České vysoké učení technické v Praze
Fakulta jaderná a fyzikálně inženýrská

Katedra softwarového inženýrství
Studijní program: Aplikace informatiky v přírodních vědách

Texturní analýza 3D obrazů

Texture-Based Analysis of 3D Images

DIPLOMOVÁ PRÁCE

Vypracoval:	Bc. Matěj Pokorný
Vedoucí práce:	doc. Ing. Jaromír Kukal, Ph.D.
Rok:	2023

České vysoké učení technické v Praze
Fakulta jaderná a fyzikálně inženýrská

Katedra softwarového inženýrství   Akademický rok 2022/2023

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Bc. Matěj Pokorný

Studijní program: Aplikace informatiky v přírodních vědách

Název práce česky: Texturní analýza 3D obrazů

Název práce anglicky: Texture-Based Analysis of 3D Images

Jazyk práce: angličtina

Pokyny pro vypracování:

1. Seznamte se s tématem textur obrazů, jejich vlastností a rozšířením tohoto tématu do tří dimenzí.

2. Seznamte se s možnostmi dekompozice a segmentace 3D obrazů pomocí zpracování textur.

3. Seznamte se s tématem analýzy 3D obrazů založené na extrakci a vyhodnocování strukturálních vlastností textur.

4. V prostředí MATLAB realizujte knihovnu funkcí určených pro analýzu a vyhodnocování vlastností textur 3D obrazů.

5. Otestujte implementovanou knihovnu funkcí provedením experimentů na 3D obrazech a to zejména na 3D SPECT snímcích lidského mozku.

Doporučená literatura:

[1] GONZALES, R. C., WOODS, R. E. *Digital Image Processing*. 4th ed. New York, NY: Pearson, 2018. ISBN 978-0133356724.

[2] IP, H. H. S., LAM, S. W. C. Three-Dimensional Structural Texture Modeling and Segmentation. In: *Pattern Recognition*., Vol. 28, Iss. 9, 1995, pp. 1299–1319.

[3] BOZKURT, F., KÖSE, C., SARI, A. A Texture-Based 3D Region Growing Approach for Segmentation of ICA through the Skull Base in CTA. In: *Multimedia Tools and Applications*., Vol. 79, 2020, pp. 33253–33278.

[4] HUMEAU-HEURTIER, A. Texture Feature Extraction Methods: A Survey. In: *IEEE Access*., Vol. 7, 2019, pp. 8975-9000.

[5] BARBURICEANU, S., TEREBES, R., MEZA, S. 3D Texture Feature Extraction and Classification Using GLCM and LBP-Based Descriptors. In: *Applied Sciences*., Vol. 11, No. 5, 2021, pp. 2332-2357.

[6] KOVALEV, V. A., KRUGGEL, F., GERTZ, H.-J., VON CRAMON, D. Y. Three-Dimensional Texture Analysis of MRI Brain Datasets. In: *IEEE Transactions on Medical Imaging*., Vol. 20, No. 5, 2001, pp. 424-433.

[7] TESAŘ, L., SHIMIZU, A., SMUTEK, D., KOBATAKE, H., NAWANO, S. Medical Image Analysis of 3D CT Images Based on Extension of Haralick Texture Features. In: *Computerized Medical Imaging and Graphics*., Vol. 32, Iss. 6, 2008, pp. 513-520.

Jméno a pracoviště vedoucího práce:

doc. Ing. Jaromír Kukal, Ph.D.

Katedra softwarového inženýrství, Fakulta jaderná a fyzikálně inženýrská, ČVUT v Praze

....
vedoucí práce

Datum zadání diplomové práce: 14. 10. 2022

Termín odevzdání diplomové práce: 3. 5. 2023

Doba platnosti zadání je dva roky od data zadání.

....
garant oboru

....
vedoucí katedry

....
děkan

V Praze dne 14. 10. 2022

Statement of Originality

I hereby declare that I have developed this thesis independently and that, to the best of my knowledge, the content of this thesis is my own work based on research conducted between November 2022 and April 2023. I certify that I have used only the sources (literature, software, etc.) listed in the attached list of references and that I have acknowledged all the assistance received while preparing this thesis. This thesis has not been submitted for any other purpose other than as a master's thesis at the Faculty of Nuclear Sciences and Physical Engineering, Czech Technical University in Prague.

In Prague 3.5.2023

Bc. Matěj Pokorný

Acknowledgements

First and foremost, I would like to thank my thesis supervisor, doc. Ing. Jaromír Kukal, Ph.D., for his many insightful suggestions and invaluable advice, which significantly contributed to the improvement of both the form and content of this thesis. I would also like to express my deepest gratitude to my family for their support, which made possible my studies at the university. Furthermore, I would like to acknowledge and underscore the considerable and unpayable debt I feel I owe to my favourite artists and scientists for creating works that have entertained, educated, and motivated me during the whole length of my studies. I would especially like to thank all the lecturers in the Department of Software Engineering, whose steadfast leadership and stellar example have elucidated to me a great many things previously unknown. Finally, I proclaim my most heartfelt and eternal thanks to all my girlfriends, friends, acquaintances, and fellow students at the Faculty of Nuclear Sciences and Physical Engineering, whose company has created many beautiful moments and ensured that I will remember this period of my life with fondness.

Bc. Matěj Pokorný

Abstrakt: Texturní analýza obrazů je oblastí s četnými možnostmi využití, např. při analýze 3D snímků v medicíně. Většina autorů se ovšem dosud primárně zabývala pouze texturní analýzou 2D obrazů. Texturní analýza zaměřená specificky na 3D obrazy je, díky četným možnostem využítí, nyní velmi aktuálním a rychle se rozvíjejícím odvětvím. V první části této diplomové práce jsou představeny teoretické koncepty přístupu k texturní analýze, zaměřené již od počátku speciálně na 3D obrazy. Nejprve je diskutováno téma rychlé digitální filtrace obrazu, které je následně rozvedeno a využito k výpočtu rotačně invariatních, globálních vlastností textur 3D obrazů, založených na Fourierově transformaci Zernikeho polynomů. Na teoretickou část navazuje implementace v prostředí MATLAB, která umožní vypočtené vlastnosti obrazů využít ke klasifikaci SPECT snímků mozků pacientů s Alzheimerovou chorobou. Pomocí tohoto přístupu je možné dosáhnout vysoké přesnosti klasifikace pacientů, což tuto metodu činí vhodnou k využití při diagnostice v medicíně.
Klíčová slova: texturní analýza 3D obrazu, klasifikace obrazu, Zernikeho polynom, rotační invariantnost, Fourierova transformace

Abstract: Texture-based image analysis is a field with numerous applications, e.g., in the analysis of 3D medical images. In the past, however, most authors were primarily concerned with the texture-based analysis of 2D images. Texture analysis specifically focused on 3D images is, thanks to its numerous applications, now a state-of-the-art, rapidly developing discipline. In the first part of this master's thesis, theoretical concepts of an approach to texture analysis specifically tailored to 3D images are presented. First, the topic of fast digital image filtering is discussed, which is then developed and used to compute rotationally invariant global texture characteristics of 3D images based on the Fourier transform of Zernike polynomials. The theoretical part is followed by a MATLAB implementation that allows the computed image features to be used for the classification of SPECT images of the brains of patients suffering from Alzheimer's disease. Using this approach, it is possible to achieve high accuracy in patient classification, which makes this method suitable for use in medical diagnostics.
Keywords: 3D Image Texture Analysis, Image Classification, Zernike Polynomial, Rotational Invariance, Fourier Transform

Contents

List of Symbols

$\exp(x)$	exponential function e^x
$\mathcal{B}$	unit n-dimensional ball
$\mathcal{B}_0$	unit n-dimensional sphere
$\mathcal{P}$	unit n-dimensional ball in spherical coordinates
$\mathcal{P}_0$	unit n-dimensional sphere in spherical coordinates
F_1	F_1 score
$\mathbf{0}$	zero vector
i	imaginary unit
$*$	n-dimensional convolution
$\circledast$	n-dimensional correlation
$\Gamma(z)$	gamma function
$\lceil \cdot \rceil$	rounding real number to the nearest larger integer
$\lfloor \cdot \rceil$	rounding real number to the nearest integer
$\mathbb{C}$	set of complex numbers
$\mathbb{N}$	set of natural numbers
$\mathbb{R}$	set of real numbers
$\mathbb{R}^+$	interval $(0, \infty)$
$\mathbb{R}_0^+$	$\mathbb{R}^+ \cup \{0\}$
$\mathbb{Z}$	set of integers
$\mathcal{S}$	support set of image
$\det(\boldsymbol{X})$	determinant of matrix $\boldsymbol{X}$
$\mathrm{f}(\cdot)$	n-dimensional image
$\mathrm{I}_A(x)$	indicator function
Q_1	first quartile
Q_3	third quartile
$\mathscr{F}_n$	n-dimensional Fourier transform
$\mathscr{F}_n^{-1}$	n-dimensional inverse Fourier transform
$\boldsymbol{\Sigma}$	covariance matrix
$\boldsymbol{I}$	identity matrix
$\boldsymbol{J}_{m,n}$	matrix of ones of size $m \times n$
$\boldsymbol{X}$	matrix
$\boldsymbol{X}^{\mathrm{T}}$	transpose of matrix $\boldsymbol{X}$
$\oplus$	element-wise image addition
$\otimes$	element-wise image multiplication
$\bar{z}$	complex conjugate
$\prod$	product
$\mathrm{rank}(\boldsymbol{X})$	rank of matrix $\boldsymbol{X}$

σ	standard deviation
σ^2	variance
$\sum$	sum
$\lVert \boldsymbol{x} \rVert_p$	p-norm l_p
$\boldsymbol{v}$	vector
$J_\alpha(x)$	Bessel function of the first kind and order α
L^2	space of square integrable functions
$O(g(n))$	function $\lvert f(n) \rvert$ is bounded above by $g(n)$ asymptotically
$P_l^m(x)$	Legendre associated polynomial
$P_n^{[\alpha,\beta]}(x)$	Jacobi polynomial
$R_{l,n}(x)$	radial polynomial
se^*	critical sensitivity
$Y_l^m(\theta,\phi)$	spherical harmonic function
$Z_{l,n}^m(\boldsymbol{r})$	three-dimensional Zernike polynomial

List of Abbreviations

ACC	Accuracy
AD	Alzheimer's Disease
ANN	Artificial Neural Network
BACC	Balanced Accuracy
BCT	Box-Cox Transformation
CDF	Cumulative Distribution Function
CN	Cognitively Normal
CV	Cross-Validation
DFT	Discrete Fourier Transform
DOR	Diagnostic Odds Ratio
FDR	False Discovery Rate
FFT	Fast Fourier Transform
FM	Fowlkes-Mallows Index
FN	False Negative
FNR	False Negative Rate
FOR	False Omission Rate
FP	False Positive
FPR	False Positive Rate
FT	Fourier Transform
FWER	Family-Wise Error Rate
GLCM	Grey-Level Co-Occurrence Matrix
HP	High-Pass
IDFT	Inverse Discrete Fourier Transform
IQR	Inter-Quartile Range
KNN	k-Nearest Neighbours
LDA	Linear Discriminant Analysis
LP	Low-Pass
LR+	Positive Likelihood Ratio
LR−	Negative Likelihood Ratio
LT	Log Transformation
MAD	Median Absolute Deviation
MCC	Matthews Correlation Coefficient
NPV	Negative Predictive Value
PCA	Principal Component Analysis
PD	Positive Definite
PDF	Probability Distribution Function

PPV	Positive Predictive Value
PRT	Prevalence Threshold
PT	Power Transformation
QDA	Quadratic Discriminant Analysis
SPECT	Single-Photon Emission Computed Tomography
SVM	Support Vector Machine
TN	True Negative
TNR	True Negative Rate
TP	True Positive
TPR	True Positive Rate
TS	Threat Score

Introduction

Image processing is a varied and rapidly developing field with many practical uses, ranging from medical imaging and self-driving vehicles to processing data from space telescopes. We conceive of this work as an investigation into mid-level image processing methods, namely texture-based analysis and image classification. We consider the main contribution of this work to be the fact that we primarily make inquiries into texture-based analysis and classification of three-dimensional images, a modern but still relatively unexplored discipline. In this work, we introduce fast methods of producing rotationally invariant characteristics of three-dimensional images through the usage of Zernike polynomials and a fast Fourier transform (FFT). In the experimental part, we will show that even with the employment of not particularly complex global statistical characteristics, we are able to produce good image classification results for single-photon emission computed tomography (SPECT) brain imaging data of cognitively normal (CN) patients and patients suffering from Alzheimer's disease (AD), which makes our method and results potentially of interest in the field of medical diagnosis.

In Chapter 1, we introduce and briefly describe the most important aspects of three-dimensional image processing. We start the chapter with a definition of an n-dimensional greyscale image. We mention the various levels of image processing, elementary image operations, and define image noise and some of its types. We continue this chapter by defining the meaning of low-pass and high-pass image filtering and introducing the basics of global filtering in the frequency domain utilizing a fast Fourier transform (FFT). We then move onto higher-level image processing - texture-based image analysis. In this section, we introduce several popular approaches to image texture analysis. Finally, we discuss the topic of image classification. Some concepts mentioned in this introductory chapter will be elaborated on in the later chapters of this work.

Chapter 2 is concerned with descriptions of fast global filtering methods in the frequency domain. We discuss the construction of several types of low-pass filtering kernels including the binary, Gaussian, Butterworth, and α-stable kernels. We also define a simple manner in which we may utilize low-pass filtering kernels for high-pass filtering. We end this chapter with definitions of several nonlinear data transformations, which we may use as an intermediary step while filtering images in the frequency domain.

In Chapter 3, we move onto the higher-level image processing technique of texture analysis. In this chapter, we introduce a manner of constructing three-dimensional rotational invariants through the usage of FFT of Zernike polynomials. We will also define the global statistical characteristics that will act as textural image features and input data for classifiers in the experimental part of this work.

Chapter 4 describes the process of image classification. In the beginning of this chapter, we will describe a way to test the data by using the Mann-Whitney U test, the Liliefors test, and the Welch's t-test to determine which, if any, features exhibit the largest potential for dataset separation. In the rest of the chapter, we will be primarily concerned with transforming and whitening the data to serve as inputs for several types of binary classifiers, which will be used to decide whether a SPECT brain image belongs to a patient suffering from Alzheimer's disease or to a healthy, cognitively normal patient. To this end, we will use the linear discriminant analysis (LDA), quadratic discriminant analysis (QDA), k-nearest neighbours (KNN), as well as the support vector machines (SVMs) of the polynomial and Gaussian variety, and the artificial neural networks (ANNs).

Chapter 5 is concerned primarily with the explanation of certain crucial details of our image processing library, which will be implemented in MATLAB. We make comments on defined classes and explain the purpose of functions used in our implementation. We also describe the incorporation of already prepared software, such as the MATLAB Statistics and Machine Learning Toolbox™ implementation of ANNs and SVMs, as well as the incorporation of community-created functions for easier saving of MATLAB plots containing obtained results.

Chapter 6 presents results of image processing experiments obtained on a dataset containing SPECT images of patients suffering from Alzheimer's disease and patients with cognitively normal brains. We first discuss the results of statistical testing for data separation. The second and largest part of this chapter is dedicated to a discussion and analysis of image classification results. We will show that even while using relatively non-complex statistical characteristics, we are able to obtain good binary classification results while using all mentioned types of classifiers, which makes our method potentially of interest in the field of medical diagnosis.

We conclude this work with a brief summary of the theoretical approaches we decided to employ, a recapitulation of the obtained results, and a discussion of suggested future improvements, and extensions, as well as possible follow-up projects.

Chapter 1

Image Processing

In this introductory chapter, we present and briefly examine some of the principal elements and concepts of image processing, which include basic image operations in spatial and frequency domains, image filtering, texture-based image analysis, and image classification. These theoretical concepts are indispensable as they become the foundation upon which lie the practical implementation and applications found in chapters 5 and 6 of this work. Most of the concepts related to fast global image filtering, texture-based image analysis, and image classification, and hence to the main topic of this work, are also described in more depth in chapters 2, 3 and 4.

1.1 n-Dimensional Greyscale Image

Let $n \in \mathbb{N}$ be the image dimension and $\boldsymbol{r} \in \mathbb{N}^n$ be the image range vector. A set $\mathcal{S} = \{\boldsymbol{i} \,|\, \boldsymbol{i} \in \mathbb{Z}^n, \, \boldsymbol{0} \leq \boldsymbol{i} < \boldsymbol{r}\}$ of coordinate vectors is called the image support set. An n-dimensional *greyscale image* is a function f: $\mathcal{S} \to \mathbb{R}$. The elements of the set $\mathcal{S}$ are called *pixels (picture elements)* in the case of a two-dimensional image and *voxels (volume elements)* in the case of a three-dimensional image. When not specifically concerned with the number of dimensions n, we will also refer to the coordinate vector $\boldsymbol{i}$ as *image element*. We define the signal intensity of an image element as $f(\boldsymbol{i}) = f[(i_1, \ldots, i_n)]$ or $f(\boldsymbol{i}) = f(i_1, \ldots, i_n)$ using a simplified notation.

1.2 Image Processing and Enhancement

Image processing may be defined as the bulk of various procedures that take an image as their input and, by performing certain predefined operations, arrive at a result. We can classify these procedures as *low-, mid-* and *high-level processes*. Low-level processes take an image as their input and arrive at a result that is itself also an image. Popular low-level processes frequently include use of the *image filtering*, i.e. algorithms aimed at reducing the level of *noise* present within the image. Mid-level processes include image texture analysis, segmentation/partitioning and image recognition/classification. Mid-level processes therefore take an image as an input and

generally give a result that is itself not an image but rather a bundle of extracted features. Finally, high-level processes perform tasks otherwise generally reserved for the human vision, such as depth analysis and making sense of the groups of recognized objects within the image [1].

The term *image enhancement* describes an act of processing an image that facilitates using said image for a problem specific application. It should therefore be obvious that image enhancement is always a goal specific process [1]. It can, for example, be inferred that enhancing a single-photon emission computed tomography (SPECT) image is a task vastly different from enhancing a result of aerial/satellite imaging.

1.3 Spatial and Frequency Domains

In 1.1, we have defined an n-dimensional image as a function, which represents the image in the spatial domain. It is, however, also possible to represent the image in the *frequency domain*. Calculating the discrete Fourier transform (usually a fast Fourier transform) of the image, we obtain its frequency-domain representation. This can be used for *global image filtering* and even texture-based analysis, as we will see in chapters 2 and 3.

1.4 Elementary Image Operations

As we mentioned in the previous section, image processing consists of performing various procedures on an image. In terms of low-level processing in the spatial domain, these procedures or algorithms can oftentimes be decomposed into several *elementary image operations*. Remembering the fact that images are represented by a function, we define the element-wise sum of two images $f(\cdot)$ and $g(\cdot)$ as the operation $\oplus$:

$$f(\cdot) \oplus g(\cdot) = f(\boldsymbol{i}) + g(\boldsymbol{i}),\ \forall \boldsymbol{i} \in \mathcal{S} \tag{1.1}$$

We define the product of two images $f(\cdot)$ and $g(\cdot)$ as the operation $\otimes$:

$$f(\cdot) \otimes g(\cdot) = f(\boldsymbol{i}) \cdot g(\boldsymbol{i}),\ \forall \boldsymbol{i} \in \mathcal{S} \tag{1.2}$$

In the scope of this work, we will generally use element-wise operations (especially when concerned with fast global image filtering) unless stated otherwise.

1.5 Noise

Provided that we could transfer a perfect captured image of a scene, we would obtain a *noiseless ideal image*. Unfortunately, due to limitations in equipment quality and extraneous environmental influences this is not possible and any captured image will therefore feature some level of *noise*, i.e. elements that are not present in the original scene itself and that negatively impact the quality of the image by random variation of intensity levels. Assuming simple element-wise additive noise, the corrupted image g($\cdot$) is the result of combining the ideal noiseless image f($\cdot$) and noise $\eta(\cdot)$ [1]:

$$g(\cdot) = f(\cdot) \oplus \eta(\cdot),$$

where we assume that $\rho_{\eta,\eta} = \boldsymbol{I}$, $\mu(\eta) = 0$, i.e. the correlation matrix of η is the identity matrix and the mean of η is equal to zero. The behaviour and features of each type of noise are defined by its probability distribution function (PDF).

1.5.1 Gaussian Noise

Gaussian noise is a frequently employed noise model because of its easiness of use in both the spatial and frequency domains. It is defined by the well-known Gaussian PDF:

$$p(u) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(u-\mu)^2}{2\sigma^2}\right) \quad -\infty < u < \infty, \tag{1.3}$$

where u represents the intensity, μ the mean value and σ the standard deviation. The tractability of Gaussian noise is, however, often balanced out by its low level of applicability for a large portion of real-life problems [1].

1.5.2 Salt-and-Pepper Noise

The *salt-and-pepper* or *impulse* noise receives its name after the salt-and-pepper granules or specks of generally maximum or minimum possible intensity it distributes within the corrupted image. Assuming that the original image contains integer intensity values u in the range $u \in \{0, 1, \ldots, 2^m - 1\}$ where $m \in \mathbb{N}$, this type of noise is defined by the PDF [1]:

$$p(u) = \begin{cases} P_{\text{high}} & \text{for } u = 2^m - 1 \\ P_{\text{low}} & \text{for } u = 0 \\ 1 - (P_{\text{high}} + P_{\text{low}}) & \text{for } u \in \{1, 2, \ldots, 2^m - 2\} \end{cases} \tag{1.4}$$

We can image the situation as following: the ideal image $f(\cdot)$ is corrupted by the impulse noise by observing the corresponding random noise image $\eta(\cdot)$ of a same size and assigning the value $f(\boldsymbol{i}) = 2^m - 1, \forall \boldsymbol{i}: \eta(\boldsymbol{i}) = 2^m - 1$. This process is done analogously for the value zero. Values within the range $\{1, 2, \ldots, 2^m - 2\}$ remain unchanged. Thus, we can see that the resulting corrupted image will feature specks of maximum or minimum intensity distributed according to the salt-and-pepper PDF.

1.6   Image Filtering

In this section, we will introduce two approaches to image filtering, which is itself a low-level procedure aimed at reducing the noise level within an image. These two paradigms are the local filtering, which utilizes the spatial representation of images, and the global filtering, which works with the frequency representation.

It is also important to note that we can differentiate between several types of filtering depending on what is our final goal with the image. If we wish to reduce sharp transitions within the image, we employ so called *low-pass filtering* (LP). This approach is most useful when dealing with the noise within images, as it tends to manifest as sharp transitions in intensity. For this reason, the low-pass image filtering, also called smoothing, will become one of the topics discussed in the later chapters of this work. If, however, we wish to highlight sharp transitions (changes in texture) within the image, we use the *high-pass filtering* (HP) approach. Combining these two approaches we can create either *band-pass* or *band-reject* filters, i.e. filters that either accentuate or understate intensities within a certain range.

Using the information from 1.4, we can also say that we are dealing with either linear or nonlinear image filtering methods. Spatial linear image filtering such as computing the (weighted) arithmetic average is perhaps the simplest way of implementing a low-pass filter. To add more variability to our library of image processing functions, we will also add an option to transform the image using a nonlinear transformation (see Chapter 2).

1.6.1   Local Filtering

Local filtering exploits the spatial representation of an image. Remembering the definition in 1.1, it follows naturally that local filtering constructs for each element of the image a *element neighbourhood* of predefined size and shape, whereupon a certain *mask* also known as a *kernel* can be used to modify the intensity $f(\boldsymbol{i})$ using the intensities of its neighbours. The process of moving the centre of a kernel across an image is called a *convolution*. This process can, however, be rather time-inefficient for large two-dimensional and most three-dimensional images. In this section, we will thus provide only definitions that are also useful for defining the fast global filtering.

Discrete Convolution and Correlation

Convolution and correlation are the names of fundamentally similar operations that describe the movement of a mask across an image. Let $f(\cdot)$ be an n-dimensional function (not necessarily an image) and $h(\cdot)$ be an n-dimensional kernel function. We define the n-dimensional convolution $*$ as:

$$(h * f)(\boldsymbol{i}) = \sum_{\boldsymbol{k} \in \mathbb{Z}^n} h(\boldsymbol{k}) f(\boldsymbol{i} - \boldsymbol{k}) \tag{1.5}$$

We define the n-dimensional correlation $\circledast$ as:

$$(h \circledast f)(\boldsymbol{i}) = \sum_{\boldsymbol{k} \in \mathbb{Z}^n} \overline{h(\boldsymbol{k})} f(\boldsymbol{i} + \boldsymbol{k}) \tag{1.6}$$

The convolution is commutative, associative, and distributive, while correlation upholds only the property of distributivity. From both of these definitions, where the sum indices are technically infinite, we can see that for convolving images with kernels we first have to restrict indices' values to be appropriately bounded and finite. It is also obvious that without a special preparation, it would be impossible to convolve the mask with the elements close to the rim of the image. We call this preparation *padding*.

Padding

As defined in the previous sections, padding is a preprocessing operation that enables full movement of the centre of a mask across all elements of an image. We will define three kinds of padding - the *zero*, *copy* and *periodic padding*. The zero padding is the most basic of the padding methods, as it consists of appending rows and columns of zeros to all sides of the image. The copy padding appends values equal to values of the nearest image element. The periodic padding appends values obtained by mirror reflecting the image across its border. Illustrating the concept on a matrix $\boldsymbol{M}$:

$$\boldsymbol{M} = \begin{bmatrix} m_{1,1} & m_{1,2} & m_{1,3} \\ m_{2,1} & m_{2,2} & m_{2,3} \\ m_{3,1} & m_{3,2} & m_{3,3} \end{bmatrix}$$

Assuming a kernel of size 3×3, the results of zero, copy and periodic padding methods respectively would then be matrices of size 5×5:

$$\boldsymbol{M}_{\mathrm{zero}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & m_{1,1} & m_{1,2} & m_{1,3} & 0 \\ 0 & m_{2,1} & m_{2,2} & m_{2,3} & 0 \\ 0 & m_{3,1} & m_{3,2} & m_{3,3} & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\boldsymbol{M}_{\text{copy}} = \begin{bmatrix} m_{1,1} & m_{1,1} & m_{1,2} & m_{1,3} & m_{1,3} \\ m_{1,1} & m_{1,1} & m_{1,2} & m_{1,3} & m_{1,3} \\ m_{2,1} & m_{2,1} & m_{2,2} & m_{2,3} & m_{2,3} \\ m_{3,1} & m_{3,1} & m_{3,2} & m_{3,3} & m_{3,3} \\ m_{3,1} & m_{3,1} & m_{3,2} & m_{3,3} & m_{3,3} \end{bmatrix}$$

$$\boldsymbol{M}_{\text{per}} = \begin{bmatrix} m_{3,3} & m_{3,1} & m_{3,2} & m_{3,3} & m_{3,1} \\ m_{1,3} & m_{1,1} & m_{1,2} & m_{1,3} & m_{1,1} \\ m_{2,3} & m_{2,1} & m_{2,2} & m_{2,3} & m_{2,1} \\ m_{3,3} & m_{3,1} & m_{3,2} & m_{3,3} & m_{3,1} \\ m_{3,3} & m_{1,1} & m_{1,2} & m_{1,3} & m_{1,1} \end{bmatrix}$$

1.6.2 Global Filtering

Global image filtering techniques utilize the *Fourier transform* (FT) and the associated computationally efficient algorithms such as a *fast Fourier transform* (FFT) that enable us to work with the frequency representation of the image. Apart from an introduction and a brief exploration, a rigorous definition of Fourier transform and proofs of its properties remain beyond the scope of this work and can be found in [2] or [1], where its direct applications in the image filtering field are also discussed in some detail. For the same reasons, we will also generally discuss only the case of one-dimensional Fourier transform and outline the n-dimensional extensions where necessary for applications in image processing.

Fourier Series and Transform

For our purposes and level of detail, we can say that it is possible to represent any periodic function satisfying some mild conditions as a sum of sines and cosines weighted with calculable coefficients. We call this sum a *Fourier series*. Functions that are not periodic but whose area under a curve is finite can be similarly represented using integrals of weighted sines and cosines. We call this integral representation a *Fourier transform* of a function. Most importantly, both representations offer inverse operations that enable us to fully reconstruct the original function from either its series or transform [1][2].

Using the Euler's identity formula, we define the Fourier series of a periodic one-dimensional function $f(t)$ with a period T as [1][2]:

$$f(t) = \sum_{n=-\infty}^{\infty} C_n \cdot \exp\left(\frac{2\pi i n t}{T}\right), \tag{1.7}$$

where

$$C_n = \frac{1}{T} \int_{-T/2}^{T/2} f(t) \cdot \exp\left(-\frac{2\pi \mathrm{i} n t}{T}\right) \mathrm{d}t \quad \text{for } n = 0, \pm 1, \pm 2, \ldots \tag{1.8}$$

are the series coefficients and i denotes the imaginary unit.

We define the Fourier transform $\mathscr{F}$ of a continuous one-dimensional function $f(t)$ of a continuous variable t as [1][2]:

$$\mathscr{F}(f(t)) = \int_{-\infty}^{\infty} f(t) \cdot \exp(-2\pi \mathrm{i}\mu t) \mathrm{d}t, \tag{1.9}$$

where μ is also a continuous variable. Because the variable t is annihilated during integration, we may write (1.9) as [1]:

$$\mathrm{F}(\mu) = \int_{-\infty}^{\infty} f(t) \cdot \exp(-2\pi \mathrm{i}\mu t) \mathrm{d}t \tag{1.10}$$

Using the above-mentioned invertibility property or $f(t) = \mathscr{F}^{-1}(\mathrm{F}(\mu))$, we use the inverse Fourier transform to obtain the function $f(t)$ as [1][2]:

$$f(t) = \int_{-\infty}^{\infty} \mathrm{F}(\mu) \cdot \exp(2\pi \mathrm{i}\mu t) \mathrm{d}\mu \tag{1.11}$$

Continuous Convolution and Correlation

Remembering the convolution, which describes the movement of a mask across an image, we may take interest in expressing this operation for continuous functions. Similarly to (1.5), we define the convolution of two continuous functions $f(t)$ and $k(t)$ as [1][2]:

$$(f * k)(t) = \int_{-\infty}^{\infty} f(\tau)k(t - \tau)\mathrm{d}\tau, \tag{1.12}$$

where τ represents a dummy variable annihilated during integration.

We define the continuous correlation of two continuous functions $f(t)$ and $k(t)$ as:

$$(f \circledast k)(t) = \int_{-\infty}^{\infty} \overline{f(\tau)}k(t + \tau)\mathrm{d}\tau, \tag{1.13}$$

The Fourier transform of $(f * k)(t)$ is then:

$$\mathscr{F}((f * k)(t)) = \int_{-\infty}^{\infty} \left[\int_{-\infty}^{\infty} f(\tau)k(t - \tau)\mathrm{d}\tau \right] \exp(-2\pi \mathrm{i}\mu t) \mathrm{d}t \tag{1.14}$$

Switching the order of integration and using the result $\mathscr{F}(k(t-\tau)) = \mathrm{K}(\mu) \exp(-i2\pi\mu t)$ from [1], we may simplify (1.14) as:

$$\mathscr{F}((f * k)(t)) = \int_{-\infty}^{\infty} f(\tau)[\mathrm{K}(\mu)\exp(-2\pi\mathrm{i}\mu t)]\mathrm{d}\tau \qquad (1.15)$$

Seeing that $\mathrm{K}(\mu)$ does not depend on τ and from the definition of Fourier transform of a function $f(\tau)$, we may observe the convolutional theorem [1]:

$$\mathscr{F}((f * k)(t)) = \mathrm{F}(\mu)\mathrm{K}(\mu) = (\mathrm{F} \otimes \mathrm{K})(\mu) \qquad (1.16)$$

Because the convolution is commutative, so is also the multiplication of these respective transforms. We now see that the convolution of two continuous functions of a continuous variable is equal to the multiplication of Fourier transforms of said functions.

The correlation equivalent to the convolutional theorem is:

$$\mathscr{F}((f \circledast k)(t)) = \overline{\mathrm{F}(\mu)}\mathrm{K}(\mu) = (\overline{\mathrm{F}} \otimes \mathrm{K})(\mu), \qquad (1.17)$$

Correlation, however, is only commutative when both functions are hermitian, i.e. they uphold the property $\overline{f(t)} = f(-t)$.

Discretization, Sampling and Aliasing

To employ the theory established for continuous functions in practical use, we must first discretize the continuous functions in question. To do so, we sample the function at a certain sampling rate according to the Nyquist-Shannon Sampling theorem [1] to prevent over- or under-sampling of the function as well as related problems such as aliasing. The sampling theorem dictates that a function is fully recoverable from its discrete samples provided that the sampling rate exceeds double of the highest frequency of the function [1]. Such sampling rate prevents aliasing, which is the situation when it is impossible to recover the original function purely from its discrete samples. We may see such situation in the fig. 1.1.

Supposing that we are sampling a function $f(t)$ every ΔT units apart, the sampled function $\tilde{f}(t)$ is defined as [1][2]:

$$\tilde{f}(t) = \sum_{n=-\infty}^{\infty} f(t)\delta(t - n\Delta T), \qquad (1.18)$$

where $\delta(t)$ is an unit impulse defined as [1] [2]:

$$\delta(t) = 0 \quad \text{if } t \neq 0 \qquad (1.19)$$

that satisfies [1][2]:
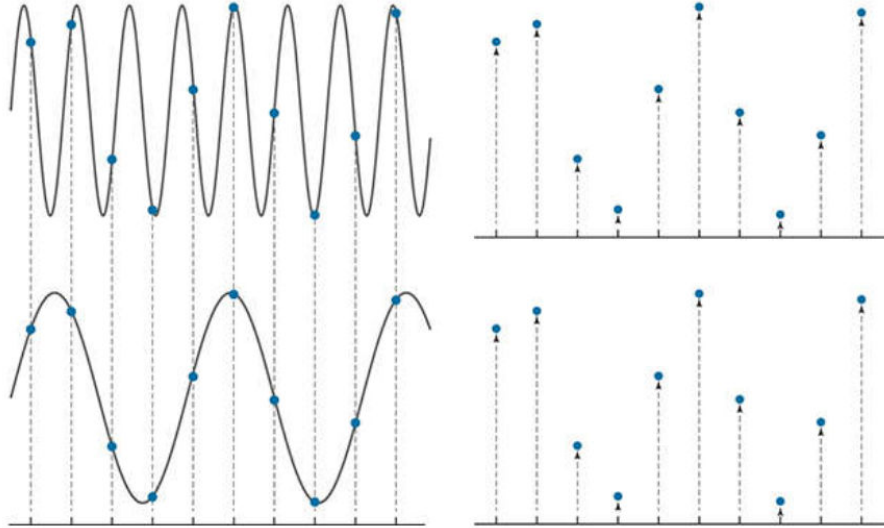
Figure 1.1: Two signals whose sampling exhibits aliasing. Retrieved from [1].

$$\int_{-\infty}^{\infty} \delta(t)\mathrm{d}t = 1 \tag{1.20}$$

Following the sampling, we are then able to obtain the continuous Fourier transform $\tilde{\mathrm{F}}(\mu)$ of the function $\tilde{f}(t)$ [1][2]:

$$\tilde{\mathrm{F}}(\mu) = \int_{-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f(t)\delta(t - n\Delta T)\exp(-2\pi\mathrm{i}\mu t)\mathrm{d}t, \tag{1.21}$$

where the order of summation and integration can be switched and using certain properties of the impulse described in [1] or [2] we obtain from (1.21) [1]:

$$\tilde{\mathrm{F}}(\mu) = \sum_{n=-\infty}^{\infty} f_n \exp(-2\pi\mathrm{i}\mu n\Delta T), \tag{1.22}$$

where f_n is a discrete function. The transform $\tilde{\mathrm{F}}(\mu)$, however, is a continuous periodic function, which enables us to describe it by only one of its periods. Assuming that we choose the period interval $\mu = [0, 1/\Delta T]$ and want to obtain N equally spaced samples of the transform, we obtain the *discrete Fourier transform* (DFT) F_m of the function $f(t)$ [1][2]:

$$F_m = \sum_{n=0}^{N-1} f_n \exp\left(-\frac{2\pi\mathrm{i}mn}{N}\right) \quad \text{for } m = 0, 1, 2, \ldots, N-1 \tag{1.23}$$

We may also return from F_m to the original sample set by using the *inverse discrete Fourier transform* (IDFT) [1][2]:

$$f_n = \frac{1}{N}\sum_{m=0}^{N-1} F_m \exp\left(\frac{2\pi\mathrm{i}mn}{N}\right) \quad \text{for } n = 0, 1, 2, \ldots, N-1 \tag{1.24}$$

It is simple to see that by substituting F_m and f_n into their respective equations, they form a transform pair. It is also interesting to observe that this pair exists for any set of finitely valued uniformly spaced samples [1].

We will finish this section by noting it is customary to write $f(x)$ and $\mathrm{F}(u)$ instead of f_n and F_m, as this manner of notation is better suited to convey the spatial- and frequency-domain variables when extended to the case of n-dimensional images. Therefore, while dealing with a one-dimensional case, we write:

$$\mathrm{F}(u) = \sum_{n=0}^{N-1} f(x) \exp\left(-\frac{2\pi \mathrm{i} u x}{N}\right) \quad \text{for } u = 0, 1, 2, \ldots, N-1 \qquad (1.25)$$

and

$$f(x) = \frac{1}{N} \sum_{m=0}^{N-1} \mathrm{F}(u) \exp\left(\frac{2\pi \mathrm{i} u x}{N}\right) \quad \text{for } x = 0, 1, 2, \ldots, N-1, \qquad (1.26)$$

where x and u denote the spatial or frequency variables.

n-Dimensional Image Fourier Transform

The theory introduced in previous sections can be naturally extended to functions in more than one dimension. If extended to n dimensions, it can be used for our purposes of Fourier transform and global image filtering of n-dimensional images. As the Fourier transform assumes periodicity of the given function, it is customary to first pad the image using one of the techniques introduced in 1.6.1. In the case of global filtering, we use the padding to enlarge the original n-dimensional image $\mathrm{f}(\cdot)$ to a 2^n times larger padded image $\mathrm{f_p}(\cdot)$. This padded image is then used to compute the DFT $\mathrm{F}(\cdot)$ of the image. After constructing the kernel $\mathrm{K}(\cdot)$ (see Chapter 2) and multiplying $\mathrm{F}(\cdot) \otimes \mathrm{K}(\cdot)$, based on the convolutional theorem, we obtain the filtered image $\mathrm{G}(\cdot)$ in the frequency domain:

$$\mathrm{G}(\cdot) = \mathrm{K}(\cdot) \otimes \mathrm{F}(\cdot), \qquad (1.27)$$

We obtain the spatial representation of the filtered padded image as:

$$g_{\mathrm{p}}(\cdot) = \mathscr{F}^{-1}(\mathrm{G}(\cdot)),$$

We then obtain the final filtered result $\mathrm{g}(\cdot)$ by cropping the upper left region of $\mathrm{g_p}(\cdot)$ of size corresponding to the size of the original image.

Fast Fourier Transform (FFT)

Fast Fourier transform is a computationally efficient method that simplifies and speeds up otherwise cumbersome calculation of the DFT. It uses primarily the separability of DFT - the three-dimensional transform of a function image f($\boldsymbol{k}$) can be computed by first computing one-dimensional transform of f($\boldsymbol{k}$) for each row and column along the grid depth of the obtained result [1]. Computing the Fourier transform of any three-dimensional image is therefore reduced to sequential computation of one-dimensional transforms.

The second simplification used in FFT is that it is in fact possible to compute the IDFT using the forward DFT algorithm. Analogously to (1.25), we define the n-dimensional DFT as [1]:

$$F(\boldsymbol{u}) = \sum_{\boldsymbol{k} \in \mathcal{S}} f(\boldsymbol{k}) \exp\left(-2\pi i \sum_{j=1}^{n} \frac{u_j k_j}{r_j}\right) \tag{1.28}$$

and assuming $N = \prod_{j=1}^{n} r_j$ the IDFT as [1]:

$$f(\boldsymbol{k}) = \mathscr{F}_n^{-1}(F(\boldsymbol{u})) = \frac{1}{N} \sum_{\boldsymbol{u} \in \mathcal{S}} F(\boldsymbol{u}) \exp\left(2\pi i \sum_{j=1}^{n} \frac{u_j k_j}{r_j}\right) \tag{1.29}$$

Taking the complex conjugate of both sides of IDFT multiplied by N we get from (1.29) [1]:

$$N \cdot \overline{f(\boldsymbol{k})} = \mathscr{F}_n(\overline{F(\boldsymbol{u})}) = \sum_{\boldsymbol{u} \in \mathcal{S}} \overline{F(\boldsymbol{u})} \exp\left(-2\pi i \sum_{j=1}^{n} \frac{u_j k_j}{r_j}\right) \tag{1.30}$$

We are now able to see that to obtain the complex conjugate of f($\boldsymbol{k}$) multiplied by N, we can simply calculate the DFT of the complex conjugate $\overline{F(\boldsymbol{u})}$. Taking the complex conjugate of this result divided by $\prod_{j=1}^{n} r_j$ then gives us the function f($\boldsymbol{k}$). Thus, we are able to obtain the IDFT using the DFT algorithm. In the case of f : $\mathcal{S} \to \mathbb{R}$, this computation is further simplified, as f($\boldsymbol{k}$) = $\overline{f(\boldsymbol{k})}$.

There are many FFT algorithms, most of which reduce the computation complexity from $O(N^2)$ of the DFT to $O(N \cdot \log(N))$, assuming we compute a one-dimensional transform. This simplification gives us computational advantage of $\dfrac{N}{\log_2(N)}$ when compared to classical DFT algorithm [1]. For example, if the DFT requires N^2 operations, and we know that $N = 1024 = 2^{10}$, we may assume computational advantage of $\dfrac{2^{10}}{\log_2 2^{10}} = \dfrac{2^{10}}{10} \approx 100$. Therefore, we may expect that a FFT will arrive at a result approximately a hundred times faster than the classical DFT algorithm.

1.7 Texture-Based Image Analysis

In this section, we will briefly discuss the notion of image texture and several approaches to its analysis. These will include statistical approaches, which aim to quantify certain perceived local qualities such as smoothness, directionality and regularity [1][3][4][5]. Popular statistical approaches include grey-level co-occurrence matrix (GLCM) (utilizing some or all of features proposed by Haralick et al. in [4]) and features proposed by Tamura et al. in [5]. Transform-based approaches aim to represent the image through a coordinate system that might be closely related to perceived image textures. These approaches include techniques based on the FT, Gabor decomposition and the wavelet transform.

1.7.1 Image Texture

Even though it is generally easy to distinguish between different textures for the human eye, the notion of image texture is not rigorously defined [1][3]. Using statistical approaches we may define texture as being described by certain local qualities such as smoothness (or coarseness), directionality, regularity, line-likeness or contrast [3][4][5]. No matter which approach we choose, the core of texture-based analysis is the extraction of quantifiable qualities, which is only possible through devising mathematical definitions of texture descriptors.

1.7.2 Statistical Approaches

Statistical approaches attempt to come up with mathematical approximations of qualities perceived by the human eye. As already mentioned, these include smoothness (or coarseness), directionality, regularity, line-likeness or contrast [3][4][5]. The disadvantage of these methods is that they are generally not well-defined in three-dimensional images and their use of local spatial approach to texture, which, especially in three-dimensional images, results in a large number of calculations and time inefficiency (see 1.6.1).

Grey-Level Co-Occurrence Matrix (GLCM)

Grey-Level Co-Occurrence Matrix considers second order statistics, i.e. it studies the pairs of pixels, which have predefined spatial relation to each other. We utilize co-occurrence matrices $\boldsymbol{P}(i, j | d, \theta)$ [3]. By this notation, we understand that the matrix element on position (i, j) represents the number of occurrences of a situation, in which one pixel has the grey-level intensity i and is separated by the distance d and the angle θ from another pixel having intensity j. We may utilize a symmetric version (counting if pairs are separated by $\pm d$) or a nonsymmetric version considering only positive distances. After computing the co-occurrence matrices, we calculate various of the 14 features proposed by Haralick et al. in [4]. Conners and Harlow have however shown that only five of these original features are necessary to obtain

satisfactory results [6]. These are energy, entropy, correlation, local homogeneity and contrast (see [3, p. 3] or [4, pp. 10–11] for definitions). This inherently local (based on neighbourhood distance and angle) approach may be time-consuming for even larger two-dimensional images, and is especially limited by the choice of investigated of directions θ in the three-dimensional case. Because of these limitations, we consider this approach not suitable to analysis of three-dimensional images (with some exceptions, such as [7]).

Tamura Features

In 1978 Tamura et al. [5] proposed six textural features designed to correspond to human visual perception [3]. These are coarseness, contrast, directionality, line-likeness, regularity and roughness. For precise definitions, see [3, p. 6] or [5, pp. 6–9]. These features (particularly coarseness or line-likeness) again require extension into three-dimensions (see for example [8]) and their usage in three-dimensions can be mired by complex and time-consuming computations.

1.7.3 Transform-Based Approaches

Transform-based approaches convert an image into a new form, such as the frequency-domain representation, whose coordinate system can be interpreted as relevant to the features of the image texture [3][9].

Fourier Transform-Based Approaches

Fourier transform-based approaches decompose the original image into its frequency components using DFT (see 1.6.2). Spatial edges of the original image exhibit low frequency in one direction, whereas there are multiple frequencies in the orthogonal direction. This fact is represented by a straight line in the frequency domain. The further we are from the centre of the image spectrum, the higher the frequency. Smooth texture will, therefore, show high values about the spectrum centrer, whereas coarse texture will show values spread over the spectrum [3].

The Fourier transform-based approaches are also useful for when we want to compute rotationally invariant features. Fourier transform cannot meaningfully describe local variations in image texture. This is, however, not a grave hindrance for our usage of FT-based approach, as our final aim is the image classification based on feature extraction. We will use an approach somewhat similar to the one used by Maani et al. [10], who utilized a neighbouring function with circular radius leading to rotation invariant texture features.

Gabor Decomposition-Based Approaches

Gabor decomposition-based approaches perform space-frequency decomposition. A Gabor filter is a Gaussian kernel function modulated by a sinusoidal plane wave (see [3, p. 11]), which can be seen as a tunable bandpass filter. The general principle is to decompose the original image into several filtered images using Gabor filter with differently set parameters. Each of these images then possesses information limited to a certain part of the spectrum. We are able to extract textural features from the Gabor-filtered images. These features, however, have the disadvantage of being non-orthogonal, which may lead to calculation of redundant (not scale invariant) features when considering more than one filter scale [3]. Calculation of Gabor features is thus hindered by high storage requirements and computational demands [11] when compared to other methods.

Wavelet Transform-Based Approaches

Similar to Gabor analysis, the Wavelet transform-based approaches also consider the textural content in both frequency and spatial domains. The wavelet transform approximates the image by tuned local wavelets based on the mother wavelet (see [3, p. 11]). During each stage of analysis, the transform decomposes the original input into several sub-images containing different information about the texture. Each of these sub-images is then treated as a separate image and analysed in the next iteration. This leads to feature extraction, which is not scale invariant. As the wavelet transform uses a dataset-independent basis/mother function wavelet, which may be considered a type of fixed dictionary, it is less flexible than various other methods. While wavelet has the better ability to represent textures at different scales when compared to Gabor-based approaches, it is not translationally nor rotationally invariant [3].

1.8 Image Classification

In the scope of this work, we will understand image classification mainly as a task of utilizing calculated image textural features as inputs of various classifiers to decide whether the image data (dataset of SPECT images of brains of CN and AD patients) belongs to a patient suffering from Alzheimer's disease (binary classification). This process may first include preprocessing of the extracted features, consisting of data transformation and whitening. Resulting data will be used as inputs of binary classifiers - the linear discriminant analysis (LDA), quadratic discriminant analysis (QDA), k-nearest neighbours (KNN), the support vector machines (SVMs) of the polynomial and Gaussian variety, and the artificial neural networks (ANNs) with one hidden layer. All of these classifiers will be cross-validated using primarily the leave-one-out method. We will also try the statistical testing approach of utilizing the Mann-Whitney U test to determine which, if any, features have the largest potential for dataset separation.

Chapter 2

Fast Global Filtering

In this chapter, we introduce and describe some of the techniques of global filtering in the frequency domain, which will make use of the already introduced FFT. MATLAB implementation of all the methods described in this chapter is one of the topics discussed in Chapter 5. Obtained results are then discussed and analysed in Chapter 6.

We begin this chapter by introducing several types of frequency-domain low-pass (LP) kernels - binary, Gaussian, Butterworth, and α-stable. We will also mention a manner in which we can utilise these types of kernels for high-pass (HP) filtering. Apart from 'inverting' the low-pass filters, we also define the high-boost frequency filters. Finally, we describe several nonlinear input and output data transformations that can be used as an intermediary step during frequency-domain image filtering.

2.1   Low-Pass Filtering (LP)

We will begin this chapter by defining several kernels that result in image smoothing in the frequency domain. We will start with a simple cut-off binary low-pass filter, then define filters based on Gaussian and Butterworth functions, and finish this section with filters based on the α-stable characteristic function. As we will observe in the next section, it will be easy to invert the low-pass filters and define sharpening high-pass filters using smoothers as a basis.

2.1.1   Binary Kernel

We define the *binary low-pass filter* as a filter with kernel that passes all frequencies within a predefined radius d_0 without change and cuts off all frequencies outside said circular radius. Extending the definition found [1] to n dimensions, we write:

$$K_{\mathrm{LP}}(\boldsymbol{u}) = \mathrm{I}_{\mathrm{d}(\boldsymbol{u}) \leq d_0}(\boldsymbol{u}),\tag{2.1}$$

where I is the indicator function and by d($\cdot$) we understand the Euclidean distance from the centre of the centred and padded frequency representation of an n-dimensional image:

$$\mathrm{d}(\boldsymbol{u}) = \left(\sum_{j=1}^{n} \left(\frac{u_j}{r_j} \right)^2 \right)^{\frac{1}{2}} \tag{2.2}$$

where the original image f($\cdot$) is bounded by the range vector $\boldsymbol{r}$ and $d_0 > 0$ is a tunable critical frequency threshold.

2.1.2 Gaussian Kernel

Extending the definition found in [1], we define the Gaussian low-pass filter kernel by utilizing the n dimensional transfer function:

$$K_{\mathrm{LP}}(\boldsymbol{u}) = \exp\left(- \frac{\mathrm{d}^2(\boldsymbol{u})}{2\sigma^2} \right), \tag{2.3}$$

where d($\boldsymbol{u}$) is the Euclidean distance from the centre of the frequency representation of the original image, which was defined in (2.2). The standard deviation σ controls the spread around the centre of the kernel.

2.1.3 Butterworth Kernel

Extending the definition found in [1], we define the n-dimensional Butterworth low-pass filter kernel using a transfer function [1]:

$$K_{\mathrm{LP}}(\boldsymbol{u}) = \frac{1}{\sqrt{1 + \left(\frac{\mathrm{d}(\boldsymbol{u})}{d_0} \right)^{2m}}}, \tag{2.4}$$

where $m \in \mathbb{N}$ is the order of the transfer function and d_0 the frequency threshold.

2.1.4 α-Stable Kernel

The one-dimensional α-stable filter kernel arises from the α-stable characteristic function [13]:

$$\phi(\omega) = \exp(-\gamma|\omega|^{\alpha}), \tag{2.5}$$

where $\gamma \in \mathbb{R}^+$ is called the *dispersion* and $\alpha \in (0, 2)$ the *characteristic exponent* that determines the tail heaviness of the distribution [14]. In the case of $\alpha = 2$, the

distribution becomes a Gaussian distribution with a mean of zero and a variance of 2γ. The case $\alpha = 1$ coincides with a Cauchy distribution centred at zero with a PDF [12][13]:

$$f(x) = \frac{\gamma}{\pi} \frac{1}{\gamma^2 + x^2} \tag{2.6}$$

For cases $\alpha \notin \{1, 2\}$, there is no analytical expression of the corresponding density functions [15]. These densities are, however, smooth, unimodal, symmetric about the mode and bell shaped [12]. The α-stable characteristic function can be extended to a n-dimensional joint characteristic function [16]:

$$\phi(\omega_1, \omega_2, \ldots, \omega_n) = \exp\left(-\gamma|\omega_1^2 + \omega_2^2 + \ldots + \omega_n|^{\frac{\alpha}{2}}\right), \tag{2.7}$$

where $\gamma \in \mathbb{R}^+$ and $\alpha \in (0, 2)$.

Utilizing the concept of distance within the frequency representation of the image and cut-off distance, we may use (2.7) and define the n-dimensional α-stable kernel as:

$$K_{\text{LP}}(\boldsymbol{u}) = \exp\left(-\left(\frac{\mathrm{d}(\boldsymbol{u})}{d_0}\right)^\alpha\right) \tag{2.8}$$

2.2 High-Pass Filtering (HP)

2.2.1 Kernel Transformation

There are many ways how to define the high-pass filters. We utilise the already defined low-pass filters and transform the general low-pass kernel $K_{\text{LP}}(\boldsymbol{u})$, into a n-dimensional high-pass kernel $K_{\text{HP}}(\boldsymbol{u})$ by 'inverting' the filter and subtracting the value of the low-pass kernel from one:

$$K_{\text{HP}}(\boldsymbol{u}) = 1 - K_{\text{LP}}(\boldsymbol{u}) \tag{2.9}$$

2.2.2 Unsharp Masking, High-boost Filtering

The high-boost filtering and its special case, the unsharp masking, use a simple idea. In the process of sharpening the image $\mathrm{f}(\cdot)$, we first blur the image using a low-pass filter then subtract the blurred result from the original and finally add the difference to the original image, thus sharpening it. The sharpened image $\mathrm{g}(\cdot)$ is therefore equal to [1]:

$$\mathrm{g}(\cdot) = \mathrm{f}(\cdot) + c \cdot (\mathrm{f}(\cdot) - \mathrm{f}_{\text{blur}}(\cdot)), \tag{2.10}$$

where $c \in \mathrm{R}_0^+$ is a filtering constant. If $c = 1$, we speak of the unsharp masking. When $c > 1$, we describe the method as high-boost filtering.

Using fast global filtering, we obtain the blurred image $\mathrm{f}_{\mathrm{blur}}(\cdot)$ as [1]:

$$\mathrm{f}_{\mathrm{blur}}(\cdot) = \mathscr{F}^{-1}[K_{\mathrm{LP}}(\cdot)F(\cdot)], \tag{2.11}$$

where $K_{\mathrm{LP}}(\cdot)$ is the chosen low-pass kernel.

We obtain the result $\mathrm{g}(\cdot)$ from the frequency domain [1]:

$$
\begin{aligned}
\mathrm{g}(\cdot) &= \mathscr{F}^{-1}[(1 + c \cdot (1 - K_{\mathrm{LP}}(\cdot)))F(\cdot)] \\
&= \mathscr{F}^{-1}[(1 + c \cdot K_{\mathrm{HP}}(\cdot))F(\cdot)],
\end{aligned}
\tag{2.12}
$$

where we use the previously defined relation $K_{\mathrm{HP}}(\cdot) = 1 - K_{\mathrm{LP}}(\cdot)$.

2.3   Nonlinear Transformations

In the last section of this chapter, we introduce several nonlinear data transformations and their corresponding inverse functions, which, although not frequently used in this context [18], can be naturally extended to digital images. These transformations will be the *power transformation* (PT), the *log transformation* (LT) and the *Box-Cox transformation* (BCT), all of which transform intensity values of the digital image and can lead to image quality enhancement [1] [18]. Another use of transformation can be found in the field of image segmentation and feature extraction [19], which means that these transformations may prove useful for our purposes.

2.3.1   Power Transformation (PT)

Let f: $\mathcal{S} \rightarrow \mathbb{R}_0^+$ be an n-dimensional image. We transform this image to an image $\mathrm{f}_{\mathrm{tran}}(\cdot)$ by changing its intensity element-wise, which we achieve by using the power transformation (PT) [1]:

$$\mathrm{f}_{\mathrm{tran}}(\boldsymbol{i}) = c \cdot \mathrm{f}(\boldsymbol{i})^{\gamma}, \tag{2.13}$$

where we assume $c, \gamma \in \mathbb{R}^+$. The operation $\mathrm{f}(\boldsymbol{i})^{\gamma}$ signifies element-wise exponentiation of the image intensity in a location described by the coordinate vector $\boldsymbol{i}$.

We define the inverse of the (2.13) as:

$$f_{\text{tran}}(\boldsymbol{i}) = \left(\frac{f(\boldsymbol{i})}{c}\right)^{\frac{1}{\gamma}} \tag{2.14}$$

2.3.2 Log Transformation (LT)

Let f: $\mathcal{S} \to \mathbb{R}_0^+$ be an n-dimensional image. We transform this image to an image $f_{\text{tran}}(\cdot)$ by changing its gamma intensity element-wise by using a log transformation (LT) [1]:

$$f_{\text{tran}}(\boldsymbol{i}) = c \cdot \log(f(\boldsymbol{i}) + \lambda), \tag{2.15}$$

where we assume $c \in \mathbb{R}^+$, $\lambda > -\min_{i \in \mathcal{S}}\{f(\boldsymbol{i})\}$.

We define the inverse of (2.15) as:

$$f_{\text{tran}}(\boldsymbol{i}) = \exp\left(\frac{f(\boldsymbol{i})}{c}\right) - \lambda \tag{2.16}$$

2.3.3 Box-Cox Transformation (BCT)

Let f: $\mathcal{S} \to \mathbb{R}_0^+$ be an n-dimensional image f($\cdot$). We transform this image to an image $f_{\text{tran}}(\cdot)$ by changing its gamma intensity element-wise by using a Box-Cox transformation defined in [17]. We must, however, remember the fact that we assume the image to only possess non-negative intensity values. These values can therefore be from the interval $[0, 1)$, which could result in a transformed image having negative intensity values. Consequently, when values smaller than one occur, we substitute them with $\epsilon > 0$ by using the $\max\{f(\boldsymbol{i}), \epsilon\}$:

$$f_{\text{tran}}(\boldsymbol{i}) = \begin{cases} \dfrac{\max\{f(\boldsymbol{i}), \epsilon\}^\lambda - 1}{\lambda} & \text{for } \lambda \neq 0 \\ \log(\max\{f(\boldsymbol{i}), \epsilon\}) & \text{for } \lambda = 0, \end{cases} \tag{2.17}$$

where $\lambda \in \mathbb{R}$, $\epsilon > 0$ are tunable parameters.

We define the inverse of (2.17):

$$f_{\text{tran}}(\boldsymbol{i}) = \begin{cases} (\lambda \cdot f(\boldsymbol{i}) + 1)^{\frac{1}{\lambda}} & \text{for } \lambda \neq 0 \\ \exp[f(\boldsymbol{i})] & \text{for } \lambda = 0 \end{cases} \tag{2.18}$$

Chapter 3

Texture-Based Image Analysis

In this chapter, we introduce the concepts underlying our approach to the texture-based analysis of three-dimensional images. We first define and discuss Zernike polynomials, whose FT will be later used for the calculation of rotationally invariant image features. We then describe several global invariant characteristics, which will serve as our features and inputs for binary classifiers in the later chapters of this work. The MATLAB implementation of the theoretical concepts located in this chapter is the topic of Chapter 5 and Appendix A.

3.1 Zernike Polynomials

Let $\mathcal{B}$ be a unit ball in L^2, $\mathcal{B} = \{\boldsymbol{x} \in \mathbb{R}^3 \colon \|\boldsymbol{x}\|_2 \leq 1\}$. Let $\mathcal{P} = [0,1] \times [0,\pi] \times (-\pi,\pi]$ be the representation of $\mathcal{B}$ in spherical coordinates. Let $\mathcal{B}_0$ be a unit sphere in L^2, $\mathcal{B}_0 = \{\boldsymbol{x} \in \mathbb{R}^3 \colon \|\boldsymbol{x}\|_2 = 1\}$. Let $r = 1$ and let $\mathcal{P}_0 = [0,\pi] \times (-\pi,\pi]$ be the representation of $\mathcal{B}_0$ in spherical coordinates.

3.1.1 Definition of Zernike Polynomial

Let $\boldsymbol{r} = (r,\theta,\phi)$ be a vector of three-dimensional spherical coordinates, $r \in [0,1]$, $\phi \in (-\pi,\pi]$, $\theta \in [0,\pi]$. Let $l,n \in \mathbb{N}_0$ and $m \in \mathbb{Z}$, $-l \leq m \leq l$. The complex-valued three-dimensional Zernike polynomial $Z_{l,n}^m \colon \mathcal{P} \to \mathbb{C}$ is defined [20]:

$$Z_{l,n}^m(r,\theta,\phi) = R_{l,n}(r)Y_l^m(\theta,\phi) \tag{3.1}$$

The Zernike polynomial $Z_{l,n}^m(r,\theta,\phi)$ thus consists of a real-valued radial polynomial $R_{l,n} \colon [0,1] \to \mathbb{R}$ and a complex-valued spherical harmonic function $Y_l^m \colon \mathcal{P}_0 \to \mathbb{C}$. The radial polynomial $R_{l,n}(x)$ is defined [20]:

$$R_{l,n}(x) = \sqrt{4n+2l+3}(-1)^n x^l P_n^{[l+\frac{1}{2},0]}(1-2x^2) \tag{3.2}$$

Let $\alpha, \beta \in \mathbb{R}$. A Jacobi polynomial $P_n^{[\alpha,\beta]} \colon [-1,1] \to \mathbb{R}$ is defined by the formula [20]:

$$P_n^{[\alpha,\beta]}(x) = \frac{(-1)^n}{2^n n!}(1-x)^{-\alpha}(1+x)^{-\beta}\frac{\mathrm{d}^n}{\mathrm{d}x^n}[(1-x)^\alpha(1+x)^\beta(1-x^2)^n] \tag{3.3}$$

The polynomials $R_{l,n}(x)$ are orthonormal in the natural inner product on the unit ball $\mathcal{B}$ [20].

The complex-valued spherical harmonic function $Y_l^m(\theta,\phi)$ is defined by the formula [20]:

$$Y_l^m(\theta,\phi) = \sqrt{\frac{2l+1}{4\pi}\frac{(l-m)!}{(l+m)!}}P_l^m(\cos\theta)\exp(\mathrm{i}m\phi) \tag{3.4}$$

where $P_l^m\colon [-1,1] \to \mathbb{R}$ is a Legendre Associated polynomial [20]:

$$P_l^m(x) = \frac{(-1)^m}{2^l l!}(1-x^2)^{\frac{m}{2}}\frac{\mathrm{d}^{l+m}}{\mathrm{d}x^{l+m}}(x^2-1)^l \tag{3.5}$$

The spherical harmonics form a complete orthonormal basis of the surface of the unit sphere $\mathcal{B}_0$ [20]. We thus see that Zernike polynomials $Z_{l,n}^m(\boldsymbol{r})$ form a complete orthonormal basis of a function inside the unit ball $\mathcal{B}$ with respect to the $l_2 = \|\boldsymbol{x}\|_2$ norm on the unit ball [20].

3.1.2 Fourier Transform of Zernike Polynomial

Let $l,n \in \mathbb{N}_0$, $m \in \mathbb{Z}$, $-l \le m \le l$. The Fourier transform of the complex-valued three-dimensional Zernike polynomial $Z_{l,n}^m(\boldsymbol{r})$ is given by the formula [20]:

$$\mathscr{F}(Z_{l,n}^m(\boldsymbol{r})) = \frac{(-1)^n}{\mathrm{i}^l 2\pi^{l+\frac{1}{2}}}Y_l^m\left(\frac{\boldsymbol{k}}{k}\right)\frac{J_{2n+l+\frac{3}{2}}(k)}{k}, \tag{3.6}$$

where $\boldsymbol{r} = (r,\theta,\phi)$ is the coordinate vector of a point in the three-dimensional spatial domain and $\boldsymbol{k} = (k,\psi,\mu)$ is the coordinate vector of a point in the three-dimensional frequency domain, where k is the magnitude of $\boldsymbol{k}$, $k = \|\boldsymbol{k}\|_2 = \sqrt{(k\sin\psi\cos\mu)^2 + (k\sin\psi\sin\mu)^2 + (k\cos\psi)^2}$. $J_\alpha\colon \mathbb{R} \to \mathbb{R}$ is a Bessel function of the first kind and order $\alpha \in \mathbb{R}_0^+$ [21]:

$$J_\alpha(x) = \sum_{j=0}^\infty \frac{(-1)^j}{j!\,\Gamma(j+\alpha+1)}\left(\frac{x}{2}\right)^{2j+\alpha} \tag{3.7}$$

and $\Gamma(z)$ is a gamma function:

$$\Gamma(z) = \int_0^\infty t^{z-1}\exp(-t)\mathrm{d}t \tag{3.8}$$

3.2   Rotational Invariants

3.2.1   Rotational Invariants Inside Unit Ball

Let $f\colon \mathcal{P} \to \mathbb{C}$ be a general complex-valued function. The linear expansion of the function $f(\boldsymbol{r}) \in \mathcal{B}$ is defined by the formula [20][22]:

$$f(\boldsymbol{r}) = \sum_{l=0}^{\infty} \sum_{n=0}^{\infty} \sum_{m=-l}^{l} a_{l,n,m} Z_{l,n}^{m}(\boldsymbol{r}), \tag{3.9}$$

where $a_{l,n,m} \in \mathbb{C}$ are coefficients of the linear expansion and $Z_{l,n}^{m} : \mathcal{P} \to \mathbb{C}$ are three-dimensional Zernike polynomials. The linear combination (3.9) is $\mathbf{SO}(3)$ rotationally invariant [20]. The linear coefficients $a_{l,n,m}$ are given by the formula stemming from the orthonormality of Zernike polynomials [22][23][24]:

$$a_{l,n,m} = \int_{0}^{1} \int_{0}^{\pi} \int_{-\pi}^{\pi} f(\boldsymbol{r}) \overline{Z_{l,n}^{m}(\boldsymbol{r})} \mathrm{d}\boldsymbol{r}, \tag{3.10}$$

where $\mathrm{d}\boldsymbol{r} = r^2 \sin(\theta) \mathrm{d}r \mathrm{d}\theta \mathrm{d}\phi$.

Let f: $\mathcal{S} \to \mathbb{C}$ be a complex-valued three-dimensional image with a support set $\mathcal{S}$ and a range vector $\boldsymbol{r}$. Let $\mathrm{A}_{l,n,m} : \mathcal{S} \to \mathbb{C}$ be a three-dimensional image, where the element intensities correspond to spatial representation of coefficients $a_{n,l,m}$. Remembering the convolutional theorem (1.16) and applying it to three-dimensional images, we may see that:

$$\mathrm{A}_{l,n,m}(\cdot) = \mathscr{F}^{-1}(\mathrm{Q}_{l,n}^{m}(\cdot) \otimes \mathrm{F}(\cdot)), \tag{3.11}$$

where $\mathrm{Q}_{l,n}^{m}(\cdot)$ is a three-dimensional image consisting of sampled values of the Fourier transform $Q_{l,n}^{m}(\boldsymbol{k})$ of the Zernike polynomial $\overline{Z_{l,n}^{m}(\boldsymbol{r})}$ and $\mathrm{F}(\cdot)$ is the three-dimensional image consisting of values of the Fourier transform of the original image f($\cdot$).

We define the l, n-th invariant $b_{l,n}$ as a sum of squared linear coefficients $a_{l,n,m}$:

$$b_{l,n} = \sum_{m=-l}^{l} |a_{l,n,m}|^2, \tag{3.12}$$

and understand that $\mathrm{b}_{l,n} \in \mathbb{R}$ is a $\mathbf{SO}(3)$ rotationally invariant [25]. We thus see that $\mathrm{B}_{l,n}\colon \mathcal{S} \to \mathbb{R}$ consisting of values $b_{l,n}$ is a three-dimensional real-valued image, whose intensities correspond to a discrete sampling of the rotational invariant. We will use this image to calculate several textural characteristics of the original image f($\cdot$).

3.2.2 Scaled Rotational Invariants

As shown above, the rotational invariants are defined inside a three-dimensional L_2 unit ball, where orthonormality holds. However, as this is too restrictive a definition for our purposes of describing three-dimensional images, we also find it useful to generalize the expansion (3.9) to a ball described by a three-dimensional coordinate vector $\boldsymbol{s}$ with an arbitrary maximum radius $\rho > 1$ and calculate the generalized coefficients $a_{l,n,m}^*$ and thus also the invariants $b_{l,n}^*$ more flexibly for this broader case. This may be done by scaling the coordinate vector $\boldsymbol{r}$ of the unit ball to the general ball case [23]. The radius ρ is thus also a scaling parameter, which converts the model to the general ball case (or, in fact, transforms the general cases to the known case of the unit ball). We transform the original vector to the general case by substituting $\boldsymbol{s} = \rho \cdot \boldsymbol{r}$, where $r = |\boldsymbol{r}| \leq 1$, as noted in previous sections. The expression (3.9) thus becomes:

$$f(\boldsymbol{s}) = \sum_{l=0}^{\infty} \sum_{n=0}^{\infty} \sum_{m=-l}^{l} a_{l,n,m}^* Z_{l,n}^m \left(\frac{\boldsymbol{s}}{\rho} \right) \tag{3.13}$$

We also rewrite the formula (3.10) as:

$$a_{l,n,m}^* = \int_0^{\rho} \int_0^{\pi} \int_{-\pi}^{\pi} f \left(\frac{\boldsymbol{s}}{\rho} \right) \overline{Z_{l,n}^m \left(\frac{\boldsymbol{s}}{\rho} \right)} \mathrm{d}\boldsymbol{s}, \tag{3.14}$$

where $a_{l,n,m}^*$ signifies the generalized expansion coefficient. We define the generalized invariants $b_{l,n}^*$ in the same manner as above, while keeping in mind the declared substitution.

3.3 Image Characteristics

In this section, we define the statistical characteristics calculated on image invariants, which will serve as image textural characteristics. These simple properties such as the mean, the median, etc. will be utilised as input data (in both reduced, whitened and non-whitened versions) for image classification. As we can perceive the invariant as an n-dimensional image (see above), we will define these characteristics for images.

Let f: $\mathcal{S} \to \mathbb{C}$ be an n-dimensional image with a support set $\mathcal{S}$. We define the image maximum $\max[\mathrm{f}(\cdot)]$:

$$\max[\mathrm{f}(\cdot)] = \max_{\boldsymbol{i} \in \mathcal{S}} [\mathrm{f}(\boldsymbol{i})] \tag{3.15}$$

We define the image minimum $\min[\mathrm{f}(\cdot)]$:

$$\min[f(\cdot)] = \min_{i \in \mathcal{S}}[f(i)] \tag{3.16}$$

We define the image range $\text{range}[f(\cdot)]$:

$$\text{range}[f(\cdot)] = \max[f(\cdot)] - \min[f(\cdot)] \tag{3.17}$$

Let r be the range vector of the image $f(\cdot)$. Let $N = \prod_{i=1}^{n} r_i$ be the number of image elements. We define the image mean $\text{mean}[f(\cdot)]$:

$$\text{mean}[f(\cdot)] = \frac{1}{N} \sum_{i \in \mathcal{S}} f(i) \tag{3.18}$$

We define the image median $\text{med}[f(\cdot)]$:

$$\text{med}[f(\cdot)] = \begin{cases} f(i_{(N+1)/2}) & \text{if } N \text{ is odd} \\[2mm] \dfrac{f(i_{(N/2)}) + f(i_{(N/2)})}{2} & \text{if } N \text{ is even,} \end{cases} \tag{3.19}$$

where $f(i_{(N/2)})$ signifies the $\frac{N}{2}$-th element of the ordering of image element intensities from smallest to largest.

Let $g(\cdot)$ be an n-dimensional image, $g(i) = f(i) - \text{med}[f(\cdot)]$, $\forall i \in \mathcal{S}$. We define the image median absolute deviation (MAD) $\text{mad}[f(\cdot)]$:

$$\text{mad}[f(\cdot)] = \text{med}[g(\cdot)] \tag{3.20}$$

Let $k \in \mathbb{N}_0$, $k \leq 100$. We define the k-th percentile of the image $f(\cdot)$ as $\text{perc}[f(\cdot)](k)$:

$$\text{perc}[f(\cdot)](k) = f(i_{(\lceil \frac{k \cdot N}{100} \rceil)}) \tag{3.21}$$

We define the first quartile of the image $f(\cdot)$ as $Q_1[f(\cdot)]$:

$$Q_1[f(\cdot)] = \text{perc}[f(\cdot)](25) \tag{3.22}$$

41

We define the third quartile of the image f($\cdot$) as Q$_3$[f($\cdot$)]:

$$Q_3[f(\cdot)] = \text{perc}[f(\cdot)](75) \tag{3.23}$$

We define the inter-quartile range of the image f($\cdot$) as IQR[f($\cdot$)]:

$$\text{IQR}[f(\cdot)] = Q_3[f(\cdot)] - Q_1[f(\cdot)] \tag{3.24}$$

We define the variance of the image f($\cdot$) as var[f($\cdot$)]:

$$\text{var}[f(\cdot)] = \frac{1}{N} \sum_{i \in \mathcal{S}} [f(\boldsymbol{i}) - \text{mean}[f(\cdot)]]^2 \tag{3.25}$$

We define the skewness of the image f($\cdot$) as skew[f($\cdot$)]:

$$\text{skew}[f(\cdot)] = \frac{\frac{1}{N} \sum_{i \in \mathcal{S}} [f(\boldsymbol{i}) - \text{mean}[f(\cdot)]]^3}{\text{var}[f(\cdot)]^{\frac{3}{2}}} \tag{3.26}$$

We define the kurtosis of the image f($\cdot$) as kurt[f($\cdot$)]:

$$\text{kurt}[f(\cdot)] = \frac{\frac{1}{N} \sum_{i \in \mathcal{S}} [f(\boldsymbol{i}) - \text{mean}[f(\cdot)]]^4}{\text{var}[f(\cdot)]^2} \tag{3.27}$$

Chapter 4

Image Classification

In this chapter, we describe our approach to image classification. This approach is tailored to the binary classification of medical SPECT images of healthy, cognitively normal (CN) brains as well as the brains of patients suffering from Alzheimer's disease (AD). We first introduce the approach of statistically testing the data for separability. We then introduce the data transformations, resulting in three approaches to image classification - classification using one-dimensional (based on one calculated feature) inputs, classification using the full data inputs (calculated using the theory described in Chapter 3) and classification using data inputs, which were first whitened using the principal component analysis (PCA). We continue by describing the types of classifiers we will be using, viz. the linear discriminant analysis (LDA), quadratic discriminant analysis (QDA), k-nearest neighbours (KNN), the support vector machines (SVMs), and the artificial neural networks (ANNs). Lastly, we describe the cross-validation (CV) process - the metrics we will use to measure the quality of classification, and the leave-one-out and stratified-k-fold approaches to cross-validation. The MATLAB implementation of the theoretical concepts located in this chapter is the topic of Chapter 5 and Appendix A. The results are shown and discussed in detail in Chapter 6.

4.1 Data Separability Testing

We will first approach image classification in terms of testing the statistical separability of the two classes, represented in our case by the invariant characteristics calculated on data from AD and CN patients. First, we will try to discern whether the two populations have equal or different medians by using the non-parametric Mann-Whitney U test. If we find a characteristic, where the medians of the two populations differ, we may argue it is possible to separate the data by using this characteristic. We will also try to test whether the data come from normally distributed populations. In the case that they do, we will be able to use the Welch's t-test to discern whether the populations have the same mean. Using the same logic as with the Mann-Whitney U test, we will also try to find characteristics, which would allow for data separation.

4.1.1 Mann-Whitney U Test

Let $X_1, X_2, \ldots, X_n$ be an *i.i.d.* sample from X and $Y_1, Y_2, \ldots, Y_m$ an *i.i.d.* sample from Y and let both samples be independent of each other and at least ordinal, i.e. possible to be ordered with respect to size. We test for the null hypothesis H_0: the distributions of both populations are identical, against the alternative hypothesis H_a: the distributions of the two populations are not identical. The Mann-Whitney U statistic is defined as $U = \min(U_X, U_Y)$ with U_X and U_Y being defined by the formulas [26]:

$$
U_X = R_X - \frac{n(n+1)}{2}
$$

$$
U_Y = R_Y - \frac{m(m+1)}{2},
$$

$$(4.1)$$

where R_X and R_Y are the sums of the ranks in samples X and Y respectively:

$$
R_X = \sum_{i=1}^{n} r(X_i)
$$

$$
R_Y = \sum_{j=1}^{m} r(Y_j),
$$

$$(4.2)$$

where $r(\cdot)$ is the rank of the data from X or Y assuming that the data (both samples put together) are ordered from the smallest value to the largest. In the case of tied ranks, we assign a rank value equal to the midpoint of unadjusted rankings, e.g. the adjusted ranks of a sample $(1, 2, 2, 2, 3)$ are $(1, 3, 3, 3, 5)$, where the unadjusted ranks would be $(1, 2, 3, 4, 5)$.

The statistic U tends to be tabulated for sample sizes $n \leq 20$. For larger samples, the statistic U is approximately normally distributed. The standardized value z, an approximation of the standard normal deviate, whose significance can be checked against tables of normal distribution, is given by the formula [27][28]:

$$
z = \frac{U - \mu_U}{\sigma_U},
$$

$$(4.3)$$

where μ_U and σ_U are the mean and the standard deviation of U given by the formulas [27][28]:

$$
\mu_U = \frac{nm}{2}
$$

$$(4.4)$$

and

$$\sigma_U = \sqrt{\frac{nm}{12}\left((N+1) - \frac{\sum_{k=1}^{K}(t_k^3 - t_k)}{N(N-1)}\right)}, \tag{4.5}$$

where $N = n + m$, $K \leq N$ is the total number of unique ranks with ties and t_k is the number of ties for the k-th rank.

4.1.2 Liliefors Test

The Liliefors test is a normality test based on the goodness of fit one sample Kolmogorov-Smirnov test. The Kolmogorov-Smirnov test statistic D_n is given by the formula [29]:

$$D_n = \sup_x |F_n(x) - F(x)|, \tag{4.6}$$

where $F(x)$ is a given cumulative distribution function (CDF) (e.g. normal CDF) and $F_n(x)$ is an empirical distribution function of an *i.i.d.* sample $X_1, X_2, \ldots, X_n$ from X [29]:

$$F_n(x) = \frac{1}{n}\sum_{i=1}^{n} I_{(-\infty, x]}(X_i), \tag{4.7}$$

The Liliefors test tests for the null hypothesis H_0: the data come from a normally distributed population, against the alternative hypothesis H_a: the data do not come from a normally distributed population. The Liliefors statistic D_n is similar to the Kolmogorov-Smirnov statistic [30]:

$$D_n = \sup_x |F_n(x) - F^*(x)|, \tag{4.8}$$

where $F^*(x)$ is now the CDF of the normal distribution with $\mu = \mu_X$, where μ_X is the sample mean, and $\sigma^2 = s^2$, where s^2 is the sample variance with a denominator $n-1$ to eliminate bias. Since the normal CDF has been moved closer to the data (because of the estimation based on sample mean and variance), the value of the statistic D_n has been made smaller than it would have been if simply testing using the classical Kolmogorov-Smirnov test (standardised sample against standard normal CDF). The null distribution of the test statistic, the Liliefors distribution, is thus stochastically smaller than the Kolmogorov-Smirnov distribution. Tables for Liliefors distribution have been computed using Monte Carlo methods [31].

4.1.3 Welch's t-Test

Let $X_1, X_2, \ldots, X_n$ be an *i.i.d.* sample from X and $Y_1, Y_2, \ldots, Y_m$ an *i.i.d.* sample from Y. Let us assume that the two populations come from a normal distribution,

and let us not assume that the populations have equal variances. We test the null hypothesis (assuming two-tailed test) H_0: the means of both populations are equal, against the alternative hypothesis H_a: the means of the two populations are not equal. The t statistic of the two-sample Welch's test is given by the formula [32]:

$$t = \frac{\mu_X - \mu_Y}{\sqrt{\dfrac{s_X^2}{n} + \dfrac{s_Y^2}{m}}}, \tag{4.9}$$

The μ_X and μ_Y are population means. The s_X and s_Y are the sample standard deviations corrected by the denominator $n-1$.

We can see that Welch's test is a version of Student's t-test for samples assumed not to have equal variances. The degrees of freedom ν are approximated using the Welch-Satterthwaite equation [33]:

$$\nu \approx \frac{\left(\dfrac{s_X^2}{n} + \dfrac{s_Y^2}{m}\right)^2}{\dfrac{s_X^4}{n^2(n-1)} + \dfrac{s_Y^4}{m^2(m-1)}}, \tag{4.10}$$

where $n-1$ is the degrees of freedom associated with the variance estimate of the sample X.

The statistics t and ν can be used with the t-distribution to test the hypotheses. Welch's t-test is more robust than Student's t-test [32][34] and its power comes close to that of Student's t-test even when the populations have equal variances and similar sample sizes are chosen [32]. It has also been suggested that Welch's t-test performs as well as Mann-Whitney U test when sample variances are equal and better when variances are unequal [32]. Assuming normal distribution of the data samples, it is thus considered good practice to prefer the Welch's t-test to Mann-Whitney U test.

4.1.4   Controlling False Discovery Rate (FDR)

The false discovery rate (FDR), first conceptualized by Benjamini and Hochberg in [35], signifies the rate of Type I errors - false positives (FP) - in null hypothesis testing [35]:

$$\text{FDR} = \frac{\text{FP}}{\text{FP} + \text{TP}}, \tag{4.11}$$

assuming a multiple testing problem, where TP signifies true positive cases (see 4.4). The FDR was invented as a more powerful alternative to family-wise error

rate (FWER) [35] to identify the important few from the unimportant many hypotheses tested. The FDR-controlling procedure were then designed to provide less stringent control of Type I errors when compared to previously predominant FWER procedures.

Benjamini-Yekutieli Controlling Procedure

Let $H_1, H_2, \ldots, H_m$ be null hypotheses and $P_1, P_2, \ldots, P_m$ their corresponding p-values. The Benjamini-Yekutieli procedure controls the FDR under arbitrary dependence assumptions and defines the acceptance threshold refinement by finding the largest index $k \in \mathbb{N}$ such that [36]:

$$P_{(k)} \leq \frac{k}{m \cdot c(m)} \alpha, \tag{4.12}$$

where $P(k)$ is the k-th p-value, assuming an ordering from smallest to largest, α is the chosen level of significance (most commonly we choose $\alpha = 0.05$), and $c(m)$ is (in the case of arbitrary dependence) the harmonic number:

$$c(m) = \sum_{i=1}^{m} \frac{1}{i}, \tag{4.13}$$

which can be approximated using the Taylor series expansion and the Euler-Mascheroni constant $\gamma = 0.57721 \ldots$ [37]:

$$\sum_{i=1}^{m} \frac{1}{i} \approx \ln(m) + \gamma + \frac{1}{2m} \tag{4.14}$$

After finding the threshold k_0, the largest k for which (4.12) holds, the null hypotheses with p-values $P_{(1)}, P_{(2)}, \ldots, P_{(k_0)}$ are rejected, while the null hypotheses with p-values $P_{(k_0+1)}, P_{(k_0+2)}, \ldots, P_{(m)}$ are accepted.

4.2   Data Transformations

Before we begin using binary classifiers, we must first appropriately transform the results of calculating characteristics of image invariants. We will test three distinct approaches. First, the full table of all global characteristics (see 3.3) of all calculated invariants (see 3.2) will be used as inputs of our binary classifiers. We will also test the one-dimensional approach - using each feature of each invariant as the single input. We expect lower classification accuracy when using these trivial inputs, however, they will be interesting to compare with the results of statistical testing from 4.1. The last approach will be to use principal component analysis (PCA) data whitening procedure on the full data table and then using the whitened data with reduced dimension as inputs of binary classifiers.

4.2.1 Input Image Normalization

Before we proceed with any operations on the input images, we must first perform several transforming operations. First, we will normalize the SPECT image f: $\mathcal{S} \to \mathbb{R}$ by dividing it by the maximum intensity found in the image:

$$
\text{f}_{\text{norm}}(\boldsymbol{i}) = \frac{\text{f}(\boldsymbol{i})}{\max[\text{f}(\cdot)]}, \quad \forall \boldsymbol{i} \in \mathcal{S}, \tag{4.15}
$$

where f_{norm}: $\mathcal{S} \to [0, 1]$ is the normalized SPECT image.

Following the normalization, we will use a binary cut-off mask based on an input threshold $\theta \in [0, 1]$, cutting off any normalized image values smaller than θ. Such normalized and reduced images will serve as the basis of invariant characteristic calculation (see Chapter 3).

4.2.2 Full Characteristics

The calculated invariant global characteristics will be initially saved into an array with several dimensions based on configuration parameters such as the number of input images, values of binary mask cut-off threshold θ, the radius of Zernike polynomial ρ, and the utilised fast global filters (see Chapter 5 and Appendix A for more details). This higher dimensional array will be transformed into a two-dimensional array using the MATLAB function `reshape`. We can now image the array as a matrix of size $n_{\text{img}} \times n_{\text{chars}}$ signifying the number of images times the number of global characteristics, where each element indexed (i, j) will be the j-th global invariant characteristic of the i-th input image. As we will be calculating the features of CN and AD images separately, we will then combine these two transformed two-dimensional arrays for both classes into a two-dimensional array of size $n_{\text{img}} \times n_{\text{chars}}$, where now $n_{\text{img}} = (n_{\text{AD}} + n_{\text{CN}})$ with n_{AD}, n_{CN} being the number of AD and CN images. Such two-dimensional array will serve as the input for binary classifiers using the full transformed data approach.

4.2.3 One-Dimensional Characteristics

To use this trivial one-dimensional approach, we will utilise the prepared two-dimensional full data array from 4.2.2. For the one-feature classification, we will iterate through the columns (as each column is a vector of values of a one global characteristic for all AD and CN images) of the calculated two-dimensional array. We expect this approach to yield lower classification accuracy than utilizing the approaches from 4.2.2 and 4.2.4. However, it will be useful when comparing the true classification accuracy of classifiers trained on single characteristic with the results obtained by statistically testing the data separation based on individual characteristics (see 4.1). In other words, we may see whether the characteristics judged to

be good for data separability (if indeed there are any) also act as particularly good inputs of binary classifiers.

4.2.4 Principal Component Analysis (PCA)

We will start the data-whitening approach with a two-dimensional array of all features prepared according to 4.2.2. To escape the potential curse of dimensionality (depending on number of invariants, number of features, binary mask cut-offs, etc. we may be dealing with an array featuring hundreds or even thousands of distinct characteristics for each image, see also 6.1) we will reduce the dimensionality by deriving a significantly smaller number of variables (principal components) with a hope of maximizing variance and preserving most of the information present in the original data [38][39].

Let $\boldsymbol{X}_0$ be a $n \times p$ matrix, whose each row corresponds to all $p \in \mathbb{N}$ characteristics of one image and each column corresponds to all values of one particular characteristic of all $n \in \mathbb{N}$ images. Let $\boldsymbol{X} = \boldsymbol{X}_0 - \boldsymbol{J}_{n,1}\boldsymbol{\mu}^{\mathrm{T}}$, where $\boldsymbol{\mu} \in \mathbb{R}^p$ is the centre of mass of $\boldsymbol{X}_0$, be the matrix of deviations from the mean along each column. The $p \times p$ covariance matrix $\boldsymbol{S}$ of $\boldsymbol{X}$ can be expressed as [39]:

$$\boldsymbol{S} = \frac{1}{n-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} \tag{4.16}$$

Let $\boldsymbol{S} = \boldsymbol{V}\boldsymbol{D}\boldsymbol{V}^{-1}$ be the eigendecomposition of the matrix $\boldsymbol{S}$, where $\boldsymbol{V}$ is a $p \times p$ matrix, whose each column corresponds to one of the mutually orthonormal unit eigenvectors of $\boldsymbol{S}$ that can be interpreted as the principal axes of a p-dimensional ellipsoid fitted to the original data [39], and $\boldsymbol{D}$ is the $p \times p$ diagonal matrix of eigenvalues of $\boldsymbol{S}$. Let $k, l \in \mathbb{N}$, $k, l \leq p$. Following from the above, the elements of the diagonal matrix $\boldsymbol{D}$ are formally expressed:

$$d_{kl} = \begin{cases} \lambda_{(k)} & \text{if } k = l \\ 0 & \text{otherwise,} \end{cases} \tag{4.17}$$

where $\lambda_{(k)}$ is the k-th largest eigenvalue of the covariance matrix $\boldsymbol{S}$.

The k-th principal component of $\boldsymbol{X}$ is now the k-th column of the matrix $\boldsymbol{V}$, i.e. the eigenvector corresponding to the k-th largest eigenvalue $\lambda_{(k)}$ [39]. Let us consider the first K principal components, $K \in \mathbb{N}, K \leq \mathrm{rank}(\boldsymbol{X}_0)$, thus creating the $p \times K$ matrix $\boldsymbol{W}$ from the matrix $\boldsymbol{V}$. The new data matrix $\boldsymbol{Y}$, consisting of the original data projected onto a subspace of the first K principal components, is [39]:

$$\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{W} \tag{4.18}$$

We will compute several iterations of the matrix $\boldsymbol{Y}$ based on differing numbers of principal components K and utilise each of these matrices $\boldsymbol{Y}$ as an input for a binary classifier, which will allow us to compare classification accuracy with other data inputs (especially the full data) as well as find the optimal number of components K_{opt}.

4.3 Classifiers

In this section, we provide a brief and by no means exhaustive overview of the several types of classifiers we will be using for the task of binary image classification, assuming supervised learning. We will first describe the two discriminant analysis methods - the linear discriminant analysis (LDA) and quadratic discriminant analysis (QDA). We will continue with the k-nearest neighbours (KNN), the support vector machines (SVMs) of the polynomial and Gaussian variety, and finish the chapter with the artificial neural networks (ANNs).

4.3.1 Discriminant Analysis

The discriminant analysis assumes that the data are perfectly separable by linear hyperplanes (or affine sets) called *decision boundaries*, which divide the space into regions of constant classification. Let us assume that the PDF of the k-th class in the *i.i.d.* sample, $k \in \mathbb{N}$, $k \in \{1, 2, \ldots, K\}$, where $K \in \mathbb{N}$ is the total number of classes, is a multivariate Gaussian [40][41]:

$$f_k(\boldsymbol{x}) = \frac{1}{(2\pi)^{\frac{p}{2}} \det(\boldsymbol{\Sigma}_k)^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_k)^{\mathrm{T}} \boldsymbol{\Sigma}_k^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_k)\right), \qquad (4.19)$$

where $\boldsymbol{x} \in \mathbb{R}^p$ is the classification data matrix $n \in \mathbb{N}$ data inputs of $p \in N$ features coming from a normally distributed population, $(\boldsymbol{x} - \boldsymbol{\mu}_k)^{\mathrm{T}} \boldsymbol{\Sigma}_k^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_k)$ is the Mahalanobis distance [40][41], $\boldsymbol{\mu}_k$ is the mean vector of k-th class and $\boldsymbol{\Sigma}_k$ the covariance matrix of the k-th class.

It is possible to show [40] that we may minimize the risk (expected loss) related to classifying the data from normal distribution by using a rule of choosing such a class prediction $\hat{y} \in \mathbb{N}$, $\hat{y} \in \{1, 2, \ldots, K\}$ that [40]:

$$f_{\hat{y}}(\boldsymbol{x})\pi_{\hat{y}} = \max_{1 \le k \le K} f_k(\boldsymbol{x})\pi_k, \qquad (4.20)$$

where π_k is the unconditional prior probability of observing data coming from the class k. The discriminant score for the k-th class, is defined by the formula [40]:

$$d_k(\boldsymbol{x}) = (\boldsymbol{x} - \boldsymbol{\mu}_k)^{\mathrm{T}} \boldsymbol{\Sigma}_k^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_k) + \ln \det(\boldsymbol{\Sigma}_k), \qquad (4.21)$$

Using (4.21) results in the quadratic discriminant analysis (QDA). Using an additional assumption that all covariance matrices are equal, $\boldsymbol{\Sigma}_k = \boldsymbol{\Sigma}$, $1 \leq k \leq K$, results in the LDA [40][41]. Substituting (4.19) into (4.20) leads to the overall classification rule [40]:

$$\hat{y} = \min_{1 \leq k \leq K} d_k(\boldsymbol{x}) \qquad (4.22)$$

In practical uses, we may often deal with a sample size that is smaller than the number of characteristics of each data vector. In such cases, it is not possible to invert the covariance matrix, and we resort to calculating a pseudo-inverse (such as the Moore-Penrose pseudo-inverse [42]). To achieve a better numerical stability, we may first consider replacing all the class samples' covariance matrices $\boldsymbol{S}_k$ by a pooled covariance matrix $\boldsymbol{S}$ [43]:

$$\boldsymbol{S} = \frac{\sum_{k=1}^{K} n_k \boldsymbol{S}_k}{\sum_{k=1}^{K} n_k}, \qquad (4.23)$$

where n_k is the number of data coming from the k-th class. The regularized sample covariance matrix is then defined $\boldsymbol{S}_{\text{reg}}(\lambda)$ [43][44]:

$$\boldsymbol{S}_{\text{reg}}(\lambda) = \boldsymbol{S} + \lambda \boldsymbol{I}, \qquad (4.24)$$

where $\lambda \in \mathbb{R}^+$ is the regularization (bias) parameter. The (4.21) thus becomes:

$$d_k(\boldsymbol{x}) = (\boldsymbol{x} - \boldsymbol{\mu}_k)^{\text{T}} \boldsymbol{S}_{\text{reg}}(\lambda)^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_k) + \ln \det(\boldsymbol{S}_{\text{reg}}(\lambda)), \qquad (4.25)$$

4.3.2 k-Nearest Neighbours (KNN)

Let $\mathcal{T}$ be the classification training set. Let $\boldsymbol{x} \in \mathbb{R}^p$, $p \in \mathbb{N}$ be a classification data vector, $\hat{y} \in \mathbb{N}$, $\hat{y} \in \{1, 2, \ldots, K\}$ its predicted class and $y^* \in \mathbb{N}$, $\leq y^* \in \{1, 2, \ldots, K\}$ its true class out of $K \in \mathbb{N}$ total classes. Let $N_k(\boldsymbol{x})$ be the k-neighbourhood of $k \in \mathbb{N}$ vectors $\boldsymbol{x}_i$, $i \in \mathbb{N}, i \in \{1, 2, \ldots, k\}$, from $\mathcal{T}$, closest to the input $\boldsymbol{x}$ according to a chosen metric function (for continuous variables commonly the Euclidean distance). We define the input class prediction $\hat{y}$ as [41]:

$$\hat{y} = \left\lfloor \frac{1}{k} \sum_{\boldsymbol{x}_i \in N_k(\boldsymbol{x})} y_i^* \right\rceil, \qquad (4.26)$$

and thus understand that the predicted class $\hat{y}$ of input $\boldsymbol{x}$ is the average of the true class values of its k closest neighbours rounded to the closest integer. We may also reformulate the average to an equivalent histogram criterion - we assign the input to the class most frequently represented in the neighbourhood of the input $\boldsymbol{x}$.

4.3.3 Support Vector Machines (SVM)

The support vector machines allow for constructing an optimal *hard margin* affine set in the case of perfectly linearly separable data, as well as considering the more general, *soft margin* case, when the classification data are not linearly separable. Let $\boldsymbol{x}_i \in \mathbb{R}^p$, $i \in \mathbb{N}$, $i \in \{1, 2, \ldots, n\}$, $p \in \mathbb{N}$ be vectors of $n \in \mathbb{N}$ data inputs consisting of p features each. Let $\boldsymbol{y}^*$ be a vector of true classes, where $y_i^* \in \{-1, 1\}$, $1 \leq i \leq n$. We will first discuss the case of hard margin, i.e. we will assume that for the training set $\mathcal{T}$ there exist an affine set defined by a vector $\boldsymbol{w} \in \mathbb{R}^p$, $\|\boldsymbol{w}\| = 1$ and a bias scalar $b \in \mathbb{R}$ that perfectly separates the data. In other words, such weights and a bias can be found that the following holds [41][45]:

$$(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b) y_i^* > 0 \quad i = 1, 2, \ldots, n, \tag{4.27}$$

We are now trying to find the largest margin M between the two classification sets. Hence, we may also call this problem the *max-margin* and optimize [41]:

$$\max_{b,\, \boldsymbol{w},\, \|\boldsymbol{w}\|=1} M$$
$$\text{subject to } (\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b) y_i^* \geq M, \quad i = 1, 2, \ldots, n \tag{4.28}$$

It is also possible to convert this problem into a minimization form [41]:

$$\min_{b,\, \boldsymbol{w}} \|\boldsymbol{w}\|$$
$$\text{subject to } (\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b) y_i^* \geq 1, \quad i = 1, 2, \ldots, n, \tag{4.29}$$

as $M = \frac{1}{\|\boldsymbol{w}\|}$. Before we discuss the solution, let us also define the soft margin case, where we cannot assume that the classes are separable in the feature space. In [46] Cortes and Vapnik propose to introduce a slack variable vector $\boldsymbol{\xi} \in \mathbb{R}^n$, $\xi_i \geq 0$, $i = 1, 2, \ldots, n$, which ensures that we still maximize the margin while taking into account the fact that due to the class overlap some data might end up on the incorrect side of the margin. Therefore, we rephrase (4.29) as: [41][45][46]:

$$\min_{b,\, \boldsymbol{w}} \|\boldsymbol{w}\|$$
$$\text{subject to } \begin{cases} (\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b) y_i^* \geq 1 - \xi_i, & i = 1, 2, \ldots, n \\ \xi_i \geq 0, & i = 1, 2, \ldots, n \\ \sum_{i=1}^n \xi_i \leq C \end{cases} \tag{4.30}$$

where $C > 0$ is a fixed bounding constant that prevents the relaxed constraints from being trivially satisfied. The formulation (4.30) is a quadratic and convex optimization problem, which can be solved using Lagrange multipliers [41]. This problem is better reformulated as [45][47]:

$$\min_{b,\boldsymbol{w}} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\xi_i$$

$$\text{subject to} \begin{cases} (\langle\boldsymbol{w},\boldsymbol{x}_i\rangle + b)y_i^* \geq 1 - \xi_i, & i = 1,2,\ldots,n \\ \xi_i \geq 0, & i = 1,2,\ldots,n \end{cases} \tag{4.31}$$

where $C = \infty$, in the case of perfect class separability. This formulation allows us to finally arrive at the Lagrangian objective function L_{P} [41][47]:

$$L_{\mathrm{P}} = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i[(\langle\boldsymbol{w},\boldsymbol{x}_i\rangle + b)y_i^* - (1 - \xi_i)] - \sum_{i=1}^{n}\mu_i\xi_i, \tag{4.32}$$

which we minimize with respect to b, $\boldsymbol{w}$ and ξ_i. By setting the derivatives equal to zero, we obtain [41][47]:

$$\sum_{i=1}^{n}\alpha_i y_i^* \boldsymbol{x}_i = \boldsymbol{w},$$

$$\sum_{i=1}^{n}\alpha_i y_i^* = 0, \tag{4.33}$$

$$C - \mu_i = \alpha_i, \quad i = 1,2,\ldots,n,$$

under constraints α_i, μ_i, $\xi_i \geq 0$, $i = 1,2,\ldots,n$. By substituting (4.33) back into (4.32), we obtain the dual objective function L_{D} [41][47]:

$$L_{\mathrm{D}} = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i^* y_j^* \langle\boldsymbol{x}_i,\boldsymbol{x}_j\rangle, \tag{4.34}$$

which signifies the lower bound of the original objective function. We maximize the dual function under constraints $0 \leq \alpha_i \leq C$ and $\sum_{i=1}^{n}\alpha_i y_i^* = 0$ as well as the Karush-Kuhn-Tucker constraints [41][47]:

$$\alpha_i[(\langle\boldsymbol{w},\boldsymbol{x}_i\rangle + b)y_i^* - (1 - \xi_i)] = 0,$$

$$(\langle\boldsymbol{w},\boldsymbol{x}_i\rangle + b)y_i^* - (1 - \xi_i) \geq 0, \tag{4.35}$$

$$\mu_i\xi_i = 0,$$

Furthermore, we observe from (4.33) that the sought solution has a form:

$$\hat{\boldsymbol{w}} = \sum_{i=1}^{n}\hat{\alpha}_i y_i^* \boldsymbol{x}_i, \tag{4.36}$$

where (coming from the first condition in 4.35) the Lagrangian coefficients $\hat{\alpha}_i$ are non-zero only when the second condition in (4.35) becomes equality. As the observations corresponding to such indices describe the solution $\hat{\boldsymbol{w}}$, they are called the *support vectors*.

SVM Kernels

Using the definitions above, the SVM is still only a linear classifier. We might transform it to a nonlinear classifier by using a nonlinear kernel, which allows us to map the original input space onto a high-dimensional feature space (allowing for better class separation), where we again construct the margin as a linear affine set. This new affine set, however, may be nonlinear in the original input space, thus creating a nonlinear classifier. It is obvious that it is necessary to transform the data $\boldsymbol{x}_i \in \mathbb{R}^p$ by utilizing some basis functions $h_m(\boldsymbol{x})$, $m = 1, 2, \ldots r$, $r \in \mathbb{N}$, where $h(x_i) = (h_1(x_i), h_2(x_i), \ldots, h_m(x_i))$. This converts the initial formulation (4.27) of the problem into [41]:

$$(\langle \boldsymbol{w}, h(\boldsymbol{x}_i) \rangle + b) y_i^* > 0 \quad i = 1, 2, \ldots, n \tag{4.37}$$

The corresponding dual objective function (4.34) is now [41]:

$$L_{\mathrm{D}} = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i^* y_j^* \langle h(\boldsymbol{x}_i), h(\boldsymbol{x}_j) \rangle, \tag{4.38}$$

The kernel trick is that we do not need to specify or even know the transformation function $h(x)$, because we may utilise the kernel formulation of the inner product $\langle h(\boldsymbol{x}_i), h(\boldsymbol{x}_j) \rangle$ [41]:

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle h(\boldsymbol{x}_i), h(\boldsymbol{x}_j) \rangle, \tag{4.39}$$

where, in general, the $K(\boldsymbol{x}, \boldsymbol{y})$ is a symmetric positive definite (PD) kernel function. We will utilise kernels in the form of d-th degree (inhomogeneous) polynomials and the Gaussian radial basis function [41]:

$$K(\boldsymbol{x}, \boldsymbol{y}) = \begin{cases} (\langle \boldsymbol{x}, \boldsymbol{y} \rangle + 1)^d \\ \exp\left(\dfrac{\|\boldsymbol{x} - \boldsymbol{y}\|^2}{2\sigma^2}\right) \end{cases} \tag{4.40}$$

4.3.4 Artificial Neural Networks (ANN)

In our work, we will be utilizing a simple and commonly used single hidden layer neural network. This model can be described as a two-step classification model governed by a chosen nonlinear, differentiable *activation function* $\sigma(\cdot)$. Let $\boldsymbol{x} \in \mathbb{R}^p$, $p \in \mathbb{N}$ be the input vector and $\boldsymbol{y}^*$, $y_i^* \in \{1, 2, \ldots, K\}$, $i = 1, 2, \ldots, n$, where $K \in \mathbb{N}$ is the number of classes, be the vector of target true classes. The model first takes the linear combination of the input vector $\boldsymbol{x}$ with added bias [41][47]:

$$a_j = \sum_{i=1}^{p} w_{ij}^{(1)} x_i + w_{0j}^{(1)}, \quad j = 1, 2, \ldots m, \tag{4.41}$$

where $m \in \mathbb{N}$ is the number of parameters or *neurons* in the input layer of the network, superscript $^{(1)}$ indicates that the weight parameters belong to the first (input) layer, and $w_{ij}^{(1)}, w_{0j}^{(1)}$ are thus the weight and bias parameters of this layer. Each of these derived features a_j are then transformed using the activation function $\sigma(\cdot)$ in the hidden (as these values are not directly observed by the user) layer and again linearly combined using the weight parameters of the output layer of the network [41][47]:

$$b_k = \sum_{j=1}^{m} w_{jk}^{(2)} \sigma(a_j) + w_{0k}^{(2)}, \quad k = 1, 2, \ldots, K, \tag{4.42}$$

where K now signifies the K neurons of the output layer corresponding to the K true classes. Common choices for the activation function include the hyperbolic tangent, or the logistic sigmoid function [47]:

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \tag{4.43}$$

In the case of K-classification, we also use the *softmax* function $s_i(b_k)$ [41]:

$$s_i(b_k) = \frac{\exp(b_k)}{\sum_{l=1}^{K} \exp(b_l)} \tag{4.44}$$

to convert the output values for the i-th input x_i into K-class softmax probability distribution with values $y_{ik} = s_i(b_k)$, where we seek the maximum likelihood, i.e. the class prediction $\hat{y}_i$ of the i-th input data x_i is the maximum likelihood out of all likelihoods y_{ik} that the i-th input belongs to the k-th class, $\hat{y}_i = \arg\max_k y_{ik} = \arg\max_k s_i(b_k)$.

We fit the model by optimizing the weight vector $\boldsymbol{w}$ values, which we achieve by minimizing the cross-entropy error function $E(\boldsymbol{w})$ [41]:

$$E(\boldsymbol{w}) = -\sum_{i=1}^{n} \sum_{k=1}^{K} y_{ik}^* \log(y_{ik}(\boldsymbol{w})), \tag{4.45}$$

where we now write $y_{ik}(\boldsymbol{w})$, because we may observe that the probability values y_{ik} are dependent on weight values from the vector $\boldsymbol{w}$, as is evident from (4.44). This problem is solved by employing the gradient descent approach, which may be summarised as [47]:

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta^{(t)} \nabla E(\boldsymbol{w}^{(t)}), \tag{4.46}$$

where $\boldsymbol{w}^{(t)}$ signifies the value of the weight vector $\boldsymbol{w}$ in the t-th step and $\eta^{(t)} > 0$ is the descent step size or *learning rate* in the t-th step.

4.4 Cross-Validation (CV)

In this section, we will discuss two topics related to measuring the performance of binary classifiers. First, we will give an overview of various classification statistics used for performance evaluation. Second, we will comment on two common CV methods - the leave-one-out and the stratified-k-fold methods, which we will use to assess the performance of our classifiers on an 'independent' data set.

4.4.1 Classification Statistics

Let P, $N \in \mathbb{N}$ be the true number of images corresponding to AD positive and negative (CN) patients. Coming from the definition of the confusion matrix, the true positive (TP) is defined as a result of classification that correctly indicates the presence of AD in the AD patient [48]. Similarly, we define as true negative (TN) the result that correctly indicates the absence of AD in the CN patient. We define as false positive (FP) the result that falsely indicates the presence of AD in a CN patient. The FP results correspond to the Type I error. We define as false negative (FN) the result that falsely indicates the absence of AD in an AD diagnosed patient. The FN results correspond to the Type II error. We will also be taking the TP, TN, FP, FN as signifying the number of results, which belong to each corresponding category. In this manner and based on [48], we will be able to define classification statistics, which will enable us to evaluate the binary classifier performance.

Sensitivity or True Positive Rate (TPR)

The sensitivity, also known as the true positive rate (TPR), is defined by the formula:

$$\text{TPR} = \frac{\text{TP}}{P} \tag{4.47}$$

The TPR is bounded by the interval $[0, 1]$. The ideal classification has a value of TPR equal to one.

Specificity or True Negative Rate (TNR)

The specificity, also known as the true negative rate (TNR), is defined by the formula:

$$\text{TNR} = \frac{\text{TN}}{N} \tag{4.48}$$

The TNR is bounded by the interval $[0, 1]$. The ideal classification has a value of TNR equal to one.

False Positive Rate (FPR)

The false positive rate (FPR) is defined by the formula:

$$\text{FPR} = \frac{\text{FP}}{N} = 1 - \text{TNR} \tag{4.49}$$

The FPR is bounded by the interval $[0, 1]$. The ideal classification has a value of FPR equal to zero.

False Negative Rate (FNR)

The false negative rate (FNR) is defined by the formula:

$$\text{FNR} = \frac{\text{FN}}{P} = 1 - \text{TPR} \tag{4.50}$$

The FNR is bounded by the interval $[0, 1]$. The ideal classification has a value of FNR equal to zero.

Critical sensitivity se^*

We define the critical sensitivity se^* as the minimum of TPR and TNR [49]:

$$se^* = \min\{\text{TPR}, \text{TNR}\} \tag{4.51}$$

The se^* is bounded by the interval $[0, 1]$. The ideal classification has a value of se^* equal to one.

Precision or Positive Predictive Value (PPV)

The precision, also known as the positive predictive value (PPV), is defined by the formula:

$$\text{PPV} = \frac{\text{TP}}{\text{TP} + \text{FP}} = 1 - \text{FDR}, \tag{4.52}$$

where FDR is the false discovery rate defined in (4.11). The PPV is bounded by the interval $[0, 1]$. The ideal classification has a value of PPV equal to one.

Negative Predictive Value (NPV)

The negative predictive value (NPV) is defined by the formula:

$$\text{NPV} = \frac{\text{TN}}{\text{TN} + \text{FN}} = 1 - \text{FOR} \tag{4.53}$$

The NPV is bounded by the interval $[0, 1]$. The ideal classification has a value of NPV equal to one.

False Omission Rate (FOR)

The false omission rate (FOR) is defined by the formula:

$$\text{NPV} = \frac{\text{FN}}{\text{TN} + \text{FN}} = 1 - \text{NPV} \tag{4.54}$$

The FOR is bounded by the interval $[0, 1]$. The ideal classification has a value of FOR equal to zero.

Positive Likelihood Ratio (LR+)

The positive likelihood ratio (LR+) is defined by the formula:

$$\text{LR+} = \frac{\text{TPR}}{\text{FPR}} \tag{4.55}$$

The LR+ is bounded by the interval $[0, \infty)$. The ideal classification has a value of LR+ approaching infinity.

Negative Likelihood ratio (LR−)

The negative likelihood ratio (LR−) is defined by the formula:

$$\text{LR−} = \frac{\text{FNR}}{\text{TNR}} \tag{4.56}$$

The LR− is bounded by the interval $[0, \infty)$. The ideal classification has a value of LR− equal to zero.

Prevalence Threshold (PRT)

The prevalence threshold (PRT) is defined by the formula [50]:

$$\text{PT} = \frac{\sqrt{\text{FPR}}}{\sqrt{\text{TPR}} + \sqrt{\text{FPR}}} \tag{4.57}$$

The PRT is bounded by the interval $[0, 1]$. The ideal classification has a value of PRT equal to zero.

Threat Score (TS)

The threat score (TS), also known as the Jaccard index, is defined by the formula [51]:

$$\text{TS} = \frac{\text{TP}}{\text{TP} + \text{FN} + \text{FP}} \tag{4.58}$$

The TS is bounded by the interval $[0, 1]$. The ideal classification has a value of TS equal to one.

Accuracy (ACC)

The accuracy (ACC) is defined by the formula:

$$\text{ACC} = \frac{\text{TP} + \text{TN}}{P + N} \tag{4.59}$$

The ACC is bounded by the interval $[0, 1]$. The ideal classification has a value of ACC equal to one.

Balanced Accuracy (BACC)

The balanced accuracy (BACC) is defined by the formula:

$$\text{BACC} = \frac{\text{TPR} + \text{TNR}}{2} \tag{4.60}$$

The BACC is bounded by the interval $[0, 1]$. The ideal classification has a value of BACC equal to one.

F_1 Score

The F_1 score is defined by the formula:

$$F_1 = \frac{2 \cdot \text{PPV} \cdot \text{TPR}}{\text{PPV} + \text{TPR}} = \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}} \tag{4.61}$$

The F_1 is bounded by the interval $[0, 1]$. The ideal classification has a value of F_1 equal to one.

Matthews Correlation Coefficient (MCC)

The Matthews correlation coefficient (MCC) is defined by the formula:

$$\text{MCC} = \frac{\text{TP} \cdot \text{TN} - \text{FP} \cdot \text{FN}}{\sqrt{(\text{TP} + \text{FP})(\text{TP} + \text{FN})(\text{TN} + \text{FP})(\text{TN} + \text{FN})}} \tag{4.62}$$

The MCC is bounded by the interval $[-1, 1]$. The ideal classification has a value of MCC equal to one.

Fowlkes-Mallows Index (FM)

The Fowlkes-Mallows index (FM) is defined by the formula [52]:

$$\text{FM} = \sqrt{\text{PPV} \cdot \text{TPR}} \tag{4.63}$$

The FM is bounded by the interval $[0, 1]$. The ideal classification has a value of FM equal to one.

Diagnostic Odds Ratio (DOR)

The diagnostic odds ratio (DOR) is defined by the formula:

$$\text{DOR} = \frac{\text{LR}+}{\text{LR}-} \tag{4.64}$$

The DOR is bounded by the interval $[0, \infty)$. The ideal classification has a value of DOR approaching infinity.

4.4.2   Leave-One-Out Technique

Leave-one-out cross-validation is the special case of the k-fold cross-validation, which divides the $n \in \mathbb{N}$ data inputs into $k \in \mathbb{N}$, $k \leq n$ approximately equal sized parts (folds) [41]. In each iteration of the cross-validation method, $k - 1$ folds are used to train the chosen classifier while the remaining one fold is used for prediction [41], i.e. the data are treated as being from an unknown, independent set, which we try to classify using the trained classifier. In the case of leave-one-out, we set $k = n$ and thus treat every classification input as a separate fold. In every iteration, we therefore use $n - 1$ data inputs to train the classifier, which we use to predict the class of the one remaining input. The leave-one-out cross-validation estimator is approximately unbiased for the true prediction error, but may suffer from high variance because we might use a large number of similar folds [41].

4.4.3   Stratified-k-Fold Technique

The stratified-k-fold cross-validation is akin to the k-fold method with the added condition that we try to construct the folds in a manner that every fold contains

roughly the same number of data inputs from both classes. To illustrate, if we have a hundred classification inputs and choose one of the commonly recommended k-fold values, e.g. $k = 5$ [41], we will divide the data into five folds of twenty values each. We will thus try to construct the folds so that each of the folds contains ten AD patients and ten CN patients. By adding this condition, we ensure that each fold is representative of the whole dataset and prevent situations when we could train the classifier on unbalanced data folds.

Chapter 5

Implementation

In this chapter, we provide a non-exhaustive overview of the MATLAB [53] implementation of our function library. The library can be chiefly divided into seven more or less independent components, which are: reading inputs, global filtering, invariant calculation, data separability testing, data transformations, image classification, and saving results. In the following chapter, we will go over all of these parts of our library and describe some of the implemented functions. We will not directly describe any code excerpts in this chapter. A brief commentary on the most interesting parts of the code is instead the topic of Appendix A.

5.1   Inputs, Configurations, Results

In this section, we provide an overview of the functions necessary to read the inputs, as well as the two classes defined for the purposes of our library. We may see such general overview of in the tabs. 5.1 and 5.2.

Table 5.1: Functions used in the data reading part of the library.

Function	Description
`testDriver`	Sequentially calls functions from the library
`loadImage`	Loads a 3D SPECT image
`normSpectImage`	Normalizes a 3D SPECT image
`calcBinMask`	Calculates a binary mask of a 3D SPECT image

As we can see, the functions we consider to be related to reading of inputs cover loading and normalizing a SPECT image, calculating a binary mask, which is used to filter out non-essential values based on an input threshold and a driver function, which is used to run the whole (all seven parts of the) library.

Table 5.2: Configuration and result classes. See also A.1.

Class	Description
configurationClass	Predefined configurations used to run the library
resultClass	Results of invariant calculation and image classification

We define two classes specifically for purposes of our library - the `configuration-Class` containing predefined testing configurations (number of invariants to be calculated, classifier parameters' settings, etc.), and the `resultClass` used to store results, which will be shown and discussed in Chapter 6. We chose to predefine several configurations as it would otherwise be rather cumbersome to enter a new configuration in every run (configurations are composed of many settings related to feature calculation as well as image classification) and the fact that we assumed there would be only a handful of truly interesting configurations, which would achieve good classification results. Still, these configurations are easily ready for editing in the corresponding `configurationClass` file.

testDriver

Table 5.3: `testDriver` sequentially calls functions from the library.

Function	Calling	
testDriver	results = testDriver(conf_number)	
Input/Output	Data Type	Description
conf_num	integer	Runs the `testDriver` using the configuration `conf_number`
results	resultClass	Instance of the `resultClass` class

The `testDriver` function documented in the tab. 5.3 is the primary driver function of the library. Based on user input, it loads one of the predefined configurations from the `configurationClass` and then sequentially runs all seven (unless user specifies otherwise in configuration) principal components of the library mentioned above. Finally, the function returns an instance of the `resultClass` containing results calculated during the run of the used configuration.

loadImage

The function `loadImage` documented in the tab. 5.4 is used to load one image based on the input filename. This function is thus used in a cycle to load all input images. As we can see, we also return a boolean showing whether the image we have loaded is a SPECT image (recognized by reading a `.img` file). This is used later to decide whether we will also normalize the image (see below). The part of the code relevant to loading a 3D SPECT image can be found in A.1.

Table 5.4: The `loadImage` function loads one SPECT image based on input filename.

Function	Calling	
loadImage	[input_image, spect_bool] = loadImage(image_filename)	
Input/Output	Data Type	Description
image_filename	string	Image filename
input_image spect_bool	3D array of double boolean	Loaded input image Shows whether the image is a SPECT image

normSpectImage

Table 5.5: `normSpectImage` normalizes a 3D SPECT image according to (4.15).

Function	Calling	
normSpectImage	input_image = normSpectImage(input_image, norm)	
Input/Output	Data Type	Description
input_image norm	3D array of double integer $\in \{1, 2\}$	Unnormalized input image Normalization type
input_image	3D array of double $\in [0, 1]$	Normalized image

If `norm = 1`, the function `normSpectImage` documented in the tab. 5.5 normalizes a 3D SPECT image according to the definition (4.15). The image is thus divided by its maximum intensity value and all its values transformed to the interval $[0, 1]$. An alternative is provided by `norm = 2`, when the image is divided by the sum of all of its intensities.

calcBinMask

Table 5.6: `calcBinMask` calculates a binary image mask based on an input threshold `theta` (see 4.2.1).

Function	Calling	
calcBinMask	bin_mask = calcBinMask(X, theta)	
Input/Output	Data Type	Description
X theta	3D array of double $\in [0, 1]$ double $\in [0, 1]$	Normalized input image Binary mask threshold
bin_mask	1D array of boolean	Binary mask

The function `calcBinMask` documented in the tab. 5.6 calculates a binary mask based on input threshold `theta` (see 4.2.1). We may notice that the mask is a

one-dimensional array calculated based on values of the normalized image. We will use this array of booleans for logical indexing of the eventually calculated invariants, i.e. we will eventually only use those values of the calculated invariant, which correspond to `true` values in the binary mask. It should be also obvious that we eventually transform the calculated invariant into a one-dimensional array, which is frequently easier to utilise for some of the operations (such as calculating invariant's global features).

5.2   Global Filtering

In this section, we provide an overview of the functions necessary to filter a three-dimensional image in the frequency domain. These functions are used for both global invariant filtering (see 2, used as a pre-processing for invariant feature calculation) and invariant creation (see 3) because in our approach both are calculated filtering the image/invariant in the frequency domain. We may see such general overview in the tab. 5.7.

Table 5.7: Functions used to filter the image in the frequency domain.

Function	Description
calcFilterKernel	Calculates frequency-domain kernel of the selected type
calcNonLinTran	Calculates nonlinear transformation of a 3D image
filterFunctionFFT	Filters a 3D image in the frequency domain
getPaddingFFT	Pads a 3D image as a preparation for filtering
imageProcessingDriver	Runs functions related to image filtering

calcFilterKernel

Table 5.8: `calcFilterKernel` calculates the frequency-domain filtering kernel.

Function	Calling	
calcFilterKernel	K = calcFilterKernel(X, filter_name, kernel_style, varargin)	
Input/Output	Data Type	Description
X	3D array of double	Input image
filter_name	string	Type of filter
kernel_style	string	Type of kernel
varargin	double	Kernel settings
K	3D array of double	Filtering kernel

The function `calcFilterKernel` documented in the tab. 5.8 calculates the frequency-domain filtering kernel. We have seen the definitions pertaining to these kernels in 2.1 and 2.2. We thus see that we create a three-dimensional array of doubles used for

element-wise multiplication with the original image (convolution in the frequency domain).

calcNonLinTran

Table 5.9: `calcNonLinTran` calculates a nonlinear transformation of the 3D image.

Function	Calling	
calcNonLinTran	X = calcNonLinTran(X, tran_name, direction, varargin)	
Input/Output	Data Type	Description
X	3D array of double	Input image
tran_name	string	Type of nonlinear transformation
direction	string $\in \{\texttt{dir}, \texttt{rev}\}$	Direct or reverse transformation
varargin	double	Transformation settings
X	Transformed 3D image	

The function `calcNonLinTran` documented in the tab. 5.9 calculates a nonlinear transformation of the spatial representation of the 3D image. This is used as both pre- and post-processing of the frequency-domain filtering, depending on the selected 'direction' of the transformation. Definitions related to nonlinear transformations can be seen in 2.3.

filterFunctionFFT

Table 5.10: `filterFunctionFFT` filters a 3D image in the frequency domain. See also A.2.

Function	Calling	
filterFunctionFFT	Y = filterFunctionFFT(X, K, frame_style)	
Input/Output	Data Type	Description
X	3D array of double	Input image
K	3D array of double	Filtering kernel
frame_style	integer $\in \{0, 1, 2\}$	Type of padding
Y	3D array of complex numbers	Filtered image in the spat. domain

The `filterFunctionFFT` documented in the tab. 5.10 is one of the core functions of our library, as it filters a 3D image in the frequency domain using the selected kernel. As such, we show the implementation of this function in A.2, where we also provide more comments on its workings.

Table 5.11: `getPaddingFFT` pads a 3D image in the spatial domain.

Function	Calling	
`getPaddingFFT`	`X = getPaddingFFT(X, style)`	
Input/Output	Data Type	Description
`X`	3D array of double	Input image
`style`	integer $\in \{0, 1, 2\}$	Type of padding
`X`	3D array of double	Padded image

getPaddingFFT

The function `getPaddingFFT` documented in the tab. 5.11 is used by `filterFunctionFFT` (see also A.2) to pad the input image in the spatial domain (FT assumes periodicity) using a selected padding method (see 1.6.1) here symbolized by the variable `style`. When `style` is equal to 0 we use zero padding. When it is equal to 1 we use copy padding. Finally, 2 means usage of mirror padding.

imageProcessingDriver

Table 5.12: `imageProcessingDriver` runs functions `calcFilterKernel`, `calcNonLinTran`, `filterFunctionFFT`.

Function	Calling	
`imageProcessingDriver`	`Y = imageProcessingDriver(X, frame_style, ...` `kernel_style, filter_name, nonlin_trans, varargin)`	
Input/Output	Data Type	Description
`X`	3D array of double	Input image
`frame_style`	integer $\in \{0, 1, 2\}$	Type of padding
`kernel_style`	string	Type of kernel
`filter_name`	string	Type of filter
`nonlin_trans`	string	Type of non-lin. trans.
`varargin`	double	Kernel and non-lin. trans. params.
`Y`	3D array of double	Filtered image

The function `imageProcessingDriver` documented in the tab. 5.12 is the principal driver function of the global filtering part of the library, as it runs functions `calcFilterKernel`, `calcNonLinTran`, and `filterFunctionFFT`. These functions are used as a pre-processing for invariant global feature calculation, i.e. the calculated invariant is first filtered before the feature calculation.

5.3 Invariant Calculation

In this section, we provide an overview of the functions used to calculate rotational invariants and their global features. We may see such general overview in the tab. 5.13.

Table 5.13: Functions used to calculate rotational invariants and their global features.

Function	Description
calcFreqCoords	Calculates 3D spherical coordinates
calcImageFeatures	Calculates all features pertaining to one invariant (see 3.3)
calcInvarFeature	Calculates one global feature
calcInvarFunc	Calculates FFT of a Zernike Polynomial (see 3.1.2)
calcInvars	Calculates rotational invariants by summing (see 3.2)
calcSpherHarmFunc	Calculates value of a spherical harmonic function (see 3.1.1)
featureCalculationDriver	Runs functions related to invariant feature calculation

This part of the library thus deals with calculating spherical coordinates and calculating all the invariants and their features.

calcFreqCoords

Table 5.14: `calcFreqCoords` calculates 3D spherical coordinates for a Zernike polynomial.

Function	Calling	
calcFreqCoords	[k, mu, psi] = calcFreqCoords(M, N, P)	
Input/Output	Data Type	Description
M	integer ≥ 0	Image height
N	integer ≥ 0	Image width
P	integer ≥ 0	Image depth
k	3D array of double ≥ 0	Vector modulus
mu	3D array of double $\in (-\pi, \pi]$	Azimuth angle
psi	3D array of double $\in [0, \pi]$	Polar angle

The function `calcFreqCoords` documented in the tab. 5.14 calculates 3D spherical coordinates for the Zernike polynomial sampling. These coordinates are later used to calculate the FFT of the Zernike polynomial (through spherical harmonic and Bessel functions) needed to create rotational invariants.

Table 5.15: `calcImageFeatures` calculates global features pertaining to one rotational invariant.

Function	Calling

calcImageFeatures	image_features = calcImageFeatures(invars, bin_mask, ...
	  bin_theta, func_name, l_max, n_max, rho, frame_style, ...
	  kernel_style, filter_name, nonlin_trans, features)

Input/Output	Data Type	Description
invars	4D array of double	All calculated invariants
bin_mask	1D array of boolean	Image binary mask
bin_theta	double $\in [0,1]$	Binary mask threshold
func_name	string	Function used to calculate invariants
l_max	integer ≥ 0	Maximum l of invariant (see 3.2)
n_max	integer ≥ 0	Maximum n of invariant (see 3.2)
rho	double > 1	Radius of Zernike polynomial
frame_style	integer $\in \{0, 1, 2\}$	Padding style
kernel_style	string	Filtering kernel style
filter_name	string	Type of filter
nonlin_trans	string	Type of nonlinear transformation
features	1D array of strings	Features to be calculated
image_features	2D array of double	Calculated invariant features

calcImageFeatures

The function `calcImageFeatures` documented in the tab. 5.15 uses a cycle to preprocess calculated invariants (see 5.2) and calculate global selected global features (see 3.3). In our case, the values of calculated invariants tend to be small ($< 10^{-4}$) and we thus also use invariant normalization and a natural logarithm as a preprocessing to increase invariant size and improve classification results.

calcInvarFeature

Table 5.16: `calcInvarFeature` calculates one global invariant feature.

Function	Calling
calcInvarFeature	invar_feature = calcInvarFeature(invar, feature_name, varargin)

Input/Output	Data Type	Description
invar	3D array of double	Rotational invariant
feature_name	string	Feature to be calculated
varargin	integer	Add. params. related to perc. and MAD
invar_feature	double	Calculated invariant feature

The function `calcInvarFeature` documented in the tab. 5.16 is used to define and calculate the invariant features/characteristics (see 3.3). It is used in a cycle (see above) to calculate all selected features of all invariants. The `varargin` signifies the

additional parameters, such as the *k*-th percentile and MATLAB implementation `mad` of MAD related calculation specification (to median).

calcInvarFunc

Table 5.17: `calcInvarFunc` calculates a FFT of a Zernike polynomial.

Function	Calling	
calcInvarFunc	invar_func = calcInvarFunc(func_name, l, m, n, rho, M, N, P)	
Input/Output	Data Type	Description
func_name	string	Function used to calculate invariant
l	integer ≥ 0	See 3.1.1
m	integer $\in \{-l, -l+1, \ldots, l-1, l\}$	See 3.1.1
n	integer ≥ 0	See 3.1.1
rho	double > 1	Zernike polynomial radius
M	integer ≥ 0	Image height
N	integer ≥ 0	Image width
P	integer ≥ 0	Image depth
invar_func	3D array of complex numbers	Sampled FFT of Zernike polynomial

The function `calcInvarFunc` documented in the tab. 5.17 calculates a FFT of a chosen function `func_name` (in our case only a Zernike polynomial) (see 3.1.2) and thus enables us to calculate the rotational invariants by filtering the image in the frequency domain with the calculated, sampled FFT of the Zernike polynomial.

calcInvars

Table 5.18: `calcInvars` calculates a rotational invariant by summing (see 3.2). See also A.3.

Function	Calling	
calcInvars	invars_nl = calcInvars(input_image, func_name, ... frame_style, l_max, n_max, rho)	
Input/Output	Data Type	Description
input_image	3D array of double	Input image
func_name	string	Function used to calculate invariant
frame_style	integer $\in \{0, 1, 2\}$	Padding style
l_max	integer ≥ 0	Maximum *l*. See 3.1.1
n_max	integer ≥ 0	Maximum *n*. See 3.1.1
rho	double > 1	Zernike polynomial radius
invars_nl	4D array of double	Calculated invariants

The function `calcInvars` documented in the tab. 5.18 is the key function of this part of the library, as it calculates the rotational invariants by summing the expansion coefficients (see 3.2). As such, we show the relevant code and provide more comments in A.3.

calcSpherHarmFunc

Table 5.19: `calcSpherHarmFunc` calculates a spherical harmonic function (see 3.1.1).

Function	Calling	
calcSpherHarmFunc	Y_lm = calcSpherHarmFunc(l, m, phi, theta)	
Input/Output	Data Type	Description
l	integer ≥ 0	See 3.1.1
m	integer $\in \{-l, -l+1, \ldots, l-1, l\}$	See 3.1.1
phi	3D array of double $\in (-\pi, \pi]$	Azimuth angle
theta	3D array of double $\in [0, \pi]$	Polar angle
Y_lm	3D array of complex numbers	Spherical harmonic function

The function `calcSpherHarmFunc` documented in the tab. 5.19 is used to calculate the spherical harmonic function, which forms a part of the FFT of a Zernike polynomials. This function incorporates the MATLAB function `legendre`, which is used to calculate Legendre associated polynomials.

featureCalculationDriver

Table 5.20: `featureCalculationDriver` runs functions pertaining to invariant calculation.

Function	Calling	
featureCalculationDriver	imgs_features = featureCalculationDriver(img_datastore, num_imgs, ... bin_theta, norm, func_name, l_max, n_max, rho, frame_style, kernel_style, ... filter_name, nonlin_trans, features)	
Input/Output	Data Type	Description
img_datastore	MATLAB Datastore	Filenames of images to be loaded
num_imgs	integer ≥ 0	Number of images to be loaded
bin_theta	1D array of double $\in [0, 1]$	Binary mask threshold
norm	integer $\in \{1, 2\}$	SPECT normalization type
func_name	1D array of string	Functions used to calculate invariants
l_max	integer ≥ 0	See 3.1.1
n_max	integer ≥ 0	See 3.1.1
rho	1D array of double > 1	Function radius
frame_style	integer $\in \{0, 1, 2\}$	Padding style
kernel_style	1D array of string	Kernel style names
filter_name	1D array of string	Filter style names
nonlin_trans	1D array of string	nonlinear transformation names
features	1D array of string	Features to be calculated
imgs_features	9D array of double	Features of all invariants

The function `featureCalculationDriver` documented in the tab. 5.20 is the driver function of the part of our library that deals with calculating rotational invariants and their global features. It is used to calculate the features of a one kind of input data (AD or CN images). We thus run this driver twice for both AD and CN images to obtain invariant features for both classes.

5.4 Data Separability Testing

In this section, we provide an overview of the functions used for data separability testing (see 4.1). We may see such general overview of in the tab. 5.21.

Table 5.21: Functions used to calculate the data separability.

Function	Description
calcStatImportance	Calculates the number of accepted hypotheses
statTestingDriver	Runs functions needed for data separability testing

calcStatImportance

Table 5.22: `calcStatImportance` counts the number of accepted hypotheses, creates tables, etc.

Function	Calling
calcStatImportance	[stat_importance_func, stat_importance_theta, stat_importance_rho, stat_importance_filt, ... stat_importance_ker, stat_importance_tran, stat_importance_invar, stat_importance_feature, ... stat_importance_all] = calcStatImportance(test_h, func_name, bin_theta, rho, filter_name, ... kernel_style, nonlin_trans, n_max, l_max, features)

Input/Output	Data Type	Description
test_h	8D array of boolean	Booleans whether null hypotheses were accepted
func_name	1D array of string	Functions used for invariant calculation
bin_theta	1D array of double $\in [0,1]$	Binary mask thresholds
rho	1D array of double > 1	Invariant function radius
filter_name	1D array of string	Filters used for image filtering
kernel_style	1D array of string	Kernels used for image filtering
nonlin_trans	1D array of string	nonlinear transformations used during image filtering
n_max	integer ≥ 0	Maximum n. See 3.1.1
l_max	integer ≥ 0	Maximum l. See 3.1.1
features	1D array of string	Names of calculated features
stat_importance_func	MATLAB table	Functions sorted according to relative number of accepted hypotheses
stat_importance_theta	MATLAB table	Binary thresholds sorted acc. to rel. number of accepted hypotheses
stat_importance_rho	MATLAB table	Function radii sorted acc. to rel. number of accepted hypotheses
stat_importance_filt	MATLAB table	Filters sorted acc. to rel. number of accepted hypotheses
stat_importance_ker	MATLAB table	Kernels sorted acc. to rel. number of accepted hypotheses
stat_importance_tran	MATLAB table	Non-lin. transformations sorted acc. to rel. number of accepted hypotheses
stat_importance_invar	MATLAB table	Invariants sorted acc. to rel. number of accepted hypotheses
stat_importance_feature	MATLAB table	Features sorted acc. to rel. number of accepted hypotheses
stat_importance_all	MATLAB table	Configurations sorted acc. to rel. number of accepted hypotheses

The function `calcStatImportance` documented in the tab. 5.22 calculates the statistical 'importance' of the used invariants, features, configurations, etc. It does so by summing the respective number of accepted null hypotheses (see 4.1) in Mann-Whitney U test implemented in MATLAB through function `ranksum` [54] and

Welch's *t*-test implemented in MATLAB through function `ttest2` [54], calculating the relative number of accepted hypotheses, putting them in a table and sorting this table from the highest relative number of accepted hypotheses to smallest. We may understand this process as trying to find the best configuration, which enables us to calculate features that make it possible to separate the AD and CN data into two distinct classes.

`statTestingDiver`

Table 5.23: `statTestingDriver` tests the data for separability (see 4.1).

Function	Calling
statTestingDriver	[pos_imgs_feats_signif, neg_imgs_feats_signif, stat_importance_all_rank, ... stat_importance_all_t, select_num_hyps_false_corr] = statTestingDriver(func_name, ... bin_theta, rho, filter_name, kernel_style, nonlin_trans, n_max, l_max, features, ... pos_imgs_feats, neg_imgs_feats, alpha, fdr_refine)

Input/Output	Data Type	Description
func_name	1D array of string	Functions used for invariant calculation
bin_theta	1D array of double $\in [0, 1]$	Binary mask thresholds
rho	1D array of double > 1	Invariant function radius
filter_name	1D array of string	Filters used for image filtering
kernel_style	1D array of string	Kernels used for image filtering
nonlin_trans	1D array of string	Non-lin. trans. used during image filtering
n_max	integer ≥ 0	Maximum n. See 3.1.1
l_max	integer ≥ 0	Maximum l. See 3.1.1
features	1D array of string	Names of calculated features
pos_imgs_feats	9D array of double	Features of AD images
neg_imgs_feats	9D array of double	Features of CN images
alpha	double $\in [0, 1]$	Test significance level
fdr_refine	boolean	Decide whether to use FDR refinement procedure
pos_imgs_feat_signif	MATLAB table	Statistically significant features of AD images
neg_imgs_feat_signif	MATLAB table	Statistically significant features of CN images
stat_importance_all_rank	MATLAB table	Sorted configurations when using Mann-Whitney U test
stat_importance_all_t	MATLAB table	Sorted configurations when using Welch's *t*-test
select_num_hyps_false_corr	integer ≥ 0	Number of accepted null hypotheses

The function `statTestingDriver` documented in the tab. 5.23 is the principal driver function of the part of our library that deals with testing the data separability. It tests the null hypotheses (see 4.1), uses a FDR refinement procedure and finally sorts the calculation configurations (see above) from best to worst.

5.5 Data Transformation

In this section, we provide an overview of the functions used for data transformations and whitening (see 4.2). We may see such general overview of in the tab. 5.24.

Table 5.24: Functions used to transform and whiten data.

Function	Description
dataWhiteningDriver	Runs functions related to data transformations and whitening.
whitenDataPCA	Whitens data using PCA

The trivial transformations (such as creating a two-dimensional array of full features) are done as a part of the `dataWhiteningDriver`, which also iteratively calls the data whitening function `whitenDataPCA`.

dataWhiteningDriver

Table 5.25: `dataWhiteningDriver` transforms the data to a form that can be used as classifier inputs.

Function	Calling
dataWhiteningDriver	[X_train, X_train_whit] = dataWhiteningDriver(pos_imgs_features, ... neg_imgs_features, func_name, bin_theta, rho, filter_name, kernel_style, ... nonlin_trans, n_max, l_max, features, num_whit_features)

Input/Output	Data Type	Description
pos_imgs_feats	9D array of double	Features of AD images
neg_imgs_feats	9D array of double	Features of CN images
func_name	1D array of string	Functions used for invariant calculation
bin_theta	1D array of double $\in [0,1]$	Binary mask thresholds
rho	1D array of double > 1	Invariant function radius
filter_name	1D array of string	Filters used for image filtering
kernel_style	1D array of string	Kernels used for image filtering
nonlin_trans	1D array of string	Non-lin. trans. used during image filtering
n_max	integer ≥ 0	Maximum n. See 3.1.1
l_max	integer ≥ 0	Maximum l. See 3.1.1
features	1D array of string	Names of calculated features
num_whit_features	1D array of integer ≥ 2	Number of principal components
X_train	2D array of double	All features of all images (see 4.2.2)
X_train_whit	3D array of double	Features after PCA (see 4.2.4)

The function `dataWhiteningDriver` documented in the tab. 5.25 is the principal driver function of the part of our library that deals with transforming data, which will go on to serve as inputs of binary classifiers. This function first transforms the original nine-dimensional invariant features into two-dimensional arrays (where each row corresponds to all calculated features of one image). Afterwards, it iteratively calls the function `whitenDataPCA` to transform the full data into a selected number of principal components (we generally want to try to find the best number of components and thus whiten the full two-dimensional data into a different number of principal components, which means we run `whitenDataPCA` more than once in a cycle using a different number of components each time).

whitenDataPCA

The function `whitenDataPCA` documented in the tab. 5.26 perform the PCA data whitening (see 4.2.4) on input data, which are transformed into `num_dims` principal components.

Table 5.26: `whitenDataPCA` whitens data using PCA.

Function	Calling	
whitenDataPCA	[mean_X, lambda, W, Y] = whitenDataPCA(X, num_dims)	

Input/Output	Data Type	Description
X	2D array of double	Full image features transformed into 2D array
num_dims	integer ≥ 2	Number of principal components
mean_X	1D array of double	Mean of each features
lambda	1D array of double	Principal component percentage
W	2D array of double	Chosen eigenvectors
Y	2D array of double	Principal components

5.6 Image Classification

In this section, we provide an overview of the functions used for binary image classification. We may see such general overview of in the tab. 5.24.

Table 5.27: Functions used for data transformation and whitening.

Function	Description
classANN	Classifies images using trained ANN MATLAB model [54]
classificationDriver	Runs functions related to image classification
classImages	Runs functions related to cross-validation
classKNN	Classifies images using KNN
classLDA	Classifies images using LDA
classQDA	Classifies images using QDA
classSVM	Classifies images using SVM MATLAB model [54]
createClassStatsTable	Creates table of classification statistics
trainANN	Trains ANN MATLAB model [54]
trainKNN	Trains KNN
trainLDA	Trains LDA
trainQDA	Trains QDA
trainSVM	Trains SVM MATLAB model [54]

As we can see, the bulk of this part of the library is formed by functions responsible for training and using classifiers (see 4.3), which is done through training and cross-validation function `classImages`.

classANN

The function `classANN` documented in the tab. 5.28 uses trained ANN model for binary image classification based on invariant global features. This model is created by using the MATLAB Statistics and Machine Learning Toolbox™ [54] and features

Table 5.28: `classANN` classifies images using trained MATLAB ANN model [54].

Function	Calling	
classANN	y = classANN(x, res_train)	
Input/Output	Data Type	Description
x	1D array of double	Classification input
res_train	1D `cell` array	Trained ANN model
y	integer $\in \{0, 1\}$	Predicted class

a classification neural network with one hidden layer. We will use the activation function and the number of neurons in the one hidden layer later in Chapter 6 as a basis for accuracy plots. We see that we use this classification function iteratively to always classify precisely one input image, and the output is thus an integer (in our binary case also a boolean) classifying the image into one of the two classes.

classificationDriver

Table 5.29: `classificationDriver` runs functions related to image classification.

Function	Calling	
classificationDriver	[class_stats_leave1, class_stats_stratkfold, class_acc_leave1_lda, ... class_acc_leave1_qda, class_acc_leave1_knn, knn_opt, class_acc_leave1_svm, ... class_acc_leave1_sigmoid, class_acc_leave1_tanh] = ... classificationDriver(X_train, num_whit_features, classifier_train_name, ... classifier_class_name, k_fold, train_params, num_pos_imgs, num_neg_imgs)	
Input/Output	Data Type	Description
X_train	2D or 3D array of double	Classification data
num_whit_features	1D array of integer	Data inputs dimensions
classifier_train_name	string	Training function name
classifier_class_name	string	Classification function name
k_fold	integer > 0	Size of each k-fold
train_params	2D `cell` array	Classifier parameters
num_pos_imgs	integer ≥ 0	Number of AD images
num_neg_imgs	integer ≥ 0	Number of CN images
class_stats_leave1	MATLAB `table`	Cross-val. stats. using leave-one-out
class_stats_kfold	MATLAB `table`	Cross-val. stats. using stratified-k-fold
class_acc_leave1_lda	MATLAB `figure`	Plot of the best LDA classifier's accuracy
class_acc_leave1_qda	MATLAB `figure`	Plot of the best QDA classifier's accuracy
class_acc_leave1_knn	MATLAB `figure`	Plot of the best KNN classifier's accuracy
knn_opt	integer	Optimum number of KNN neighbours
class_acc_leave1_svm	MATLAB `figure`	Plot of the best SVM classifier's accuracy
class_acc_leave1_sigmoid	MATLAB `figure`	Plot of the best ACC of a ANN with sigmoid act. func.
class_acc_leave1_tanh	MATLAB `figure`	Plot of the best ACC of a ANN with tanh act. func.

The function `classificationDriver` documented in the tab. 5.29 is the principal driver function of the part of our library that deals with binary image classification. It runs the cross-validation function `classImages`, creates sorted tables of classification statistics and also makes plots of the best classifier results based on classification critical sensitivity se^* and accuracy (ACC). It also returns the variable `knn_opt`, which signifies the number of neighbours, for which the best classification results

of full data were achieved. This number of neighbours is then used a basis of using KNN for testing the one-dimensional data inputs as described in 4.2.

classImages

Table 5.30: `classImages` runs functions related to cross-validation and computes classification statistics.

Function	Calling	
classImages	`[class_stats_leave1, class_stats_stratkfold] =...` `classImages(X_train, y_star_train, num_pos_imgs_train,...` `num_neg_imgs_train, classifier_train_name,...` `classifier_class_name, k_fold, train_params)`	
Input/Output	Data Type	Description
X_train	2D or 3D array of double	Classification data
y_stat_train	1D array of integer $\in \{0,1\}$	Array of true classes
num_pos_imgs_train	integer ≥ 0	Number of AD images
num_neg_imgs_train	integer ≥ 0	Number of CN images
classifier_train_name	string	Training function name
classifier_class_name	string	Classification function name
k_fold	integer > 0	Size of each k-fold
train_params	2D cell array	Classifier parameters
class_stats_leave1	MATLAB table	Cross-val. stats. using leave-one-out
class_stats_kfold	MATLAB table	Cross-val. stats. using strat.-k-fold

The function `classImages` documented in the tab. 5.30 is the classifier training and cross-validation function. It tests the selected classifier on the input data using leave-one-out and stratified-k-fold techniques (here we note that we use a non-standard definition for the k - it signifies the number of samples taken from the dataset rather than the total number of folds) and computes classification statistics (see 4.4).

classKNN

Table 5.31: `classKNN` classifies images using KNN.

Function	Calling	
classKNN	`y = classKNN(x, res_train)`	
Input/Output	Data Type	Description
x	1D array of double	Classification input
res_train	1D cell array	Classification data and KNN parameters
y	integer $\in \{0,1\}$	Predicted class

The function `classKNN` documented in the tab. 5.31 classifies the input data using a k-nearest neighbours (KNN). We use our own implementation of KNN and define the closest neighbours in the sense of the Euclidean $l_2 = \|x\|_2$ distance.

classLDA

Table 5.32: `classLDA` classifies images using LDA.

Function	Calling	
classLDA	y = classLDA(x, res_train)	
Input/Output	Data Type	Description
x	1D array of double	Classification input
res_train	1D `cell` array	Classification data and LDA parameters
y	integer $\in \{0, 1\}$	Predicted class

The function `classLDA` documented in the tab. 5.32 classifies the input data using the linear discriminant analysis (LDA). We use our own implementation of LDA with regularization defined in 4.3.1.

classQDA

Table 5.33: `classQDA` classifies images using QDA.

Function	Calling	
classQDA	y = classQDA(x, res_train)	
Input/Output	Data Type	Description
x	1D array of double	Classification input
res_train	1D `cell` array	Classification data and QDA parameters
y	integer $\in \{0, 1\}$	Predicted class

The function `classQDA` documented in the tab. 5.33 classifies the input data using the quadratic discriminant analysis (QDA). We use our own implementation of QDA with regularization defined in 4.3.1.

classSVM

The function `classSVM` documented in the tab. 5.34 uses trained SVM model for binary image classification based on invariant global features. This model is created by using the MATLAB Statistics and Machine Learning Toolbox™ [54] and features a SVM with a linear, quadratic or Gaussian kernel (see 4.3.3). We will use the scale

Table 5.34: `classSVM` classifies images using trained MATLAB SVM model [54].

Function	Calling	
classSVM	y = classSVM(x, res_train)	
Input/Output	Data Type	Description
x	1D array of double	Classification input
res_train	1D `cell` array	Trained SVM model
y	integer $\in \{0, 1\}$	Predicted class

of the Gaussian SVMs as a basis for accuracy plots in Chapter 6. We see that we use this classification function iteratively to always classify precisely one input image, and the output is thus an integer (in this case also a boolean) classifying the image into one of the two classes.

createClassStatsTable

Table 5.35: `createClassStatsTable` creates a sorted MATLAB table of classification statistics.

Function	Calling	
createClassStatsTable	class_stats_table = createClassStatsTable(class_stats_arr, var_names)	
Input/Output	Data Type	Description
class_stat_array	2D array of double	Classification statistics
var_names	1D array of string	Used classifiers
class_stats_table	MATLAB `table`	Sorted table with class. stats.

The function `createClassStatsTable` documented in the tab. 5.35 is used to create a sorted MATLAB table featuring all used classifiers as variable names (each table column is a one-dimensional array of classification statistics for one binary classifier in some specific configuration). The classifiers are sorted primarily according to critical sensitivity se^* and then according to ACC to break ties.

trainANN

The function `trainANN` documented in the tab. 5.36 incorporates the functions predefined in [54] to create and train a classification ANN with one hidden layer. This layer features either sigmoid or hyperbolic tangent activation function and differing number of neurons. The trained model is then wrapped in a `cell` array `res_train` and passed back. The function type and number of neurons will later serve as a basis of plots shown in Chapter 6.

Table 5.36: `trainANN` trains MATLAB ANN model [54].

Function	Calling	
`trainANN`	`res_train = trainANN(X, y_star, train_params)`	
Input/Output	Data Type	Description
`X`	2D array of double	Training inputs
`y_star`	1D array of integer $\in \{0, 1\}$	True classes
`train_params`	1D `cell` array	Training parameters
`res_train`	1D `cell` array	Trained ANN model

trainKNN

Table 5.37: `trainKNN` trains KNN classifier.

Function	Calling	
`trainKNN`	`res_train = trainANN(X, y_star, train_params)`	
Input/Output	Data Type	Description
`X`	2D array of double	Training inputs
`y_star`	1D array of integer $\in \{0, 1\}$	True classes
`train_params`	1D `cell` array	Training parameters
`res_train`	1D `cell` array	`X`, `y_star` and KNN params.

The function `trainKNN` documented in the tab. 5.37 only passes the necessary arguments to function `classKNN` in form of a `cell` array `res_train` as there is no need to train the KNN.

trainLDA

Table 5.38: `trainLDA` trains an LDA classifier.

Function	Calling	
`trainLDA`	`res_train = trainLDA(X, y_star, lambda)`	
Input/Output	Data Type	Description
`X`	2D array of double	Training inputs
`y_star`	1D array of integer $\in \{0, 1\}$	True classes
`lambda`	double > 0	Regularization param.
`res_train`	1D `cell` array	Params. of trained LDA

The function `trainLDA` documented in the tab. 5.38 trains an LDA classifier using the theory from 4.3.1. It also performs regularization established in the same section. The trained LDA classifier is then passed back wrapped in a `cell` array `res_train`.

trainQDA

Table 5.39: `trainQDA` trains QDA classifier.

Function	Calling	
trainQDA	res_train = trainQDA(X, y_star, lambda)	
Input/Output	Data Type	Description
X	2D array of double	Training inputs
y_star	1D array of integer $\in \{0, 1\}$	True classes
lambda	double > 0	Regularization param.
res_train	1D `cell` array	Params. of trained QDA

The function `trainQDA` documented in the tab. 5.39 trains an QDA classifier using the theory from 4.3.1. It also performs regularization established in the same section. The trained QDA classifier is then passed back wrapped in a `cell` array `res_train`.

trainSVM

Table 5.40: `trainSVM` trains MATLAB SVM model [54].

Function	Calling	
trainSVM	res_train = trainSVM(X, y_star, train_params)	
Input/Output	Data Type	Description
X	2D array of double	Training inputs
y_star	1D array of integer $\in \{0, 1\}$	True classes
train_params	1D `cell` array	Training parameters
res_train	1D `cell` array	Trained SVM model

The function `trainSVM` documented in the tab. 5.40 incorporates the functions predefined in [54] to create and train a classification support vector machine (SVM) featuring a linear, quadratic or Gaussian kernels (see 4.3.3). This trained model is then wrapped in `cell` array `res_train` and passed back. The scale of Gaussian kernels will later serve as a basis of plots shown in Chapter 6.

5.7 Saving Results

In this section, we provide an overview of the single function used for saving results. We may see this overview of in the tab. 5.41.

Table 5.41: Function used to save results. See also A.6.

Function	Description
saveResults	Saves results stored in an instance of resultClass

saveResults

Table 5.42: saveResults saves results with appropriate filenames. See also A.6.

Function	Calling	
saveResults	saveResults(configuration, results)	
Input/Output	Data Type	Description
configuration	configurationClass	Instance of configurationClass (to create filenames)
results	resultClass	Instance of resultClass with full results to save

The function saveResults is the only function with a separate file used to save results. As such, we may see that one of its parameters is an instance of configurationClass, which is used to create a filename (using the configuration parameters) for the results stored in an instance of resultClass. We may see a short part of the code related to saving the data files in A.6, where we also provide more comments. This part of the library incorporates functions [55] and [56], which is also shown and described in the appendix.

Chapter 6

Experimental Part

In this chapter, we will present a compendium of the most interesting results for the several branches of our inquiry. In the whole chapter, we will conduct our experiment on a AD/CN dataset [57] containing 55 AD and 56 CN SPECT ^{99m}Tc-HMPAO images of size $79 \times 95 \times 69$ resulting in 517 845 total voxels, where the voxel size is $1 \times 1 \times 1$ mm. All results will be obtained by randomly selecting fifty images from both of these classes (one hundred in total, a balanced dataset) and using them as data inputs. First, in 6.1, we describe the results obtained while testing the calculated invariant features for data separability, which will enable us to decide whether the data is statistically separable as well as to choose promising configurations (using all configurations would be too time-consuming and would also precipitate the curse of dimensionality, as we would be using hundreds or even thousands of calculated features for each image) to calculate data inputs of binary classifiers. We will continue by discussing the results of image classification, a section divided into three parts based on utilised data inputs - full data, one-dimensional data and whitened data (see 4.2). In the section on one-dimensional data, we provide only tables showing the ten best results obtained using a KNN with the optimum number of neighbours determined while classifying the full data. In this manner, we will find out whether it is possible to successfully classify the data using only the individual, statistically significant characteristics. In the sections discussing the full data and the whitened data, we present and comment on tables of various classification configurations (classifiers and their parameters) ranked by their resulting critical sensitivity se^* and accuracy (ACC). We also present graphs of ACC of the best overall classifier from each category, viz. linear discriminant analysis (LDA), quadratic discriminant analysis (QDA), k-nearest neighbours (KNN), support vector machine (SVM), artificial neural network (ANN), based on selected input parameter, e.g. the regularization parameter λ in the case of discriminant analysis classifiers. We used a HP Pavilion Gaming 15 (15-cx0015nc) laptop with Intel®Core™i5-8300H CPU clocked at 2.30 GHz for obtaining all results shown in this chapter.

6.1 Data Separability Testing

We first tested the data for statistical separability, i.e. the difference of medians or means. We did this using a maximalist configuration featuring several combinations of input parameters to find the potentially best constellation of parameters. Specifically, we tried $\theta \in \{0.05, 0.1, 0.2, 0.3\}$ for binary mask threshold, $\rho \in \{3, 5, 7.5, 10\}$ for function radii with maximum $l_{\max} = 2$ and $n_{\max} = 2$ (setting larger maximum l and n would be too time costly). We tried simple invariant calculation with no filter (thus also no filtering kernel) and no nonlinear transformation, as well as with low-pass (LP) and high-pass (HP) filters with Gaussian, α-stable and Butterworth kernels used without nonlinear transformation as well as together with log transformation (LT). We also utilised all twelve image characteristics introduced in 3.3. Using the combinations of all these parameters as well as choosing a total of one hundred three-dimensional images, these calculations lasted approximately 47 hours to complete and resulted in more than 22 000 calculated features.

It was first necessary to decide whether we will focus on using Welch's t-test or use the non-parametric Mann-Whitney U test. Therefore, we first tested the data for normality with the Lilliefors test and found that only in 0.0452 or 1016 hypotheses out of 22 464 could the normality of the data be accepted. Seeing that the data mostly could not be accepted as normally distributed, we resorted to using the non-parametric Mann-Whitney U test.

6.1.1 Mann-Whitney U Test

As the normality of tested data could not generally be accepted, we only present the results of testing the data for separability with the non-parametric Mann-Whitney U test. Even though we initially obtained the best separability results with configurations featuring predominantly high-pass filters, this situation changed after using the FDR refinement procedure described in 4.1.4. The five best overall configurations in terms of relative number of accepted hypotheses of differing medians can be seen in the tab. 6.1.

Table 6.1: Five best configurations rated according to relative number of differing medians.

Configuration	Acc. Hyps. [%]
$\theta = 0.3$, $\rho = 10$, LP, Gauss., LT	14.8
$\theta = 0.3$, $\rho = 10$, LP, Gauss., no trans.	13.9
$\theta = 0.3$, $\rho = 10$, LP, α-st., no trans.	13.9
$\theta = 0.3$, $\rho = 10$, LP, Butt., no trans.	13.9
$\theta = 0.3$, $\rho = 10$, no filt., no ker., no trans.	13.0

We thus see that we achieved best results with largest tested θ and ρ values coupled

with LP filters, where it generally did not matter, which kernel we used. The best overall result was achieved using a log transformation, but we were also able to achieve similar results without nonlinear transformations. Hence, to save computing time, we decided to continue using configurations that do not feature nonlinear transformations.

In the tab. 6.2, we may see similar rating of image characteristics (we use the notation est. in 3.3) based on number of relative accepted hypotheses of differing medians.

Table 6.2: Rating of image characteristics according to accepted hypotheses.

Characteristic	Acc. Hyps. [%]
skew	10.5
med	8.5
kurt	6.8
mad	6.2
Q_3	5.8
Q_1	4.9
IQR	4.6
var	4.3
range	3.4
mean	3.3
min	2.5
max	1.9

Unsurprisingly, we are able to see that worst results are achieved with simplest non-robust characteristics such as the maximum, minimum, and mean. On the other hand, moments such as skewness and kurtosis, as well as the robust image median (compare to non-robust mean) achieve the best results.

From above, we surmise that the potential for clean data separation using only the statistical testing is quite low, and we will thus continue by also presenting the results of attempting to classify the images with binary classifiers. We will further utilise the results of this section as a basis when attempting to find the best configurations to calculate classification data inputs.

6.2   Image Classification

In this section, we present the results of attempting to classify the three-dimensional images into two classes - AD or CN patients. While we initially tried the configurations, which achieved the best data separability (see above), with $\theta = 0.3$ and $\rho = 10$, we quickly found out that we achieve the best classification results while using configurations with $\theta = 0.2$ and $\rho \approx 7$ together with no filter or with a LP Gaussian

kernel with $\sigma \in [0.1, 0.5]$ or a Butterworth LP kernel filter with $d_0 \in [0.15, 0.25]$ and $m = 1$ (see 2.1) and no nonlinear transformations. While the configurations featuring a Gaussian kernel also achieved very good classification results, they were narrowly edged out by the configurations featuring a Butterworth kernel. Therefore, in the next section, we present the results obtained using a configuration featuring invariants calculated using no filter and invariants filtered with a Butterworth LP kernel with parameters set to $d_0 = 0.25$ and $m = 1$. Due to computational costliness of utilizing invariants with $n, l > 2$, we again set $n_{\max} = l_{\max} = 2$. Using such reduced configurations (no filter and one low-pass filter, number of characteristics $p \approx 200$, computation time c. 2 hours) we were able to achieve much better results than when using maximalist configurations, presumably because we were able to escape the curse of dimensionality. We divide this section into three parts - first we try to use the full data (see 4.2, number of characteristics $p \approx 200$) with discriminant analysis and k-nearest neighbours (in this section we do not use SVMs and ANNs because of computational costliness of using these classifiers on such high-dimensional data). Using the full data, we also determine the optimum number of neighbours in KNN (setting it as the number, for which best ACC was achieved when classifying full data). In the next section, we use the KNN with the optimum number of neighbours while classifying the data based on one-dimensional inputs - we iterate through all $p \approx 200$ characteristics and try to determine whether we can achieve good classification results based on only individual characteristics as inputs. Finally, we reduce the dimension of the full input data (see 4.2) using the PCA data whitening and use these principal components as inputs of all mentioned classifiers, including SVMs and ANNs. We present only the cross-validation results obtained using the leave-one-out technique for the sake of brevity and also because the results obtained using the stratified-k-fold technique were almost the same in the sense of results interpretation.

6.2.1 Full Data

In this section, we present the results of classifying the full data (number of characteristics $p = 9 \cdot 12 \cdot 2 = 216$, where we use nine invariants, twelve characteristics and a configuration with no filtering as well as with Butterworth LP filter). First, we compare the best classifiers ranked primarily according to the highest critical sensitivity, se^*, which is the minimum of TPR and TNR (see (4.51)), and secondarily according to accuracy (ACC) to break possible ties. In this section, we compare three types of classifiers useful for higher dimensional data because of their relatively low computational costs, viz. the linear discriminant analysis (LDA), quadratic discriminant analysis (QDA), and k-nearest neighbours (KNN). We may see such comparison for the leave-one-out cross-validation technique in the tab. 6.3. We note that the value of se^* is not explicitly shown in the table because of its simple definition, making it easy to deduce from the first two rows of the table. We also do not show results for the balanced accuracy (BACC) because we use a balanced dataset, and thus the value of BACC is always equal to the corresponding value of ACC.

Table 6.3: Comparison of the best classifiers - the LDA, QDA and KNN.

	LDA, $\lambda = 0.03$	QDA, $\lambda = 0.05$	KNN, $k = 11$
TPR	0.820	0.900	0.700
TNR	0.880	0.880	0.800
FPR	0.120	0.120	0.200
FNR	0.180	0.100	0.300
PPV	0.872	0.882	0.778
NPV	0.830	0.898	0.727
FDR	0.128	0.118	0.222
FOR	0.170	0.102	0.273
LR+	6.833	7.500	3.500
LR−	0.205	0.114	0.375
PRT	0.277	0.267	0.348
TS	0.732	0.804	0.583
ACC	0.850	0.890	0.750
F_1	0.845	0.891	0.737
MCC	0.701	0.780	0.503
FM	0.846	0.891	0.738
DOR	33.407	66	9.333

As we may see, with both the linear discriminant analysis and quadratic discriminant analysis we achieve quite similar results in terms of accuracy (ACC) equal to or greater than 0.850. The QDA, however, offers somewhat better negative predictive value (NPV) and significantly better diagnostic odds ratio (DOR). The KNN utilizing Euclidean distance (absolute distance was also tested but achieved similar or somewhat worse results) achieved significantly poorer results with accuracy equal to 0.750. It is also important to note that the QDAs classifiers with parameter $\lambda \in \{0.05, 0.075, 0.1, 0.4\}$ are all ranked the same according to $se^* = 0.880$ and ACC = 0.890. Therefore, we may consider these classifiers equivalent and show only the classification results for the smallest value of regularization parameter, $\lambda = 0.05$.

Linear Discriminant Analysis

In the fig. 6.1, we may see the plot of classification accuracy of the linear discriminant analysis for full data, where number of characteristics $p = 216$, based on the value of the regularization parameter λ.

For small values of the regularization parameter, $\lambda \approx 0.01$, we achieve low classification accuracy, which then rises sharply with greater values of α. We achieve the best results, ACC = 0.850, for value of regularization parameter $\lambda = 0.03$, as we have already mentioned above. When $\lambda > 1$ we may note a fast decline in accuracy to values ACC ≈ 0.650.
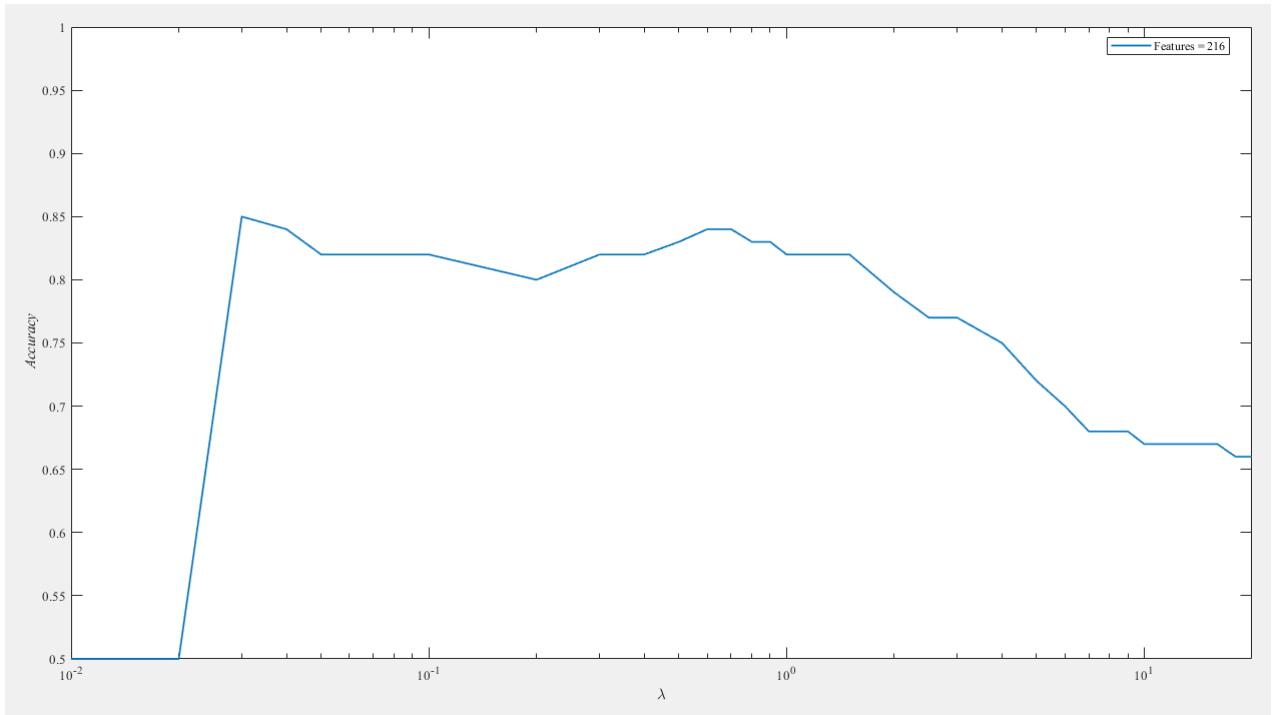
Figure 6.1: Accuracy of classifying full data with LDA.

Quadratic Discriminant Analysis

In the fig. 6.2, we may see the plot of classification accuracy of the quadratic discriminant analysis for the full data, where number of characteristics $p = 216$, based on the value of the regularization parameter λ.

For small values of the regularization parameter, $\lambda \approx 0.01$, we achieve low classification accuracy, which then rises sharply as λ approaches 0.03 (see commentary below the tab. 6.3). We achieve the best accuracy ACC = 0.890 for values of $\lambda \in \{0.05, 0.075, 0.1, 0.4\}$. When $\lambda > 1$ we may again note the fast decline in accuracy to values ACC ≈ 0.650.

k-Nearest Neighbours

In the fig. 6.3, we may see the plot of classification accuracy of the KNN for full data, where number of characteristics $p = 216$, based on the number of considered nearest neighbours.

We may clearly see that the KNN achieves the lowest classification accuracy out of the three used types of classifiers. We achieve the best accuracy results, ACC ≈ 0.770, when considering fifteen neighbours of each data input. We did not show this setting in the tables because it suffers from a lower critical sensitivity, $se^* = 0.680$, than the setting using eleven neighbours (seen in the tab. 6.3), as the fifteen neighbour setting has values of true positive rate and true negative rate of
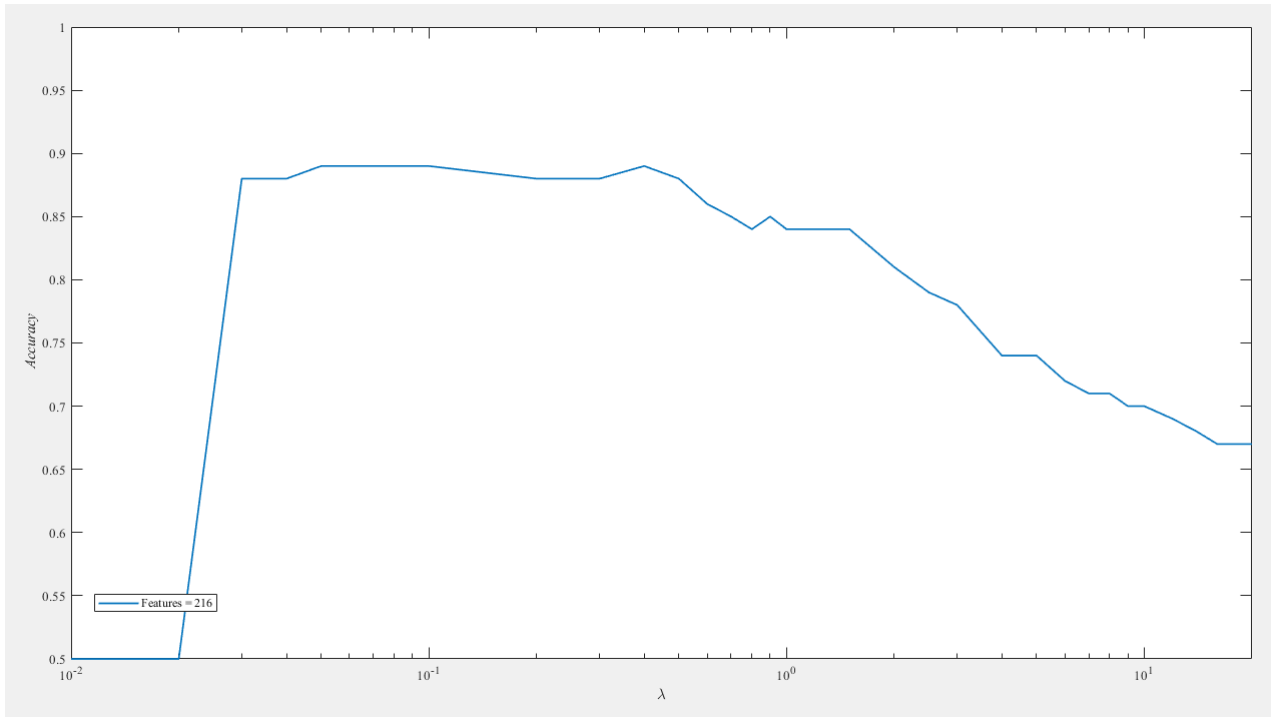
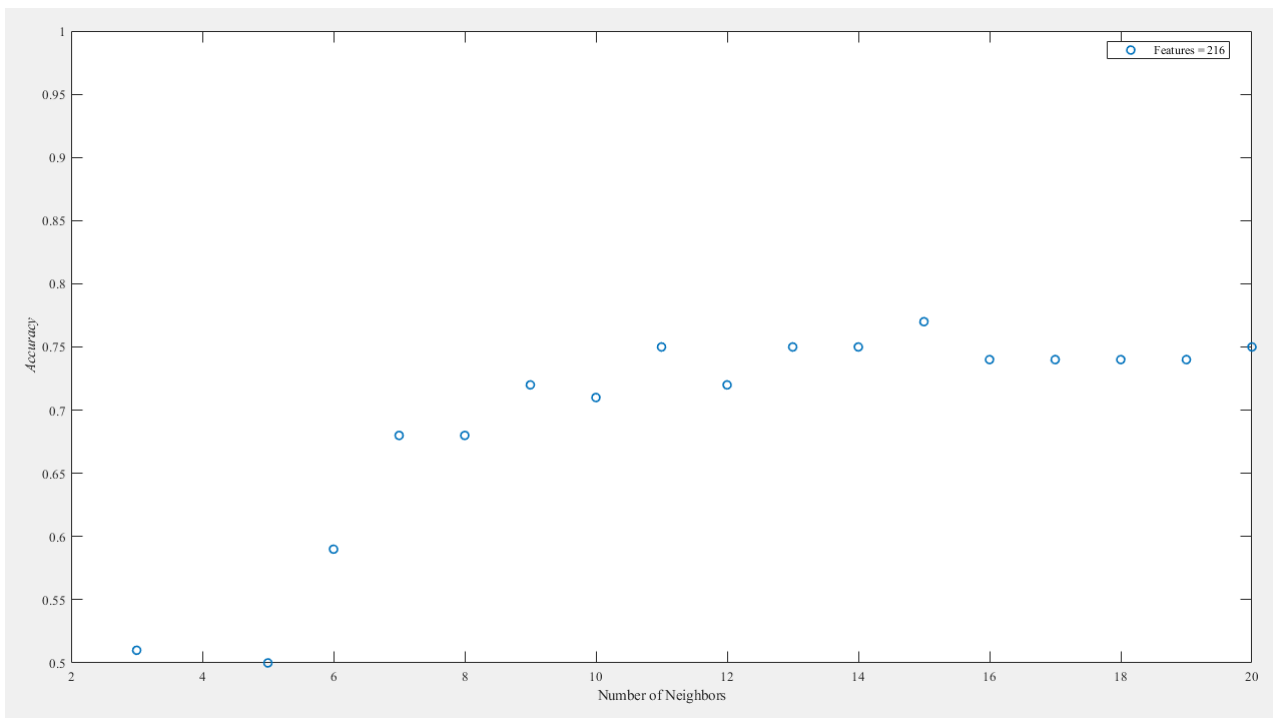Figure 6.2: Accuracy of classifying full data with QDA.

Figure 6.3: Accuracy of classifying full data with KNN.

TPR = 0.680, TNR = 0.860. However, as the se^* when using fifteen neighbours was quite similar to the one using eleven neighbours and achieved slightly better accuracy and we also see from the figure above that we tended to achieve consistently higher accuracy when considering larger number of neighbours, we declare the larger

91

number fifteen to be the optimum number of neighbours for our purposes of testing one-dimensional input data in the following section.

6.2.2 One-Dimensional Data

In this section, we briefly discuss the possibility of classifying the data based solely on one-dimensional data inputs. As the number of characteristics is $p = 216$, we iterate through all these individual invariant characteristics, attempting to find out whether it is possible to separate the images into two non-overlapping classes. To this end, we use the KNN with number of neighbours set to fifteen (optimum number of neighbours from the previous section, see above) and utilizing the Euclidean distance. We provide two tables - in tabs. 6.4 and 6.5 we may see the ten best overall invariant features ranked according to se^* and ACC. Results in both tables were (as stated above) obtained using configuration $\theta = 0.2$, $\rho = 7$ and we thus do not explicitly state this fact in the table header. We denote the usage of the low-pass Butterworth filter with parameters set to $d_0 = 0.25$ and $m = 1$ by adding LP in the table header where necessary. The $b_{l,n}$ in the table header denotes the rotational invariant (see 3.2).

Table 6.4: Comparison of the first five best invariant features.

	$b_{0,0}$, kurt	$b_{0,0}$, skew	LP, $b_{0,0}$, kurt	LP, $b_{0,0}$, med	LP, $b_{0,0}$, Q$_1$
TPR	0.820	0.760	0.820	0.680	0.680
TNR	0.720	0.720	0.700	0.800	0.800
FPR	0.280	0.280	0.300	0.200	0.200
FNR	0.180	0.240	0.180	0.320	0.320
PPV	0.745	0.731	0.732	0.773	0.773
NPV	0.800	0.750	0.795	0.714	0.714
FDR	0.255	0.269	0.268	0.227	0.227
FOR	0.200	0.250	0.205	0.286	0.286
LR+	2.929	2.714	2.733	3.400	3.400
LR−	0.250	0.333	0.257	0.400	0.400
PRT	0.369	0.378	0.377	0.352	0.352
TS	0.641	0.594	0.631	0.567	0.567
ACC	0.770	0.740	0.760	0.740	0.740
F$_1$	0.781	0.745	0.774	0.723	0.723
MCC	0.543	0.480	0.524	0.483	0.483
FM	0.782	0.745	0.775	0.725	0.725
DOR	11.714	8.143	10.630	8.500	8.500

The first five features (all features from the tab. 6.4) and the fourth feature in the second table were also determined as statistically significant (allowing for data separation) while testing for data separability. The fact that we also obtained the best classification results while using these features as single data inputs further confirms that kurtosis, skewness, and median (and perhaps the first quartile) calculated on

Table 6.5: Comparison of the second five best invariant features.

	$b_{2,0}$, med	$b_{2,2}$, skew	LP, $b_{2,2}$, skew	$b_{0,0}$, med	LP, $b_{2,0}$, med
TPR	0.680	0.680	0.680	0.660	0.660
TNR	0.720	0.720	0.700	0.800	0.720
FPR	0.280	0.280	0.300	0.200	0.280
FNR	0.320	0.320	0.320	0.340	0.340
PPV	0.708	0.708	0.694	0.767	0.702
NPV	0.692	0.692	0.686	0.702	0.679
FDR	0.292	0.292	0.306	0.233	0.298
FOR	0.308	0.308	0.314	0.298	0.321
LR+	2.429	2.429	2.267	3.300	2.357
LR−	0.444	0.444	0.457	0.425	0.472
PRT	0.391	0.391	0.399	0.355	0.394
TS	0.531	0.531	0.523	0.550	0.516
ACC	0.700	0.700	0.690	0.730	0.690
F_1	0.694	0.694	0.687	0.710	0.680
MCC	0.400	0.400	0.380	0.465	0.381
FM	0.694	0.694	0.687	0.712	0.681
DOR	5.464	5.464	4.958	7.765	4.992

rotational invariants form the core characteristics that can be used for separating AD and CN images into two distinct classes. We, however, note that it is also possible to achieve decent results by using features that were not initially labelled as allowing for data separability, such as the median of the invariant $b_{2,0}$ or the skewness of the invariant $b_{2,2}$.

6.2.3 Whitened Data

In this section, we present the results of classification while using the reduced, whitened data inputs. Using the PCA, we first reduce the number of dimensions from $p = 216$ to $p \in \{2, 3, \ldots, 10\}$. We then attempt to find the best classifiers (ranked again according to se^* and ACC) as well as the optimum number of principal components (dimensions). In the tab. 6.6, we present the results for the three classifiers, which we also used initially for the full data - the linear discriminant analysis, quadratic discriminant analysis and k-nearest neighbours. In the tab. 6.7, we present the results for the newly used SVMs and ANNs.

As we may see in the tab. 6.6, the QDA maintains or slightly improves its critical sensitivity, while the DOR increases to by more than fifty percent from the original value DOR = 66 to DOR = 103.500. The performance is also noticeably enhanced for the KNN - improvement of 0.040 for se^* and of approximately 4 for DOR. The QDA with the regularization parameter $\lambda = 0.2$ also achieves the second-best overall performance out of all classifiers (after a Gaussian SVMs, see below). The perform-

Table 6.6: Comparison of the best classifiers - LDA, QDA and KNN.

	LDA, $\lambda = 0.01$, PCA = 10	QDA, $\lambda = 0.2$, PCA = 7	KNN, $k = 16$, PCA = 3
TPR	0.820	0.900	0.760
TNR	0.820	0.920	0.820
FPR	0.180	0.080	0.180
FNR	0.180	0.10	0.240
PPV	0.820	0.918	0.809
NPV	0.820	0.902	0.774
FDR	0.180	0.082	0.191
FOR	0.180	0.098	0.226
LR+	4.556	11.250	4.222
LR−	0.220	0.109	0.293
PRT	0.319	0.230	0.327
TS	0.695	0.833	0.644
ACC	0.820	0.910	0.790
F_1	0.909	0.92	0.784
MCC	0.640	0.820	0.581
FM	0.820	0.909	0.784
DOR	20.753	103.500	14.426

ance of LDA stays similar if slightly worse, worsening by 0.030 in critical sensitivity, when compared to the one shown in 6.3.

Table 6.7: Comparison of the best classifiers - SVM and ANN.

	SVM, Gauss., $\sigma = 1$, PCA = 7	ANN, Sigm., Neur. = 5, PCA = 3
TPR	0.940	0.920
TNR	0.900	0.880
FPR	0.100	0.120
FNR	0.060	0.080
PPV	0.904	0.885
NPV	0.938	0.917
FDR	0.096	0.115
FOR	0.063	0.083
LR+	9.400	7.667
LR−	0.067	0.091
PRT	0.246	0.265
TS	0.855	0.821
ACC	0.920	0.900
F_1	0.922	0.902
MCC	0.841	0.801
FM	0.922	0.902
DOR	141	84.333

In the tab. 6.6, we may observe that we were able to obtain good classification results

for both support vector machines, particularly with a Gaussian kernel, and artificial neural networks. As above, with a similarly well performing QDA, the best result for the SVM was obtained while using input data composed of seven principal components. Furthermore, the results for this Gaussian SVM are the best overall results of all used classifiers. These were obtained using a Gaussian kernel with $\sigma = 1$ (we may note the steep rise and decline in classification accuracy based on the value of σ in the fig. 6.7).

We only needed five neurons and three principal components to obtain the best result for ANN. It is important to note, however, that we also obtained exactly the same classification results for a sigmoid activation function, seven neurons and three principal components, as well as for three neurons, hyperbolic tangent activation function and three principal components. Apart from this, however, the ANNs with a hyperbolic tangent activation tended to achieve slightly worse performance on average, and we thus opt to show the best results obtained with a sigmoid activation.

From the tabs. 6.6 and 6.7, we may see that we achieved the best classification performance while using either the QDA, SVMs or ANNs. Observing what we consider to be good classification results, ACC ≈ 0.900, our proposed method of texture analysis coupled with one of these classifiers (for example the QDA because of its comparably low computational costs) can be considered a promising approach, which could, if further developed and tested, find its usage as a fast consultative tool in the field of medical diagnostics.

Linear Discriminant Analysis

We obtained the best accuracy results, ACC $= 0.820$, with the linear discriminant analysis while using the data reduced into the maximum tested value of ten principal components. It is also interesting to note that the accuracy of the LDA used on whitened data does not change with different values of the regularization parameter λ, facing no rise or decline (compare the fig. 6.1), which we may see in the fig. 6.4. The results for LDA in the tab. 6.6 are thus the same for all values of λ and six principal components, and we merely choose the value $\lambda = 0.01$ as a placeholder for all other tested values of the regularization parameter.

While invariant in respect to the value of the regularization parameter, the LDA used on the whitened data also achieves a slightly worse top accuracy performance than the LDA used on the full data (compare tabs. 6.3 and 6.6). We also observe that this is in contrast with the QDA used on the whitened data, which achieves a significantly better accuracy performance (second-best performance of all classifiers) for several specific values of the regularization parameter, but also faces a decline in performance (see the fig. 6.5).
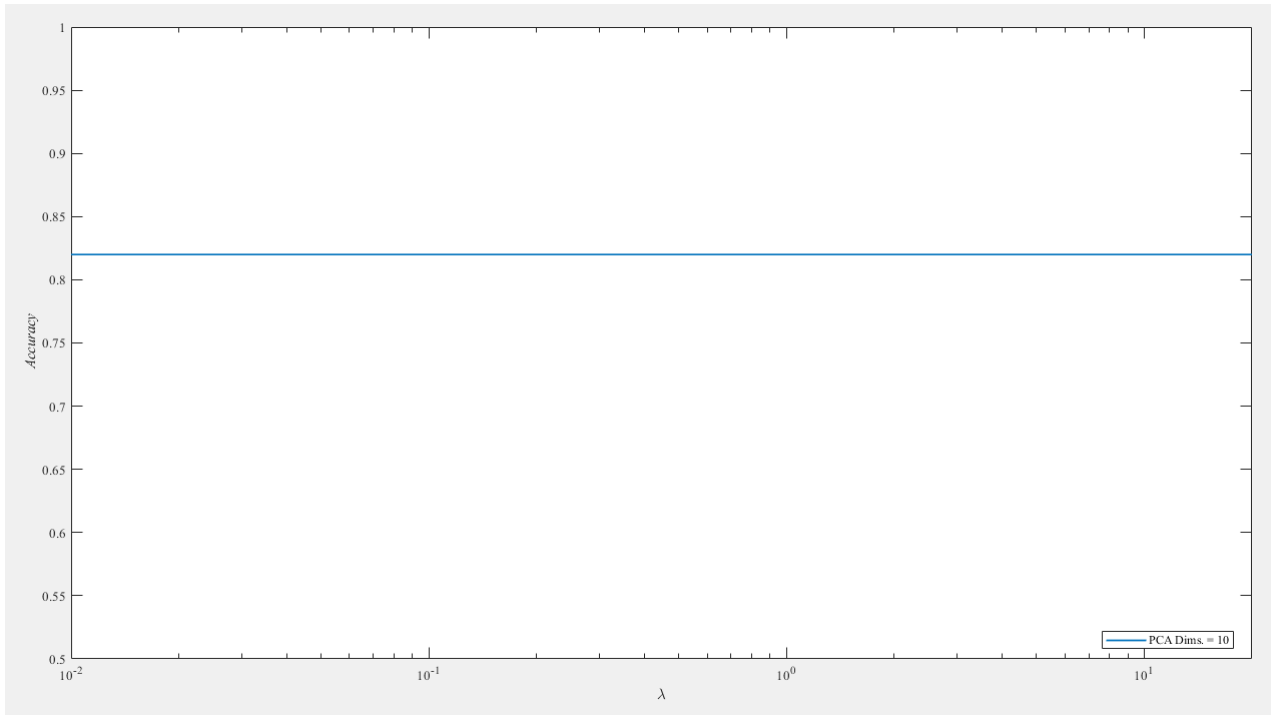
Figure 6.4: Accuracy of classifying whitened data with LDA.

Quadratic Discriminant Analysis

The quadratic discriminant analysis achieves the second-best top classification results in terms of classification accuracy, ACC = 0.910, of all tested classifiers (compare tabs. 6.6 and 6.7). These results were obtained while using seven principal components of the reduced data and experimentally optimal values of the regularization parameter $\lambda \approx 0.2$. However, in terms of averaged accuracy we achieved the best results with six principal components. This setting also achieved best accuracy, ACC = 0.900, with the regularization parameter set $\lambda = 0.2$, thus confirming its closeness to optimum, and placed third-best overall for top performance. We may see the accuracy plot in the fig. 6.5.

In contrast to the linear discriminant analysis (see above), the quadratic discriminant analysis still exhibits a decline in performance due to the rising value of the regularization parameter, especially for values of regularization parameter $\lambda > 1$. This decline is, however, not as steep as while using the QDA for classification of the full data (compare the fig. 6.2).

k-Nearest Neighbours

The k-nearest neighbours again achieved the weakest results of all tested classifiers (compare tabs. 6.6 and 6.7). However, in the fig. 6.6, we may still highlight a noticeable improvement in accuracy, ACC = 0.800, from the original value ACC = 0.750 while considering eighteen neighbours, on the data reduced into three principal com-
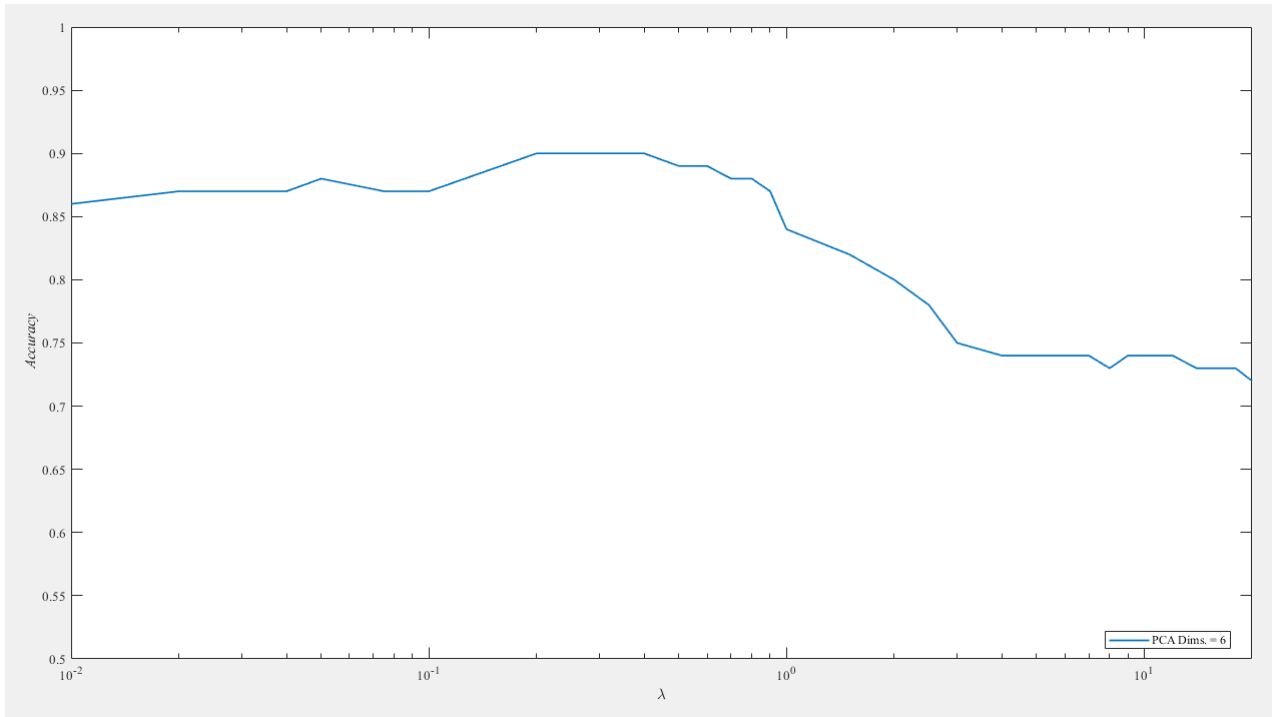
Figure 6.5: Accuracy of classifying whitened data with QDA.

ponents. As with the full data, this result, however, suffers from lower critical sensitivity, $se^* = 0.740$, and we thus select the sixteen neighbours as the optimum shown in the tab. 6.6.
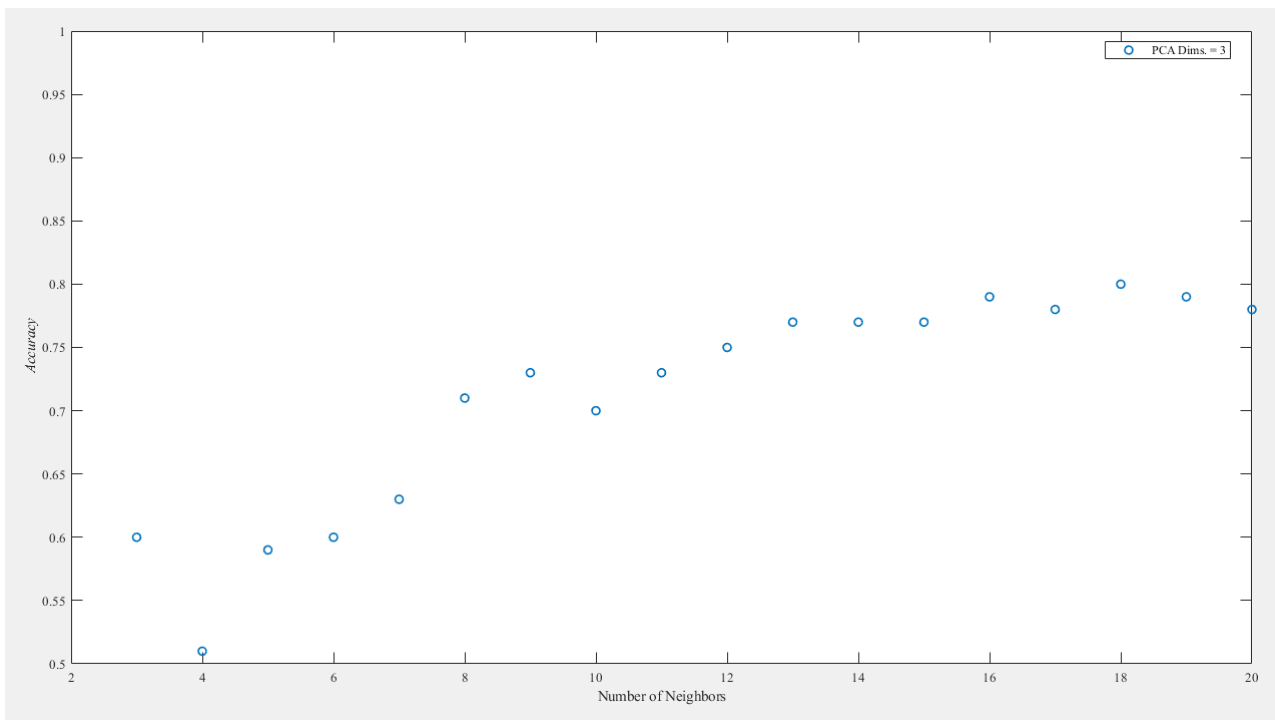

Figure 6.6: Accuracy of classifying whitened data with KNN.

97

While we may pose a question whether it is necessary to perform the principal component analysis as a preprocessing for the linear discriminant analysis and quadratic discriminant analysis, as the accuracy performance stays quite similar (or even worsens) for both the full and the whitened data, it seems that the data whitening noticeably improves the performance of the *k*-nearest neighbours.

Support Vector Machines with Gaussian Kernel

We may see the classification performance of a nonlinear support vector machine with a Gaussian kernel and seven principal components as an input in the fig. 6.7. We observe a steep rise and decline in classification accuracy based on the value of the scaling parameter σ, whose optimum value is $\sigma \approx 1$ (see the tab. 6.7), where the Gaussian SVMs achieved the best classification results of all tested classifiers, comparable with perhaps only the QDA and the best performing ANNs. For values of σ significantly differing from this experimental optimum, the Gaussian SVMs generally achieved very poor results of ACC < 0.500.
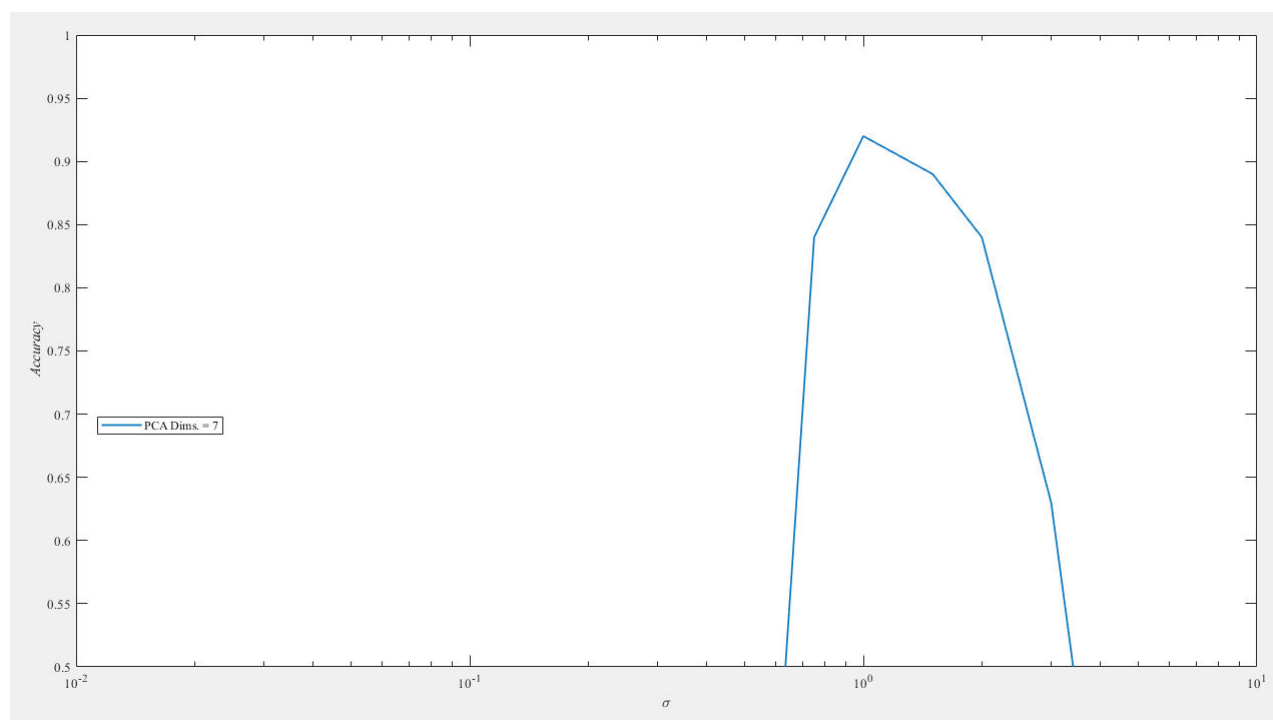

Figure 6.7: Accuracy of classifying whitened data with SVM with a Gaussian kernel.

Apart from the Gaussian SVMs, we also tested the SVMs featuring polynomial (linear and quadratic) kernels. In general, the top performance of these SVMs, ACC ≥ 0.850, was similar to, if slightly worse than, the performance achieved with Gaussian kernels. These types of kernels, however, did not exhibit the same steep decline in performance. Even the worst polynomial kernels still maintained accuracy ACC ≥ 0.550.

Artificial Neural Networks with Sigmoid Activation Function

Due to the high computational cost of training the artificial neural networks, we only utilised this type of classifiers on the preprocessed, whitened data. We may see the results for a artificial neural network with one hidden layer with a sigmoid activation function in the fig. 6.8, where we show the classification accuracy based on the number of neurons in the hidden layer. We may note that we needed a smaller number of principal components - three - than with most other previously mentioned classifiers.
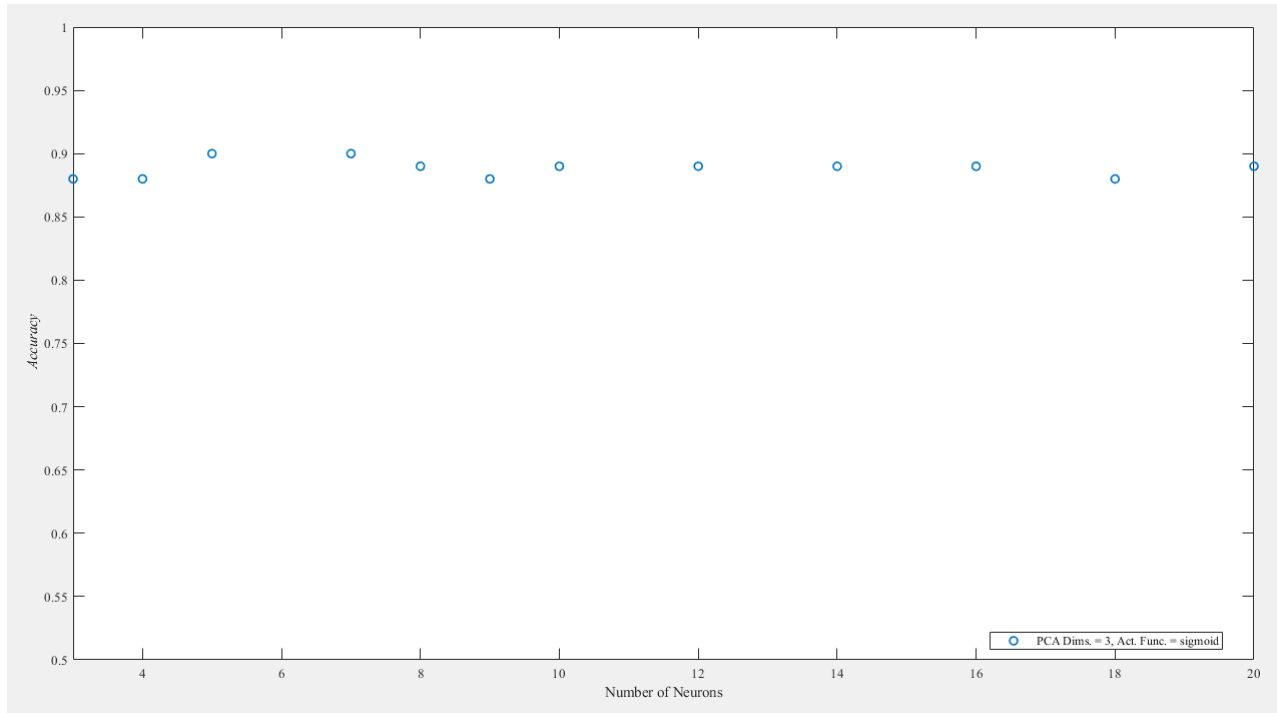

Figure 6.8: Accuracy of classifying whitened data with ANN with a sigmoid activation function.

We also observe the general invariance of high classification performance, ACC ≥ 0.850, to the changing number of neurons. This is further evidenced by the fact that we obtained exactly the same results when featuring five and seven neurons in the hidden layer (this lower number of neurons can thus be considered an optimum). The ANNs with a sigmoid activation function achieved slightly better performance than the ANNs with a hyperbolic tangent activation function (see below) with the best performance shown above and the worst performance, ACC ≈ 0.650, achieved while using a large number of neurons (≥ 15) and a low number of principal components (≤ 3).

Artificial Neural Networks with Hyperbolic Tangent Activation Function

We may see the results for a ANN with one hidden layer with a hyperbolic tangent activation function in the fig. 6.9, where we show the classification accuracy based

on the number of neurons in the hidden layer. We may note that we needed a smaller number of principal components - three - than with other non-ANN classifiers but the same number of components as while using an ANN with a sigmoid activation function.
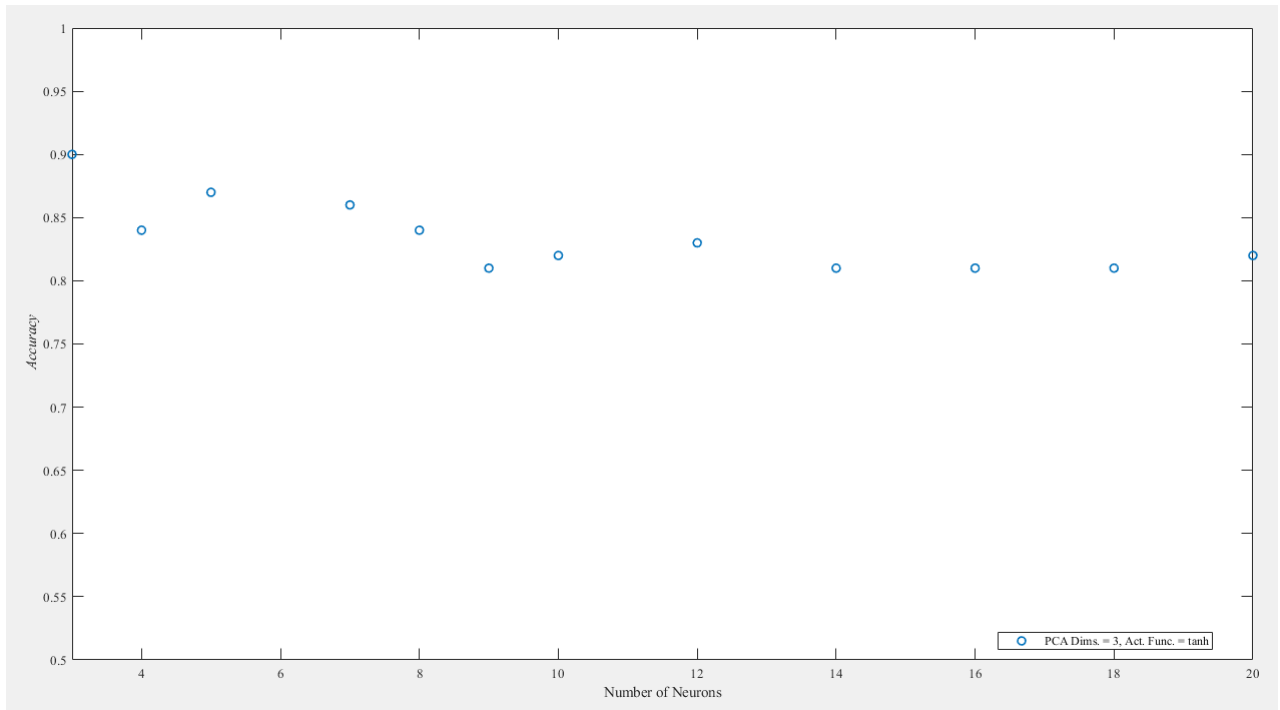

Figure 6.9: Accuracy of classifying whitened data with ANN with a hyperbolic tangent activation function.

In contrast to ANNs with a sigmoid classification function, we observe a higher inconsistency in classification performance, which drops off from ACC ≥ 0.850 to ACC ≈ 0.800. The ANNs with a hyperbolic tangent activation thus performed somewhat worse than the ANNs with a sigmoid activation function. The best performance was achieved for a low number of neurons and approximately a lower to middle (from three to five) number of principal components. The worst results, ACC ≈ 0.600, were achieved with only two principal components and an arbitrary number of neurons.

Conclusions

We conceived this master's thesis as a project developing a novel method of texture-based image analysis, focused first and primarily on three-dimensional images. After introducing our theoretical approach, which uses image filtering, Zernike polynomials, and the Fourier transform to calculate rotationally invariant image characteristics in the first part of this work and commenting on our MATLAB implementation, we also set out to test our approach to texture-based analysis while experimenting on the task of binary image classification of SPECT brain scans of patients suffering from Alzheimer's disease.

Chapters 1–4 are concerned with developing a theoretical approach to texture-based analysis of three-dimensional images. In the first chapter, we introduce some of the basic concepts of image processing, such as the elementary image operations or the Fourier transform used for fast image filtering. These concepts are further developed and elaborated on in the following chapters. Chapter 2 is concerned with introducing both low-pass and high-pass image filters, as well as nonlinear image transformations. Together, these two concepts allow us to preprocess three-dimensional images and achieve fast filtering in the frequency domain. The approach of filtering in the frequency domain is also used in Chapter 3 to develop rotationally invariant image characteristics based on the Fourier transform of Zernike polynomials. These characteristics then serve as inputs of binary image classifiers - the discriminant analysis classifiers, k-nearest neighbours, support vector machines and artificial neural networks - introduced in Chapter 4 together with our approach to statistically testing data separability, whitening the data, and measuring the performance of binary classifiers.

In Chapter 5 and Appendix A, we provide some details of our MATLAB library implementation and comment on some of the most important code excerpts. We divide our library into several more or less overlapping parts. We first define the two classes defined in our implementation, designed to hold the predefined configurations and the calculated results. Likewise, we also define the functions used to read and normalize SPECT images. We continue by describing the functions related to image filtering in the frequency domain, some of which are also used while calculating the rotationally invariant characteristics described in the next section of the chapter. We then move on to describing the functions related to testing the data separability and classifying three-dimensional images. Finally, we describe the function concerned with saving the obtained results.

In Chapter 6, we put our method to practical use while testing it on SPECT images of healthy, cognitively normal patients, as well as patients suffering from Alzheimer's disease. In the first part of this chapter, we attempted to find statistically significant image characteristics that would enable us to separate the two classes by their differing means, or medians. Using the Liliefors test, we generally found out that the data could not be considered coming from a normal distribution, which prevented us from using the Welch's t-test. Instead, we used the non-parametric Mann-Whitney U test and observed that the characteristics with the most potential for class separation are skewness, kurtosis, and median. In the next section, we tried to classify the images using full, non-whitened data, which resulted in good classification results of ACC ≥ 0.850 while using the discriminant analysis classifiers. In the next section, we tested the usefulness of each one-dimensional single characteristic in patient classification and concluded that while some of the characteristics that were initially labelled as statistically significant also formed good classification input data, there were also characteristics that provided decent results despite not being labelled statistically significant. The use of the combination of these characteristics thus significantly improved classification results, as we could see while classifying with full or whitened data. In the section concerned with presenting the results of classifying the whitened data, we could observe that we achieved the best classification performance while using either the quadratic discriminant analysis, support vector machines, or artificial neural networks. Observing what we consider to be good classification results, ACC ≈ 0.900, our proposed method of texture analysis coupled with one of these classifiers (for example, the QDA because of its comparably low computational costs) can be considered a promising approach, which could, if tested on a wider selection of data, find its usage as a fast consultative tool in the field of medical diagnostics.

As a conclusion to this thesis, we attempt to provide a few thoughts on possible further developments of this project. One manner in which we could further improve our method is by adding and testing more image characteristics, more types of binary classifiers, more methods of data whitening, or by attempting to develop a theoretical approach that would enable us to efficiently choose only some of the calculated characteristics as inputs to binary classifiers, which would reduce computation and memory costs. We could also acquire better hardware (or perhaps adapt this project to run on cloud computing) to slash the computation time. Another key way to develop this project further would be to acquire more medical data, not only related to Alzheimer's disease but also to other diseases such as various types of cancer, which would enable us to test our method for monitoring textural changes in cancer-afflicted tissues. We could also try to not limit ourselves to medical data and test our method of textural analysis on data from different fields of image analysis. Should our method work well with a wider selection of data, we are of the opinion that it could provide real and tangible results, not only in medical diagnostics but also as a promising new theoretical approach to texture-based image analysis.

References

[1] GONZALES, R. C., WOODS, R. E. *Digital Image Processing*. 4th Ed. NY: Pearson, 2018. ISBN: 9780133356724.

[2] BRACEWELL, R. N. *The Fourier Transform & Its Applications*. 3rd Ed. Singapore: The McGraw-Hill Companies, 2000. ISBN: 9780073039381.

[3] HUMEAU-HEURTIER, A. Texture Feature Extraction Methods: A Survey. In: *IEEE Access*, Vol. 7, 2019, pp. 8975–9000.

[4] HARALICK, R. M., SHANMUGAM, K., DINSTEIN, I. Textural Features for Image Classification. In: *IEEE Transactions on Systems, Man and Cybernetics*, Vol. SMC-3, No. 6, 1973, pp. 610–621.

[5] TAMURA, H., MORI, S., YAMAWAKI, T. Textural Features Corresponding to Visual Perception. In: *IEEE Transaction on System, Man and Cybernetics*, Vol. 8, No. 6, 1978, pp. 460–473.

[6] CONNERS, R., HARLOW, CH. A. A Theoretical Comparison of Texture Algorithms. In *IEEE Transaction on Pattern Analysis and Machine Intelligence*, Vol. PAMI-2, No. 3, 1980, pp. 204–222.

[7] TAN, J. et al. 3D-GLCM CNN: A 3-Dimensional Gray-Level Co-Occurrence Matrix-Based CNN Model for Polyp Classification via CT Colonography. In: *IEEE Transactions on Medical Imaging*, Vol. 39, No. 6, 2019, pp. 2013–2024.

[8] MAJTNER, T., SVOBODA D. Extension of Tamura Texture Features for 3D Fluorescence Microscopy. Proceedings of: *2012 Second International Conference on 3D Imaging, Modeling, Processing, Visualization & Transmission*, Zurich, Switzerland, 2012, pp. 301–307.

[9] BHARATI, M. H., LIU, J. J., MACGREGOR, J. F. Image Texture Analysis: Methods and Comparison. In: *Chemometrics and Intelligent Laboratory Systems*, Vol. 72, Iss. 1, 2004, pp. 57–71.

[10] MAANI, R., KALRA, S., YANG, Y.-H. Rotational Invariant Local Frequency Descriptors for Texture Classification. In: *IEEE Transactions on Image Processing*, Vol. 22, No. 6, 2013, pp. 2409–2419.

[11] TEUNER, A., PICHLER, O., HOSTICKA, B. J. Unsupervised Texture Segmentation of Images Using Tuned Matched Gabor Filters. In: *IEEE Transactions of Image Processing*, Vol. 4, No. 6, 1995, pp. 863–870.

[12] BARNER, K. E., ARCE, G. R. *Nonlinear Signal and Image Processing: Theory, Methods, and Applications.* Boca Raton, FL: CRC Press, 2003. ISBN: 9780849314278.

[13] LÉVY, P. *Calcul des Probabilités.* 2nd Ed. Éditions Jacques Gabay, Paris, 2006. Reprint of the original edition published by Gauthier-Villars, Paris, France, 1925. ISBN: 9782876472310.

[14] ARCE, G. R. *Nonlinear Signal Processing: A Statistical Approach.* Hoboken, NJ: John Wiley & Sons, Inc., 2004. ISBN: 9780471676249.

[15] ZOLOTAREV, V. M. One-Dimensional Stable Distributions. In: *Translations of Mathematical Monographs*, Vol. 65. Providence, RI: American Mathematical Society, 1986. ISBN: 9780821845196.

[16] TSIHRINTZIS, G. A., SHAO, M., NIKIAS, CH. L. Recent Results in Applications and Processing of α-stable-distributed Time Series. In: *Journal of the Franklin Institute*, Vol. 333, Iss. 4, 1996, pp. 467–497.

[17] BOX, G. E. P., COX, D. R. An Analysis of Transformations. In: *Journal of the Royal Statistical Society: Series B (Methodological)*, Vol. 26, Iss. 2, 1964, pp. 211–243.

[18] CHEDDAD, A. On Box-Cox Transformation for Image Normality and Pattern Classification. In: *IEEE Access*, Vol. 8, 2020, pp. 154975–154983.

[19] LEE, J.-D. et al. MR Image Segmentation Using a Power Transformation Approach. In: *IEEE Transactions on Medical Imaging*, Vol. 28, No. 6, 2009, pp. 894–905.

[20] HERREROS, D. et al. Approximating Deformation Fields for the Analysis of Continuous Heterogeneity of Biological Macromolecules by 3D Zernike Polynomials. In: *IUCrJ (International Union of Crystallography)*, Vol. 8, Pt. 6, 2021, pp. 992–1005.

[21] ABRAMOWITZ, M., STEGUN, I. A. *Handbook of Mathematical Functions: with Formulas, Graphs, and Mathematical Tables.* 9th Ed. National Bureau of Standards, 1970. ISBN: 9780486612720

[22] JANSSEN, A. J. E. M. *Generalized 3D Zernike Functions for Analytic Construction of Band-Limited Line-Detecting Wavelets.* 2015 [online] [retr. 24-03-23] Available: `https://arxiv.org/pdf/1510.04837.pdf`

[23] LIU, H. et al. Computation of Fluctuation Scattering Profiles via 3D Zernike Polynomials. In: *Acta Crystallographica Section A: Foundations of Crystallography*, Vol. 68, Iss. 5, 2012, pp. 561–567.

[24] DAI, G.-M. Zernike Aberration Coefficients Transformed to and from Fourier Series Coefficients for Wavefront Representation. In: *Optics Letters*, Vol. 31, No. 4, 2006, pp. 501–503.

[25] NOVOTNI, M., KLEIN, R. 3D Zernike Descriptors for Content Based Shaped Retrieval. In: *SM '03: Proceedings of the Eighth ACM Symposium on Solid Modelling and Applications*, New York, NY, pp. 216–225.

[26] MANN, H. B., WHITNEY, D. R. On a Test of Whether One of Two Random Variables is Stochastically Larger Than the Other. In: *The Annals of Mathematical Statistics*, Vol. 18, No. 1, 1947, pp. 50–60.

[27] SIEGAL, S. *Nonparametric Statistics for the Behavioral Sciences* 1st Ed., NY: McGraw-Hill, 1956. ISBN: 9780070573.

[28] LEHMANN, E. L., D'ABRERA, H. J. M. *Nonparametrics: Statistical Methods Based on Ranks*, 1st Ed., CA: Holden-Day, 1975. ISBN: 9780070370739.

[29] KOLMOGOROV, A. N. Sulla determinazione empirica di una legge di distribuzione. In: *Giornale dell'Instituto Italiano degli Attuari*, Vol. 4, 1933, pp. 83–91.

[30] LILIEFORS, H. W. On the Kolmogorov-Smirnov Test for Normality with Mean and Variance Unknown. In: *Journal of the American Statistical Association*, Vol. 62, Iss. 318, 1967, pp. 399–402.

[31] DALLAL, G. E., WILKINSON, L. An Analytic Approximation to the Distribution of Liliefors's Test Statistic for Normality. In: *The American Statistician*, Vol. 40, Iss. 4, 1986, pp. 294–296.

[32] RUXTON, G. D. The Unequal Variance t-Test is an Underused Alternative to Student's t-Test and the Mann-Whitney U Test. In: *Behavioral Ecology*, Vol. 17, Iss. 4, 2006, pp. 688–690.

[33] SATTERTHWAITE, F. E. An Approximate Distribution of Estimates of Variance Components. In: *Biometrics Bulletin*, Vol. 2, No. 6, 1946, pp. 110–114.

[34] DERRICK, B., TOHER, D., WHITE, P. Why Welch's Test is Type I Error Robust. In: *Quantitative Methods for Psychology*, Vol. 12, Iss. 1, 2016, pp. 30–38.

[35] BENJAMINI, Y., HOCHBERG, Y. Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing. In: *Journal of the Royal Statistical Society: Series B (Methodological)*, Vol. 57, Iss. 1, 1995, pp. 289–300.

[36] BENJAMINI, Y., YEKUTIELI, D. False Discovery Rate-Adjusted Multiple Confidence Intervals for Selected Parameters. In: *Journal of the American Statistical Association*, Vol. 100, Iss. 469, 2005, pp. 71–81.

[37] LEIBOVICI, C. *Harmonic Series Sum Approximation.* [online] [retr. 28-03-23] Available: `https://math.stackexchange.com/q/2986766`

[38] HOTELLING, H. Analysis of a Complex of Statistical Variables into Principal Components. In: *Journal of Educational Psychology*, Vol. 24, No. 6, 1933, pp. 417–441.

[39] JOLLIFE, I. T. *Principal Component Analysis.* 2nd Ed. Springer Series in Statistics. NY: Springer-Verlag, Inc., 2002. ISBN: 9781441929990.

[40] FRIEDMAN, J. H. Regularized Discriminant Analysis. In: *Journal of the American Statistical Association*, Vol. 84, Iss. 405, 1989, pp. 165–175.

[41] HASTIE, T., TIBSHIRANI, R., FRIEDMAN, J. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* 2nd Ed., 12th Corrected Printing, Springer Series in Statistics, NY: Springer New York, 2017. ISBN: 9780387848587.

[42] PENROSE, R. A Generalized Inverse for Matrices. In *Mathematical Proceedings of the Cambridge Philosophical Society*, Vol. 51, Iss. 3, 1955, pp. 406–413.

[43] MKHADRI, A., CELEUX, G., NASROALLAH, A. Regularization in Discriminant Analysis: An Overview. In: *Computational Statistics & Data Analysis*, Vol. 23, Iss. 3, 1997, pp. 403–423.

[44] DI PILLO, P. J. The Application of Bias to Discriminant Analysis. In: *Communications in Statistics - Theory and Methods*, Vol. 5, Iss. 9, 1975, pp. 843–854.

[45] CHRISTMANN, A., STEINWART, I. *Support Vector Machines.* 1st Ed., Information Science and Statistics, NY: Springer New York, 2008. ISBN: 9780387772417.

[46] CORTES, C., VAPNIK, V. N. Support-Vector Machines, In: *Machine Learning*, Vol. 20, 1995, pp. 273–297.

[47] BISHOP, CH. M. *Pattern Recognition and Machine Learning.* 1st Ed., Information Science and Statistics, NY: Springer New York, 2006. ISBN: 9780387310732.

[48] THARWAT, A. Classification Assessment Methods. In: *Applied Computing and Informatics*, Vol. 17, No. 1, 2020, pp. 168–192.

[49] HREBIK, R., KUKAL, J. Concept of Hidden Classes in Pattern Classification. In: *Artificial Intelligence Review*, 2023. [online] [retr. 22-02-2023] Available: `https://link.springer.com/content/pdf/10.1007/s10462-023-10430-6.pdf`

[50] BALAYLA, J. Prevalence Threshold ϕe and the Geometry of Screening Curves. In: *PLoS ONE*, Vol. 15, No. 10, 2020.

[51] TAHA, A. A., HANBURY, A. Metrics for Evaluating 3D Medical Image Segmentation: Analysis, Selection, and Tool. In: *BMC Medical Imaging*, Vol. 15, Article No. 29, 2015.

[52] MEILA, M. Comparing Clusterings - An Information Based Distance. In: *Journal of Multivariate Analysis*, Vol. 98, Iss. 5, 2007, pp. 873–895.

[53] The Mathworks Inc. *MATLAB*. Ver. 9.11.0 (R2021b), Natick, MA: The MathWorks Inc., 2021. [software] [retr. 14-12-2021], Available: `https://www.mathworks.com`

[54] The Mathworks Inc. *Statistics and Machine Learning Toolbox*™. R2021b, Natick, MA: The MathWorks Inc., 2021. [software] [retr. 01-02-2023], Available: `https://www.mathworks.com/help/stats/index.html?s_tid=CRUX_lftnav`

[55] GREENE, CH. `fullfig`. Ver. 1.0.0.0, MATLAB Central File Exchange, [software] [retr. 01-03-2023], Available: `https://www.mathworks.com/matlabcentral/fileexchange/48071-fullfig`

[56] ALTMAN, Y. `export_fig`. Ver. 3.33, Github, [software] [retr. 01-03-2023], Available: `https://github.com/altmany/export_fig/releases/tag/v3.33`

[57] KUKAL, J. et al. Morfologická analýza segmentů 3D SPECT obrazu mozku v diagnostice Alzheimerovy demence. In: *Psychiatrie*, Vol. 11, Suppl. 3, 2007, pp. 53–55.

Appendix A

Code Excerpts

This appendix is concerned with showing excerpts of some of the more interesting parts of our MATLAB [53] library implementation. These briefly commented excerpts are by no means exhaustive but instead try to show some of the key parts of our library.

A.1 Inputs, Configurations, Results

configurationClass

The class `configurationClass` defines configuration properties and presets several configurations used to obtain results shown in Chapter 6. An example of one preset is shown in the code A.1. We may see the definition of folders containing input data (lines 2 and 3), number of images to be randomly selected from these folders (lines 4 and 5) and also the setting of properties relevant to 3D image processing as defined in chapters 3 and 4.

```matlab
case 11
    conf.pos_img_datastore = imageDatastore("data\
        AD_BARTOS\","FileExtensions", ".img", "ReadFcn", @
        loadImage);
    conf.neg_img_datastore = imageDatastore("data\NOSO\",
        "FileExtensions", ".img", "ReadFcn", @loadImage);
    conf.num_pos_imgs = 50;
    conf.num_neg_imgs = 50;

    conf.bin_theta = 0.2;

    conf.calc_image_features = true;
    conf.func_name = "zerPoly";
    conf.l_max = 2;
    conf.n_max = 2;
```

```
13        conf.rho = 7;
14
15        conf.filter_name = ["noFilt" "fftLP"];
16        conf.kernel_style = ["noKer" "buttKer"];
17        conf.nonlin_trans = "noTran";
18
19        conf.features = ["max", "min", "range", "mean", "med
            ", "mad", "1Q", "3Q", "IQR", "var", "skew", "kurt
            "];
20
21        conf.test_stat_importance = true;
22
23        conf.whiten_data = true;
24
25        conf.class_images = true;
```

Code A.1: Example of a computing configuration.

resultClass

The class `resultClass` defines an object containing relevant results such as tables of classification statistics (lines 25–31) and classification plots (lines 34–43), which are featured in Chapter 6.

```
1  %% Result Class
2  % class defining results/outputs such as image features,
       training datasets,
3  % classification results, etc.
4
5  classdef resultClass
6      properties
7          % Calculated image features
8          pos_imgs_features = NaN;
9          neg_imgs_features = NaN;
10
11         % Significant image features after statistical
               testing, best
12         % configuration acc. to ranksum and t-test,
               number of statistically
13         % significant features after FDR refinement
14         pos_imgs_features_signif = NaN;
15         neg_imgs_features_signif = NaN;
16         ranksum_stat_importance_all = NaN;
17         t_stat_importance_all = NaN;
18         select_num_hyps_false_corr = NaN;
19
```

```
20          % Classifier inputs
21          X_train = NaN;
22          X_train_whit = NaN;
23
24          % Tables of classification statistics
25          class_stats_leave1 = NaN;
26          class_stats_stratkfold = NaN;
27          class_stats_leave1_triv = NaN;
28          signif_feats_class_stats_leave1 = NaN;
29          class_stats_stratkfold_triv = NaN;
30          class_stats_leave1_whit = NaN;
31          class_stats_stratkfold_whit = NaN;
32
33          % Accuracy figures for best classifiers
34          class_acc_leave1_lda = [];
35          class_acc_leave1_qda = [];
36          class_acc_leave1_knn = [];
37          class_acc_leave1_lda_whit = [];
38          class_acc_leave1_qda_whit = [];
39          class_acc_leave1_knn_whit = [];
40          class_acc_leave1_svm_whit = [];
41          class_acc_leave1_sigmoid_whit = [];
42          class_acc_leave1_tanh_whit = [];
43      end
44  end
```

Code A.2: `resultClass` and its properties.

loadImage

The function `loadImage` reads a SPECT image as a 1D vector from a `.img` file specified by a `.hdr` header (lines 4–12) and transforms it into a 3D image (lines 20–32).

```
1   image_filename = image_filename(1:dot_ind-1);
2
3   % read .hdr header
4   header_file = fopen([image_filename '.hdr'],'r');
5   input_header = fread(header_file);
6   fclose(header_file);
7
8   % 3D .img image sizes and define image
9   m = double(input_header(43) + 256*input_header(44));
10  n = double(input_header(45) + 256*input_header(46));
11  p = double(input_header(47) + 256*input_header(48));
12  X = uint32(zeros(m, n, p));
13
```

```
14  % read .img 1D vector
15  img_file = fopen([image_filename '.img'],'r');
16  input_image = fread(img_file);
17  fclose(img_file);
18
19  % create 3D image by transforming data with step size
20  step_size = length(input_image)/m/n/p;
21  cur_ind = 1;
22  for k = 1:p
23      for j = 1:n
24          for i = 1:m
25              X(i, j, k) = uint32(input_image(cur_ind));
26              if(step_size >= 2 && step_size <= 4)
27                  X(i, j, k) = X(i, j, k) + uint32(256*sum(
                        input_image(cur_ind + step_size - 1:
                        cur_ind+1)));
28              end
29              cur_ind = cur_ind + step_size;
30          end
31      end
32  end
33
34  input_image = X;
35  spect_bool = true;
```
<div align="center">Code A.3: Reading a 3D SPECT image.</div>

A.2   Global Filtering

filterFunctionFFT

The function `filterFunctionFFT` pads the input image `X` (line 12), calculates the FFT using the MATLAB function `fftn` (line 15) and filters the image in the frequency domain using a kernel `K`. Furthermore, the function then transforms the filtered image back into spatial domain using MATLAB function `ifftn` (lines 19–26).

```
1  %% Frequency domain filtering driver
2  % takes spatial input image X and frequency kernel K,
       transforms X into
3  % frequency domain and performs frequency domain
       filtering
4
5  % inputs: X - input image in spatial domain
6  %        : K - filtering kernel in frequency domain
7  %        : frame_style - padding style
```

```
 8  % outputs: Y - spatial domain filtered image
 9
10  function Y = filterFunctionFFT(X, K, frame_style)
11      [m, n, p] = size(X);
12      X = getPaddingFFT(X, m, n, p, frame_style);
13
14      % transform input image X to frequency domain
15      X = fftshift(fftn(X));
16
17      % perform filtering as element-wise matrix
18          multiplication in frequency
19      % domain
19      Y = X .* K;
20
21      % transform filtered image Y back to spatial domain
22      Y = ifftn(ifftshift(Y));
23
24      % select the part of the filtered image corresponding
                to the original
25      % input image
26      Y = Y(1:m, 1:n, 1:p);
27  end
```

Code A.4: Filtering 3D image in the frequency domain.

A.3   Invariant Calculation

calcInvars

The function `calcInvars` computes (scaled) rotational invariants according to 3.2 (lines 16–34). We see that this is done according to (3.12) through using three `for` cycles, where the coefficients are calculated using `filterFunctionFFT` (see above).

```
 1  %% Calculate Invariants
 2  % calculates image invariants by summing invariant
        coefficients
 3
 4  % inputs: input_image - normalized input image
 5  %       : func_name - function to be used for invariant
        calculation
 6  %       : frame_style - frame style to be used for
        invariant FFT
 7  %                       calculation
 8  %       : l_max - max. spherical harmonic l
 9  %       : n_max - max. spherical harmonic n
```

```matlab
10 %         : rho - function radius
11 % outputs: invars_nl - all image invariants from 0 to n
     and 0 to l
12
13 function invars_nl = calcInvars(input_image, func_name,
     frame_style, l_max, n_max, rho)
14     [M, N, P] = size(input_image);
15     invars_nl = zeros((n_max + 1) * (l_max + 1), M, N, P)
         ;
16   for n = 0:n_max
17       for l = 0:l_max
18           coeffs_nlm = zeros(2*l + 1, M, N, P);
19
20           ft_func_m_zero = calcInvarFunc(func_name, l,
               0, n, rho, 2*N, 2*M, 2*P);
21           coeffs_nlm(l+1, :, :, :) = abs(
               filterFunctionFFT(input_image,
               ft_func_m_zero, frame_style)).^2;
22
23           if(l > 0)
24               for m = 1:l
25                   ft_func_m_minus = calcInvarFunc(
                       func_name, l, -m, n, rho, 2*N, 2*M
                       , 2*P);
26                   ft_func_m_plus = calcInvarFunc(
                       func_name, l, m, n, rho, 2*N, 2*M,
                        2*P);
27
28                   coeffs_nlm(m, :, :, :) = abs(
                       filterFunctionFFT(input_image,
                       ft_func_m_minus, frame_style)).^2;
29                   coeffs_nlm(2*l + 2 - m, :, :, :) =
                       abs(filterFunctionFFT(input_image,
                        ft_func_m_plus, frame_style)).^2;
30               end
31           end
32           invars_nl((n_max + 1)*n + l + 1, :, :, :) =
               reshape(sum(coeffs_nlm, 1), [M N P]);
33       end
34   end
35 end
```

Code A.5: Calculating rotational invariants.

A.4 Data Transformation

dataWhiteningDriver

The function `dataWhiteningDriver` transforms higher dimensional data into 2D arrays containing features of AD and CN images. This is rather trivially achieved through MATLAB function `reshape` and therefore we only show the part of the code concerned with creating the whitened data. We can see that the whitened data is a 3D array of sizes corresponding to the total number of whitening dimensions setting (we may want to try whitening the data into e.g. 2, 3 or more principal components, and it is thus practical to save all these results into one array), total number of processed images, and the maximum number of whitening dimensions (for settings with smaller number of dimensions than the maximum, the remaining columns remain all zero). The whitening itself is done according to 4.2.4 and need not be further discussed here.

```
1  X_train_whit = zeros(num_num_whit_features, num_pos_imgs
       + num_neg_imgs, max(num_whit_features));
2  for i = 1:num_num_whit_features
3      [~, ~, ~, X_train_whit(i, :, 1:num_whit_features(i))]
           = whitenDataPCA(X_train, num_whit_features(i));
4  end
```

Code A.6: Data whitening cycle.

A.5 Image Classification

classImages

The function `classImages` trains and cross-validates a selected classifier by using techniques described in 4.4. Below, we show the excerpt concerned with the leave-one-out cross-validation. The calculations done in the function `calcClassStats` are somewhat trivial, sufficiently described in 4.4.1 and need not be discussed here.An example of a training function (for SVM) called through `feval` on line 7 is shown in a separate subsection below.

```
1  perm_ind_train = randperm(num_imgs_train)';
2  X_train = X_train(perm_ind_train, :);
3  y_star_train = y_star_train(perm_ind_train);
4  for i = 1:num_imgs_train
5      X_train_temp = [X_train(1:i-1, :); X_train(i+1:
           num_imgs_train, :)];
6      y_star_train_temp = [y_star_train(1:i-1);
           y_star_train(i+1:num_imgs_train)];
```

```matlab
7      res_train_leave1 = feval(classifier_train_name,
           X_train_temp, y_star_train_temp, train_params);
8      Y_train(i, 1) = feval(classifier_class_name, X_train(
           i, :)', res_train_leave1);
9  end
10 class_stats_leave1 = calcClassStats(Y_train, y_star_train
       , num_pos_imgs_train, num_neg_imgs_train);
```

Code A.7: Leave-one-out cross-validation.

trainSVM

Below, we show the `trainSVM` function utilizing the SVM implementation from [54]. The follow-up classification is then done by calling the `predict` [54] function with the trained `svm_model` and validation data as parameters.

```matlab
1  %% Train SVM
2  % train Matlab Support Vector Machine
3
4  % inputs: X - inputs for training
5  %        : y_star - class labels
6  %        : train_params - classifier training parameters
7  % outputs: res_train - trained SVM model
8
9  function res_train = trainSVM(X, y_star, train_params)
10     type = train_params(1);
11     outlier_coeff = train_params(2);
12     nu = train_params(3);
13     if(size(train_params, 2) > 3)
14         sigma = train_params(4);
15     end
16     if(type > 0)
17         svm_model = fitcsvm(X, y_star,'KernelFunction','
               polynomial', 'KernelScale','auto', ...
18             'PolynomialOrder', type, 'OutlierFraction',
                   outlier_coeff, 'Nu', nu, 'Standardize',
                   true);
19     elseif(type == 0)
20         svm_model = fitcsvm(X, y_star,'KernelFunction', '
               rbf', 'KernelScale', 2*sigma^2, ...
21             'OutlierFraction', outlier_coeff, 'Nu', nu,
                   'Standardize', true);
22     end
23
24     res_train = {svm_model};
```

```
25   end
```

Code A.8: Leave-one-out cross-validation.

A.6   Saving Results

saveResults

Below we may see the function `writeData`, which is a part of the larger `saveResults.m` file. We may see that we technically save 4 types of results - MATLAB variables `.mat`, which are useful when wanting to run the library without calculating image features but instead merely using already prepared files, `.csv` matrices or tables (depending on whether the result needs row and column descriptors), and `.fig` MATLAB figures together with `.png` exported images. On lines 29 and 32 we may see incorporation of community-created functions `fullfig` [55] and `export_fig` [56] into our library. The `fullfig` function enlarges the figure to a whole screen, achieving a better resolution when saving. In our case, the `export_fig` is used to export the figure into a `.png` image format.

```
1    %% Write Data
2    % write data into a file
3
4    % inputs: data - data array/table/figure
5    %       : folder_name - folder, into which the result
         will be saved
6    %       : filename -  filename of file to be saved
7    %       : mode - saving either array/table/figure + png
8
9    function [] = writeData(data, folder_name, filename, mode
         )
10       folder_name = "results/" + folder_name;
11       if(~exist(folder_name, 'dir'))
12           mkdir(folder_name);
13       end
14
15       if(mode == 0)
16           filename = folder_name + "/" + filename + ".mat";
17           save(filename, 'data');
18       elseif(mode == 1)
19           filename = folder_name + "/" + filename + ".csv";
20           save(filename, 'data');
21           writematrix(data, filename);
22       elseif(mode == 2)
23           filename = folder_name + "/" + filename + ".csv";
```

```
24          writetable(data, filename, 'WriteRowNames', true)
                ;
25      elseif(mode == 3)
26          filename_fig = folder_name + "/" + filename + ".
                fig";
27          filename_png = folder_name + "/" + filename + ".
                png";
28          fig = figure(data);
29          fullfig(fig);
30          data.Visible = 'off';
31          saveas(fig, filename_fig);
32          export_fig(fig, filename_png, '-p0.02');
33          close(data);
34      end
35  end
```

Code A.9: Saving results.