the source of this processed image is the left or the right bottom image.

Figure 3.1 *Image Thresholding describes the image from a global point of view and offers no specificity regarding the skin condition.*

Semantic segmentation on the other hand, is superior to segmentation alone such as image thresholding, as it aims to partition the image into semantically meaningful parts and to assign a class-label to each pixel. In order to apply semantic segmentation algorithms, classification-based models need to be trained on labeled data. Firstly, it is essential to manually map images in a semantic way that will provide ground truth labels for supervised machine learning algorithms. Therefore, tools to annotate images are needed in order to train an efficient classifier model. To this end, a web-browser based tool that enables annotation of skin images by experts was created.

This chapter starts with the user story that explains a potential usage scenario of the image annotation tool. We define function and non-functional requirements that drive the choice of implementation. Available tools and technologies are presented and we explain the implementation details. Lastly, the user interface and the user’s manual of the annotation tool are presented together with screenshots of the application.

3.1 User Story

The following user scenario is described in order to understand the functionalities of the image annotation tool and to define the requirements.

Raluca is using a smartphone application for personal care in order to track the appearance and the changes in pores and wrinkles. She is using this application to check if the new cream she is applying or the treatment device she is using has any measurable effect on her skin. She is taking a picture, uploads it to the application which displays physiological structures and skin related metrics: wrinkles and pore per surface area, wrinkle and pores depth and dimensions, skin tone color and other metrics. The application is also showing Raluca trends over time so she could monitor how her skin changes. After using a treatment device for some time, Raluca notices some redness in her skin. Nevertheless, she dismisses it as a harmless reaction of the skin. She is still using the smartphone application to track changes in the wrinkles appearance. This time it also raises an alert that her redness might represent the onset of dermatitis. Raluca does not know what dermatitis is, so she goes online to check. Afterwards, she decides to visit a medical professional.

Due to the prompt notification from the smartphone application, Raluca goes timely to the doctor to investigate her skin problems. While waiting in the doctor's office, she notices that one of the medical staff people is using an online web application. On a closer look, she notices how the person is uploading an image from the computer and then it proceeds to mark areas of the skin with specific colors. Raluca asks the medical staff what is the web application for. The person explains to her that she is using this application to label physiological skin structures by coloring them. This is more or less like coloring a kids book in a way that is meaningful in order to describe what the objects are - for example trees will have green leaves and a brown trunk.

Being a web application, it empowers collaborative work so that expert personnel can label skin structures across different areas of research at the same time. The annotations made in dermatology department are related to skin diseases like dermatitis or acne, but also moles, pores and blood vessels. The latter serve as skin markers for image guided therapy. Moreover, doctors could also this application in order to track remotely the progress of moles or malignant melanoma. In the department of infectious disease epidemiology, the medical staff annotates skin characteristics ranging from ring-worm diseases to Lyme disease.

All in all, labeled images are at the foundation of any skin-tracking application. Annotation tools are needed so that a knowledge based database of labeled images could serve as learning material for automatic image analytics. Thousands of skin images could be mapped instantaneously at the smallest level and as accurately as possible.

3.2 Requirements

The requirements for the *Image Annotation Tool (IAT)* were categorized into two parts, functional and non-functional. Items under each category are presented in the following subsections.

3.2.1 Functional Requirements

Currently *IAT* is in a state that allows annotation of images for analytic purposes. The annotation of an image can be done for various skin structures that are identified by different semi-transparent colours. IAT is designed with flexibility and robustness in mind allowing editing, saving and correction of layers over time. Table 1 contains the functional requirements formulated for IAT.

Table 1 *Functional Requirements described for IAT*

No.	Name	Description
F.1.1	**Upload Image**	*IAT* must be able to upload images from the file system or database and display it so that the image can be annotated.
F.1.2	**Draw on Image**	*IAT* must provide means for drawing on top of the loaded image with the purpose of annotating several skin structures.
F.1.3	**Erase from Image**	*IAT* must have the option to erase strokes drawn on the image, so that a drawing mistake can be removed. The option to clear all the strokes has to exist as well.
F.1.4	**Retrieve Deleted Strokes**	*IAT* must enable undoing and redoing action of erasing and drawing strokes so that editing is simple.
F.1.5	**Magnification factor**	*IAT* web tool must provide several means to zoom in and out the image simultaneously, without disturbing the strokes drawn.
F.1.6	**Scrolling the Image**	*IAT* must allow scrolling an image both vertically and horizontally, in the case the image dimensions exceed those of the default drawing window.
F.1.7	**Export and Save Layers**	*IAT* must save drawn annotation layers as binary masks and export them separately to file.
F.1.8	**Export Layers for editing**	*IAT* must save and export all annotation layers in a format that allows loading and editing over time.
F.1.9	**Load Layers**	*IAT* must be able to load previously made annotations layers on the related skin image so that they can be edited.
F.1.10	**Data aggregation**	*IAT* must manage data in a way that allows to reproduce, which data belong to a common record.
F.1.11	**Common Format**	*IAT* has to store annotated data in several common formats that allows further processing of data with other tools.

3.2.2 Non-Functional Requirements

The application will be utilized by people with different backgrounds and different technology knowledge. *IAT* has to persist its interface functionality, scale and adapt to the user's screen. *IAT* must also be able to handle the addition of new features. The application has to meet several requirements regarding usability, so that people are able to use it without any problems on any type of screen. Table 2 presents the non-functional requirements defined for *IAT*.

Table 2 *Non Functional requirements of IAT*

No.	Name	Description
NF.1.1	**Easy Starting**	The users must be able to start the *IAT* in a non-complicated manner.
NF.1.2	**Availability**	*IAT* should be available and able to run 100% of the time. Exceptions are a new update or version being pushed to the server.
NF.1.3	**Ease of Use**	The target group of application are people with different backgrounds that have been assumed to be layman regarding computer technology. *IAT* user interface should be intuitive to use for everyone.
NF.1.4	**Sufficient Feedback**	*IAT* must provide sufficient feedback to users, so that they understand how they should operate the application and what the restrictions or limitations are.
NF.1.5	**Scalability**	*IAT* should be able to accommodate a growing amount of features at minimal effort and changes.
NF.1.6	**Extensibility**	*IAT* software framework should have as many separate components as possible. Adding new features and enhancements should be part of a continuous development process.
NF.1.7	**Interoperability**	*IAT* should be able to interact with other devices like camera devices. *IAT* has to interact with other information systems or tools. These can be database platforms or executable programs that enable automatic annotation.
NF.1.8	**Response Time**	*IAT* must run without any particularly lag or delay, particularly when drawing. Using the application should imitate real-life pen on paper drawing as close as possible.
NF.1.9	**Data Storage**	*IAT* must allow the user to store data to the internal memory, external memory or to a cloud storing service.
NF.1.10	**Data Security**	In health related applications, privacy is of paramount importance. *IAT* must handle data in such a way so that that no risk of data breach exists.
NF.1.11	**Data Administration**	*IAT* must guarantee minimal authentication so that data are not accessible by any unauthorized instance.

3.3 Architecture

Application architecture attempts to link business requirements and technical requirements by understanding use cases, and then finding ways to implement them in the available software options. The goal of software architecture is to identify the requirements that affect the structure of the application. A good design is sufficiently flexible to be able to handle the natural changes that will occur over time in hardware and software technology, as well as in user scenarios and requirements. The architecture must consider the overall effect of design decisions, the tradeoffs between performance and those required to address user, security, and business requirements [14].

Bearing these points in mind, there are several aspects that influence the choice of architecture for *IAT.* First of all, it has to be lightweight and portable, needs a simple user interface, should be available to many users at the same time and must be extensible and maintainable over time. The users should be able to use the application independent of the operating system. As long as the device has an internet connection and a Web browser, *IAT* should be ready to use.

Furthermore, *IAT* was designed to be a web editor tool that replicates real drawing, but annotating complex structures (e.g. edges, curved lines) using the mouse does not provide sufficient precision. The devices that provide the natural pen on paper drawing experience are tablets with digital pens. Consequently, the application should be independent on the type of the screen and its size, whether it has touchscreen technology or only mouse interaction (Figure 3.2).

Figure 3.2 *IAT requires a responsive website that can scale and interact with different devices. Image adopted from namwebsites.com/website-development/.*

The architecture of *IAT* is not standalone, instead it is built on top of two main vector graphics libraries, *RaphaëlSketchpad.js*[1] and *Raphaël .js*[2]*.* These web drawing editors are at the foundation of the annotation functionality in the *IAT*. They represent the core application, providing the essential functionality. Other libraries that enable additional functional requirements are added alongside the core application. The HTML5 Boilerplate front-end template is the scaffold that brings these libraries together, using a structured and adaptable framework.

3.4 Implementation

The *Image Annotation Tool* is implemented as an application, available on computer devices, tablets and smartphones through the Web browser. It is an online drawing editor that uses vector graphics elements. *IAT* is based on HTML5, CSS, JavaScript, jQuery and in this work we only describe the client-side functionality attached to it.

3.4.1 Tools and technologies

HTML5 is the latest standard programming language for creating web-based applications. The appearance of the website layout is defined by CSS. The CSS elements become interactive to the user with jQuery and JavaScript (JS) programming languages. The annotations made on the images are stored in the JSON[3] format. The annotations can be exported offline in SVG or Raster (Canvas) format. In a further step the annotations are processed for analytics purposes using Python programming language (Figure 3.3).

[1] https://ianli.com/sketchpad/

[2] https://github.com/DmitryBaranovskiy/Raphaël

[3] http://www.json.org/js.html

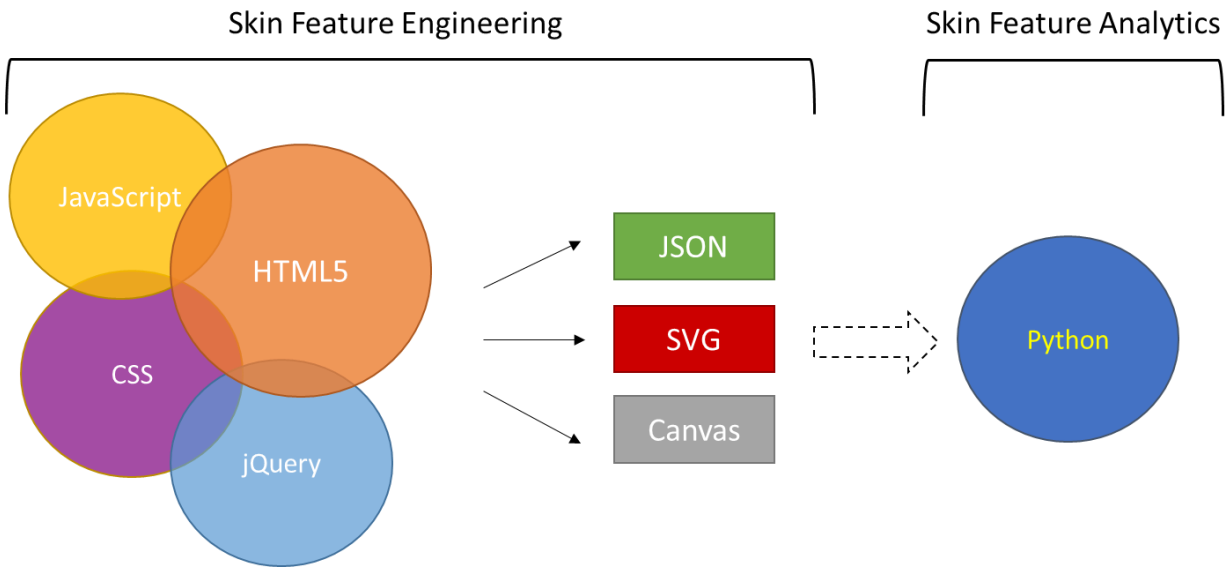

Figure 3.3 *Programming Languages and File Formats used in IAT. The dashed line indicates a transfer to the Python environment through segmentation maps of JSON, SVG or Canvas format.*

At the foundation of *IAT* are two main libraries written in JavaScript: *Raphaël .js*[2] and *RaphaëlSketchpad.js*[1], which was derived from the former. These libraries implement one of two main technologies used for web drawing, this is the SVG (Scalable Vector Graphics), a 2D format for images based on the XML mark-up language. Vector images are defined using algorithms that implement shape and path definitions (straight and curved lines) [15]. They are basically parametric functions that take a number of points in order to build lines in several directions.

The other element provided by HTML5 is Canvas, a resolution-dependent bitmap canvas [16] that can be used to draw graphics via JavaScript. Raster images are defined using a grid of pixels, where each pixel is defined by location coordinates and color information. Popular formats include Bitmap (.bmp), PNG (.png), JPEG (.jpg), and GIF (.gif.)[15].

To choose between HTML5 SVG and Canvas HTML5 for building *IAT* the following aspects which derived from the requirements were taken into consideration:

- *Image Annotation Tool* is an application-oriented tool with tailored drawing capabilities
- Interaction with annotations is desired through click and select
- Separate level between the source image and the annotations
- Fast update of annotations layers internally for saving purposes
- Possibility to edit annotated layers at a later date by loading them onto the associated image
- Keep *IAT* lightweight considering extensibility : as few code lines as possible

Table 3 presents a comparison between HTML5 SVG and Canvas HTML5 as defined by [15],[16],[17],[18]. Taking into consideration all the above specifications plus the widespread availability of open-source vector graphics libraries SVG was chosen for the *IAT.*

Table 3 *Comparison between the main HTML5 technologies used for web drawing*

Canvas	SVG
Canvas has no knowledge of the object drawn, everything is pixels with no information about location or content	Every SVG object is a HTML element that is part of the DOM (Document Object Model) so JavaScript events can be attached instantly
Canvas works faster with thousands of objects and precise bitmap manipulation but all the concepts of managed state have to be programmed	Selection and moving around of objects is already implemented so much less code is needed to keep track of the objects and manipulate them through mouse click events
Event model/user interaction is rigid because interactions with the canvas elements must be manually programmed from mouse coordinates	Event model/user interaction is object-based at the level of primitive graphic elements—lines, rectangles, paths
Canvas based images look pixilated when zoomed in because each pixel is increased in size to fill the new dimensions	The SVG image can be zoomed without looking pixilated, because each shape is scaled based on algorithms

3.4.2 Libraries

Raphaël.js[2] is a JavaScript library that allows to create SVG scenes programmatically through a unified API (application programming interface). It uses the SVG W3C Recommendation [17] and VML as a base for creating web graphics[2]. To create a vector graphics drawing we simply start by instantiating a *Raphaël* object. This library has a lot of functions for drawing basic shapes like circles, rectangles and paths[4] (lines and curves) using only very few code lines. Going a step further, *RaphaëlSketchpad.js*[1] is a library that already implements a basic drawing editor that creates path[4] elements from translating the mouse events and movements. There are several libraries that implement web drawing editors, but these libraries were selected because of several reasons which are detailed further.

The integration of *Raphaël* libraries into *IAT* supports multiple drawing editors on the same web-page. These editors could act whether as a drawing pad or as a viewing pad by simple switching between the True/False states. The editor/viewer modes were adapted for *IAT* as 2 layers, where the former allows drawing annotations on the loaded image, while the latter saves the annotations for exporting purposes (Figure 3.4). Secondly, *Raphaël Sketchpad* stores the path or strokes information in JSON[3] (JavaScript Object Notation) format. This format consists of attribute-value pairs that are easy to read and write. The viewer layer stores the annotated *path* information into a string in SVG format. Next, the JSON representation of the string can be exported to a *.txt file that preserves the annotation data for later editing purposes.

Libraries used for building *IAT* are presented in Table 4.

[4] https://www.w3.org/TR/SVG/paths.html#PathElement

Code for Sketchpad Editor and Skethchpad Viewer

```
<div style="background-color:#FFF; height:653px; overflow: scroll;" class="widget">
                    <div id="sketchpad_editor"></div>
                </div>

                <div id="viewer" style="display:none">
                    <h3>Viewer</h3>
                    <p>
                        <a href="javascript:void(0);"
id="update_sketchpad_viewer"></a>
                    </p>

<div style="height:653px;" class="widget">
                        <div id="sketchpad_viewer"></div>
                    </div>
                </div>

                 <div id="result_div" style="display: none">
                    <h3>Result</h3>
                    <p>
                        The sketch is stored as JSON in an input field.
                    </p>
                    <form action="" method="post" onsubmit="return false;">
                        <textarea id="input1" name="input1"></textarea>
                    </form>
                </div>
```

Code 1 *Sketchpad Editor and Sketchpad Viewer as defined in the index.html page of IAT.*

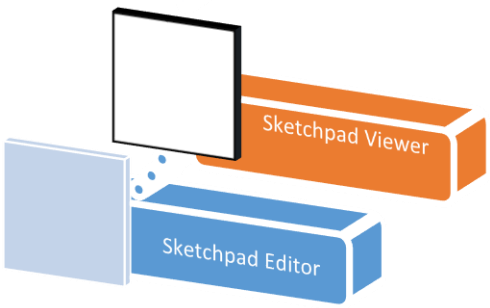

Figure 3.4 *The Raphaël Sketchpad library implements viewers and editors. These were adapted for the IAT into a 2-layered structure. The editor is displayed to the user, while the viewer is hidden and used for exporting the annotations.*

Table 4 *Main libraries used in building the IAT*

No.	Library/Tool	Purpose
1.	*HTML5 Boilerplate*	Front-end template framework for building a website
2.	*Bootstrap*[5]	CSS, HTML and JavaScript framework or developing responsive, mobile first projects on the web
3.	*Bootstrap Toggle* [6]	Bootstrap Toggle plugin that converts checkboxes into toggles
4.	*jQuery*[7]	DOM (Document Object Model) manipulation library
5.	*JQuery Migrate*[8]	Migrate older jQuery (v.1.4.2) needed for *Raphaël Sketchpad* to newer version compatible with the *Bootstrap* library
6.	*Raphaël .js*[2]	Vector drawing library that enables SVG drawing through an API
7.	*RaphaëlSketchpad.js*[1]	Drawing editor built on top of *Raphaël* library that translates mouse events to path elements. Implements basic drawing features like stroke size, stroke color, erase, undo/redo.
8.	*JSON Package*[3]	Stores the path information in JSON format
9.	*Raphaël Export*[9]	Export *Raphaël* paper objects to SVG strings. Used together with *Canvg.js* to convert SVG strings to canvas element.
10.	*Canvg*[10]	JavaScript SVG parser and renderer. It takes an URL to a SVG file or the text of an SVG file, parses it in JavaScript, and renders the result on a canvas element. Enables SVG → Canvas→ *.PNG transition on the client side.

3.4.3 Zooming Annotations

Raphaël Sketchpad implements a web editor with basic drawing functions, but it lacks more advanced features that fulfill the requirements defined for *IAT* (F.1.5). Zooming the pictures is essential in order to draw detailed and precise annotations for various skin structures. Although touchscreen devices have native zooming functionality triggered by pinching events, if the entire webpage is zoomed a drawing offset appears in the sketchpad editor window. In order to avoid incorrect drawing coordinates, we disable the browser's native zooming by setting the properties of the viewport meta-tag to `user-scalable = no`. The idea is to use to zoom in/out solely in the *Raphaël* sketchpad editor window, while the webpage stays the same size. To this end a scaling function was implemented and attached to zooming in and out buttons through mouse click events or mouse wheel events (where mouse is present).

[5] http://getbootstrap.com/
[6] http://www.bootstraptoggle.com/
[7] https://jquery.com/
[8] https://plugins.jquery.com/migrate/
[9] https://github.com/AliasIO/Raphaël .Export
[10] https://github.com/canvg/canvg

Algorithm for scaling the surface of the drawing window

```
_scaling_ratio = 1; // reset the scaling ratio
self.scaling_ratio = function() {
    return _scaling_ratio;
};
//----------------SCALE FUNCTION--------////
self.scale = function(width, height) {
    // set new width and height to paper
    _paper.setSize(width, height);
    // scale according to old dimensions
    _paper.setViewBox(0, 0, _options.width, _options.height,true);
    // scale stroke width
    _scaling_ratio = width / _options.width;
    _strokes = _action_history.current_strokes();
    _redraw_strokes();
    _fire_change();
    return self;
};
```

Code 2 *Scaling function written in JavaScript for zooming in the image and its annotations.*

3.4.4 Saving and Exporting Annotations

In *IAT* exporting the drawn annotation layers is of paramount importance since these will be later processed with other tools for analytics purposes. The annotated data is stored internally by updating an input field with the values from the *JSON function* when a change event on is triggered. The sketchpad viewer role is highlighted when exporting the annotations drawn. The sketchpad viewer is designed to store only the stroke information, without the source loaded image as canvas. A white background is added on the viewer in place of the initially loaded image. Since the image from the viewer contains only white pixels and one another single color, the product of this export is identified as the binary mask (Figure 3.5).

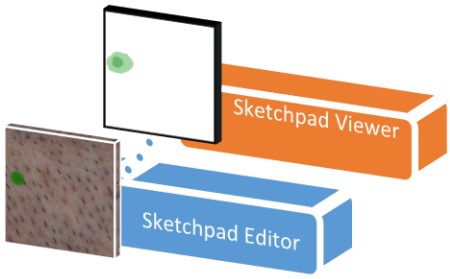

Figure 3.5 *Layer structure of IAT. Sketchpad editor displayed to the user and hidden sketchpad viewer layer used for exporting only the annotations.*

The shortcoming posed by *Raphaël Sketchpad* is that it erases the selected strokes only visually (F.1.3). This means that the internal structure of the strokes is not updated with what the user sees on the screen. The problem appears when the user desires to export the annotations offline. In order to render correct annotations when exporting, the selected for erasing strokes (SVG path elements) are converted to string objects. Next, a loop iterates through the stored strokes and removes the strokes selected for deletion if the string comparison condition is met.

Once the strokes data are saved in a consistent way, we can proceed to export the annotations offline (**F.1.7**). After studying the requirements defined earlier, the raster format (*.PNG) was chosen as being the easiest format that could be processed further for analytics purposes. Since the application is based on vector graphics, some conversions are necessary to obtain the PNG annotations.

We start by exporting the drawn JSON stroke objects to SVG string format using the library *Raphaël Export*[9]. Next, it is desired to have each type of annotation exported to a separate file. Skin structures are labeled with a different stroke color in order to tell them apart. Since each structure present will be identified by its unique color, we will split and group the SVG strokes string based on that. At the moment the separation of strokes based on color has to be done manually, because each layer color has a semantic meaning. This means that after we parse each SVG string layer to a canvas element (using *Canvg*[10]*)* we convert them to separate PNG files named in concordance with their semantic labels. Thus, when exported offline, all the annotated content are represented as raster binary masks that contain only the pixel information of a single skin structure in one color and white noise the rest.

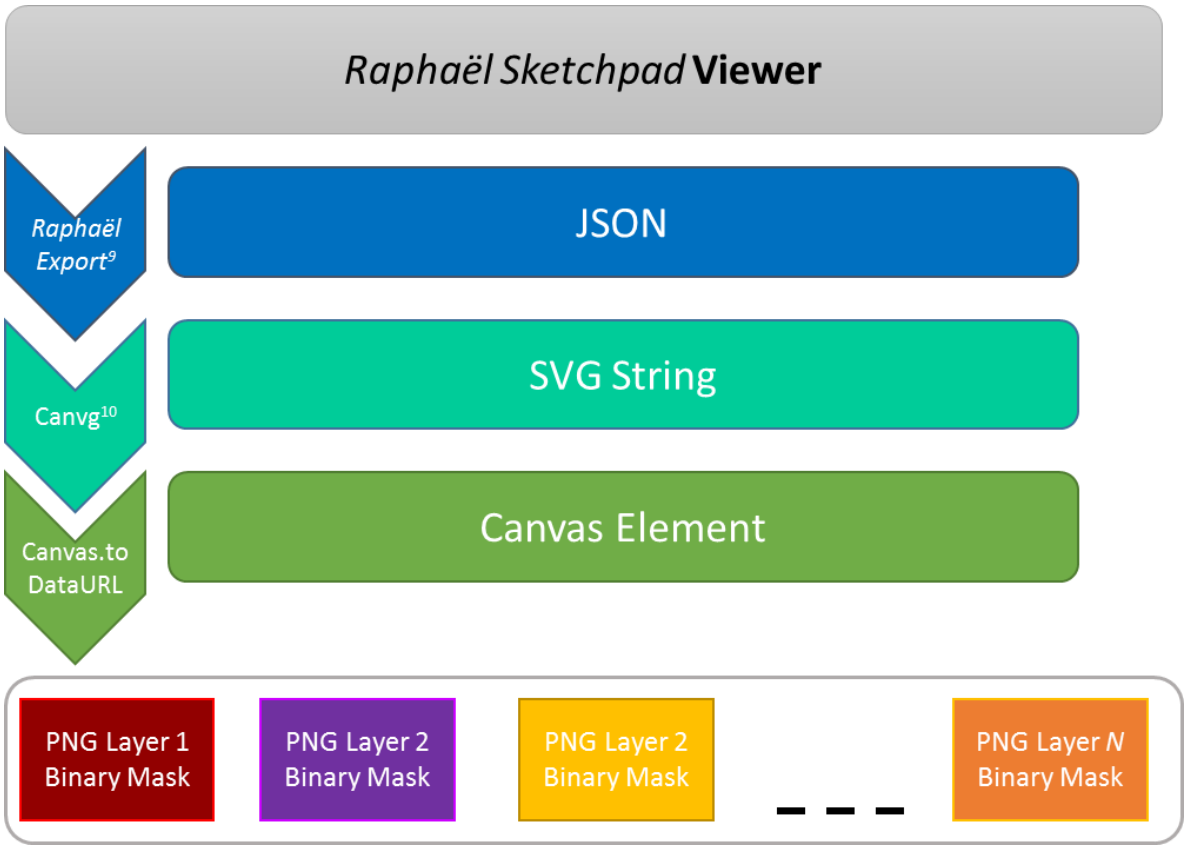

Figure 3.6 *Workflow diagram for saving and exporting the annotation from JSON to Raster Layer Image. The arrows depict which libraries or functions were used for the respective data conversion.*

3.4.5 Exporting and Uploading Layers for Editing

As defined in the requirements, we want to make annotations on an image and have the possibility to edit them over time (F.1.8). The current drawn strokes from *Raphaël Sketchpad* were retrieved from the *JSON function* and exported as a string into a text file that can be downloaded (Figure 3.7). This time, the annotation layers are not exported into separate files, but saved in only one file. The text file containing the annotation data can be loaded at a later time onto the related image (F.1.9). The user can pick up the annotations from the last state, erase strokes or add new ones.

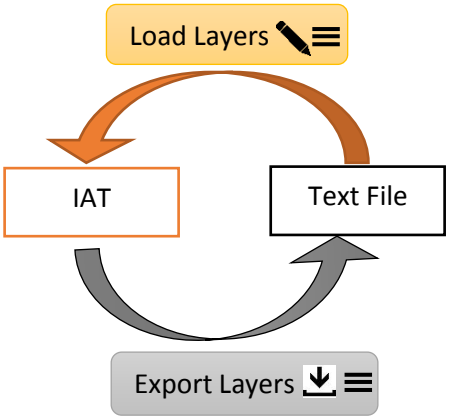

Figure 3.7 *Exporting the JSON annotations from IAT to text file. The layers can be loaded later into IAT from the respective text file.*

3.5 User Interface

IAT is a browser based SVG editor that provides a user interface reminiscent of MS Paint. The interface is built on top of the CSS Bootstrap[5] framework. The Bootstrap grid system was used to create a responsive HTML website meaning the content will re-arrange depending on the screen size. At the moment, *IAT* has only one web page with the main features in the navigation header, the drawing options displayed on the sides and the drawing canvas in the center.

3.5.1 Use case

The user starts the application from the browser and authenticates himself with the correct credentials. Not all browsers parse the webpage correctly due to obsolete technology or syntax incompatibility. The *IAT* has been adapted for the following browsers: Google Chrome (version >7.0), Mozilla Firefox (version > 3.6), Opera (version > 12.0) and Safari (version > 6.0).

When the user enters the application there is no image displayed in the central editing window and the drawing options are disabled (Figure 3.8). Next the user loads an image from file, clicking *Upload Image* from the menu *Load File,* displayed in Figure 3.9. The image is then displayed in the central editor in its source dimensions. To navigate the image, the user can click the native zooming icons from the left side menu (or use the mouse wheel if available) and use the scroll bars (Figure 3.8).

In order to start annotating, the user selects a color from the right menu and proceeds to draw on the image. The size of the brush can be changed also from the right menu, from the slider bar or from the textbox input. If a mistake is made, it can be deleted by clicking the toggle button *Draw/Erase stroke* (Figure 3.10). Also in the left menu the user has the option the modify strokes already drawn, by erasing or undoing/undoing them. There is also the option to clear all the strokes from the canvas image with *Clear All*.

After the annotations have been made they can be exported to several file formats for further processing or later editing. In the *Export File* menu (Figure 3.9) are 3 options for exporting the annotations: as PNG Image, Text file or Vector Image. The PNG images are processed with Python in the next step. Even though the user might consider that the annotation for a certain image is finished, it is recommend to export it also in text format to be able to visualize or edit later using *Export File* → *JSON Layers* (Figure 3.9).

Feedback alerts have implemented to act as a help guide for the user and make the application more interactive. Alert boxes pop up when the wrong file types are loaded, e.g. the layer file can only be loaded

in *.TXT format. Moreover, we do not want malicious files to be loaded, so only the most common image file types (*.PNG, *.JPEG, *.BMP, *.SVG) are allowed. The user is also prompted to take action if changes made have not been saved to file before the application is closed.

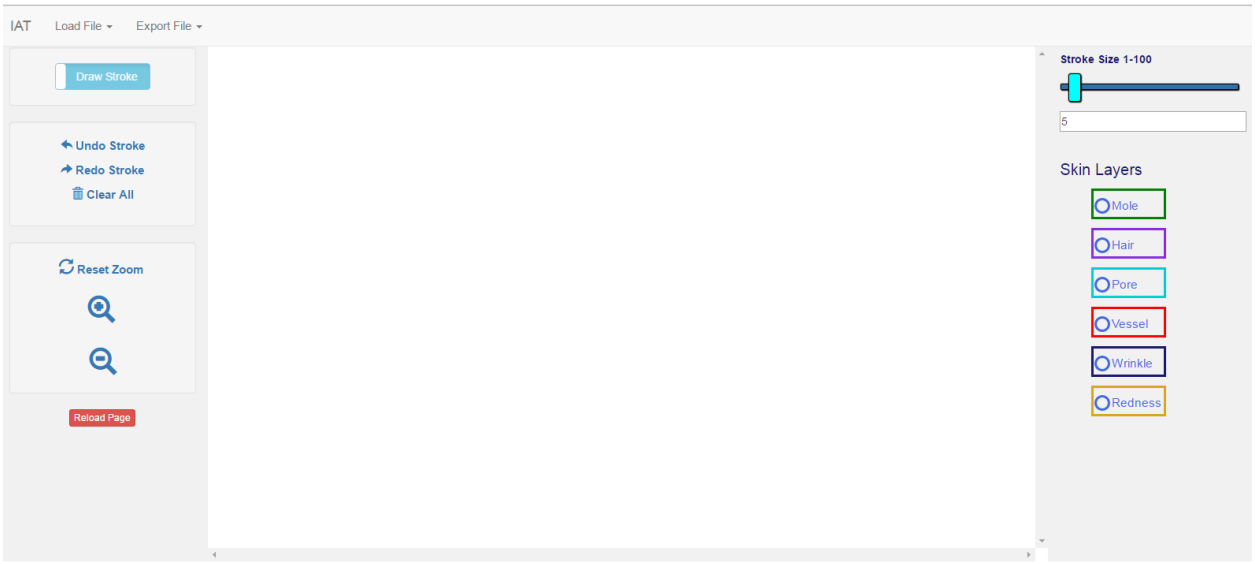

Figure 3.8 *The IAT user interface immediately after being opened in the browser.*

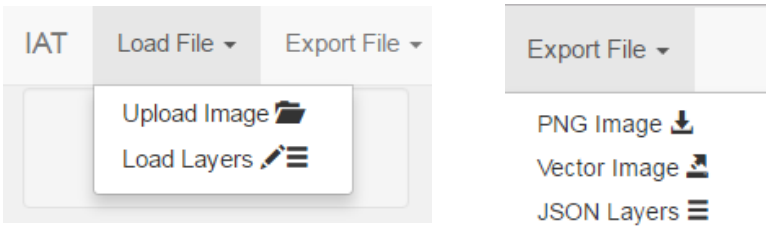

Figure 3.9 *The loading and exporting menus from the navigation header.*

Figure 3.10 *IAT user interface after an image has been loaded. A mole structure has been annotated (green color). The user desires to erase some strokes so the toggle button displays the state "Erase Stroke".*

3.6 Conclusion

This chapter discusses why annotation tools are needed which leads to defining the functional and non-functional requirements. The end result was the design of a website – a drawing tool that allows the segmentation of skin structures on individual layers so they can be later used as training labels for supervised learning algorithms.

After annotating a picture, several layers can be outputted for each skin structure. After annotations have been created on a picture, they can be exported to several file formats for further processing or later editing. Annotations can be made with different brush sizes and can be erased or modified overtime by the user. To make processing of source images and layers consistent and error-free, annotations are stored as separate segmentation maps with the same resolution as the loaded image, while preserving the filename of the source image for the layer files. This way, the annotated segmentation maps can be retrieved back to the source image that was used.

The combination of different segmentation layers will form the semantic description of the image's pixels. These annotated layers will be used to train algorithms for automatic image analytics methods described in the subsequent chapters.

Chapter 4

Feature Engineering for Image Segmentation

Image classification algorithms take several images as input and outputs the contents of an image in the form of class labels (e.g. pore, mole, hair). For semantic segmentation the desired classification output is pixel-wise labelling. After images are acquired and their labels are assigned through annotation, feature extraction is performed. This step will serve as the learning foundation on which classification models will be build. Before features are extracted, the input images are pre-processed in order to normalize contrast and brightness effects, but also to enhance local regions of interest that could serve as hallmarks of a certain class object. The feature engineering workflow is summarized in Figure 4.1.

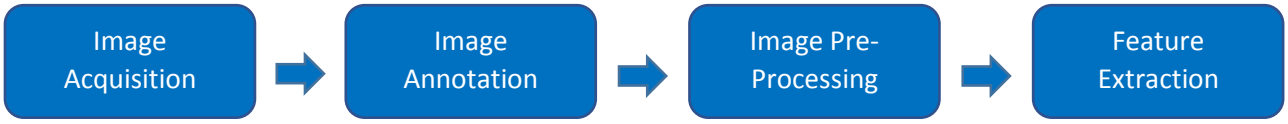

Figure 4.1 *Workflow for Feature Engineering.*

Image acquisition was carried out in using a skin analysis camera system (EH900u, 5.0 megapixels resolution sensor, Figure 4.1). The device is interfaced with the computer through an USB cable and a Python script was implemented to acquire images from the face area of several subjects.

Figure 4.2 *Device system EH900u used for acquiring skin images.*

After images of several subjects have been acquired, experts in skin care use the annotation tool website to annotate a set of images which is dubbed as the training set. The training data, consisting of labeled segmentation layers serves as ground truth labels for the machine learning classifiers. The annotated images provide not only the coordinates or key points of the pixels from the region of interest, but also class labels. Further, using the segmentation masks, true positive and true negative labels are defined using morphological image processing techniques. Since the classification goal is semantic segmentation, features will consist of raw pixels from different augmented representations (RGB, Grayscale, Gaussian pyramid). These pixels intensity values will be extracted from acquired images as blocks of pixel neighborhood around key points provided by segmentation maps.

4.1 Image Pre-Processing

In the context of image classification task, pre-processing algorithms play a significant role as they open the path to good feature engineering. These algorithms put emphasis on important characteristics and remove noise from an image. Images can be filtered for noise with various low-pass filters (e.g. Gaussian Blur, Median Filter) and augmented for contrast with high-pass filters or histogram adjustment.

Since we already have annotated data in form of segmentation image masks, location and class information has to be extracted and retrieved back to the source image in the first instance. This is achieved using morphological operations of dilation and erosion. Next, several pixel's representations are explored, including grayscale, transformations based on the image's histogram and Gaussian filters.

4.1.1 Morphological Operations

When people use the image annotation tool to label an image by coloring on it, they create a map of coordinates that point to the regions of interest present in the image. Given a task to annotate some specific skin structures, people will generally annotate the most visible ones while being inaccurate in following the outline of these structures closely. Also, annotating an entire picture is a time-consuming task so experts annotate just a few central structures.

Bearing all these points in my mind, morphological operations were chosen in order to define the true positive (1's class) and true negative labels (0's class). Thus, we consider the surrounding of the annotated structure to form the negative class (0) and we retrieve the associated pixels using the dilation operation. For the positive class (1) the erosion operation is used to determine the pixels considered a core characteristic of the structure in discussion.

By applying the operations of erosion and dilation we remove noise induced by human annotation and achieve a roughly equal data distribution between the positive and negative classes. Moreover we aim to preserve the natural distribution of the classes' occurrence (positive class being less frequent than negative class, i.e. pores pixels will be fewer than skin pixels).

Morphological operations are typically applied on binary images by convolving the source image with a structuring matrix element, which is also called kernel. The kernel size can be matrix of ones with different shapes (square matrix, cross kernel, elliptical kernel). This pre-defined kernel is sliding across the source image (as in 2D convolution) and the value of any given pixel in the output image is pre-determined by a rule and the values of all the pixels in its neighborhood. The formulation of this rule decides between the operation of erosion or dilation [19].

Erosion removes pixels from the boundary of the foreground objects (with foreground in white and background in black as a common convention). Much like its name, it minimizes the area of objects in cause (Figure 4.3). A pixel in the source image (either 1 or 0) will become 1 in the output image only if all the pixels under the kernel are 1, otherwise it is eroded [20]. So, if for every pixel in the structuring element, the corresponding pixel in the image underneath is a foreground pixel, then the input pixel is left as it is, otherwise is set to 0 [21]. Erosion is similar to the logical AND operation. The number of pixels discarded from the boundary depends on the size of the kernel.

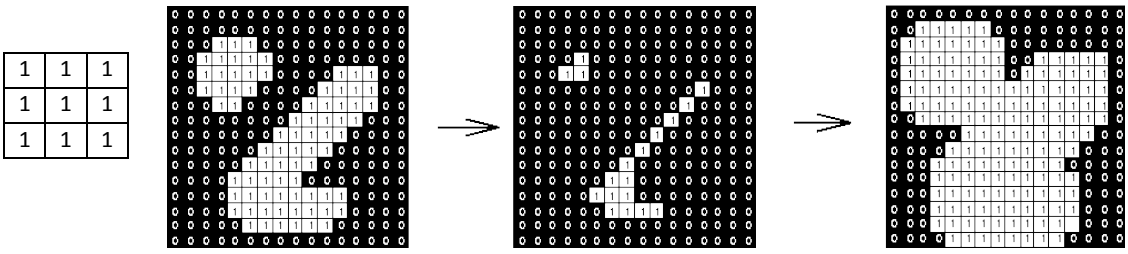

Figure 4.3 *Left side: a 3x3 structuring square element or kernel. Right Side: source binary image with example of morphological erosion (center) and dilation (right). Source : [8], [9].*

Dilation is the contrasting operation to erosion, as it adds pixels to the boundaries of the objects in the image. It enlarges the area of the object in foreground so it includes the surrounding neighborhood around the regions of interest which form the negative class in our assumption scenario [20]. If at least one pixel in the structuring element coincides with a foreground pixel (1's) in the image underneath, then the input pixel is set to 1, otherwise if all the corresponding pixels are 0 (background), the value is set to 0 [20], [22]. Dilation is similar to the logical OR operation (Figure 4.3).

Figure 4.4 shows the output of the erosion and dilation operations on a sample image from the device system used (Figure 4.2). For erosion, a square structuring element matrix of the size 2x2 (*m x n)* has been used, while for dilation a 20x20 square matrix was used. The contours of the objects retrieved from the binary image (second image from left, Figure 4.4) were displayed on the source image [23]. Although a square kernel shape has been used in this case, an ellipse kernel might be more suitable to approximate the shape of circular structures like pores. Moreover, another assumption made in this case was to consider the annotations as a "gold standard", even though every annotation will be person's dependent. This is why, if available, it is recommended to aggregate the annotations from several experts on the same dataset.

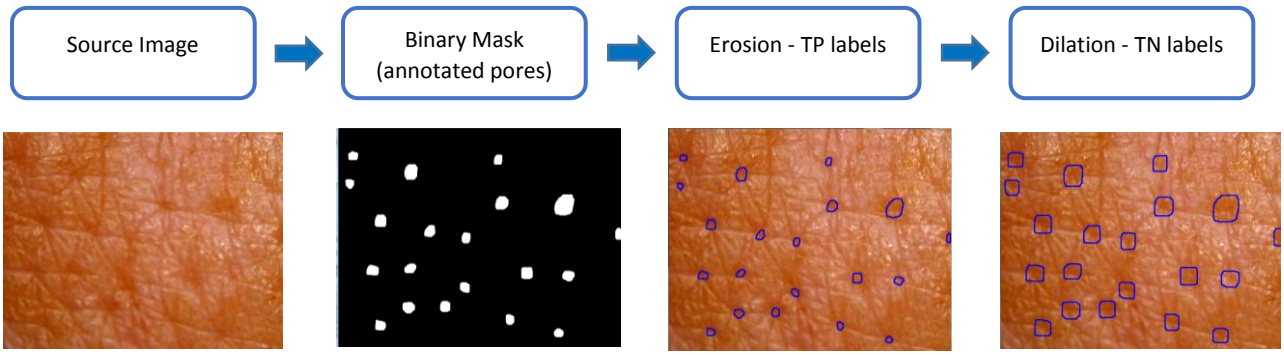

Figure 4.4 *Workflow for obtaining the positive and negative classes from the segmentation map and source image. Starting with an image and its segmentation mask, erosion is applied to obtain true positive labels (TP), followed by dilation to get the true negative labels.*

4.1.2 Contrast adjustment

Digital colored images are stored in the computer memory in the common RGB color model, where each pixels holds intensity values for red, green, and blue channel. In Python, images are stored in BGR (blue, green, red) order in a `numpy.ndarray` format of the 8 bits unsigned integer type. Color values are restricted to range 2^0 to (2^8-1), i.e. 0 to 255.

A grayscale image is a 2D representation of an image that summarizes the three RGB colors into a single value. Most image processing algorithms assume a two dimensional matrix, so the grayscale image is the starting point for further transformations on an image. The OpenCV library uses the following standard formula presented in Equation (4.1) to convert an RGB image to grayscale intensities [24].

$$Grayscale = 0.299 \times R + 0.587 \times G + 0.0114 \times B \qquad (4.1)$$

In the Python environment each pixel of a grayscale image holds a natural integer value between 0 (black) and 255 (white). However, the image does not utilize the entire spectrum so the distribution will mostly concentrate around specific values [25].

It is often necessary to emphasize certain structures of interest from an image to improve the quality of the features that will be used in the machine learning algorithms. Enhancing these features, while minimizing noise is a crucial step for good and robust feature design. A helpful tool to investigate ways of adjusting the contrast of an image is the histogram. An image histogram is a graphical representation of the intensity distribution of a grayscale image where the number of pixels is plotted (on the vertical axis) for each intensity value considered (horizontal axis) [26]. Using the image's histogram, contrast can be adjusted using techniques like histogram equalization or histogram stretching.

Histogram equalization takes a grayscale image and spreads out evenly the most frequent intensity values following a cumulative distribution function (Figure 4.5, right side). Using this non-linear and monotonic

transformation function it maps the given histogram's distribution to a wider and more uniform distribution of intensity values (i.e. an almost flattened histogram shape) [27]. The pixels intensity will keep the same ranking, but the whole image will become far more contrasted and normalized. This methods is effective for detail enhancement and correction of non-linear effects introduced by devices [25], [26], [27], [28].

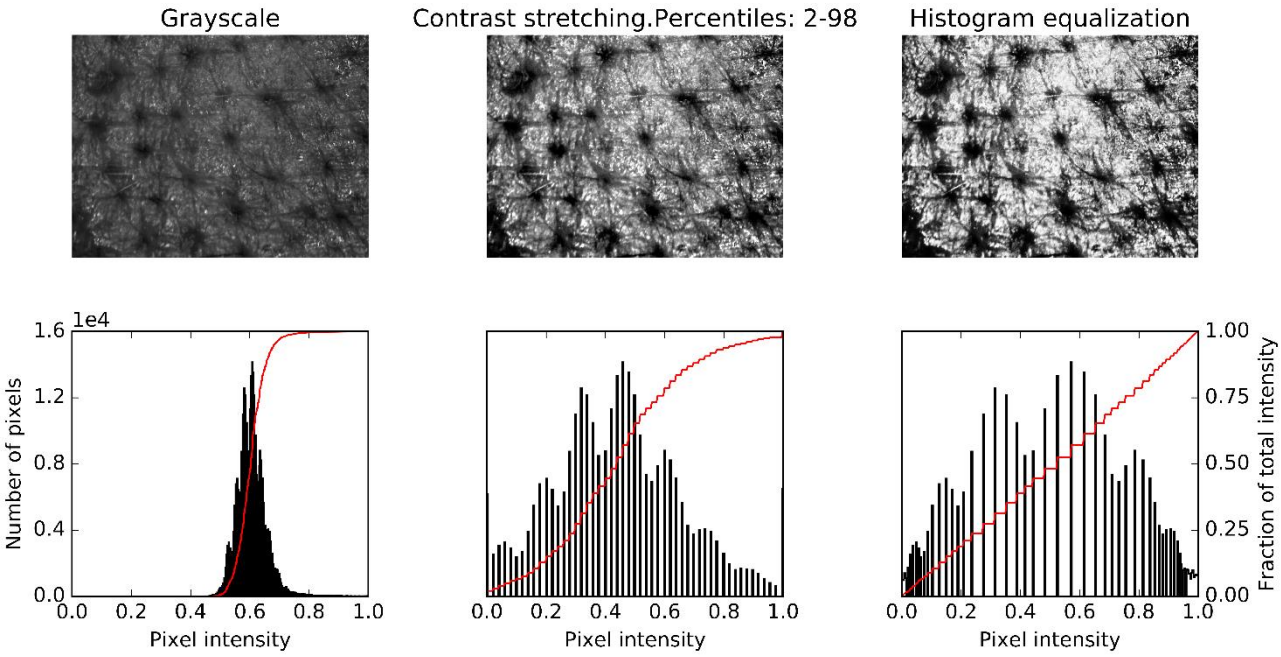

Figure 4.5 *Histogram distribution of a sample image for grayscale, contrast stretching and histogram equalization operations (left to right in this order). Cumulative distribution function plotted in red.*

While histogram equalization requires no parametrization, sometimes it might yield unnatural looking images if the histogram values are not aggregated to a particular region [28]. Contrast stretching attempts to improve the contrast in an image by stretching the range of intensity values that it contains to span a desired range of values. This interval might range between the minimum and maximum intensity from an image, a cut-off fraction below the histogram peak, or a range of percentiles. Since it only applies a linear scaling function to the pixels intensities, the end result is less harsh than in histogram equalization [29]. In the current task, the image was contrast stretched to include all intensities that fall within the 2nd and 98th percentile [28], shown in Figure 4.5, central picture.

4.2 Feature augmentation

Scenes in the images contain objects of many sizes, which in turn contain features of many sizes. These scenes can be taken at various distances from the viewer. Therefore any analysis procedure should be applied at several scales simultaneously in order to capture as much information as possible and account for future variability in image's objects sizes. In pattern recognition cases based on texture analysis, the feature space is augmented by reducing the resolution of the source image. Given different resolutions, the size of an object present in the image will change. On one hand, some features will be better visualized and contrasted at a smaller resolution and generally noise will be attenuated. On the other hand, at high resolutions details are maximized and objects might lose their context and semantic meaning. Finally, increasing the resolution of an image is much more costly operation than reducing the size of the image, as it requires s^4 more arithmetic operations, where s is the scaling factor [30]. These specifications lead to the creation of a data structure designed to support efficient scaled convolution through reduced image representation, also known as Gaussian Image Pyramid [30].

4.2.1 Gaussian Filtering

A Gaussian Filter also known as a Gaussian blur is a type of low-pass filter that reduces image noise and details, resulting into a smoother, blurrier image. This filter applies a transformation based on the normal distribution function to each pixel in the image. The normal distribution function is shown in Figure 4.6.

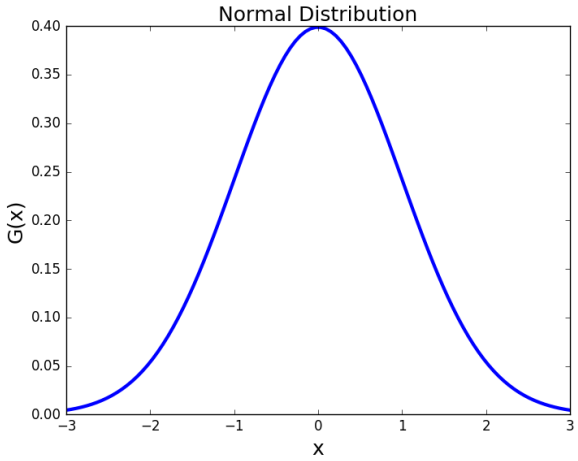

Figure 4.6 *Normal distribution, where G(x) represent the Gaussian Function and x is the value of a pixel.*

The equation of a Gaussian Function in one dimension is presented in Equation (4.2):

$$G(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{x^2}{2\sigma^2}}. \tag{4.2}$$

The equation of a Gaussian Function in two dimensions is the product of two such Gaussians is shown in Equation (4.3).

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}. \tag{4.3}$$

In Equation (4.3), *x* is the distance from the center of the image in the horizontal axis and *y* is the distance from the center in the vertical axis. Sigma, denoted with σ, is the standard deviation of the Gaussian distribution [31]. Values from this distribution will be used to build a matrix of integers, also called a kernel that will be convolved with the source image. Each pixel's new value will be set to a weighted average of the defined neighborhood. The central pixel will have the largest weight while the neighboring pixels weights decrease as their spatial distance to the central pixel increases [32]. This kind of averaging operation will smooth out the image, while preserving edges and boundaries.

Some useful properties of the Gaussian filters are related to the Fourier transform. The Fourier Transform into frequency domain of a Gaussian is another Gaussian. This infers the features computed by multiplying an image with the Fourier transform of a Gaussian are localized both in space and frequency. Convolution with a Gaussian kernel in the spatial domain reduces high frequency image trends smoothly as spatial frequency increases [31, p170]. The spread of the variance, denoted as σ^2, will determine how broad or narrow the neighborhood is, as 95% of the total weight is contained within 2σ of center [31].

Gaussian blurring is a filter that is generally applied when reducing the size of an image through down-sampling in order to avoid aliasing (distortion of the image through high-frequencies) [33].

4.2.2 Gaussian Pyramid

A Gaussian Pyramid consists of a series of copies of a source imagine in which both sample density and resolution are decreased in regular steps [30]. Figure 4.7 shows a visual representation of an image pyramid with 5 levels, where each level was blurred and down sampled.

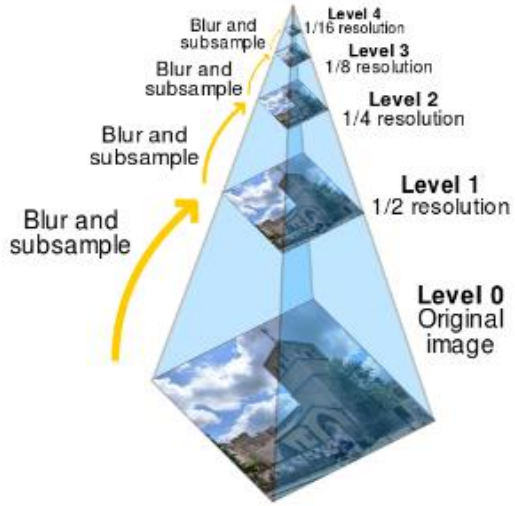

Figure 4.7 *Visual representation of an image pyramid with 5 levels*[11].

These set of images with different resolution are called image pyramids because they are stored in a stack with the largest image at the bottom, or zero level, and the smallest image at the top. This means, the higher the level in the pyramid the smaller the size of the image. Every layer is numbered from bottom to top, with level G_0 equal to the source image and $G_{i+1} < G_i$ level. To obtain the next pyramid level G_{i+1} , the G_i layer is convolved with a Gaussian kernel, then down-sampled by a factor of two by removing every even numbered rows and columns [34]. The kernel used in Python is a 5x5 matrix based on binomial coefficients as depicted in Figure 4.8 [34]. By applying these operations, an *m x n* image becomes an *m/2 x n/2* image. In other words, the image is reduced correspondingly by one octave with each level. Iterating this process on the input image G_0 produces the entire pyramid [34].

$$\frac{1}{16}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

Figure 4.8 *A 5x5 convolution matrix used for the Gaussian Filter. Source: [20]*

The pyramid representation has many advantages, from data compression to image blending or enhancement, the latter representing the property that will be used for feature augmentation and extraction part in this work. Figure 4.9 displays a sample pore image on which a four-level pyramid operation was applied. It shows that most of the smaller values (i.e. skin pattern lines) disappear, while larger values that include prominent image features (i.e. pores) are peaked in the higher levels (Figure 4.9, G_1-G_4). Pyramid representation allows tasks such as texture analysis to be computed fast and simultaneously at multiple scales. Moreover the multiresolution format matches the multiple scales found in natural visual scenes and mirrors the multiple scales of processing in the human visual perception [30].

[11] Source: By Cmglee - Own work, CC BY-SA 3.0. Website: https://commons.wikimedia.org/w/index.php?curid=42549151

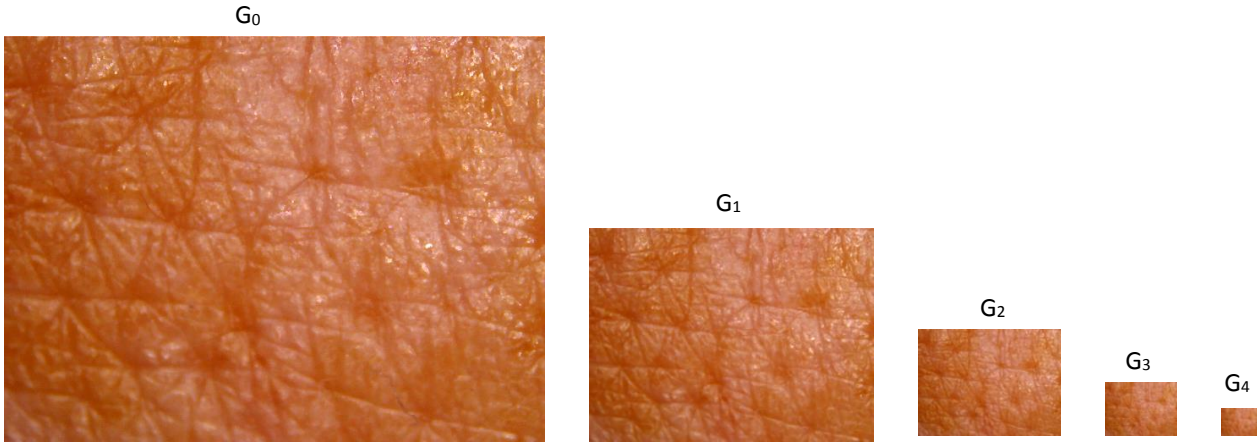

Figure 4.9 *The Gaussian Pyramid. A source image denoted G_0 (first image starting from left) is repeatedly low-pass filtered and subsampled. This generates the sequence of subsequent smaller images G_1-G_4.*

4.3 Feature Extraction

Whether image pre-processing was applied or not has a modest effect on the final prediction performance of the classification algorithm – what makes a significant difference are the selected features from an image. Literature mentions that different pixel representations, e.g. grayscale, give similar results, with only a small decrease in performance [35]. The features from an image, also known as feature descriptors are interesting parts of an image that refer to a distinct structure such as an edge, point or small image patch [36]. These regions of interest should be detected or transformed in such a way to differ from their immediate neighborhood by means of texture, color, intensity or shape. Suitable features that describe these properties rely on image gradients and intensity variations. These descriptors include edges, corners, blobs or simple raw pixel intensities. Below are a few properties that characterize a suitable feature [36]:

- **Repeatability** – given two images, the feature will be present in both of them, indifferent of the changes in viewing conditions and noise
- **Distinctive** – the neighborhood around the feature center should present an abrupt change to allow for a comparison between classes
- **Localizable** – the feature should have a unique location assigned to it

Other feature detection methods select points of interest that are rotation or scale invariant, i.e. they can still be detected even if the image is resized or rotated. Feature extraction involves detection and computing of a descriptor, which is a vector representation of a local pixel neighborhood centered on a detected key point. This new representation from a matrix of pixels to a vector enables the arrangement of the feature data into tabular form Figure 4.10. Tabular data is the most common way of representing data in machine learning problems. The pixel vectors are stacked into rows of a table, representing the data samples or instances. The labels are stored in a single column, also called a target vector, with binary values (1 and 0) defining the positive and negative class. This table serves as input feature data which will be used as learning material for the classification models described in the following chapter.

Figure 4.10 shows an example of feature extraction around a key point (marked with orange in the center of the matrix). The coordinates of the points of interest, i.e. pixels, are retrieved from the segmentation maps obtained through annotation (Chapter 3). From the binary dilated and eroded images (Chapter 4.1.1) a set of key points [x=column, y=row] is extracted with a contour retrieval algorithm [23]. For each key point or coordinates $[x_D, y_D]$ in the binary dilated segmentation mask a feature vector will be extracted and the class label 0 is assigned. The same procedure is repeated for the eroded image with coordinates $[x_E, y_E]$, instead the class label 1 is assigned (Figure 4.10).

The columns of the feature vector constitute the predictor variables. Since the algorithm implemented is modular, the number of columns is variable depending on the size of the neighborhood: if the number of neighbors is 3, then a simple 3x3 square neighborhood around [x, y] point locations is extracted and assembled into a vector. For example, if the pixels intensities from the three RGB channels are used as feature values, the feature vector will have the size 1-by-27 (row-by-columns). The feature vector for this particular example is shown in Figure 4.10. In Python the indexing starts from zero so the vector is represented accordingly.

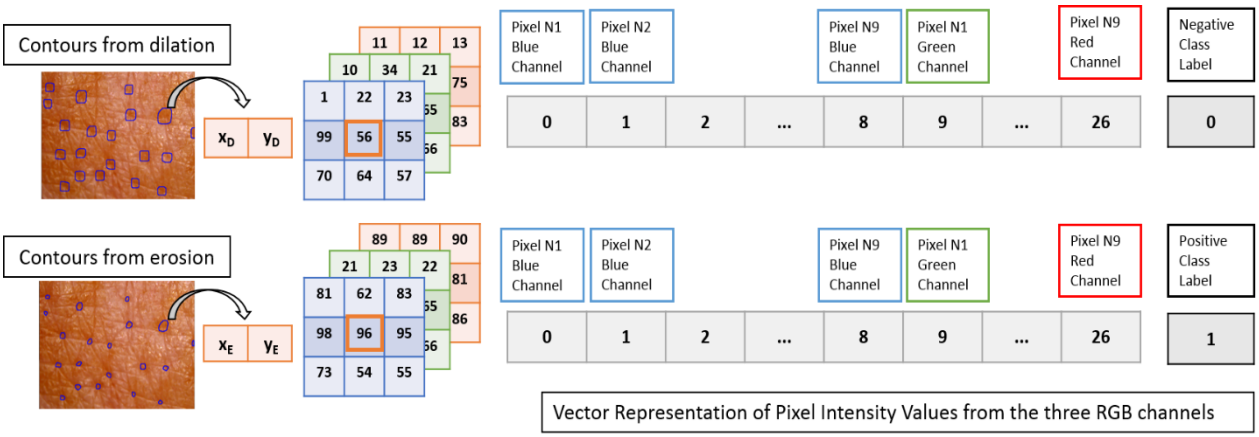

Figure 4.10 *From contours coordinates to feature vector. Binary dilated and eroded images provide a set of coordinates around which a square RGB neighborhood is extracted. The neighborhood is flattened into vector format.*

The feature values could be raw pixel intensity values from the source image, from the grayscale transformation or from the image layers of the Gaussian Pyramid (Chapter 4.2.2). All of these features increase the size of the predictor variables when added to the input feature data table as columns. Identifying the relevant features that offer the best prediction performance is a trial and error process that involves training and validation of the classification models applied (Chapter 5).

4.4 Conclusion

This chapter discusses the workflow of building a suitable feature dataset from image acquisition to feature extraction. Images are pre-processed with morphological operations to extract key points or coordinates from the region of interest as provided by annotation layers. Contrast adjustment can be applied to enhance objects present in the image. To provide context to the objects of interest and augment their characteristics, feature augmentation methods like Gaussian Pyramid are proposed. Finally feature descriptors are formed from block neighborhoods extracted around a key point. These feature descriptors are vectors, which by convention, are stacked into tabular format. Together with the labels vector column, they form the input feature matrix which is fed to machine learning classifiers. Based on these values, classifiers fit a model that can predict the label of future observations.

Chapter 5

Classification Methods for Semantic Description

This chapter discusses several supervised machine learning algorithms, their mathematical principles and algorithm implementation. Generally, the type of labels and the data distribution dictate the choice of the algorithm. Although the image annotation tool (Chapter 3) facilitates the annotation of several different skin structures, this work focuses on the classification of only one skin structure: pores. In this case, the problem becomes a binary classification task with two classes (pore vs. not pore), with one sample belonging to one class only. Since our data is highly-dimensional, i.e. has a large number of both predictor columns and observations, dimensionality reductions methods are explored in the first instance (Chapter 5.2). The machine learning algorithms are approached incrementally in this work, from parametric models like Logistic Regression to non-parametric ones like Random Forests. Figure 5.1 presents the machine learning algorithms discussed in this Chapter.

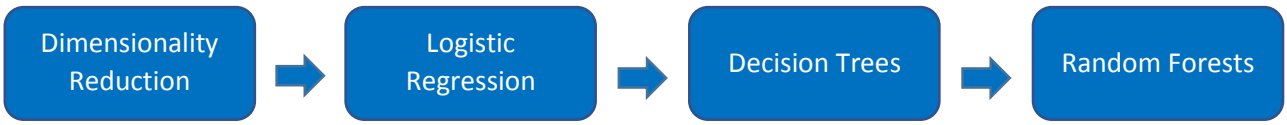

Figure 5.1 *Machine learning methods explored in this work.*

5.1 Classification Procedure

The field of machine learning is inspired from human reasoning for constructing an artificially intelligent algorithm that could learn from data the way humans do. Based on the discovered properties learned from the available data, the machine learning algorithm tries to use this knowledge to predict or classify data automatically. The learning problems are grouped in two main categories known as supervised and unsupervised learning. In supervised learning, the algorithm learns about the data from a training set with known response values and based on this information constructs a model that makes predictions about unseen data (Figure 5.2). The training dataset consists of features of a an object having a known target value number in the case of regression problems or target class label for classification tasks.

The application of an algorithm and its resulting model represents the training part of the machine learning process. The learning procedure is inductive which means that if a model estimates well the labels for a satisfactory amount of data, then it will also generalize well over future unobserved data. The current machine learning task of identifying skin structures falls under the supervised learning category since labeled images are available from annotation. In the previous sections, the labeled regions of interest were segmented from a set of images and features were extracted. The feature matrix and the labels column are the input of machine learning algorithms.

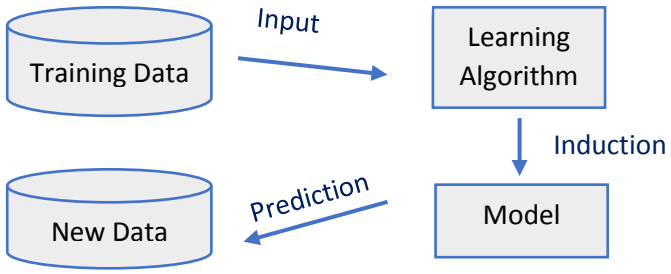

Figure 5.2 *Machine Learning Principle.*

Our data is composed of hundreds of pixels intensities arranged in a tabular form. Each row of this table is called an instance or an observation and will be denoted in this work with $x^{(i)}$. Each column represent a feature and it is also named predictor variable. In our classification task, the predictors represent a square matrix of pixel neighbors around a key point. The pixel intensities become the input variables. The prediction output is also known as a target variable and is denoted throughout this work with $y^{(i)}$. Each pair $(x^{(i)}, y^{(i)})$ represents a training example from the training set. The superscript in the notation is an index into the training set. Table 5 presents an example of a training set that follows the afore-mentioned notations.

Table 5 *Example of training set used in the present classification task*

Predictor Variable 1 (x)	Target variable (y)
Pixels Intensity	**Pore**
55	1
59	1
68	0

In a feature approach analysis, generally a machine learning classification algorithm is applied. Its result is a parametric mathematical model obtained through a fitting procedure on the training set. This model is next used to predict future instances whose classification labels are unknown. However, whether a test dataset is available or not, it is advised to perform validation to estimate to prospective prediction performance of the model.

Learning the parameters of a predictive model and testing it on the training data is a methodological mistake: a model that would just repeat the labels of the samples that it has just seen would have a perfect score but would fail to predict anything useful on yet-unseen data. This is procedure falls under a phenomena described as overfitting. To avoid it, it is common practice when performing a supervised machine learning task to hold out part of the available data as a validation set [36]. Another example of overfitting is when the model is describing noise in the data instead of the actual relations between features.

To evaluate the effectiveness of a certain model in classifying future data, classification is carried out by first decomposing the available dataset into training and validation set as depicted in Figure 5.3 [38]. This validation method is generally used if there is enough data to allow for a partition that does not built a weak model on too few data samples. The common rule is to partition the data into training and validation sets based on the following proportions: 70% and 30% or 80% and 20%, respectively. This method enables the construction of a confusion matrix where the misclassification numbers can be easily visualized. More details about this metrics are given in Chapter 6.

The splitting of data into training and validation sets should be done according to the labels distribution, indifferent of the validation method. For any type of classification problem, stratified splitting is recommended [38]. This type of splitting preserves the initial distribution of the classes in both the training and validation set [39]. Although an imbalance in the class labels might lead to bias, we preserve the natural distribution of the dataset as it reflects real-world occurrence.

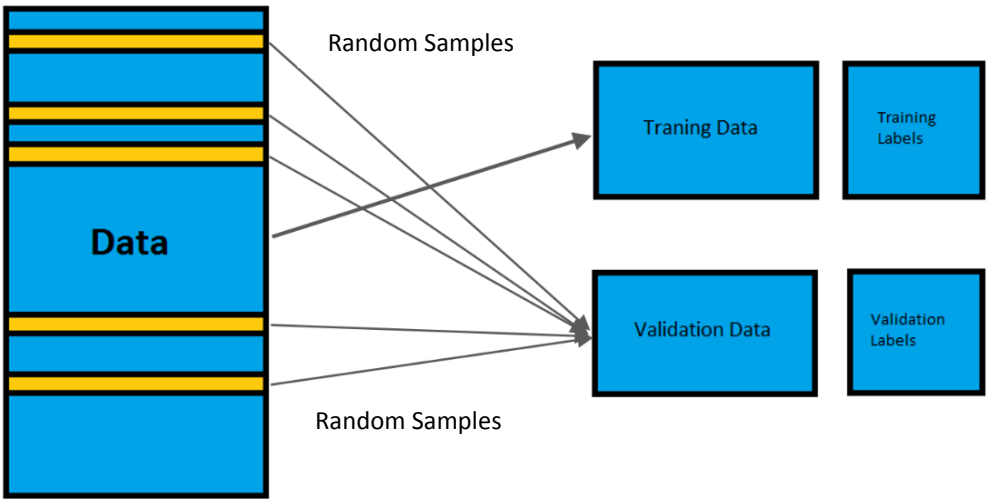

Figure 5.3 *Evaluating the predictive power of a machine learning mode through validation. The data is split into training and validation sets.*

A better way to get a sense of the predictive accuracy of the classifier for new data is cross-validation [40]. Cross-validation removes the need for a large enough amount of data where a proportion of the training set is set aside for later validation. This method directly estimates the expected sample error, which is the average generalization error when the prediction model is applied to an independent test sample [41].

Every "K-fold" method uses models trained on in-fold observations to predict labels for out-of-fold observations [26]. The dataset is split into K roughly equal sized-parts, also known as subsets or folds. For example, by taking K = 5, every training fold contains roughly K-1 parts of the data, which makes 4 folds for the given example. Every test fold contains 1/K, which is 1/5 parts of the data for this example. Thus the first model is trained on the data with the first 1/5 excluded; the second model is trained on the data with the second 1/5 excluded, and so on. The labels for each observation contained in the test fold are computed by using the model trained without this fold [26]. Thus, N different models are produced, each one having the ability to be tested against an independent subset of the data [27]. The average performance of these N models is an accurate estimate of the performance of the source model (created by using the entire dataset), on a prospective independent test set [27].

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Round 1	Validation	Train	Train	Train	Train
Round 2	Train	Validation	Train	Train	Train
Round 3	Train	Train	Validation	Train	Train
Round 4	Train	Train	Train	Validation	Train
Round 5	Train	Train	Train	Train	Validation

Figure 5.4 *Example cross-validation procedure with 5 folds.*

5.2 Dimensionality Reduction

Data coming from images is generally highly-dimensional in terms of features predictors and instances which makes the classification process slow and challenging for computer hardware. Principal component analysis or PCA is a method of summarizing data into new variables which are linear combinations of the source variables. It accomplishes this goal by converting strongly correlated variables into a set of new linearly uncorrelated variables [42], [43], [44].

PCA is an unsupervised machine learning algorithm as it attempts to describe relations in the input data without having an outcome measure. It reduces the dimensionality of the data by finding directions, called principal components along which the variation of the data is maximal. By using fewer components, each data sample can be characterized by relatively few numbers instead of by values for thousands of variables. The principal components are in fact eigenvectors and they describe the variability of the data through variance. The mathematical formula of variance is explained in Equation *(5.1)*.

$$Var(X) = \sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2 \qquad (5.1)$$

The eigenvectors determine the directions of the new feature space, and the eigenvalues determine their magnitude. Thus, the eigenvalues explain the variance of the data along the new feature axes. Mathematically speaking, variance is the average squared distance from the mean and it measures the spread of the data. In Euclidean space, the first principal component corresponds to a line that has the largest possible variance with minimum squared sum of errors of the distances of the points from the line. The second principal components is orthogonal to the first component along which the samples show the second largest variation. Each subsequent component has the highest variance possible while being orthogonal to the preceding components [42], [43], [44].

In this work the scikit-learn library from Python was used. The algorithm implements PCA using Singular Value Decomposition (SVD) by the method of Halko et al [45]. It uses this method instead of the more popular Eigen values and vectors decomposition because it has improved computational efficiency [42]. The idea is that the principal components transformation can also be associated with another matrix factorization that generalizes the Eigen decomposition. Given an *m x n* matrix $\mathbf{A}$ of rank *k*, it admits a factorization or decomposition into the following product of matrices shown in Equation (5.2):

$$\mathbf{A} = \mathbf{U}\sum\mathbf{V}^* \qquad (5.2)$$

where,

- $\mathbf{U}$ is an *m x k* unitary, orthogonal matrix
- $\sum$ is a *k x k* diagonal matrix with non-negative real numbers on the diagonal
- $\mathbf{V}$ is an *n x k* unitary, orthogonal matrix
- $\mathbf{V}^*$ is an *n x k* conjugate transpose of the $\mathbf{V}$ unitary matrix.

The diagonal entries σ_j of $\sum$ are known as the singular values of. They are arranged in a decreasing order $\sigma_1 \geq \sigma_2 \geq \ldots \sigma_k \geq 0$. The singular values are the square roots of the eigenvalues of the matrix $\mathbf{A}^T\mathbf{A}$.The columns of $\mathbf{U}$ and $\mathbf{V}$ are called left singular vectors and right singular vectors, respectively [45]. The eigenvectors are the rows of matrix $\mathbf{U}$.

From a visual point of view, PCA rotates the set of points around their mean to align them with the directions given by the principal components. Figure 5.5 shows an example of principal component analysis on artificially created data with 20 dimensions and binary labels. Using an orthogonal transformation it moves as much of the variance as possible into the first few dimensions. The values of the remaining dimensions tend to be small and may be dropped without losing much information.

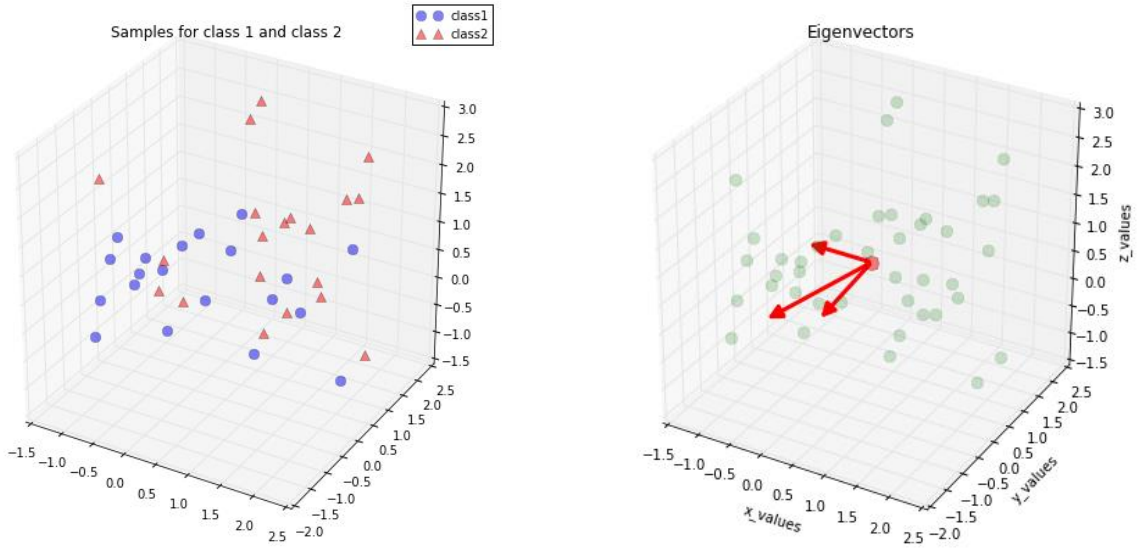

Figure 5.5 *From data samples to principal components. Data samples belonging to an artificial random dataset with binary target variables. Eigenvectors centered at the mean plotted as directed arrows on the right figure. Source: [46]*

Data normalization can be achieved with the transformation of the date onto unit scale ($\mu = 0, \sigma^2 = 1$). The data is centered by subtracting the mean and divided by the standard deviation. The rule is that if PCA is desired to be performed on the correlation matrix instead of a covariance matrix then columns of $\mathbf{A}$ should not only be centered, but standardized as well. Usually, the covariance matrix tends to be used when the variable scales are similar and the correlation matrix when the variables have a different measurement unit [47].

Whether to standardize the data prior to PCA depends on the measurement scales of the source features. If the features have different measurement units, standardization before PCA is usually recommended.. By standardizing the variables we assign them equal importance. Mean centering ensures that the first principal component describes the direction of the maximum variance. Without mean subtraction the variables with the highest variance might dominate the first principal component, while the others are ignored. Basically, the first principal component might correspond more or less to the mean of the data. Having a mean of zero has also a basis in minimizing the mean squared error [47].

All our features values are the same scale since they are pixel values in the range 0 to 255. Thus, in this case, data standardization is not necessary. To choose the optimal number of principal components that account for us much variability as possible, while retaining a small feature subspace the explained variance is used. This measure is calculated based on the eigenvalues and describes how much variance can be attributed to each of the principal components.

After the principal components have been computed using SVD, the variables are projected onto the new smaller dimensional sub-space by multiplying the eigenvectors with the transposed data matrix. The Python algorithm for computing PCA has in-built data-centering.

5.3 Logistic Regression

Logistic Regression is a classification algorithm that deals with binary categorical variables, represented by the positive class 1 or the negative class 0. It stems from linear regression and it has a probabilistic interpretation based on the logistic function.

Logistic Regression is a part of the generalized linear models intended for regression in which the target value is expected to be a linear combination of the input variables. To perform supervised learning, a hypothesis that represent the classification problem is defined in mathematical terms. Thus, if y is the target variable, $\hat{y}$ the predicted or expected value and h the hypothesis then [48]:

$$h_w(x) = \hat{y}(w, x) = w_0 + w_1 x_1 + \ldots + w_p x_p \,. \tag{5.3}$$

The weight vector $\mathbf{W} = (w_1, \ldots w_p)$ in Equation (5.3) represent the model's parameters , w_0 is the intercept of the model and p the number of input variables [48]. Leaving aside the intercept term to simplify the equation we obtain the new formulation in Equation (5.4) [49].

$$h_w(x) = \sum_{i=0}^{p} \mathbf{W}_i x_i = \mathbf{W}^T x \tag{5.4}$$

The Logistic Regression function is applicable for classification problems as it can take any real value as input and output value between $y \in \{0,1\}$. The advantage of this range is that it is interpretable as a probability, hence being useful for binary classification tasks with discrete values. In this model, the probabilities describing the possible target outcomes are modeled using a logistic function. For the classification hypothesis representation we change the form of our function to Equation (5.5).

$$h_w(x) = g(\mathbf{W}^T x) = \frac{1}{1 + e^{-w^T x}} \tag{5.5}$$

$$g(z) = \frac{1}{1 + e^{-z}} \tag{5.6}$$

Equation (5.6) represents the logistic or sigmoid function whose graphical representation is depicted in Figure 5.6. When the probability of y is greater than 0.5 we can predict $y = 1$, otherwise $y = 0$ [49].

To bring it all together in mathematical formulation we have the following probabilities for the positive class as described in Equation (5.7) and Equation (5.8) for the negative class.

$$P(y = 1 \mid x) = h_w(x) = \frac{1}{1 + e^{-w^T x}} = g(\mathbf{W}^T x) \tag{5.7}$$

$$P(y = 0 \mid x) = 1 - P(y = 1 \mid x) = 1 - h_w(x) \tag{5.8}$$

In order to determine the parameters of the weight vector that predict accurately the training binary labels, a cost function is implemented. Logistic Regression implements gradient descent as an optimization algorithm to minimize the cost function also known as a least-squares cost function [49].

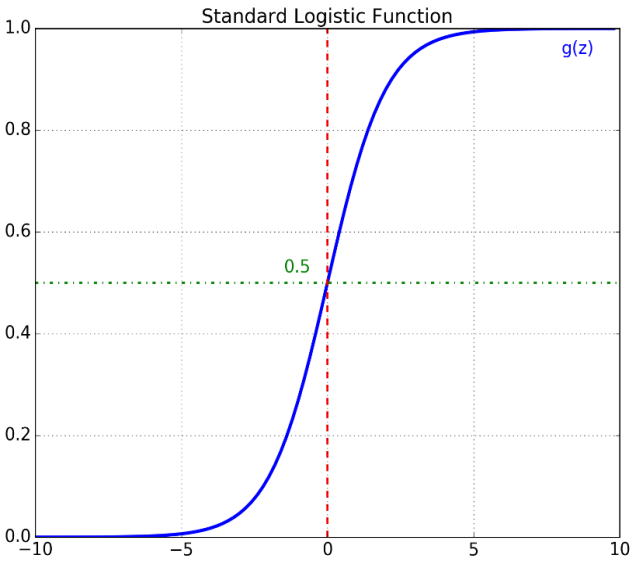

Figure 5.6 *Logistic Function g(z). Crosses the origin at 0.5 (green line) and is bounded between 0 and 1.*

5.4 Decision Trees

The main advantage of Decision Trees is that they have a non-parametric nature which implies that no assumptions are made regarding the distribution of the data [50]. Additionally, they are fast to be implemented and easy to understand as they output decision rules that can be translated to any research setting in semantic terms. The classification and regression trees (CART) algorithm as described in Breiman et al. (1984) is applied in this work using the scikit-learn library. Decision Trees constructs binary trees using the feature and threshold that yield the largest information gain at each node [51].

In order to predict the response for an instance or observation, the decision tree is followed top-down from the root node (the beginning node) through every branch which fulfills the criteria of the observation until a leaf node (terminal node) is reached. The leaf node on which the observation lands, contains the response label. Classification trees divide the data based on the input predictor values. Each node from tree, aside from the leaf nodes, is split into exactly two child nodes. This binary split is made based on the feature values. Each path from the root of the tree to one of the leaf nodes represents a set of rules of the form "if x < value then choose node i; else if x > value then choose node i+1", where x represents and observation. An example for a classification tree is depicted in Figure 5.7.

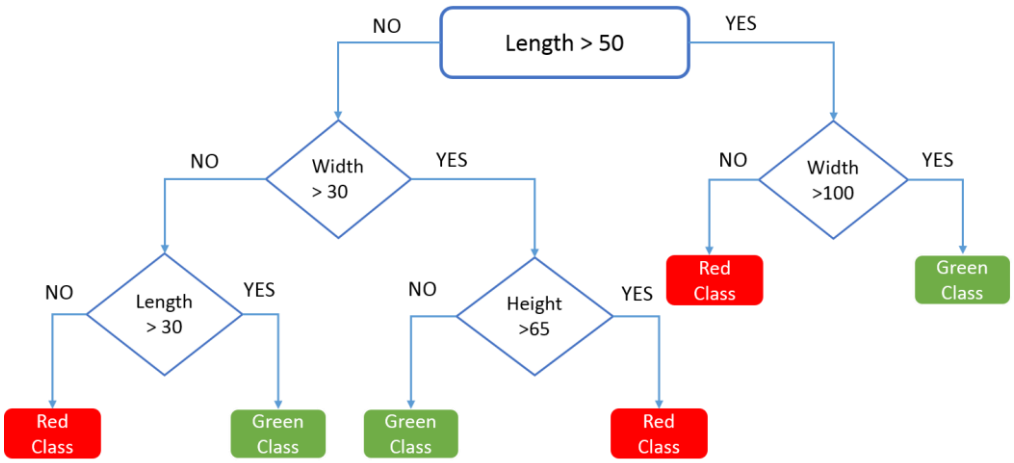

Figure 5.7 *Decision Tree example with binary classes Red and Green. The root node is represented by variable "Length>50", while the leaf nodes are the red and green colored rectangles.*

Tree-based method partition the feature space into a set of rectangles and then fit a simple model, like a constant in each one. The main idea behind a binary tree is to form the tree and then minimize the error in each leaf of the tree. Figure 5.8 depicts a partition of the feature space for a classification tree with two nominal responses, "0" and "1". The only difference from the algorithm of a regression tree is that instead of the mean of the points from a region, the majority class label from the respective region gives the output response. For example, a data point that will fall into the first region R_1, for which the split was made at $X_1 < t_1$, will have class label "1".

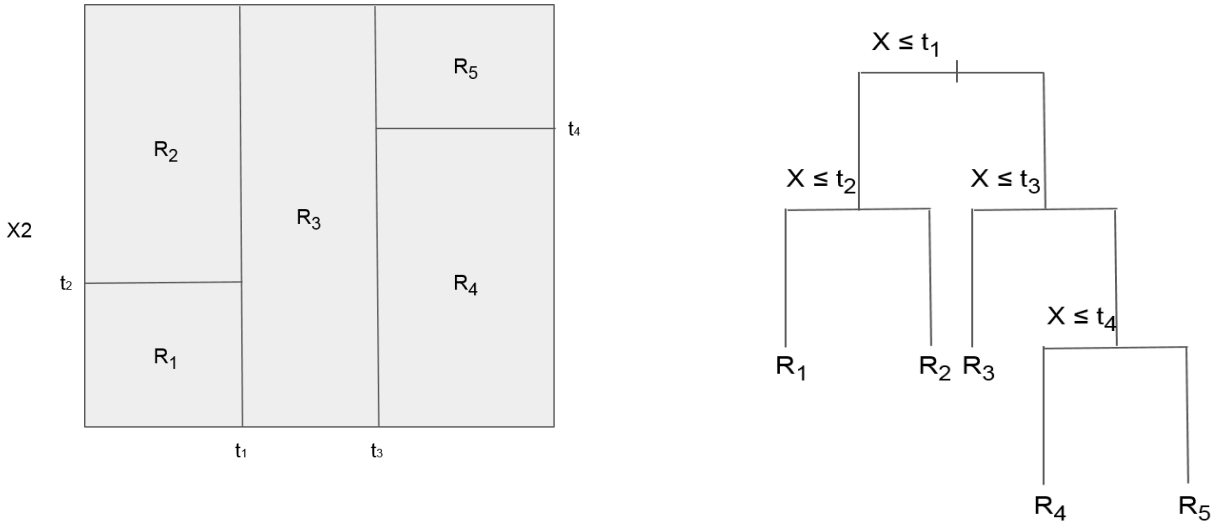

Figure 5.8 *CART decision tree example (adapted from [41]). The left figure shows a partition of two-dimensional feature space by recursive binary splitting. Right figure shows the tree corresponding to the partition on the left.*

In order to grow the entire decision tree, a greedy heuristic "divide and conquer" algorithm is applied. For this particular case, the heuristic is to find the best split by minimizing the chosen measure for node impurity. At start, all the training examples are at the root of the tree and then it starts to grow top-down in a recursive manner by replacing all the leaf nodes with simple nodes. All the nodes are created through a series of optimally local selections of selected predictors on which the partitioning or splitting of the dataset is made. The aim is to partition the dataset in subsets more and more homogenous so that the impurity is minimal [52].

If the target is a classification outcome taking on values 0, 1, … k -1 for node m , representing a region R_m with N_m observations we define the proportion of class k observations in node m [41]:

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in \Re_m} I(y_i = k) \qquad (5.9)$$

The observations in node m are classified into class $k(m) = \arg\max_k \hat{p}_{mk}$, the majority class in node m . Different measures of node impurity $Q_m(T)$ include the following metrics detailed in Table 6.

Table 6 *Different measures of node impurity used for Decision Trees*

Name	Definition	Formula
Gini's Diversity Index	A node with just one class (that is a pure node) has the Gini index equal with 0; otherwise the index is positive. A node with maximum impurity has the Gini index equal with 0.5 which means that the classes are equally distributed within the node [30].	$Q_m(T) = \sum_k \hat{p}_{mk}(1-\hat{p}_{mk})$
Cross-Entropy	Cross entropy measures how much information is required, on average, to identify random samples from the distribution. It is used as an alternative to squared error [53], [54].	$Q_m(T) = -\sum_k \hat{p}_{mk}\log(\hat{p}_{mk})$
Misclassification Error	The default classification error is the fraction of the training data that the tree misclassifies [33]. The minimum cost function predicts the label with the smallest expected misclassification cost, with the cost given by the square cost matrix property of the classifier [33]. The value *Cost(i,j)* is the cost of classifying a point into class *j* if the true class is *i* [55].	$Q_m(T) = 1 - \max \hat{p}_{mk}$

Advantages of Decision Trees include [51]:

- Trees can be visualized through plotting which makes them easy to understand and interpret (as opposed to Random Forests which are a black-box)
- input data does not need pre-processing as Decision Trees are a non-parametric method
- Ability to handle both numerical ad categorical data
- Use a white-box model, meaning the explanation for assigning a certain label to an instance is easily explained by Boolean logic. By contrast, in a black-box model (e.g. Random Forests) the basis on which classification has been achieved is not easily understood
- Permits model validation based on probability estimates obtained from the nodes of the tree

Disadvantages of Decision Trees include [51]:

- Can build complex trees that do not generalize well over future data. This results in overfitting and requires the application of methods such as pruning , setting the minimum number of samples for a leaf node or setting the maximum depth of the tree
- Can become unstable because small variations in the data might result in a completely different tree being generated. This problem is approached by using Decision Trees within an ensemble
- Decision-tree learning algorithms are based on heuristic algorithms (the greedy algorithm) where locally optimal decisions are made at each node. Such algorithms cannot guarantee to return the globally optimal decision tree. This can be addressed by training several trees in an ensemble learner, where the features and samples are randomly sampled with replacement.
- Decision tree models create biased trees if one of the classes has a dominant distribution. It is therefore suggested to balance the dataset prior to fitting with the decision tree.

The algorithm procedure for constructing CART Decision Trees is explained below [52].

Algorithm for CART Decision Trees

1. Start with all input data and examine all possible binary splits on every predictor
2. Select a split by applying the best optimization criterion
   - *Regression trees*: Mean squared error
   - *Classification trees*: Gini's diversity index
3. Impose the split
4. Repeat recursively for the two child nodes
5. Stop growing the tree if any of the following criteria are satisfied:
   - The node is pure. For classification, a node is pure if it contains only observation from or class.
   - The size of this node is less than the allowed minimal size of a parent node
   - The maximal allowed number of leaf nodes has been reached
   - The maximal allowed tree depth has been reached

5.5 Random Forests

The random forest algorithm which relies on CART Decision Trees was implemented and defined by Leo Breiman and Adele Cutler in 2001 [56], [57]. It belongs to a larger class of machine learning algorithms called ensemble methods, as it aggregates an ensemble of Decision Trees. These methods involve the combination of several machine learning models to solve a single prediction problem [58].

The aim of ensemble methods is to combine predictions of several estimators or models built with the same learning algorithms in order to improve generalizability. Random forest belong to the class of ensemble methods based on averaging methods. The principle of this method is to build several random estimators independently and then average their prediction. The idea is that on average the combined estimator is usually better than any of the single estimators alone [59].

In Random Forests each decision tree in the ensemble is built from a sample drawn with replacement (i.e. a bootstrap sample) from the training set. This method is also called bagging. This method form a class of algorithms which build several black-box estimators on random subsets of the training dataset. This means it creates Decision Trees on both subsets of random predictors and random observations. The individual predictions of these random trees are finally aggregated to a final prediction. This procedure reduces the variance present in a decision tree by introducing randomization and then creating an ensemble out of it. Generally Decision Trees have a low bias and high variance which leads to overfitting, but their aggregation in Random Forests tackles this issue. Moreover, when splitting a node during the construction of the tree, the split that is chosen is no longer the best split among all features. The chosen split is the best split among a random subset of features. As a result of this randomness, the bias of the forest usually slightly increases (with respect to the bias of a single non-random tree) but, due to

averaging, its variance also decreases. This compensates for the increase in bias, thus yielding an overall better model [59].

The main parameters that needed to be adjusted when using Random Forest is the number of trees in the forest and the maximum number of features to consider when looking for the best split. Considering the number of trees, the larger the better the performance, although there is a critical threshold by which the algorithm will not improve. The number of maximum features is the size of the random subsets of features to consider when splitting a node. If this number is low then the variance will also be kept low. However a small number of features increases the bias. Empirical good values for classification tasks is to choose the maximum number of features equal to the square root of the number of features in the data. Good results are also achieved when fully developing the trees by choosing the minimum number of samples required to split an internal node to be at least one and not limiting the depth of the trees [59], [60]. In contrast to the original publication of Breiman [56], the scikit-learn implementation combines classifiers by averaging their probabilistic prediction, instead of letting each classifier vote for a single class [59].

In conclusion, Random Forests is a collection of many CART Decision Trees that are not influenced by each other when constructed. The average of the predictions made from Decision Trees determines the overall prediction of the forest. The advantage of Random Forests over Decision Trees is that it reduces overfitting by minimizing the variance through aggregation. Random Forests is a computationally fast method that works well for the analysis of complex data structures embedded in small to moderate data sets containing less than 10,000 rows but potentially millions of columns [58].

The Random Forests algorithm procedure adopted from [33] is presented below.

Algorithm for Random Forests

1. For b=1 to B:
    - Draw a bootstrap sample Z^* of size N from the training data
    - Grow a random forest tree T_b to the bootstrapped data, by recursively repeating th following steps for each terminal node of the tree, until the minimum node size $n_{\min}$ reached
        i. select m variables at random from p variables
        ii. Pick the best variable/split-point among the p
        iii. Split the node into two daughter nodes
2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a classification prediction at a new point x:

Let $\hat{C}_b(x)$ be the class prediction of the *b*-th random forest tree. Then:

$$\hat{C}_{rf}(x) = majority - vote \ \{\hat{C}_b(x)\}_1^B$$

5.6 Conclusion

This chapter discusses what constitutes input and target variables for machine learning classifiers. If the number of input predictors is large, dimensionality reduction methods could be applied as a way to tackle overfitting (Chapter 5.2). We outline the theory of several classifiers that are going to be fitted on our input data. These classifiers are Logistic Regression, Decision Trees and Random Forests. In the subsequent chapter we build machine learning models through model fitting and we evaluate their effectiveness in making predictions. This is known as the validation step and it provides us an estimate of the predictive power of the models, before they are applied on new test images.

Chapter 6

Validation Methods

The next step in the classification procedure after model fitting is validation. This operation is performed to estimate the predictive power of the model in classifying future unseen data instances. The methods used in this work include the confusion matrix, the receiver operating characteristics, precision-recall curve and their related metrics.

6.1 Confusion Matrix

The goal of a classification algorithm is to attempt to learn a separator (classifier) that can distinguish the data points from different classes. A confusion matrix, also known as an error matrix, is a specific two-by-two table that allows analyzing the performance of an algorithm by displaying the number of false positives, false negatives, true positives, and true negatives. This allows more insight than reporting proportion of correct guesses, i.e. accuracy [61]. Given a train and validation set with known labels, after classifier training, the model will output predictions on the validation set that can be measured against their true labels. Considering a binary classification problem, with outcomes positive (P) and negative (N) of an instance, there are four possible outcome values in this case:

1. True Positive (TP): positive example is correctly classified as positive
2. True Negative (TN): negative example correctly classified as negative
3. False Positive (FP): example misclassified as positive, when the true class is negative
4. False negative (FN): example misclassified as negative, when the true class is positive

These values represent are commonly arranged in a two-by-two matrix, named confusion matrix or contingency table, as exemplified in Figure 6.1. Each instance contributes to the count in exactly one of the cells [61].

		Classifier Outcome (predicted)	
		Negative (-)	**Positive (+)**
True Class	**Negative (-)**	True Negative	False Positive
	Positive (+)	False Negative	True Positive

Figure 6.1 *Confusion Matrix.*

Thus in mathematical formulation a confusion matrix $\mathbf{C}$ is such that $\mathbf{C}_{i,j}$ is equal to the number of observations known to be in group i but predicted to be in group j. In binary classification tasks, by convention 0 is deemed as the negative class, and 1 as the positive class. Thus, the count of true negatives is $\mathbf{C}_{0,0}$, false negatives is $\mathbf{C}_{1,0}$, true positives is $\mathbf{C}_{1,1}$ and false positives is $\mathbf{C}_{0,1}$ [62].

The common performance metrics that can be calculated from the confusion matrix are formulated below. Sensitivity, also known as recall or True Positive Rate (TPR) is shown in Equation (6.1). Specificity also denoted True Negative Rate (TNR) is shown in Equation (6.2). False Positive Rate (FPR) is equal with 1-specificity.

$$sensitivity = \frac{TP}{TP + FN} \quad (6.1)$$

$$specificity = \frac{TN}{TN + FP} \quad (6.2)$$

6.2 Receiver Operating Characteristic

Receiver Operating Characteristics (ROC) graphs are a useful technique frequently employed in decision making for visualizing, organizing and selecting classifiers based on their performance [61]. This type of graph shows the performance of a binary classifier for different thresholds of the classifier output. It is created by plotting the true positive rate (TPR) versus the false positive rate (FPR) at different threshold values. A ROC curve is used to find the threshold that maximizes the classifier prediction rate or to assess how the classifier performs in regions of high sensitivity and high specificity [63].

A ROC graph could be drawn for any classifier that returns a numeric score for an instance of input data. The scikit-learn Python package, implements two methods `predict()` and `predict_proba()` to get a predicted class or predicted score, respectively on each instance. While we ultimately want to predict the class of the instance, the score of a classifier is more useful since we can choose a decision threshold other than the default 0.5. The cut-off value of 0.5 might not be the best threshold to separate the positive and negative classes, especially when data imbalance is present [40]. In mathematical terms the score of a classifier is the posterior probability estimate *p(C|X)* where *C* is the class and X is an input vector [64].

Nearly every classifier including Logistic Regression, Decision Trees and Random Forests can return both a probability estimate (score) and a hard classification label.

Logistic Regression places a line between different classes of points. Since the boundary line at 0 separates the two classes (Figure 5.6) the labels of the points near the boundary are uncertain. As a point gets further from the boundary, its score should increase, until a point far from the boundary has an extreme probability estimate (0 or 1). The logistic equation is used directly to determine a point's score [64].

In the case of Decision Trees, the score representthe posterior probability of observing an instance of the positive class at a certain point, i.e. the fractions of positive observations in a leaf of a decision tree. By convention, a high score returned by a classifier on any given instance is likely to belong to the positive class 1 and a low score suggests that the instance is likely from the negative class 0 [63].

Such a ranking or scoring classifier could be used with a threshold to draw an entire curve of the classifier performance – one of them being the ROC. If the classifier output is above the threshold it gives a positive response, otherwise a negative response. Each threshold value produces a different (FPR, TPR) point in the ROC space [61]. For each threshold, TP is the count of true positive observations with scores greater or equal to this threshold and FP is the count of FP observations with scores greater or equal to this threshold. The negative counts, TN and FN are expressed in a similar way. The thresholds are then sorted in the descending order which corresponds to the ascending order of positive counts. The first threshold value corresponds to the highest "reject all" threshold and the corresponding values for (1-specificity) and sensitivity are computed for TP=0 and FP=0. The last threshold value is the lowest "accept all" threshold for which TN=0 and FN=0[65].

Several points in the ROC spaces have a significant meaning. ROC graphs are two-dimensional graphs in which the TP rate or sensitivity is plotted on the Y axis and FP rate (1-specifificity) is plotted on the X axis. The lower left point (0, 0) represents the strategy of never returning a positive classification; such a

classifier assumes no false positive errors but also gains no true positives. In other words, (0, 0) are the coordinates where the classifier finds no positives instances so that it always gets the negative cases right but it gets all positive cases wrong. The opposite strategy, of unconditionally issuing positive classifications, is represented by the upper right point (1, 1) [61]. For the second coordinate point (1, 1), everything is classified as positive so the classifier gets all positive cases right but it gets all negative cases wrong [66][66]. One point in the ROC space is better than another if it is located more to the northwest (TP rate is higher, FP rate is lower, or both) than the other one [61]. The point (0, 1) represents perfect classification and this is the result all classifiers are aiming for.

The diagonal line y = x represents the random guess performance. For example, if a classifier guesses the positive class half the time, it can be expected to get half the positives and half the negatives correctly which produces the point (0.5, 0.5) in the ROC space [61]. The "steepness" of the ROC curve is important in analyzing a classifier, since it is ideal to maximize the true positive rate while minimizing the false positive rate. Figure 6.2 depicts an example of ROC curve.

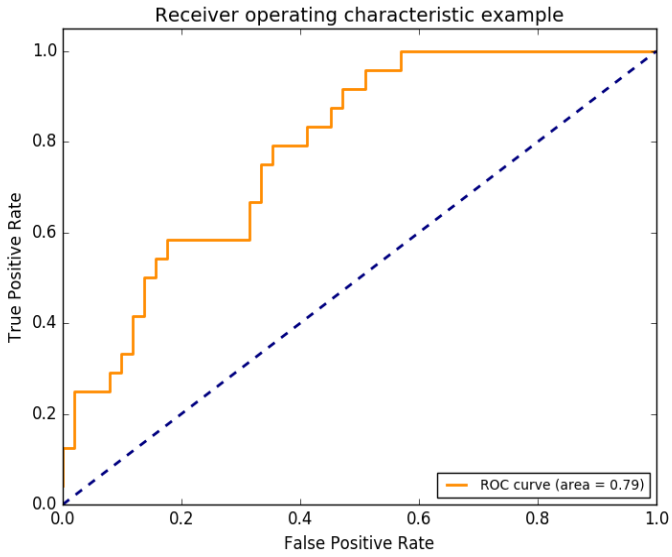

Figure 6.2 *Receiver Operating Characteristic adopted from [67]. The blue diagonal line represents the random guess.*

Area under the curve (AUC)

The area under the ROC curve, abbreviated AUC is a two-dimensional representation of classifier performance. For an easier comparison between classifiers' performance, the ROC space is reduced to a single scalar value representing the expected prediction performance [68]. Since the AUC is a portion of the area of the unit square, its value will always be between 0 and 1.0. However, because random guessing produces the diagonal line between (0, 0) and (1, 1), which has an area of 0.5, no classifier with discriminant power should have an AUC less than 0.5. A classifier that achieves perfect prediction will have an AUC equal to 1. The AUC has an important statistical property which says that is equivalent to the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance [68]. The AUC is computed in Python using trapezoidal approximation `auc(FPR,TPR)` [69]. For example, the AUC of the ROC in Figure 6.2 is equal with 0.79.

6.3 Precision-Recall Curve

Precision-recall (PR) curves represent another metric to evaluate classifier quality based on the confusion matrix parameters. Recall, also known as sensitivity, is a measure of how many relevant results are returned by the classifier (Equation (6.3)), while precision is a measure of how many selected items are relevant (Equation(6.4)) [70]. PR curves are similar to but different from the axes of the ROC curve. Interpreting their meaning for our image classification task, precision is the probability that a classifier-generated pixel label is true. Recall is the probability that a pixel with a positive class is detected. In other words, precision is a measure of how much noise is in the output of the model and recall is a measure of how much of the ground truth is detected [71].

$$precision = \frac{TP}{TP + FP} \tag{6.3}$$

$$recall = \frac{TP}{TP + FN} \tag{6.4}$$

The precision-recall graphic plots recall on the x-axis and precision on the y-axis as depicted in Figure 6.3. A precision-recall point is defined as a pair of (x,y) values, where x is recall and y is precision. A precision-recall curve is created by connecting all precision-recall points of a classifier [72]. The PR curve shows the trade-off between misses and false positives as the detector threshold changes [71]. A high area under the curve represents both high recall and high precision, where high precision refers to a low FPR, and high recall indicates to a low FNR. High scores for both metrics point out that the classifier is returning accurate results (high precision), as well as returning a majority of all positive results (high recall) [70].

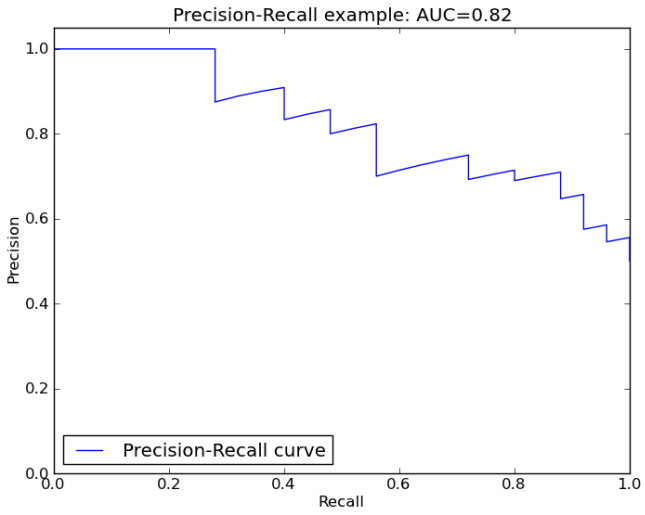

Figure 6.3 *Precision-Recall example plot with computed AUC.*

A classifier with high recall and low precision returns many results, however the majority of the predictions will be incorrect when compared to the training labels. In contrast, a system with high precision and low recall returns few results, but most of its predicted labels will be correct when compared to the training labels. The ideal classifier with high precision and high recall will return many results, with all results labeled correctly [70].

The PR curve of a random classifier shows a horizontal line that separates the graph into two areas as depicted in Figure 6.4 through the red line. While the baseline is fixed with ROC, the baseline of PR curve

is determined by the ratio of positives (P) and negatives (N) as y = P / (P + N) [73]. The area on top of the graph is where models with good performance should be placed. The area below the random line is where models would perform worse than random chance. For instance, in Figure 6.4, the random line is y = 0.5 when the ratio of positives and negatives is 1:1, whereas it becomes y= 0.25 when the ratio is 1:3 in favor of the negative class [72].

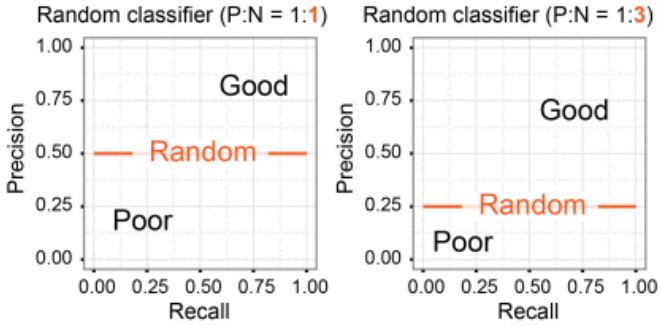

Figure 6.4 *Precision-Recall curves for different data distributions with random performance illustrated in red. In the left figure the ratio of positive to negative is balanced, while in the right figure the ration of positive to negative is inclined towards the negative class. Adopted from:[72].*

The PR curve of a perfect classifier shows a horizontal straight line from top left corner from (0.0, 1.0) to the top right corner (1.0, 1.0) and a vertical straight line to the end point (1.0, P / (P+N)). For instance, the end point is (1.0, 0.5) when the ratio of positives and negatives is 1:1, whereas (1.0, 0.25) when the ratio is 1:3 [72].

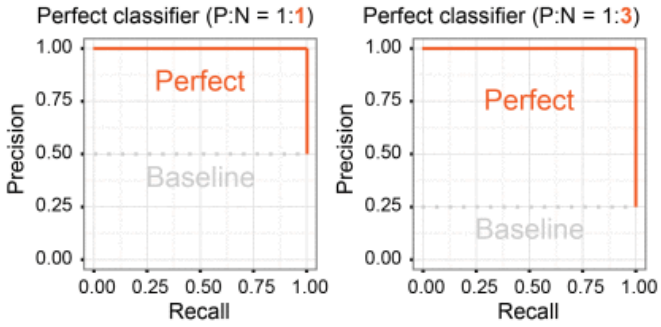

Figure 6.5 *Precision-Recall curves for different data distributions with perfect performance of the classifier illustrated in red. Left figure the ratio of positive to negative is balanced, while in the right figure the ration of positive to negative is inclined towards the negative class Adopted from:[72].*

Area under the curve (AUC)

Similar to ROC curves, the AUC can be used as a single performance measure of a precision-recall curve. Although the theoretical range of AUC score is between 0 and 1, the actual scores of meaningful classifiers are greater than P / (P + N), which is the AUC score of a random classifier [73] .

F1 score

The performance of a PR curve of a model can be summarized in a single number which is the harmonic mean of precision and recall, shown in Equation (6.5). A good classifier has the F1 score close to 1 [70].

$$F1-score = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{6.5}$$

6.4 Out-of-bag error for Random Forests

An important feature of Random Forests is its use of out-of-bag (OOB) samples that can measure the out of bag error. An out-of-bag sample is defined as follows: for each observation $z_i = (x_i, y_i)$ construct its random forest predictor by averaging only those trees corresponding to bootstrap samples in which z_i did not appear. The out-of-bag prediction error is the best approximation of the generalization (test) error. An OOB error estimate is almost identical to that obtained by K-fold cross-validation. When the OOB error stabilizes, the Random Forests training can be ended [41].

6.5 Conclusion

In this chapter we present common validation methods and their associated metrics for machine learning classifiers. We apply these methods to have a general estimate on the future classification accuracy of the algorithms. Confusion matrix is one of the validation metrics that can be created when a test set is available or a validation partition is created on the training set, as in the current scenario. This matrix is a representation of the True Positives and True Negative counts and it allows us to determine where does the classifier makes the larger error. Receiver Operating Characteristic (ROC) curve is a graphical representation obtained through variation of a threshold and plotting the resulted Sensitivity and Specificity values at each step. ROC curve could be used in combination with K-fold cross-validation in order to have a more reliable assessment on the classifiers' performance. In case of data imbalance, Precision-Recall curves are the recommended validation procedure, as they more robust than the ROC curves. Both of these graphical representations provide a metric known as the Area Under the Curve (AUC) which ranges between 0 and 1, where the latter is the value attributed to a perfect classifier.

Chapter 7

Classification Results

In the previous chapters, we present the workflow for building a machine learning classifier and the theory behind it. In this chapter we discuss the predictive power of several machine learning algorithms evaluated with the classification metrics presented in Chapter 6. These include confusion matrix, receiver operating characteristic, precision recall and their related metrics. This part is known as model validation and it evaluates the performance of the models build on a training set. Classifiers are build and validated on several input training sets. After validation, the classifier models are used to predict semantic segmentation on test images.

7.1 Validation Procedure

Key points are detected through erosion and dilation on the annotated images (Figure 7.1). The annotated images contain only pore structures as this is the aim of the classification task.

We choose an ellipsoidal kernel with a size of 10x10 for erosion and 35x35 for dilation. This shape of the kernel was chosen over others (e.g. square) since we consider a pore to be a circular structure. The key points are pixel coordinates on the contours provided by erosion and dilation. We do not select as key points all the pixels contained inside a region of interest since the block neighborhood allows to select sufficient information without overlap excess.

Next, we use as feature predictors pixels' intensities and select different combinations of their transformations to form several training sets. Images are used in their source format, converted to grayscale and histogram equalized. From these images, feature descriptor vectors are extracted. The feature descriptors form a matrix that is used as input for the classification algorithms.

The goodness of the model fit is measured through confusion matrix and cross-validation. The input data is split into 70% for training part and 30% for validation part to construct a confusion matrix.

Stratified cross-validation `StratifiedKFold()` with 5 folds as implemented in the scikit-learn module is applied in this work. This method ensures that the folds are created by preserving the percentage of samples for each class [39]. The ROC and Precision-Recall responses on different dataset combinations created from cross-validation are shown and discussed.

Additionally, we apply dimensionality reduction with Principal Component Analysis to explore how the performance of the model changes when the number of predictors is reduced.

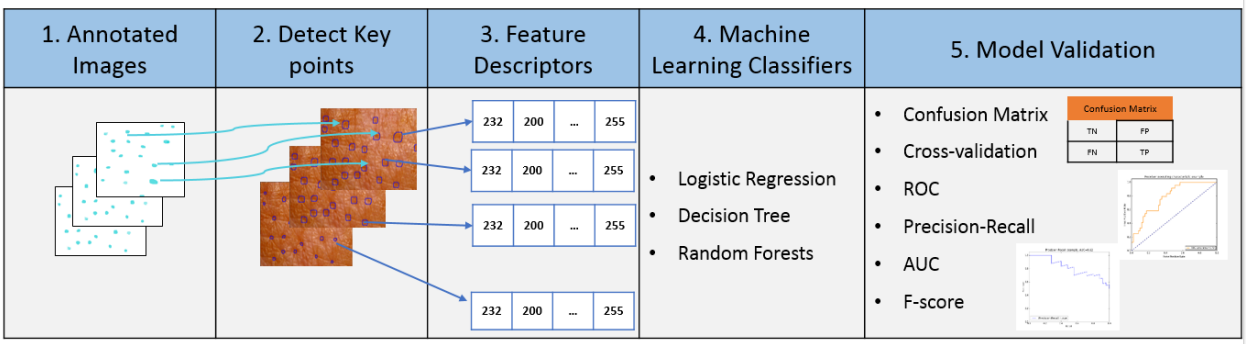

Figure 7.1 *Semantic Segmentation workflow.*

Our training set consists of 105 annotated skin images of dimensions 640x480 and 91,1 kB size. We consider our positive class 0 to be denoted as pore, represented by the pixels that define this particular skin structure in the annotations. The negative class, or class 0, is denoted as skin and it includes everything else on the skin that is not labeled as a pore, in concordance with the annotations. Table 7 shows the distribution of the classes coming from 105 annotated images. The proportion of negatives to positive is imbalanced in a ratio of 4:1, which can create a classification bias towards the majority class.

Table 7 *Data distribution*

	Instances	Ratio
Class 0 (Skin)	242041	78.7%
Class 1 (Pore)	65363	21.3%

We address this class imbalance by adjusting the class weight. The scikit-learn library from Python implements the `class_weight` parameter through which the importance of the classes can be tuned. The "balanced" mode of this parameter applied in this work uses the values of *y* (the labels vector) to automatically adjust weights inversely proportional to class frequencies in the input data as shown in Equation (7.1) [60]:

$$class\_weight = \frac{n\_samples}{n\_classes \times BinsCount(y)} \tag{7.1}$$

where $n\_samples$ represent the number observations, $n\_classes$ is the number of in the target vector and $BinsCount(y)$ is the number of occurrences for every class. Altering the class importance has an effect only on the cost of class errors (False Negatives, if the minority class is positive). This method adjusts a separating surface to decrease these errors accordingly. If the classifier makes no errors on the training set, then no adjustment may occur, so altering class weights may have no effect [40].

7.2 RGB Images

In the first instance, the pixels intensities from the RGB channels of a picture are employed as feature predictors. The size of the neighborhood is 3 x 3, which renders 27 predictors in total. We present and discuss the validation graphics for Logistic Regression, Decision Trees and Random Forest. No dimensionality reduction is employed as the numbers of predictors is considered to be small.

To demonstrate that adjusting for class imbalance has a major effect in the prediction power of a classifier, we show in Figure 7.2 and Figure 7.3 the confusion matrices of the same classifier build on the same training data.

Figure 7.2 shows the confusion matrix on the 30% of the data, with the Logistic Regression model trained on the other 70%. No methods for imbalance adjustment are applied, so the classifiers prediction are biased towards voting the majority class, i.e. the negative class (TN = 95.48%). The prediction power for the positive class is close to random guess, as demonstrated by the TP ratio (TP = 60.97%).

Figure 7.3 shows the confusion matrix of the same model, this time with weight for class importance placed on the positive class. The TP and TN ratio are similar, with a much better TP ratio of 81.06%. The

FP ratio decreases by half, while the FN error is tripled (from 4.52% to 15.0%). Consequently, the `class_weight='balanced'` parameter is applied in building all the classifiers.

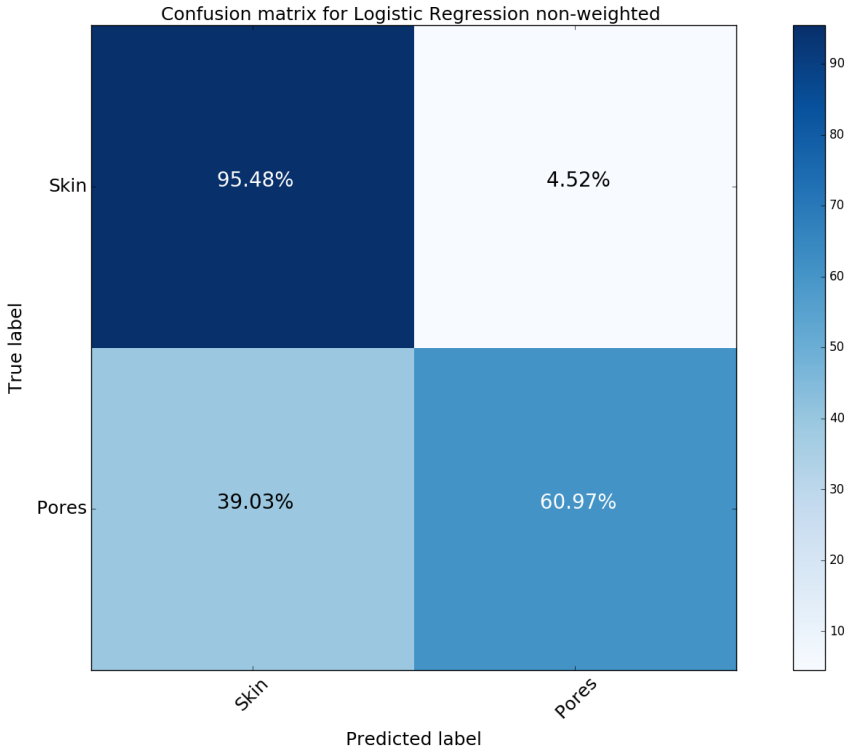

Figure 7.2 *Confusion Matrix for Logistic Regression. The weights are not adjusted for class imbalance. The classifier is biased towards the majority class, hence the high TN ratio and small TP ratio.*

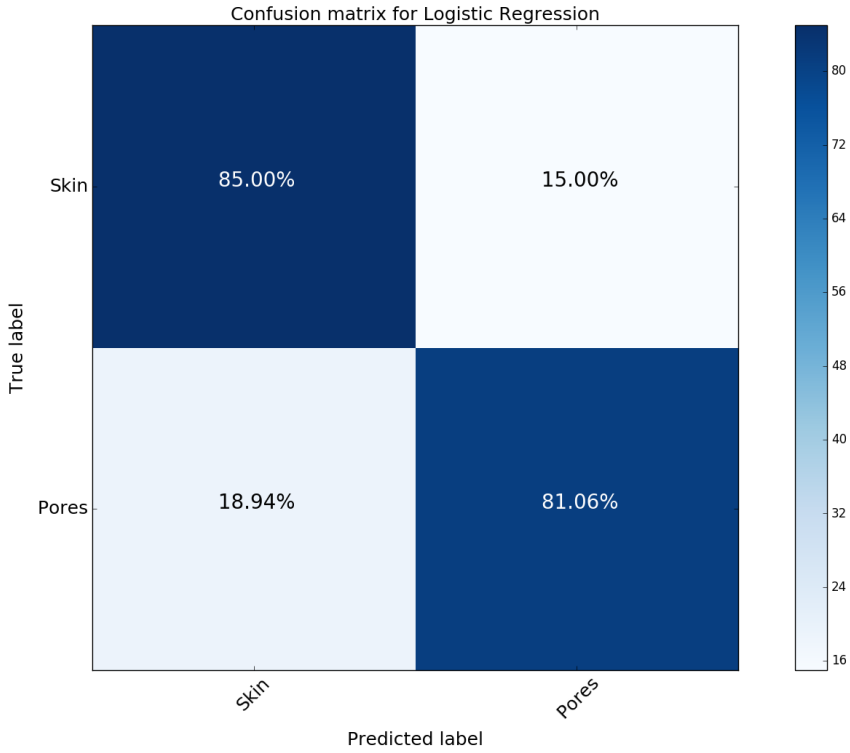

Figure 7.3 *Confusion Matrix for Logistic Regression. The weights have been adjusted accordingly for data imbalance. The TP ratio becomes as good as the TN ratio, while the FP ratio is decreased.*

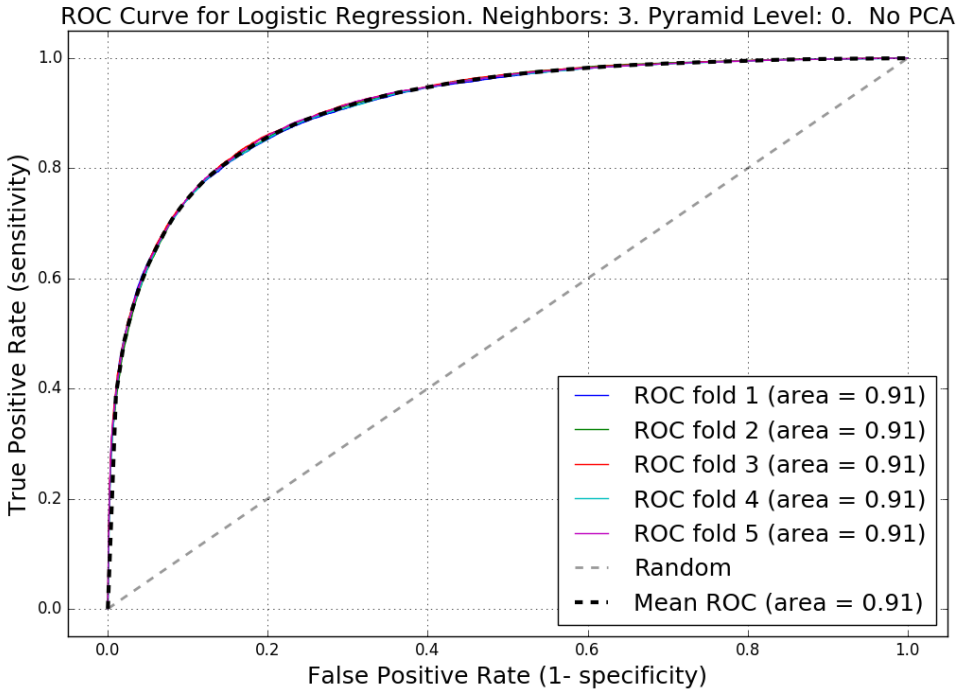

Figure 7.4 *Receiver Operating Characteristic curves for Logistic Regression. Curves drawn on cross-validated folds. The average AUC indicates a good model.*

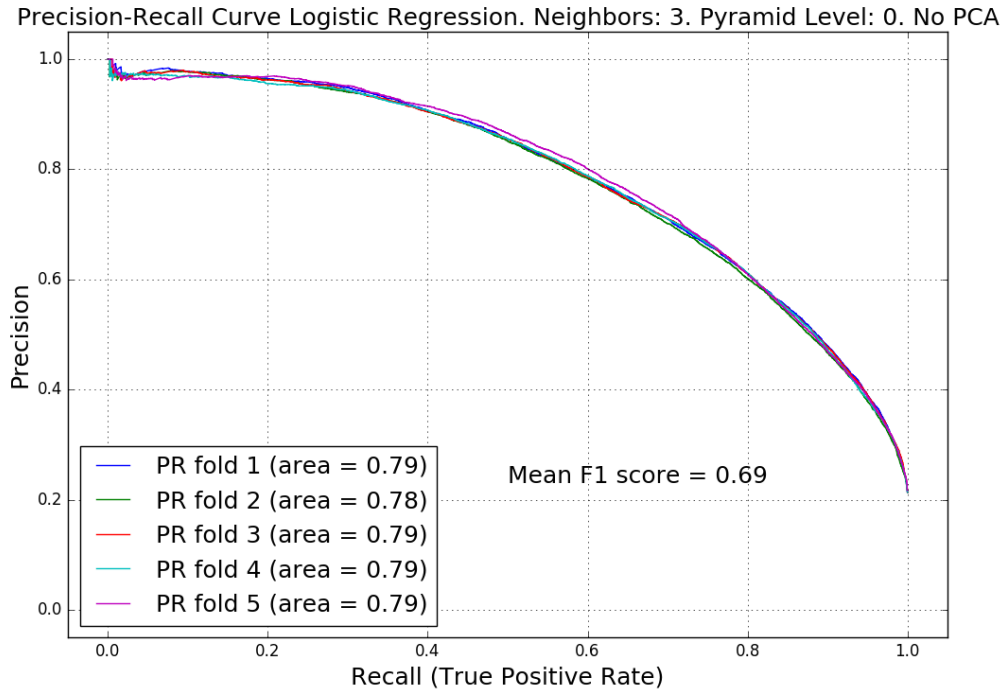

Figure 7.5 *Precision Recall curves for Logistic Regression. Curves drawn for cross-validation with 5 folds. The precision-recall curve has lower AUC than the ROC as it accounts for data imbalance.*

Figure 7.4 and Figure 7.5 show the predictive performance of the Logistic Regression model through cross-validation on 5 folds. Although the ROC (Figure 7.4) shows a very good AUC = 0.9, the Precision-Recall area of 0.79 and F1 score = 0.69 is indicative of the true performance of the model on an imbalanced dataset as in this case.

Decision Trees

Decision Trees algorithms have several parameters that influence the generalization power of the output model. To exemplify, we try different values for the `min_samples_split` and `min_samples_leaf` parameters of the `sklearn.tree.DecisionTreeClassifier`() applied in this work with the Python library. The first parameter refers to the minimum number of samples required to split an internal tree node, while the latter signifies the minimum number of samples required to be at a leaf or terminal node. Further, we show several validation metrics for different parameter values.

- `min_samples_split = 5, min_samples_leaf = 5`

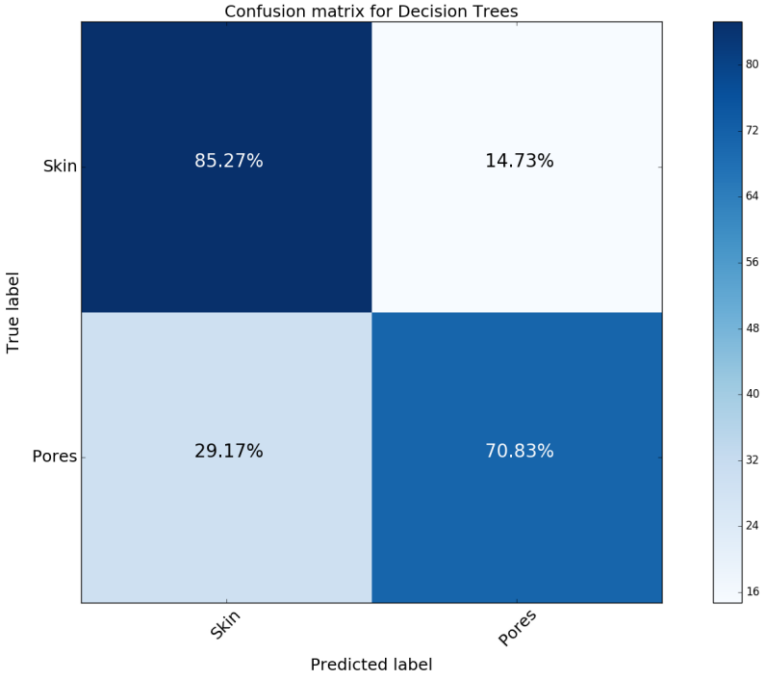

Figure 7.6 *Confusion Matrix for Decision Tree build with min_samples_split=5, min_samples_leaf=5.*

- `min_samples_split = 25, min_samples_leaf = 25`

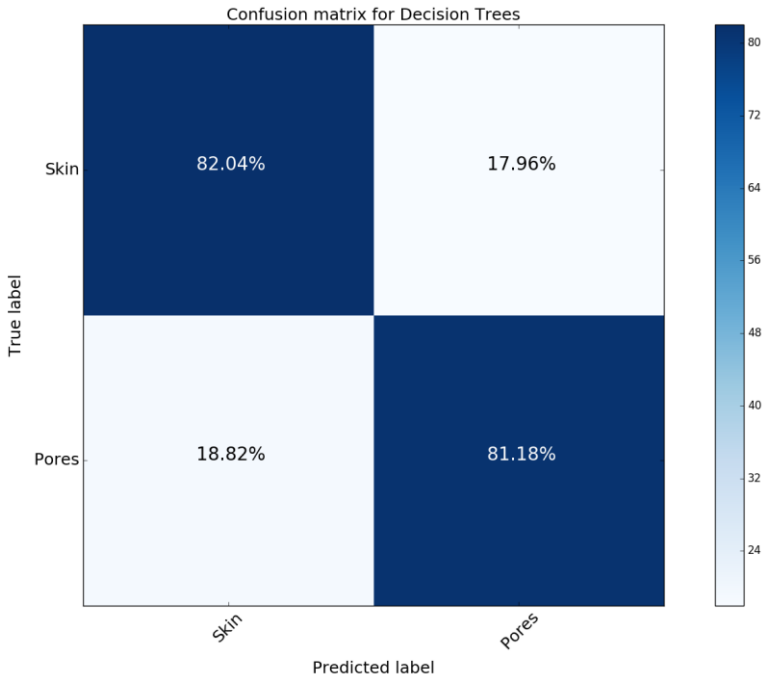

Figure 7.7 *Confusion Matrix for Decision Tree build with min_samples_split=25, min_samples_leaf=25.*

- `min_samples_split = 100, min_samples_leaf = 100`

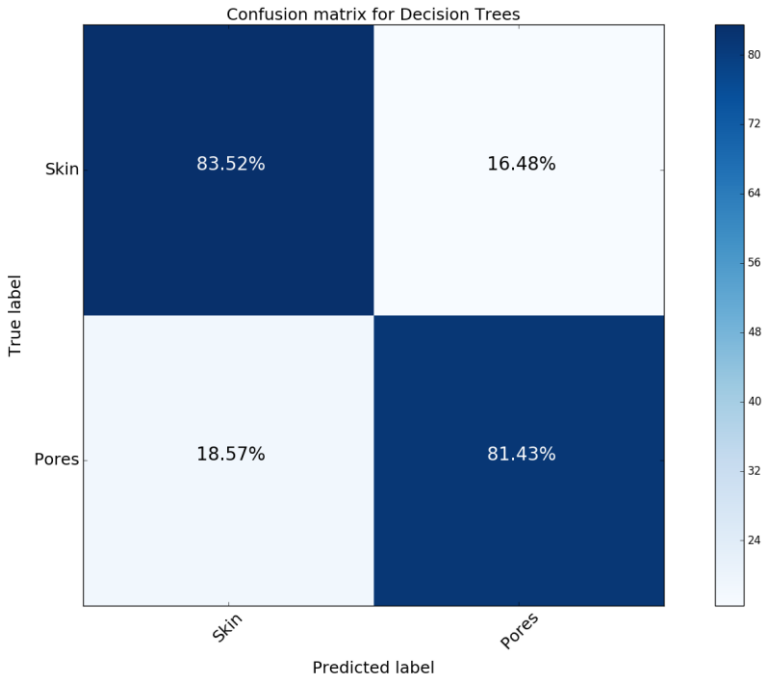

Figure 7.8 *Confusion Matrix for Decision Tree build with min_samples_split=100, min_samples_leaf=100.*

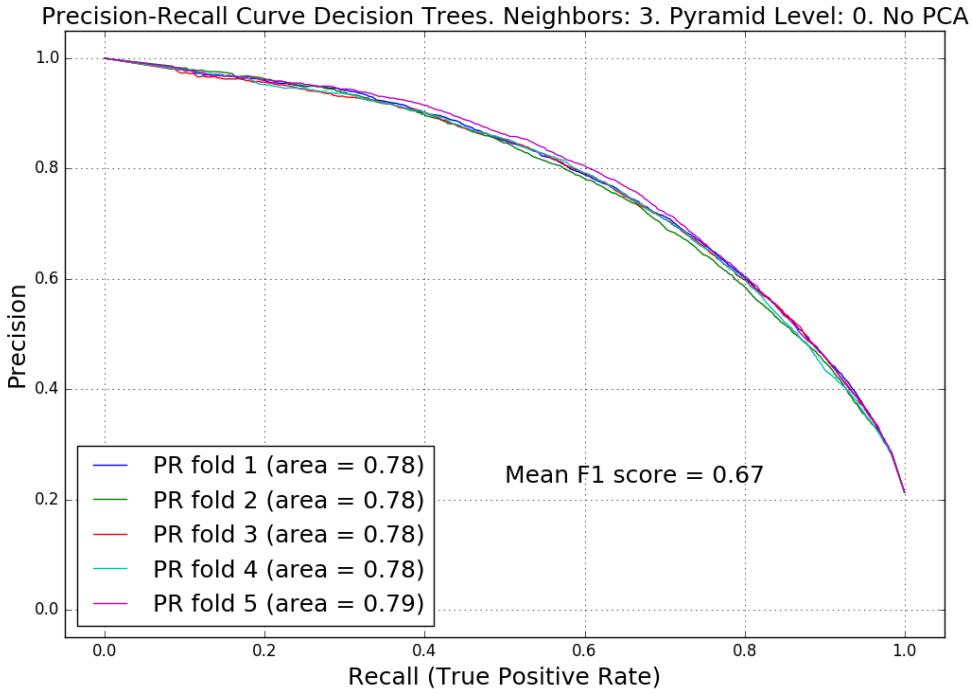

Figure 7.9 *Precision Recall curve for Decision Trees. Curves drawn for cross-validation with 5 folds.*

Increasing the number of splits and samples in a leaf node from 5 to 100 improves the performance of the model, in particular the TP counts (Figure 7.7) Nevertheless, after a certain threshold, the model stabilizes (Figure 7.8). Decision Trees are very sensitive to small changes in the dataset and tend to overfit. They seem to perform worse than Logistic Regression, as explained by the mean F1 score = 0.67, which is depicted in Figure 7.9. Perhaps the high-dimensionality of the dataset poses a problem for this algorithm.

Random Forests

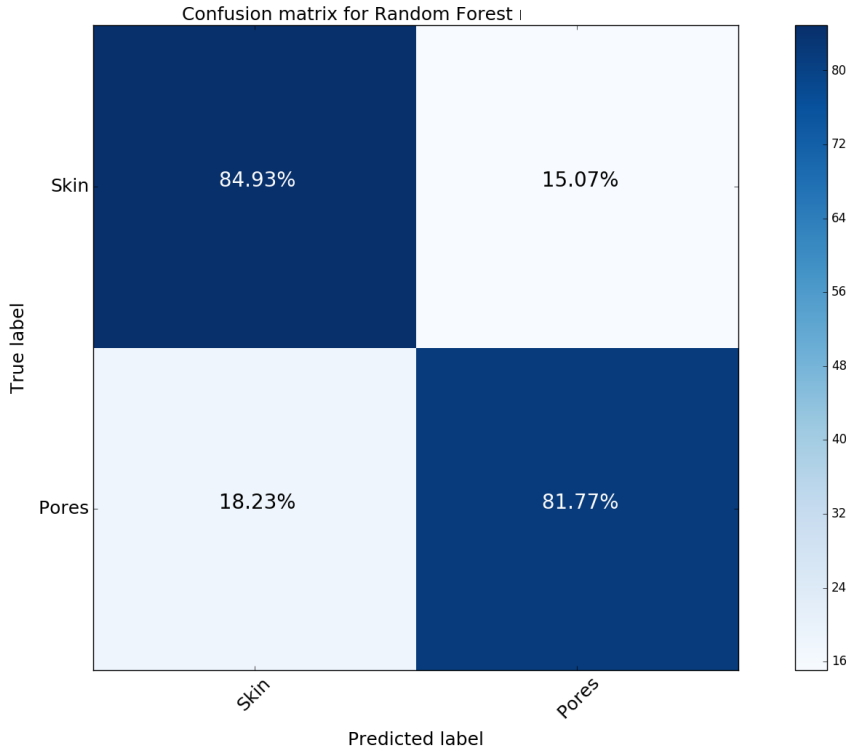

Figure 7.10 *Confusion matrix for Random Forests build with n_estimators=5, min_samples_leaf=100, min_samples_split=100.*

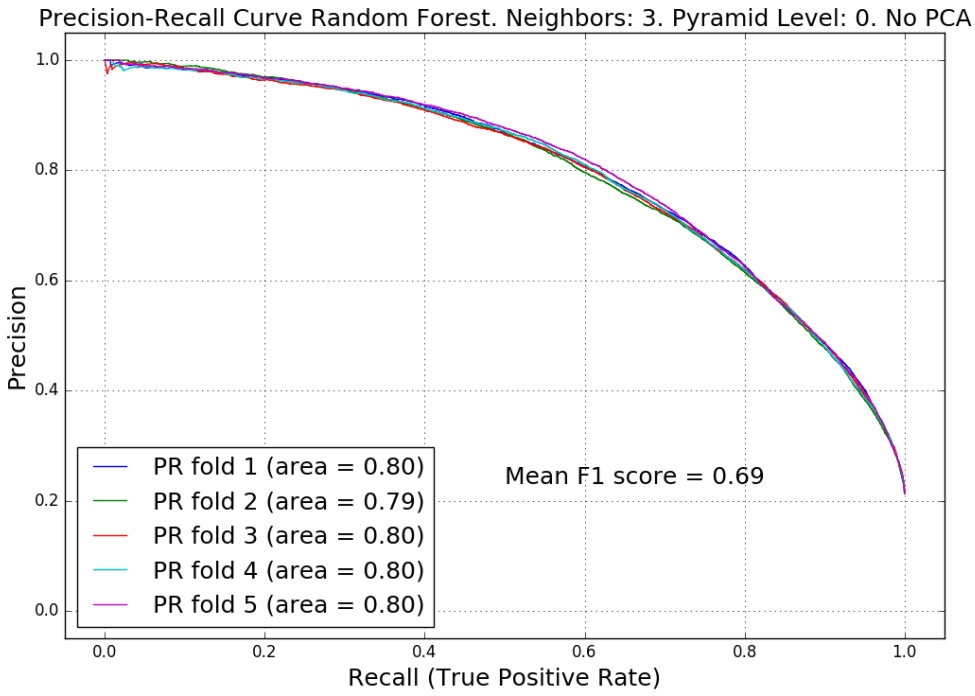

Figure 7.11 *Precision-Recall curve for Random Forests*

The confusion matrix for a Random Forests model build with a minimum number of 5 trees per forest (`n_estimators=5`) is displayed in Figure 7.10. Based on the values computed, its performance seems to be similar with the other classifiers build on the same combination of input predictors (one RGB image).

The PR curve in Figure 7.11 shows that Random Forests a have an identical F1 score as the Logistic Regression. Nevertheless, it has a greater AUC (AUC = 0.8) compared to Logistic Regression and Decision Trees. Overall, with only 27 predictors used in training all classifier models perform similarly.

7.3 RGB Images and Gaussian Pyramid

In this section we explore how adding Gaussian Pyramid to the feature descriptor influences the performance of the model. We choose 4 pyramid levels, since an image is 640 x 480 (width x height). This signifies that at the highest level in the pyramid (the 4^{th} one), the image is 40 x 30. Further increasing the pyramid level renders a small image of only a few pixels size from where is difficult to extract a block neighborhood and compute a full feature descriptor vector.

We persist with extracting a square block neighborhood of size 3x3 from each RGB channel. Since we apply Gaussian Pyramid with 4 levels the feature space increases to 135 predictors (3x3 neighborhood x total number of images x number of channels in the image). The samples size remains the same as described in Table 7.

The key points on the border of the image are ignored from the classification process to account for errors in exiting the available matrix space with the block neighborhood. The size of the image border to be ignored is defined based on the size of the neighborhood and the pyramid level applied.

Logistic Regression

Figure 7.12 shows the confusion matrix for Logistic Regression when 135 predictor columns are employed in classification. We observe that in comparison to 27 predictors being used (Figure 7.3), the number of FP is reduced greatly from 15% to 6.77%, while the TN count increases from 85% to 93.61%. The number of TP, which is central to our classification goal, is increases also from 81.06% to 92.09% compared to the previous Logistic Regression model.

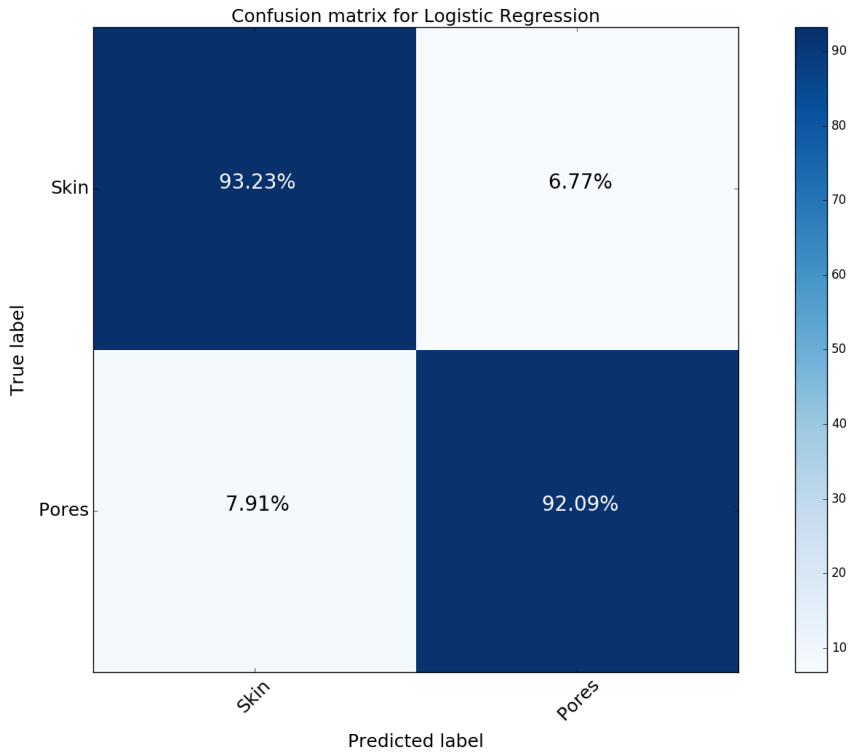

Figure 7.12 *Confusion Matrix Logistic Regression. 135 predictors used in classification.*

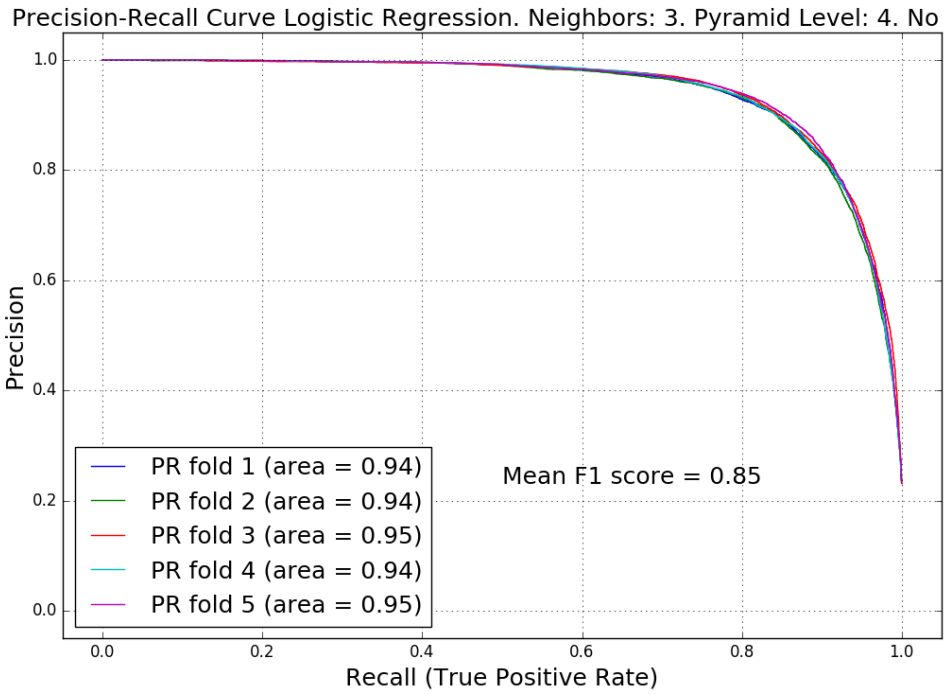

Figure 7.13 *Precision-Recall curve for Logistic Regression. 135 predictors used in classification.*

The decrease in FPR leads to an increase in precision and F1 score, supported by the Precision-Recall curve in Figure 7.13. The high-precision value indicates that there is little noise in the system with most of the predicted labels being correct. The low recall signifies that the classifier retrieves few relevant results, but those retrieved values are highly relevant (high precision value).

Logistic Regression with Principal Component Analysis

The high number of predictors (135) may pose difficulties for machine learning classifiers. Principal component analysis is applied to reduce the dimension of the feature-space as shown in Figure 7.14.

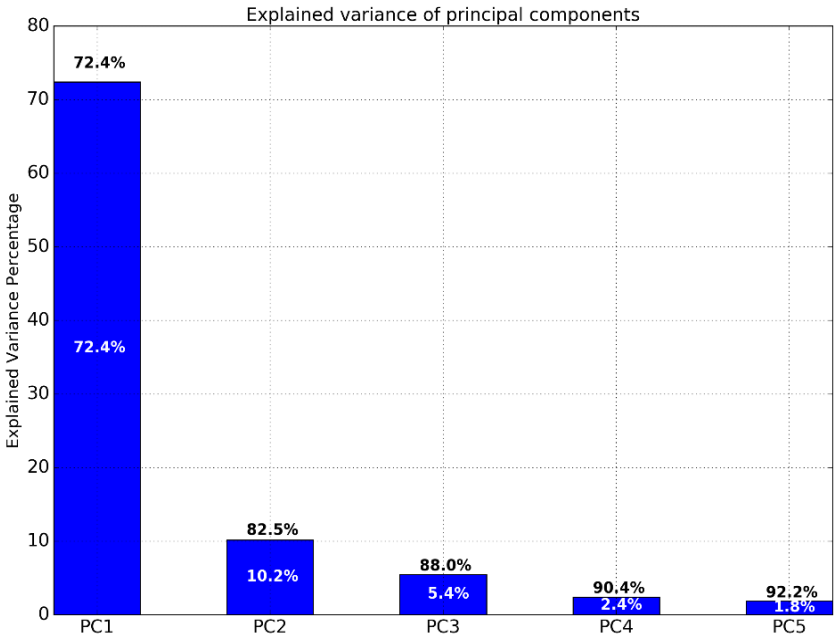

Figure 7.14 *Explained variance of 5 principal components for Logistic Regression. 135 predictors used in classification.*

The explained variance of the first five principal components is depicted in Figure 7.14, with each component plotted on the x-axis and its variance plotted in the bar (white font) and the cumulative variance above the bars (black font). The first component describes already 72% of the data variance, while the first 5 components describe 92.2% of the variance. This results suggests that selecting 5 components is sufficient to capture all the information contained by 135 predictors.

After applying PCA the performance of the Logistic Regression model decreases as exemplified by the confusion matrix in Figure 7.15 and PR curve in Figure 7.16. This result may be explained by the fact that Logistic Regression already places the data points in maximally separating hyperplanes. Another reason could be that some essential information from the 135 predictors might be discarded when using PCA.

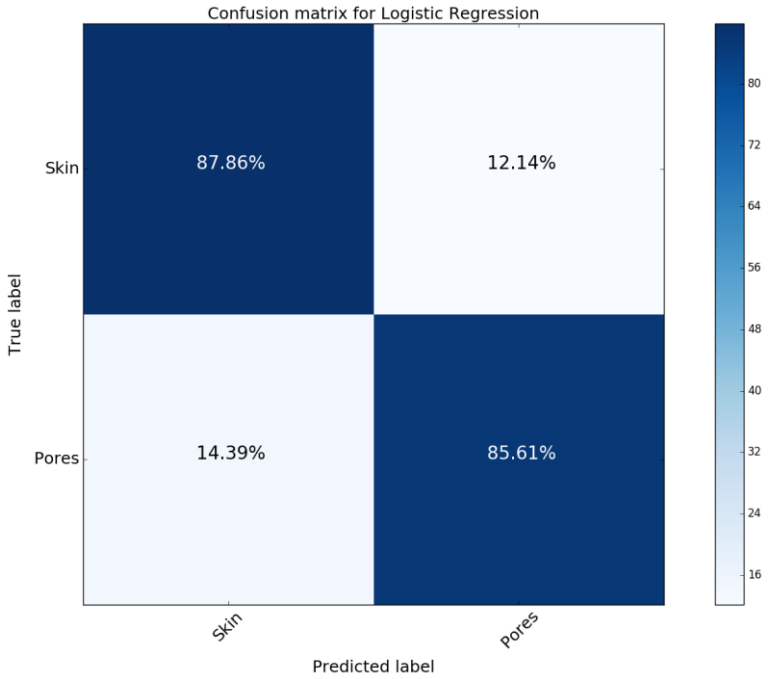

Figure 7.15 *Confusion matrix for Logistic Regression after applying Principal Component Analysis.*

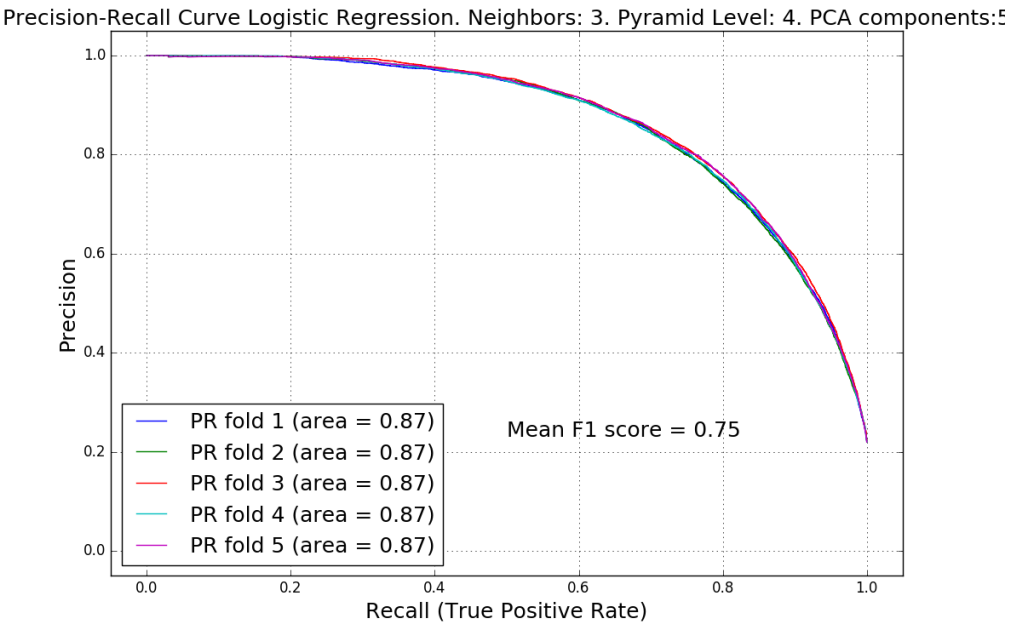

Figure 7.16 *Precision-Recall curve for Logistic Regression with 5 PCA components.*

Decision Trees

The Decision Trees model with 135 features (Figure 7.17) seems to have an improved predictive power compared to its counterpart model with just 27 predictors, but similar performance as the Logistic Regression model shown in Figure 7.12. The same parameters as before are used to fit the Decision Trees model.

- `min_samples_split=100, min_samples_leaf=100`

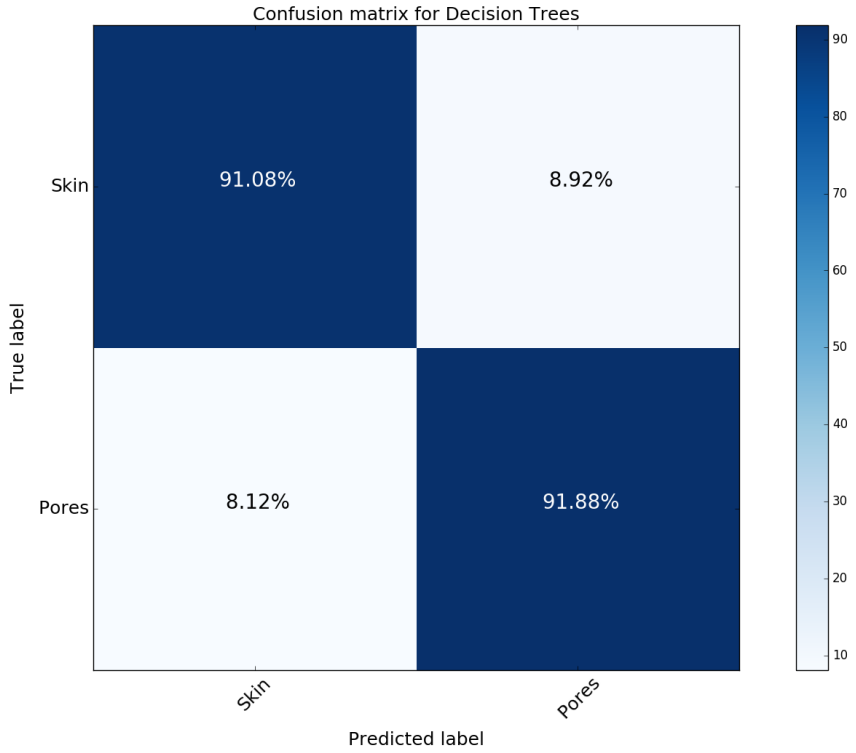

Figure 7.17. *Confusion Matrix Decision Trees. 135 Predictors used in classification.*

Random Forests

This classifier is based on bootstrap aggregation where each tree is built from a sample drawn with replacement from the training set of observations [74]. Figure 7.18 displays a graph with the Out-of-Bag error (OOB) rate plotted on the y-axis and the number of trees estimators taken into the construction of the trees on the x-axis. This error is calculated from the predictions made on the trees that do not include the respective bootstrap sample in their fitting phase [74]. The plot in Figure 7.18 shows how the OOB error changes with the number of estimators and the number of maximum allowed features. The error is lowest when using the square root or the logarithm of the number of predictors as parameter for maximum features. Moreover, it reaches its lowest value and stabilizes when employing around 100 estimators.

The validation of the Random Forest model is depicted in Figure 7.19 through the confusion matrix and through precision-recall curves, respectively shown in Figure 7.20. The TP rate is 96.49% (Figure 7.19), while the AUC is 0.99 and the average F1 score of the precision-recall values is 0.95. Both metrics are very close to 1 indicating a model with accurate predictive power. Comparing these validation metrics with the ones computed previously for the other models, we can select this model as having the best estimated predictive power. Consequently, this model will be used further for predicting the pixel-wise labelling also called semantic segmentation on new images.

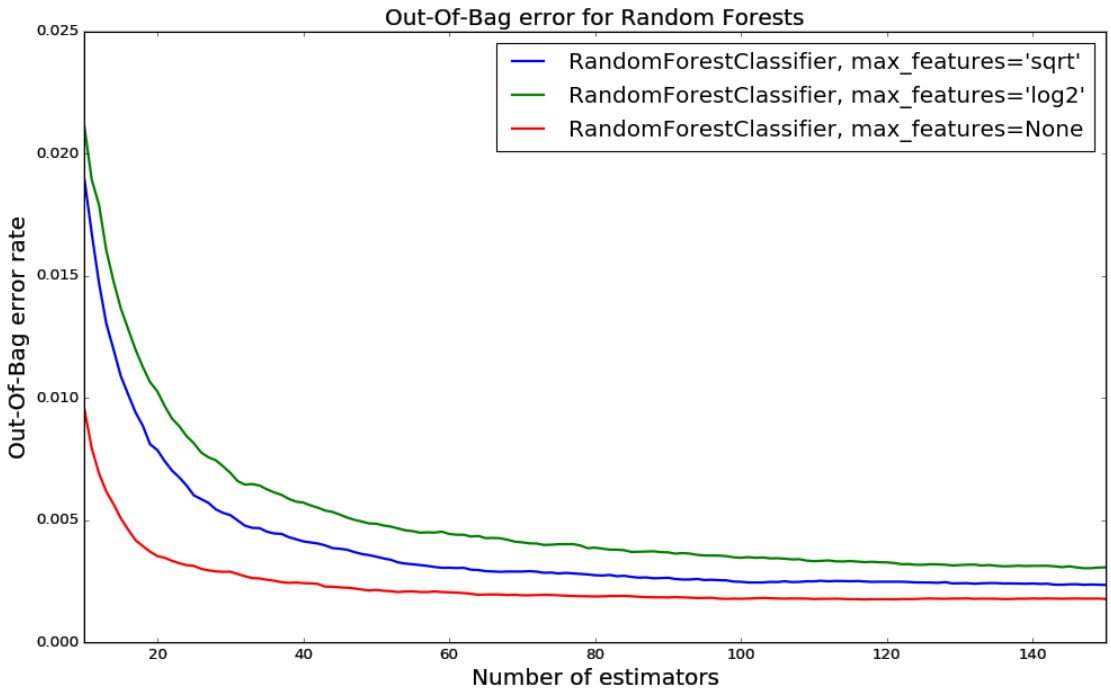

Figure 7.18 *Out-of-Bag error for Random Forests with different values for the maximum number of estimators used.*

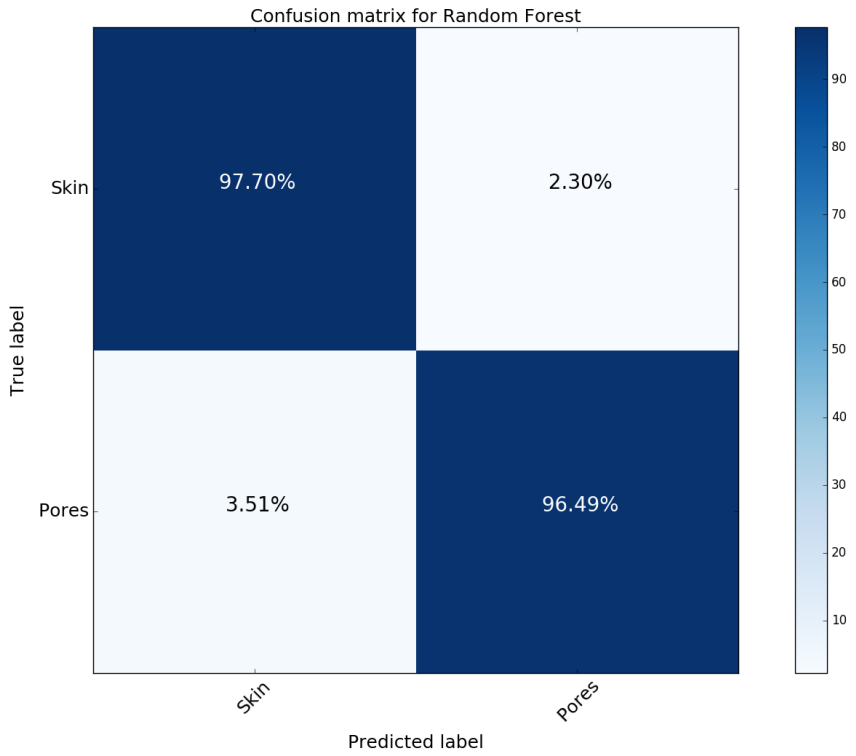

Figure 7.19 *Confusion Matrix for Random Forests using n_estimators=100, min_samples_leaf=10, min_samples_split=15.*

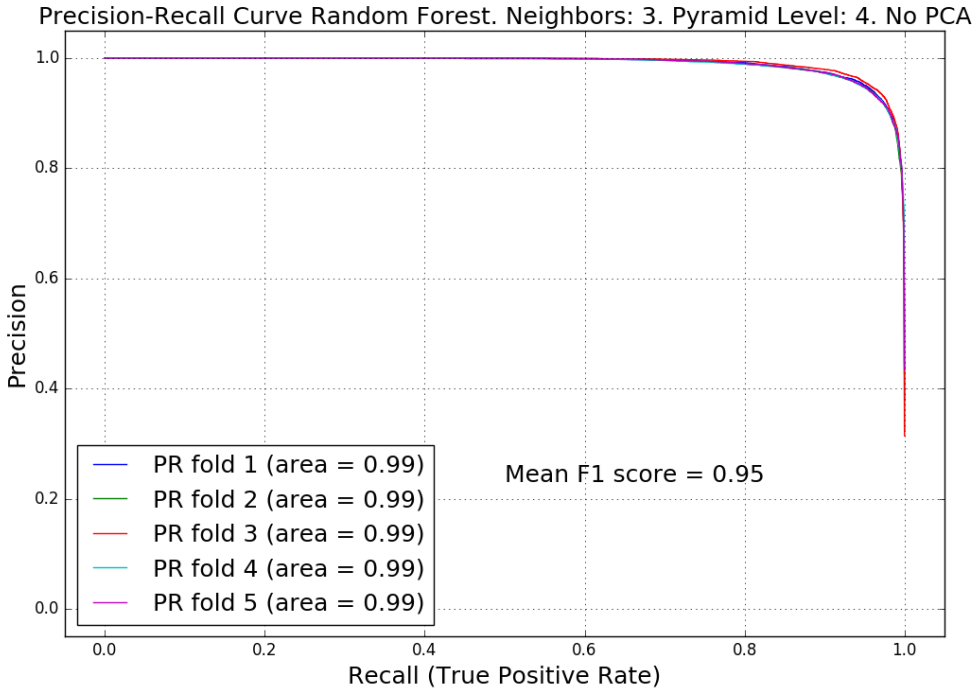

Figure 7.20 *Precision-Recall curve for Logistic Regression. 135 predictors used in classification.*

7.4 Grayscale Images and Gaussian Pyramid

A grayscale image is just a linear representation of an RGB image. It has only one channel (compared to the 3 channels in an RGB image). Consequently, the dimensionality of our feature descriptor is reduced. If we compute the grayscale equivalent of an image and extract a 3x3 pixel neighborhood from its 4 pyramid levels then we end up with 45 predictor columns. Applying the same machine learning classifiers as in the previous chapter and validating on a hold-out test that represents 30% of the dataset, the following confusion matrices in Figure 7.21, Figure 7.22 and Figure 7.23 are obtained. Again, the optimal performance is obtained with the Random Forests model (shown in Figure 7.23), where the TN ratio is 96.38% and the TP one is 94.55%. Comparing with the RGB-based model displayed in Figure 7.19, the performance in classifying TP decreases just with 2 percentages. Overall, this result suggests that Random Forests is the best model even when using grayscale images. The advantage of using this model is the small number of features (from 135 to 45) which render the prediction time much faster. The time factor could be of paramount importance especially when integration in real-time devices is desired.

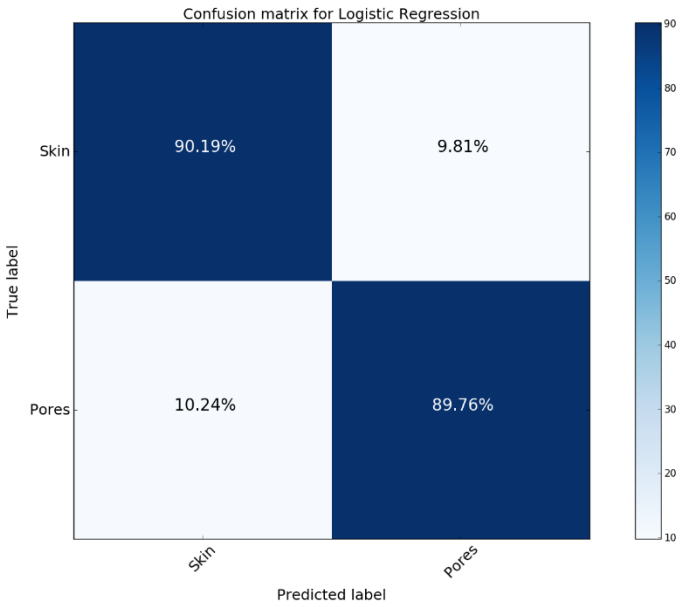

Figure 7.22 *Confusion Matrix for Logistic Regression. Grayscale images used for input predictors.*

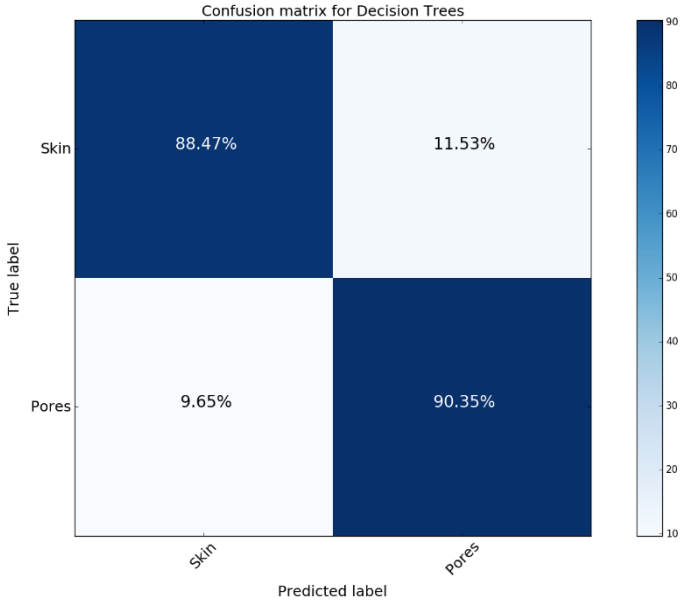

Figure 7.21 *Confusion Matrix for Decision Trees. Grayscale images used for input predictors.*

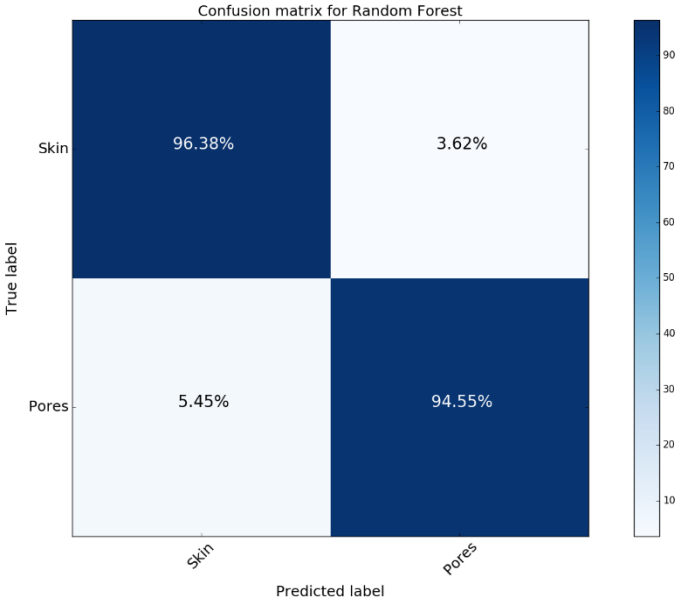

Figure 7.23 *Confusion Matrix for Random Forests. Grayscale images used for input predictors.*

7.5 Histogram Equalized Images and Gaussian Pyramid

Histogram equalization transforms a grayscale image by uniformly spreading out the most frequent intensity values. The image is normalized and becomes more contrasted. Region of interests, like pores are enhanced, while non-linear noise is mitigated. By applying histogram equalization on our images, we reduce the dimensionality of the feature descriptors. We theorize that the histogram equalized images serve as better input than the grayscale images for the machine learning classifiers. We apply the same machine learning classifiers as before and validate on a hold-out test that represent 30% of the dataset. The validation result is measured with the confusion matrices shown in Figure 7.25, Figure 7.26 and Figure 7.27. Largely, the histogram equalized based models have lower performance than the grayscale-based images (Chapter 7.4). The Random Forests model, for which the validation is shown in Figure 7.27, is the optimal model in this case as well, when comparing with the previous models. Its predictive power for positives decreases with just one percent from 94.55% (confusion matrix displayed in Figure 7.23) to 93.77%.

Histogram equalized images may also be used to reduce the area of search for regions of interest. To exemplify, given Figure 7.24, we notice that pores are stronger contrasted in the image compared with other structures. Since pores are darker, we can select as key points all pixels that have intensity closer to 0. The 25^{th} percentile could be used as threshold for approximating the pores coordinates that have intensities below a certain value. By reducing the number of candidates we remove the constraint to iterate through each pixel in the image to predict its label. Thus, the prediction algorithm can become much faster, enabling real-time applications.

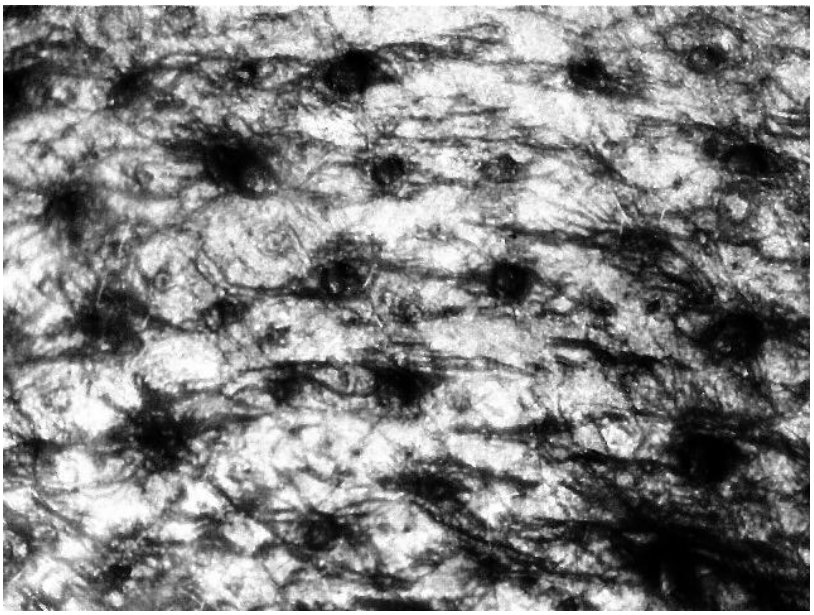

Figure 7.24 *Example of histogram equalized image*

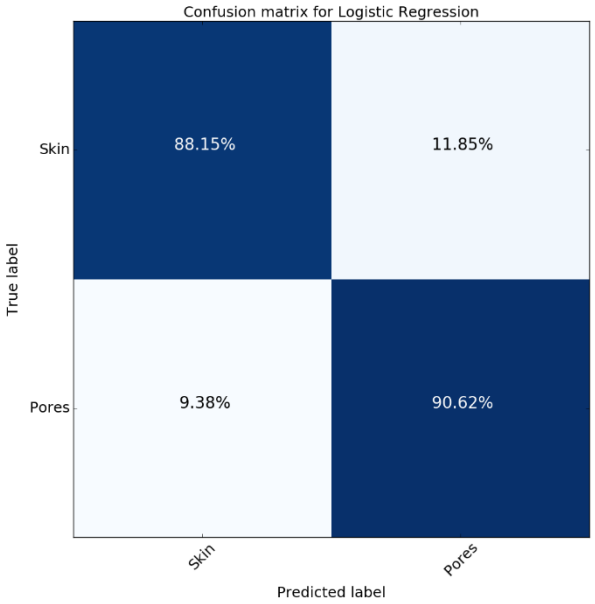

Figure 7.25 *Confusion Matrix for Logistic Regression. Histogram equalized image used for input predictors.*

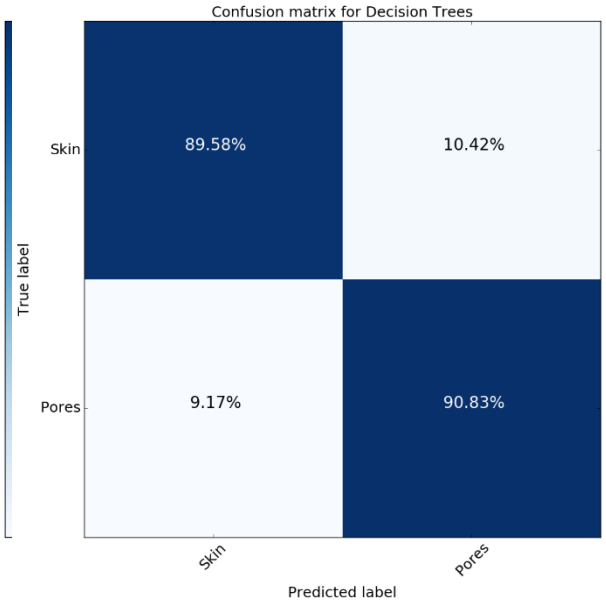

Figure 7.26 *Confusion Matrix for Decision Trees. Histogram equalized image used for input predictors.*

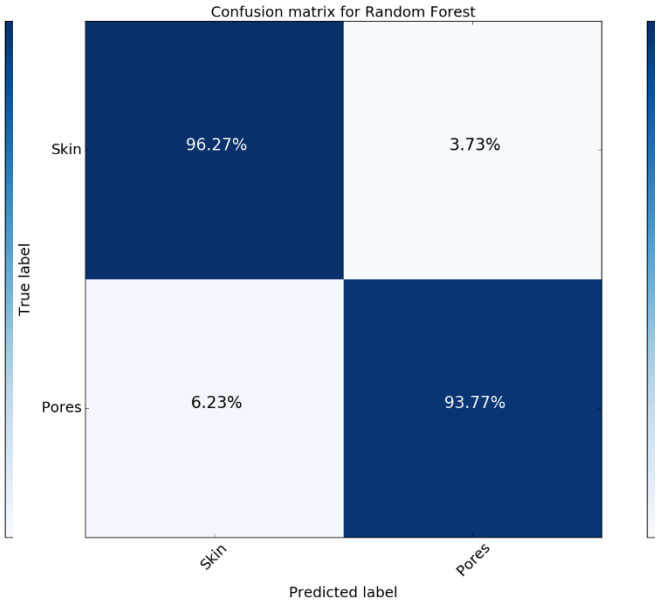

Figure 7.27 *Confusion Matrix for Random Forests. Histogram equalized image used for input predictors.*

7.6 Semantic Segmentation

In this section we test how the models perform visually on new images that have not been part of the training set. Testing is achieved by applying the previously fitted models (Chapter 7.2 to Chapter 7.5) on unseen images. The result is a semantic segmentation of an image, represented by pixel-wise labelling with either class 1 or 0. There are several ways to display the output of the classifiers, from a complete separate binary image or by color-labelling the pixels of interest onto the source image.

The predictions of the classifiers are tested on a sample image that was not included in the training phase (Figure 7.28). Analyzing the predictions on the test image, we can visually assess that indeed the Decision Trees models are weaker with respect with the other classifiers, Logistic Regression and Random Forests. This behavior is expected as supported by the validation results presented in the previous sections from Chapter 7. Their segmentations are noisy, as the models classify scattered red pixels as being part of a pore (Figure 7.30, Figure 7.33, Figure 7.36 and Figure 7.39.). It is difficult to tell in segmentations of Decision Trees-based images what the area and region of a pore is.

Logistic Regression models shown in Figure 7.29 and Figure 7.32 come close in performance to their Random Forests counterparts build on the same input predictors depicted in Figure 7.31 and Figure 7.34. However, some noise is still present in the form of scattered red pixels labeled as pore, which confirms Random Forests Models with RGB and Gaussian Pyramid predictors as the optimal model.

The grayscale and histogram equalized models both for Logistic Regression (Figure 7.35 and Figure 7.38) and Random Forest (Figure 7.37 and Figure 7.40) have less noise than the models based solely on RGB predictors. At first glance, the difference between the grayscale, the histogram equalized (Figure 7.37 and Figure 7.40) and RGB predictors based models (Figure 7.34), seem to be the size of the area considered to be a pore. The former map a pore a larger area, while the latter map a more compact and smaller area for a pore. Visually, the grayscale and the histogram equalized models suggest similar performance with the RGB and Gaussian Pyramid based models, while having the advantage of reduced dimensionality (from 135 to 45 predictors). Nevertheless, if we look at the semantic segmentation achieved on the right border of the picture, where we can manually count 5 pores (Figure 7.28), we notice the superiority of the Random Forests RGB with Gaussian Pyramid model compared with the grayscale and histogram equalized. The Random Forests models successfully identifies the 5 pores (Figure 7.34), while the other models misclassify pores under the form of scattered red pixels around the region of the pores (Figure 7.37 and Figure 7.40).

Figure 7.28 *Sample Test Image with 5 pores marked with red circles on the right side of the picture.*

Logistic Regression | **Decision Trees** | **Random Forests**

RGB

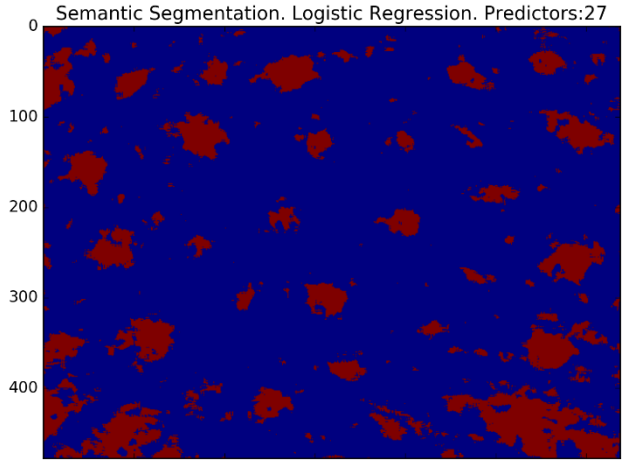

Figure 7.29 *Semantic Segmentation prediction based on RGB predictors.*

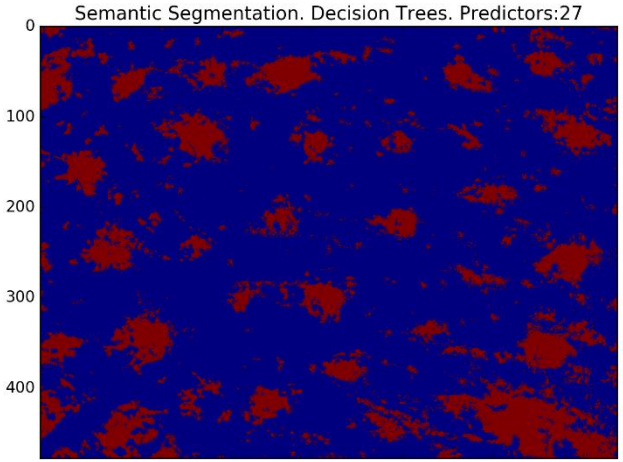

Figure 7.30 *Semantic Segmentation prediction based on RGB predictors.*

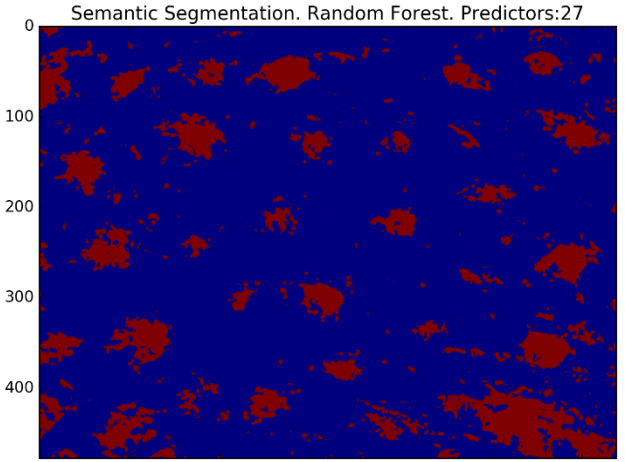

Figure 7.31 *Semantic Segmentation prediction based on RGB predictors.*

RGB + Gaussian Pyramid

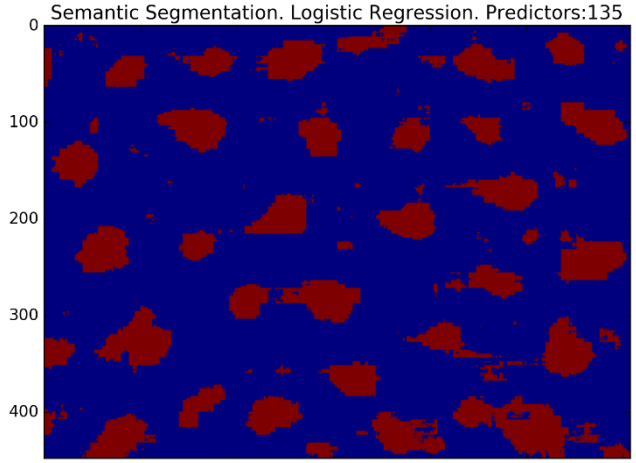

Figure 7.32 *Semantic Segmentation based on RGB and Gaussian Pyramid predictors.*

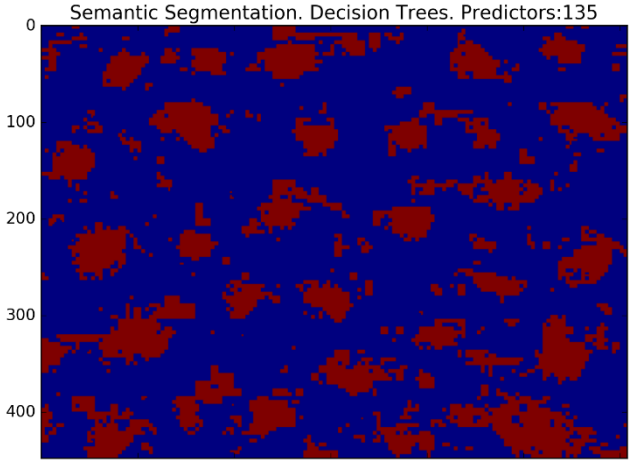

Figure 7.33 *Semantic Segmentation based on RGB and Gaussian Pyramid predictors.*

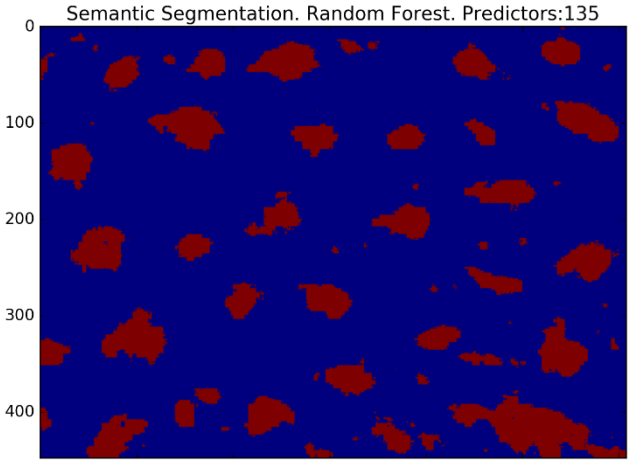

Figure 7.34 *Semantic Segmentation based on RGB and Gaussian Pyramid predictors.*

Logistic Regression | **Decision Trees** | **Random Forests**

Grayscale + Gaussian Pyramid

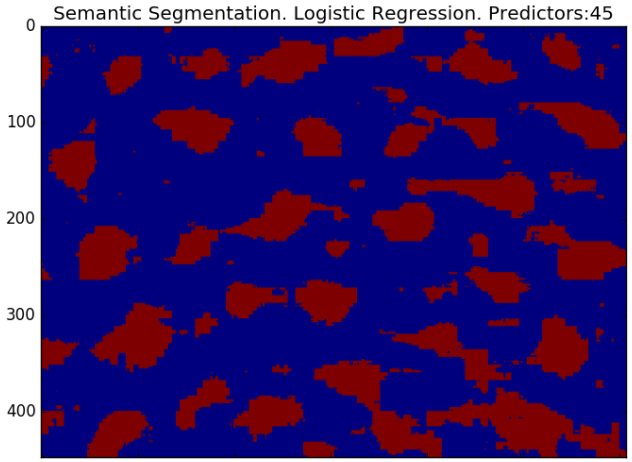

Figure 7.35 *Semantic Segmentation based on Grayscale and Gaussian Pyramid predictors.*

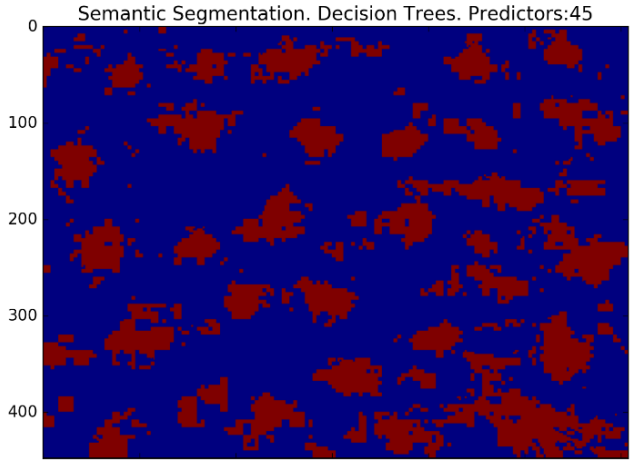

Figure 7.36 *Semantic Segmentation based on Grayscale and Gaussian Pyramid predictors.*

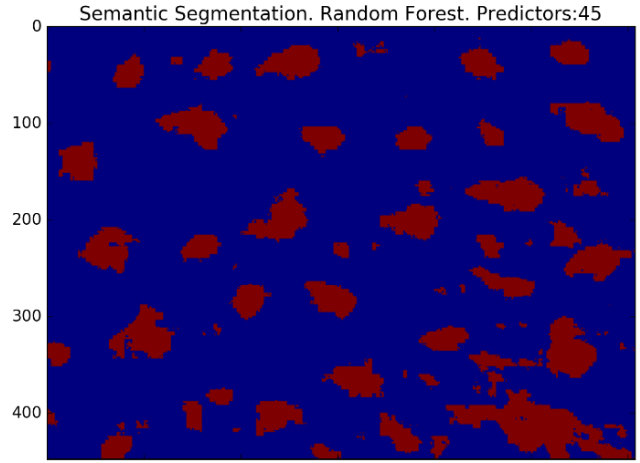

Figure 7.37 *Semantic Segmentation based on Grayscale and Gaussian Pyramid predictors.*

Histogram Equalized + Gaussian Pyramid

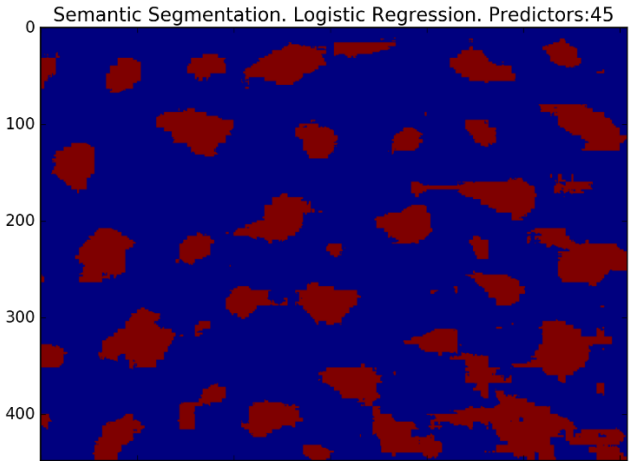

Figure 7.38 *Semantic Segmentation based on Histogram Equalization and Gaussian Pyramid.*

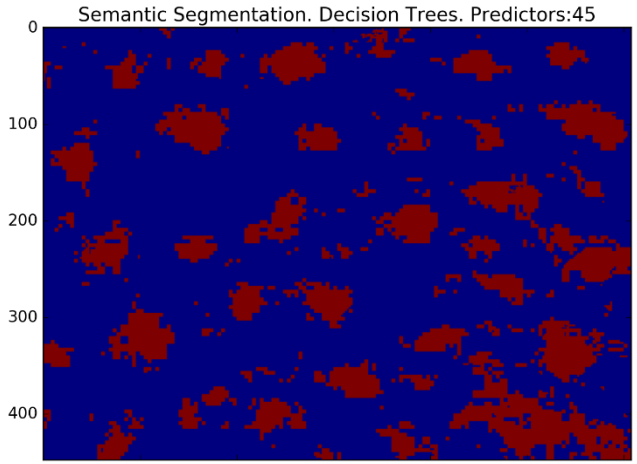

Figure 7.39 *Semantic Segmentation based on Histogram Equalization and Gaussian Pyramid*

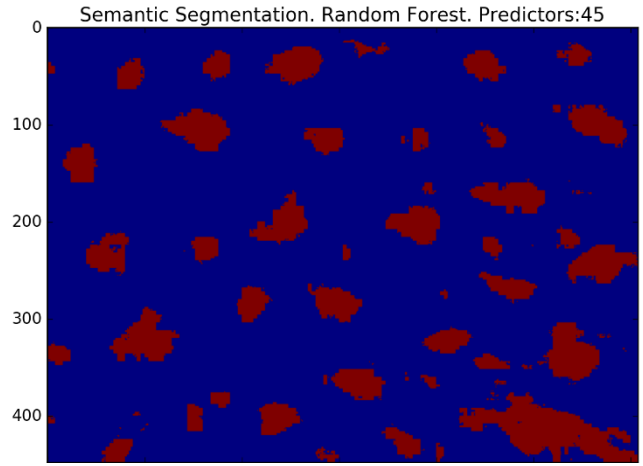

Figure 7.40 *Semantic Segmentation based on Histogram Equalization and Gaussian Pyramid*

Conclusion

Towards the goal of detecting and characterizing morphological skin structures, semantic image segmentation has the potential to quantitatively describe skin at pixel-level. Partitioning automatically an image into coherent parts, where each pixel belongs to a known category that has meaning in human terms is deemed as semantic segmentation or pixel-wise labelling. To achieve semantic image segmentation we build a comprehensive analysis framework, from dataset creation to automatic predictive classification algorithms.

The advantages of semantic segmentation over image processing techniques such as image thresholding, are automatic detection, classification and description of skin features. Semantic segmentation of skin might enable progress in several fields. The first one is digital skin care, where skin can be quantified to track changes for cosmetic purposes. Another field with potential application is that of skin diseases and conditions, where skin melanoma or dermatitis could be diagnosed timely and correctly. Finally, semantic segmentation and description at pixel-level can be employed in surgical navigation through skin context mapping.

The first part of the analysis framework was designing a tool for image annotation. The tool is a web-based application that drives collaborative creation of learning material. Drawing options for annotating several surface skin structures are available. Annotations can be exported offline into individual segmentation maps from which key points are extracted. The key points represent coordinates of regions of interest and they are traced back to the source image. We extract a square pixels' neighborhood around each deemed suitable key point. The square block matrices are then unwrapped into vector format and they become feature descriptors. The features we selected in this work concern pixels' intensities and Gaussian Pyramid. The latter gives context to objects and augments the feature space.

We apply several machine learning algorithms and estimate their predictive performance through several validation metrics. The machine learning classifiers involve Logistic Regression, Decision Trees and Random Forests. The imbalance present in dataset is addressed through class weighting methods and using the correct metric for validation, i.e. precision-recall curve instead of receiver operating characteristic. We employ several groupings of input predictors from features extracted from the RGB channels of an image to different pixels transformations combined with Gaussian Pyramid. We observe that adding a four-level Gaussian Pyramid to the RGB feature space enhances greatly the classification performance, from approximately 81% to 96% correctly classified for the positive class.

Image pre-processing algorithms are discussed as well in this work through grayscale and histogram equalized image transformations. We acknowledge a decrease in the number of input predictors, but also in the predictive power when compared to using raw RGB images. Although the predictive generalization becomes slightly lower than when using RGB predictors, the advantage of using these image transformations is a smaller feature space. This decrease in dimensionality can speed up real-time applications, both through reduced number of predictors and by restricting the search area of key points.

The validation results are confirmed by testing the algorithms on images which were not part of the training set. The output is semantic segmentation. The current classification task presents the segmentation of facial skin pores versus non-pores. On a visual assessment level, the optimal semantic segmentation result is obtained with the Random Forests algorithm which was trained on RGB and Gaussian Pyramid predictors. The final evaluation on the quality of the segmentation is up to experts' who can give more insight into what defines the area of a pore as a skin structure.

List of Figures

Figure 2.1 Pores represent skin surface features or structures. Adopted from [2]. 3
Figure 3.1 Image Thresholding 6
Figure 3.2 IAT requires a responsive website that can scale and interact with different devices. 10
Figure 3.3 Programming Languages and File Formats used in IAT 11
Figure 3.4 The Raphaël Sketchpad library implements viewers and editors. 13
Figure 3.5 Layer structure of IAT. 15
Figure 3.6 Workflow diagram for saving and exporting the annotation 16
Figure 3.7 Exporting the JSON annotations from IAT to text file.. 17
Figure 3.8 The IAT user interface immediately after being opened in the browser. 18
Figure 3.9 The loading and exporting menus from the navigation header. 18
Figure 3.10 IAT user interface after an image has been loaded 18
Figure 4.1 Workflow for Feature Engineering. 20
Figure 4.2 Device system EH900u used for acquiring skin images. 20
Figure 4.3 Left side: a 3x3 structuring square element or kernel. 21
Figure 4.4 Workflow for obtaining the positive and negative classes 22
Figure 4.5 Histogram distribution of a sample image. 23
Figure 4.6 Normal distribution. G(x) represent the Gaussian Function and x is the value of a pixel 24
Figure 4.7 Visual representation of an image pyramid with 5 levels. 25
Figure 4.8 A 5x5 convolution matrix used for the Gaussian Filter. Source: [20] 25
Figure 4.9 The Gaussian Pyramid. 26
Figure 4.10 From contours coordinates to feature vector 27
Figure 5.1 Machine learning methods explored in this work. 28
Figure 5.2 Machine Learning Principle. 28
Figure 5.3 Evaluating the predictive power of a machine learning mode through validation. 30
Figure 5.4 Example cross-validation procedure with 5 folds. 30
Figure 5.5 From data samples to principal components. 32
Figure 5.6 Logistic Function g(z).Crosses the origin at 0.5 and is bounded between 0 and 1. 34
Figure 5.7 Decision Tree example with binary classes Red and Green.. 34
Figure 5.8 CART decision tree example (adapted from [41]). 35
Figure 6.1 Confusion Matrix. 40
Figure 6.2 Receiver Operating Characteristic adopted from [67] 42
Figure 6.3 Precision-Recall example plot with computed AUC. 43

Figure 6.4 Precision-Recall curves for different data distributions..... 44
Figure 6.5 Precision-Recall curves for different data distributions..... 44
Figure 7.1 Semantic Segmentation workflow. 46
Figure 7.2 Confusion Matrix for Logistic Regression..... 48
Figure 7.3 Confusion Matrix for Logistic Regression..... 48
Figure 7.4 Receiver Operating Characteristic curves for Logistic Regression 49
Figure 7.5 Precision Recall curves for Logistic Regression. 49
Figure 7.6 Confusion Matrix for Decision Tree. 50
Figure 7.7 Confusion Matrix for Decision Tree 50
Figure 7.8 Confusion Matrix for Decision Tree 51
Figure 7.9 Precision Recall curve for Decision Trees. Curves drawn for cross-validation with 5 folds..... 51
Figure 7.10 Confusion matrix for Random Forests 52
Figure 7.11 Precision-Recall curve for Random Forests..... 52
Figure 7.12 Confusion Matrix Logistic Regression. 135 predictors used in classification. 53
Figure 7.13 Precision-Recall curve for Logistic Regression. 135 predictors used in classification. 54
Figure 7.14 Explained variance of 5 principal components for Logistic Regression 54
Figure 7.15 Confusion matrix for Logistic Regression after applying Principal Component Analysis..... 55
Figure 7.16 Precision-Recall curve for Logistic Regression with 5 PCA components. 55
Figure 7.17. Confusion Matrix Decision Trees. 135 Predictors used in classification. 56
Figure 7.18 Out-of-Bag error for Random Forests 57
Figure 7.19 Confusion Matrix for Random Forests 57
Figure 7.20 Precision-Recall curve for Logistic Regression. 135 predictors used in classification. 58
Figure 7.21 Confusion Matrix for Decision Trees. Grayscale images used for input predictors. 59
Figure 7.22 Confusion Matrix for Logistic Regression. Grayscale images used for input predictors..... 59
Figure 7.23 Confusion Matrix for Random Forests. Grayscale images used for input predictors. 59
Figure 7.24 Example of histogram equalized image 60
Figure 7.25 Confusion Matrix for Logistic Regression. Histogram equalized image used for input. 61
Figure 7.26 Confusion Matrix for Decision Trees. Histogram equalized image used for input predictors. 61
Figure 7.27 Confusion Matrix for Random Forests.Histogram equalized image used for input predictor. 61
Figure 7.28 Sample Test Image with 5 pores marked with red circles on the right side of the picture. ... 62
Figure 7.29 Semantic Segmentation prediction based on RGB predictors. 63
Figure 7.30 Semantic Segmentation prediction based on RGB predictors. 63
Figure 7.31 Semantic Segmentation prediction based on RGB predictors. 63
Figure 7.32 Semantic Segmentation based on RGB and Gaussian Pyramid predictors. 63
Figure 7.33 Semantic Segmentation based on RGB and Gaussian Pyramid predictors. 63
Figure 7.34 Semantic Segmentation based on RGB and Gaussian Pyramid predictors. 63

Figure 7.35 Semantic Segmentation based on Grayscale and Gaussian Pyramid predictors. 64
Figure 7.36 Semantic Segmentation based on Grayscale and Gaussian Pyramid predictors. 64
Figure 7.37 Semantic Segmentation based on Grayscale and Gaussian Pyramid predictors. 64
Figure 7.38 Semantic Segmentation based on Histogram Equalization and Gaussian Pyramid. 64
Figure 7.39 Semantic Segmentation based on Histogram Equalization and Gaussian Pyramid 64
Figure 7.40 Semantic Segmentation based on Histogram Equalization and Gaussian Pyramid 64

List of Tables

Table 1 Functional Requirements described for IAT 8
Table 2 Non Functional requirements of IAT 9
Table 3 Comparison between the main HTML5 technologies used for web drawing 12
Table 4 Main libraries used in building the IAT 14
Table 5 Example of training set used in the present classification task 29
Table 6 Different measures of node impurity used for Decision Trees 36
Table 7 Data distribution 47

Bibliography

[1] E. Proksch, J. M. Brandner, and J.-M. Jensen, 'The skin: an indispensable barrier', *Exp. Dermatol.*, vol. 17, no. 12, pp. 1063–1072, Dec. 2008.

[2] T. Igarashi, K. Nishino, and S. K. Nayar, '1 Why is skin appearance important?', *Found. Trends Comput. Graph. Vis.*, vol. 3, no. 1, pp. 3–3, 2007.

[3] W. J. Cunliffe, D. B. Holland, and A. Jeremy, 'Comedone formation: Etiology, clinical presentation, and treatment', *Clin. Dermatol.*, vol. 22, no. 5, pp. 367–374, Sep. 2004.

[4] 'Quantification of Facial Pores Using Image Analysis', *Cutis*, vol. 84, no. 3, Sep. 2009.

[5] 'LabelMe. The Open annotation tool'. [Online]. Available: http://labelme2.csail.mit.edu/Release3.0/index.php. [Accessed: 25-Mar-2017].

[6] 'LEAR - Image Annotation Tool - Alexander Kläser'. [Online]. Available: https://lear.inrialpes.fr/people/klaeser/software_image_annotation. [Accessed: 25-Mar-2017].

[7] 'ITK-SNAP'. [Online]. Available: http://www.itksnap.org/pmwiki/pmwiki.php. [Accessed: 25-Mar-2017].

[8] 'ISIC Archive'. [Online]. Available: https://isic-archive.com/. [Accessed: 25-Mar-2017].

[9] 'SVG-Edit/svgedit', *GitHub*. [Online]. Available: https://github.com/SVG-Edit/svgedit. [Accessed: 25-Mar-2017].

[10] P. O. Pinheiro and R. Collobert, 'From Image-level to Pixel-level Labeling with Convolutional Networks', *ArXiv14116228 Cs*, Nov. 2014.

[11] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, 'Semantic image segmentation with deep convolutional nets and fully connected crfs', *ArXiv Prepr. ArXiv14127062*, 2014.

[12] A. Esteva *et al.*, 'Dermatologist-level classification of skin cancer with deep neural networks', *Nature*, vol. 542, no. 7639, pp. 115–118, Jan. 2017.

[13] A. Shaiek *et al.*, 'A new tool to quantify the geometrical characteristics of facial skin pores. Changes with age and a making-up procedure in Caucasian women', *Skin Res. Technol. Off. J. Int. Soc. Bioeng. Skin ISBS Int. Soc. Digit. Imaging Skin ISDIS Int. Soc. Skin Imaging ISSI*, vol. 23, no. 2, pp. 249–257, May 2017.

[14] 'Chapter 1: What is Software Architecture?' [Online]. Available: https://msdn.microsoft.com/en-us/library/ee658098.aspx. [Accessed: 24-Nov-2016].

[15] 'Adding vector graphics to the Web - Learn web development | MDN'. [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/HTML/Multimedia_and_embedding/Adding_vector_graphics_to_the_Web. [Accessed: 27-Nov-2016].

[16] '4.11 Scripting — HTML5'. [Online]. Available: https://www.w3.org/TR/2014/REC-html5-20141028/scripting-1.html#the-canvas-element. [Accessed: 27-Nov-2016].

[17] 'Introduction – SVG 1.1 (Second Edition)'. [Online]. Available: https://www.w3.org/TR/SVG/intro.html. [Accessed: 27-Nov-2016].

[18] 'Thoughts on when to use Canvas and SVG – IEBlog'. [Online]. Available: https://blogs.msdn.microsoft.com/ie/2011/04/22/thoughts-on-when-to-use-canvas-and-svg/. [Accessed: 25-Nov-2016].

[19] 'Morphological Dilation and Erosion - MATLAB & Simulink - MathWorks Benelux'. [Online]. Available: https://nl.mathworks.com/help/images/morphological-dilation-and-erosion.html. [Accessed: 13-Feb-2017].

[20] 'Morphological Transformations — OpenCV-Python Tutorials 1 documentation'. [Online]. Available: http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html. [Accessed: 13-Feb-2017].

[21] 'Morphology - Erosion'. [Online]. Available: http://homepages.inf.ed.ac.uk/rbf/HIPR2/erode.htm. [Accessed: 14-Feb-2017].

[22] ‘Morphology - Dilation’. [Online]. Available: http://homepages.inf.ed.ac.uk/rbf/HIPR2/dilate.htm. [Accessed: 14-Feb-2017].
[23] ‘OpenCV: Contours : Getting Started’. [Online]. Available: http://docs.opencv.org/3.2.0/d4/d73/tutorial_py_contours_begin.html. [Accessed: 15-Feb-2017].
[24] ‘Miscellaneous Image Transformations — OpenCV 2.4.13.2 documentation’. [Online]. Available: http://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html?highlight=threshold. [Accessed: 15-Feb-2017].
[25] ‘ŷhat | Image Processing with scikit-image’. [Online]. Available: http://blog.yhat.com/posts/image-processing-with-scikit-image.html. [Accessed: 15-Feb-2017].
[26] ‘Histogram Equalization — OpenCV 2.4.13.2 documentation’. [Online]. Available: http://docs.opencv.org/2.4/doc/tutorials/imgproc/histograms/histogram_equalization/histogram_equalization.html. [Accessed: 15-Feb-2017].
[27] ‘Point Operations - Histogram Equalization’. [Online]. Available: http://homepages.inf.ed.ac.uk/rbf/HIPR2/histeq.htm. [Accessed: 15-Feb-2017].
[28] ‘Histogram Equalization — skimage v0.9.0 docs’. [Online]. Available: http://scikit-image.org/docs/0.9.x/auto_examples/plot_equalize.html. [Accessed: 15-Feb-2017].
[29] ‘Point Operations - Contrast Stretching’. [Online]. Available: http://homepages.inf.ed.ac.uk/rbf/HIPR2/stretch.htm. [Accessed: 15-Feb-2017].
[30] E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt, and J. M. Ogden, ‘Pyramid methods in image processing’, *RCA Eng.*, vol. 29, no. 6, pp. 33–41, 1984.
[31] L. G. Shapiro and G. C. Stockman, *Computer Vision*, 1 edition. Pearson, 2001.
[32] ‘Smoothing Images — OpenCV 2.4.13.2 documentation’. [Online]. Available: http://docs.opencv.org/2.4/doc/tutorials/imgproc/gausian_median_blur_bilateral_filter/gausian_median_blur_bilateral_filter.html. [Accessed: 17-Feb-2017].
[33] R. Rao, ‘Computer Vision: Edge Detection - lect3.pdf’, *Computer Vision CSE 455, Winter 2009*, 02-Jan-2009. [Online]. Available: https://courses.cs.washington.edu/courses/cse455/09wi/Lects/lect3.pdf. [Accessed: 26-Mar-2017].
[34] ‘Image Pyramids — OpenCV 2.4.13.2 documentation’. [Online]. Available: http://docs.opencv.org/2.4/doc/tutorials/imgproc/pyramids/pyramids.html. [Accessed: 13-Feb-2017].
[35] N. Dalal and B. Triggs, ‘Histograms of oriented gradients for human detection’, in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, 2005, vol. 1, pp. 886–893.
[36] ‘Local Feature Detection and Extraction - MATLAB & Simulink - MathWorks Benelux’. [Online]. Available: https://nl.mathworks.com/help/vision/ug/local-feature-detection-and-extraction.html. [Accessed: 02-Mar-2017].
[37] ‘3.1. Cross-validation: evaluating estimator performance — scikit-learn 0.18.1 documentation’. [Online]. Available: http://scikit-learn.org/stable/modules/cross_validation.html. [Accessed: 15-Mar-2017].
[38] ‘Approaching (Almost) Any Machine Learning Problem | Abhishek Thakur’, *No Free Hunch*, 21-Jul-2016. [Online]. Available: http://blog.kaggle.com/2016/07/21/approaching-almost-any-machine-learning-problem-abhishek-thakur/. [Accessed: 26-Mar-2017].
[39] ‘sklearn.model_selection.StratifiedKFold — scikit-learn 0.18.1 documentation’. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html. [Accessed: 15-Mar-2017].
[40] ‘Learning from Imbalanced Classes’, *Silicon Valley Data Science*, 25-Aug-2016. [Online]. Available: http://svds.com/learning-imbalanced-classes/. [Accessed: 22-Nov-2016].
[41] T. Hastie, J. Friedman, and R. Tibshirani, *ELements of statistical learning. Data mining, inference and prediction*. Springer Series in Statistics New York, 2001, 2009.
[42] ‘Principal Component Analysis in 3 Simple Steps’. [Online]. Available: https://plot.ly/ipython-notebooks/principal-component-analysis/. [Accessed: 12-Mar-2017].
[43] M. Ringnér, ‘What is principal component analysis?’, *Nat. Biotechnol.*, vol. 26, no. 3, pp. 303–304, Mar. 2008.

[44] ‘Principal component analysis - Wikipedia’. [Online]. Available: https://en.wikipedia.org/wiki/Principal_component_analysis. [Accessed: 12-Mar-2017].
[45] N. Halko, P.-G. Martinsson, and J. A. Tropp, ‘Finding structure with randomness: Stochastic algorithms for constructing approximate matrix decompositions’, 2009.
[46] ‘Implementing a Principal Component Analysis (PCA)’. [Online]. Available: http://sebastianraschka.com/Articles/2014_pca_step_by_step.html. [Accessed: 13-Mar-2017].
[47] A. A. Miranda, Y.-A¨el Le Borgne, and G. Bontempi, ‘New Routes from Minimal Approximation Error to Principal Components’, *Kluwer Acad. Publ.*, Sep. 2007.
[48] ‘1.1. Generalized Linear Models — scikit-learn 0.18.1 documentation’. [Online]. Available: http://scikit-learn.org/stable/modules/linear_model.html#logistic-regression. [Accessed: 13-Mar-2017].
[49] A. Ng, ‘CS229 Lecture notes’, *CS229 Lect. Notes*, vol. 1, no. 1, pp. 1–3, 2000.
[50] R. J. Lewis, ‘An introduction to classification and regression tree (CART) analysis’, *Available Accessed April*, vol. 27, p. 2008, 2008.
[51] ‘1.10. Decision Trees — scikit-learn 0.18.1 documentation’. [Online]. Available: http://scikit-learn.org/stable/modules/tree.html. [Accessed: 13-Mar-2017].
[52] ‘Decision Trees - MATLAB & Simulink - MathWorks Benelux’. [Online]. Available: http://nl.mathworks.com/help/stats/classification-trees-and-regression-trees.html. [Accessed: 14-Mar-2017].
[53] ‘What is cross-entropy, and why use it?’ [Online]. Available: http://www.cse.unsw.edu.au/~billw/cs9444/crossentropy.html. [Accessed: 14-Mar-2017].
[54] ‘Cross Entropy’. [Online]. Available: http://heliosphan.org/cross-entropy.html. [Accessed: 14-Mar-2017].
[55] ‘Classification error by resubstitution - MATLAB - MathWorks Benelux’. [Online]. Available: http://nl.mathworks.com/help/stats/classificationtree.resubloss.html. [Accessed: 14-Mar-2017].
[56] L. Breiman, ‘Random forests’, *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.
[57] ‘Random forests - classification description’. [Online]. Available: http://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm. [Accessed: 13-Mar-2017].
[58] ‘ŷhat | Random Forests in Python’, *ŷhat | Blog*. [Online]. Available: http://blog.yhat.com/posts/random-forests-in-python.html. [Accessed: 13-Mar-2017].
[59] ‘1.11. Ensemble methods — scikit-learn 0.18.1 documentation’. [Online]. Available: http://scikit-learn.org/stable/modules/ensemble.html. [Accessed: 13-Mar-2017].
[60] ‘3.2.4.3.1. sklearn.ensemble.RandomForestClassifier — scikit-learn 0.18.1 documentation’. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html. [Accessed: 13-Mar-2017].
[61] T. Fawcett, ‘An introduction to ROC analysis’, *Pattern Recognit. Lett.*, vol. 27, no. 8, pp. 861–874, Jun. 2006.
[62] ‘sklearn.metrics.confusion_matrix — scikit-learn 0.18.1 documentation’. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html. [Accessed: 15-Mar-2017].
[63] ‘Performance Curves - MATLAB & Simulink’. [Online]. Available: http://www.mathworks.com/help/stats/performance-curves.html. [Accessed: 27-Jun-2014].
[64] ‘The Basics of Classifier Evaluation, Part 2’, *Silicon Valley Data Science*, 10-Dec-2015. [Online]. Available: https://www.svds.com/classifiers2/. [Accessed: 15-Mar-2017].
[65] ‘Receiver operating characteristic (ROC) curve or other performance curve for classifier output - MATLAB perfcurve - MathWorks Benelux’. [Online]. Available: http://nl.mathworks.com/help/stats/perfcurve.html. [Accessed: 15-Mar-2017].
[66] ‘Receiver Operating Characteristics’. [Online]. Available: http://www0.cs.ucl.ac.uk/staff/ucacbbl/roc/. [Accessed: 27-Jun-2014].
[67] ‘Receiver Operating Characteristic (ROC) — scikit-learn 0.18.1 documentation’. [Online]. Available: http://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html. [Accessed: 15-Mar-2017].

[68] T. Fawcett, 'An introduction to ROC analysis', *Pattern Recognit. Lett.*, vol. 27, no. 8, pp. 861–874, 2006.
[69] 'sklearn.metrics.auc — scikit-learn 0.18.1 documentation'. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.metrics.auc.html. [Accessed: 15-Mar-2017].
[70] 'Precision-Recall — scikit-learn 0.18.1 documentation'. [Online]. Available: http://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html. [Accessed: 15-Mar-2017].
[71] 'The Berkeley Segmentation Dataset and Benchmark'. [Online]. Available: https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/. [Accessed: 22-Nov-2016].
[72] 'Introduction to the precision-recall plot', *Classifier evaluation with imbalanced datasets*, 09-Jun-2015. .
[73] T. Saito and M. Rehmsmeier, 'The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets', *PloS One*, vol. 10, no. 3, p. e0118432, 2015.
[74] 'OOB Errors for Random Forests — scikit-learn 0.18.1 documentation'. [Online]. Available: http://scikit-learn.org/stable/auto_examples/ensemble/plot_ensemble_oob.html. [Accessed: 23-Mar-2017].