Double Degree in
Computer Science and Mathematics

UNIVERSIDAD DE GRANADA

Final Degree Project

Machine Learning for Diagnosis of Alzheimer's Disease and Early Stages

Julio José Prado Muñoz

Tutor
Ignacio Rojas Ruiz

9th July 2021

Feature Extraction and Machine Learning for Diagnosis of Alzheimer's Disease and Early Stages

Julio José Prado Muñoz

Keywords: Alzheimer's disease, cognitive impairment, wavelet transform, multiresolution analysis, minimum redundancy maximum relevance, support vector machine, convolutional neuronal networks, transfer learning

Abstract

According to the WHO, around 50 million people have dementia worldwide, and there are nearly 10 million new cases every year. Alzheimer's disease is the most common form of dementia and may contribute to 60–70 % of cases. It has been proved that early diagnosis is key to promoting early and optimal management. However, early stage of dementia is often overlooked and patients are diagnosed when the disease presents a more advanced stage.

In this context, several attempts have been made to diagnose Alzheimer's disease using machine learning models and slices from MRI images. Among these investigations CNN and SVM showed to have excellent results, given scores higher than 90 % when detecting dementia in advanced stages. Since the wavelet transform replaced Fourier transform in most of today's signal and image processing tasks, SVM fed by these coefficients played a leading role in classifying images, more specifically biomedical ones. Nevertheless, the quick progress in neural networks made in the second half of the recent decade has put aside the fame and usage of this approach in today's image classification problems.

The objective of this investigation is to predict Alzheimer's early stages, not only dementia itself. For this purpose, we will compare the performance of SVM and CNN when dealing with MRI images, developing and collecting the mathematical foundation behind the algorithms used. These tools go through the basis of the wavelet transform to the mechanisms that power the modern convolutional neural networks.

The newness of the experiments conducted in this research is not only the wide range of stages that we aim to predict but also that all the available information will be processed as a whole: we will make use of multiple slices and consider different parts of the brain to give a more accurate response. This is in contrast to the previous experiments where single slices were considered for such a goal.

Overall, excellent results have been obtained and it has been proved that early stages

in Alzheimer's disease have a noticeable impact on the brain that can be used to build machine learning models.

Extracción de características y Aprendizaje Máquina para el diagnóstico del Alzheimer y estadios tempranos

Julio José Prado Muñoz

Palabras clave: enfermedad de Alzheimer, deterioro cognitivo, transformada wavelet, análisis multiresolución, mínima redundancia y máxima relevancia, máquina de soporte vectorial, redes neuronales convolucionales, transferencia de aprendizaje

Resumen

De acuerdo con la OMS, alrededor de 50 millones de personas sufren demencia en el mundo, diagnosticándose 10 millones de nuevos casos por año. La enfermedad del Alzheimer es la forma más común de demencia y está detrás del 60-70 % de los casos. Se ha demostrado que el diagnóstico temprano de dicha enfermedad es la clave para aplicar un tratamiento óptimo. No obstante, dichos estadios pasan normalmente desapercibidos y sólo se le diagnostica la enfermedad al paciente cuando presenta un estado más avanzado.

En este contexto se han realizado varios intentos para diagnosticar la enfermedad de Alzheimer usando modelos de aprendizaje máquina y cortes provenientes de imágenes MRI. Entre dichas investigaciones las redes neuronales convolucionales y las máquinas de soporte vectorial han mostrado un rendimiento excelente detectando la enfermedad por encima del 90 % de los casos en estados avanzados. Desde que la transformada wavelet sustituyera a la transformada de Fourier en las tareas actuales de procesamiento de señales e imágenes las máquinas de soporte vectorial han estado siendo alimentadas por dichos coeficientes. Estas máquinas han tenido hasta la fecha un papel predominante en las tareas de clasificación de imágenes y más especialmente, de imágenes biomédicas. Sin embargo, el rápido avance de las redes neuronales en la segunda mitad de la pasada década han mitigado el uso de dichas máquinas en los problemas de clasificación de imágenes actuales.

El objetivo de esta investigación es predecir estadios tempranos de la enfermedad de Alzheimer, no solo la propia enfermedad. Para tal propósito, compararemos el rendimiento de las máquinas de soporte vectorial y las redes neuronales convolucionales en el procesamiento de imágenes MRI, desarrollando y recopilando las bases matemáticas de dichos algoritmos. Estas herramientas van desde la transformada wavelet hasta el mecanismo que sustenta las redes neuronales convolucionales modernas.

La novedad de los experimentos realizados no es únicamente la variedad de etapas que pretendemos predecir y diagnosticar, sino también la utilización de toda la información disponible en el cerebro para realizar el procesamiento. De esta manera se usarán todos

los cortes disponibles, lo cual pretende dar un diagnóstico más exacto del estado del paciente. Esta aproximación contrasta con los estudios anteriores donde se consideraban cortes aislados para la clasificación.

En general, se han obtenido resultados excelentes y se ha podido probar que los estadios tempranos del Alzheimer tienen un impacto perceptible en el cerebro que puede ser utilizado para construir modelos de aprendizaje máquina.

La parte matemática está dividida en tres apartados principales.

La Transformada Wavelet y Extracción de Características.En esta sección se desarrolla la teoría de la transformada wavelet. Se pasa por el origen de la wavelet y su relación con la transformada de Fourier. Se define la transformada wavelet continua y se introduce el esquema estándar para construir wavelets: el análisis multiresolución (MRA). Este método se aplicará a imágenes 2-D (wavelets en el espacio $L^2(\mathbb{R}^2)$), construyendo los fundamentos del famoso algoritmo Mallat. Este algoritmo se usa en última instancia para extraer los coeficientes wavelet que alimentarán nuestro modelo.

Coeficientes Wavelet y Selección de Características. Se estudian los dos métodos que se aplicarán para seleccionar los coeficientes más relevantes de los extraídos en el apartado anterior: mínima redundancia y máxima relevancia (mRMR) y análisis de componentes principales (PCA). Para el primer método se expone la fórmula usada para encontrar la correlación entre dos variables aleatorias X e Y y cómo esta fórmula se aplica sistemáticamente para seleccionar los coeficientes con una gran relación con la clase a predecir pero poca relación con el resto de coeficientes. Por el contrario, PCA no aplica selección de características sino reducción dimensional. La idea es encontrar una base ortogonal y un conjunto de vectores ortogonales que expliquen la variación de los vectores aleatorios en pocas dimensiones. Para ello, necesitamos introducir varios conceptos, entre ellos el Teorema Espectral.

Construcción del Modelo y Clasificación. Esta es probablemente la parte más interesante desde la perspectiva de la inteligencia artificial. Las teorías y algoritmos que envuelven las máquinas de soporte vectorial (SVM) y las redes neuronales convolucionales (CNN) se desarrollan en esta sección. Con lo que respecta a las SVM, el proceso de búsqueda de un hiperplano delimitador en $\mathbb{R}^n$ implica la resolución de un problema dual usando los multiplicadores de Lagrange, extrapolando este proceso a clasificadores no lineales. Dichos clasificadores transforman los datos desde $\mathbb{R}^n$ hasta otro espacio H mediante una función $\phi : \mathbb{R}^n \longrightarrow H(H := \mathbb{R}^N,\ N > n)$. De forma similar, se introducen el método del gradiente estocástico descendente (SDG) y cómo éste se usa para entrenar la CNN mediante la computaciónde gradientes en $\mathbb{R}^n$. Además, se explican las características de cada una de las capas involucradas en una CNN desde una perspectiva matemática.

El núcleo de la investigación, el cual cubre la parte informática de la misma se divide

en dos partes fundamentales:

Investigación. En esta sección se explican todos los pasos y procedimientos realizados a lo largo del desarrollo del trabajo. El funcionamiento de los algoritmos así como las consideraciones realizadas se exponen siguiendo un orden temporal. Los resultados de la SVM, CNN y sus respectivas variaciones son comentados junto con los mecanismos usados para la extracción de características.

Anexo. Contiene los archivos (clases, funciones, constantes, scripts,...) creados para poder desarrollar los pasos en *Investigación*. Este conjunto de archivos está referenciado a lo largo del desarrollo de la investigación.

Yo, **Julio José Prado Muñoz**, alumno de la titulación DOBLE GRADO EN INGENIERÍA INFORMÁTICA Y MATEMÁTICAS de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación y la Facultad de Ciencias de la Universidad de Granada**, con DNI 77448808B, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Julio José Prado Muñoz

Granada a 9 de julio de 2021.

D. **Ignacio Rojas Ruiz**, Profesor del Departamento de Arquitectura y Tecnología de Computadores y Director del Centro de Investigación en Tecnologías de la Información y las Comunicaciones (CITIC-UGR)

Informa:

Que el presente trabajo, titulado ***Machine Learning for Diagnosis of Alzheimer's Disease and Early Stages***, ha sido realizado bajo su supervisión por **Julio José Prado Muñoz**, y autoriza la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expide y firma el presente informe en Granada a 9 de julio de 2021.

El director:

Ignacio Rojas Ruiz

Agradecimientos

Me gustaría agradecer a mi familia el apoyo ferviente e incondicional recibido desde el primer día que decidí embarcarme en este proyecto, así como la fe ciega que siempre han depositado en mí.

A todos los amigos que he podido hacer durante esta carrera, pues cada uno de ellos ha dejado una huella en mí que más grande o más pequeña, ha colaborado a la finalización de este proyecto y del Doble Grado.

Por último y más importante, a mi tutor Ignacio Rojas por la confianza prestada y la disposición a hacer de este proyecto algo más que un simple trabjo de fin de grado.

General Index

Chapter 1

Introduction

This chapter describes the contents that are discussed through the thesis. For this, we present the context that frames the investigation and the justification for the studies carried out. Finally, the objectives to be followed are specified.

1.1. Context

Wavelet transform and its applications are a powerful tool in image processing. The recent advances in neural networks and transfer learning methods opened the door to discussions about the goodness and supremacy of neural networks over the wavelet transform in pattern recognition and classification.

Since wavelet analysis rose early in 1980s, it has proved to be useful in lots of fields of applied mathematics, specially in signal and image processing. The appearance of neural networks has mitigated its fame in image-classification tasks and even more after Oxford or Google among others developed their unbeatable pre-trained models in the second half of the 2010s.

At the same time, Alzheimer's disease has established itself as one of the great epidemics of the 21st century, being the most common case of dementia. In this context, several attempts have been made to detect and diagnose Alzheimer using machine learning and brain images, typically MRI. Example of this approach can be found in [15] and [14] where specific slices were used to generally train models able to predict two binary stages: control and disease. Nowadays, the use of neural networks and support vector machine with wavelet transform as tool to extract features is still unclear and there is no common criteria on which approach performs better to diagnose diseases from biomedical images.

1.2. Justification and proposal

Alzheimer's disease has proved to present intermediate stages before the severity of the dementia becomes moderate or serious. These stages can be noticeable and sometimes patients are even aware of their own cognitive impairment. However, such symptoms can go unnoticed until the disease presents an advanced stage. Detecting and diagnosing early stages can be the key to provide patients with better prognosis and preventive treatments in order to improve their quality of life.

Although the latest experiments did not fully respond to the question of whether it is possible to detect early stages in Alzheimer's disease or not, they were able to find a correlation between the information contained in patients' MRI images and the presence of dementia as shown in [15] or [16] . Nevertheless, this correlation was based uniquely in the processing of single slices i.e., these experiments made use of a small portion of the information available.

By using all the slices we could be able to give a more precise prediction of the status of the patient. The problem of this approach is the huge amount of data to process: to train a model we need to store the information of many images together in RAM memory which is not feasible for a regular machine. Despite this issue it is possible to create smart algorithms and extraction processes to efficiently obtain the most relevant features as well as single-slice models where all the outcomes are joined together to construct a multi-criteria classifier. We could think of this approach as a prepared report that imitates the prognosis of multiple doctors and decides the most likely outcome from them.

Meanwhile, the dilemma of choosing between neural networks (more specifically CNN) and support vector machine for biomedical images processing has never been tackled using the full information of the brain and super normalized images. This is, it becomes relevant to study the performance of the 2-D MRA in $L^2(\mathbb{R}^2)$ aligned with SVM against CNN and their stochastic gradient descendent (SDG) method. To do so, all the theory behind both approaches will be study, being an excellent starting (and advanced) point to learn the functioning and mathematical basis behind them.

Overall, in this research we aim to give an answer to several questions. First, whether it is possible to predict and prevent the appearance of the Alzheimer's disease or not considering up to six stages of dementia: cognitively normal, significant memory concern, early mild cognitive impairment, mild cognitive impairment, late mild cognitive impairment and Alzheimer disease. Secondly, which of the aforementioned algorithms performs better. Thirdly, what is the result of processing multiple slices in conjunction instead of using single slices for classification. And fourthly, how to address the problem of memory management when using larger information for model training.

1.3. Contents

The **mathematical** part is grouped into three main sections.

The Wavelet Transform and Feature Extraction. Develops the theory behind the wavelet transform. This section covers the origins of the wavelets and the link between the wavelet transform and the Fourier transform. It also defines the continuous wavelet transform and establishes the standard scheme to construct wavelets: multiresolution analysis (MRA). This method will be apply to 2-D images (wavelets in the space $L^2(\mathbb{R}^2)$) to build the foundation of the famous Mallat algorithm. This algorithm is the one that we will use ultimately to extract the wavelet coefficients to use as features when feeding our models.

Wavelet Coefficients and Feature Selection. Studies the two methods that we will apply to perform feature selection once the wavelet coefficients have been extracted: Minimum Redundancy Maximum Relevance (mRMR) and Principal Component Analysis (PCA). For the first method, the formula used to find the correlation between two random variables X and Y is exposed and also how this formula is applied systematically to select the features with a high correlation with the class and low correlation between themselves. On the contrary, PCA does not apply feature selection but dimensionality reduction. The idea is to find an orthogonal basis and a subset of orthogonal vectors that explains the variation of random vectors in fewer dimensions. To do so, we need to introduce several concepts and theorems including *The Real Spectral Theorem*.

Model Building and Classification. This is probably the most interesting part from the AI's perspective. Here the theories behind Support Vector Machine (SVM) and Convolutional Neural Networks (CNN) are exposed. Regarding SVM, the process of finding a delimiter hyperplane in $\mathbb{R}^n$ implies solving a dual problem using Lagrange multipliers and how it is extrapolated to nonlinear classifiers. These nonlinear classifiers transform the data from $\mathbb{R}^n$ to some space H via a mapping function $\phi : \mathbb{R}^n \longrightarrow H$ ($H := \mathbb{R}^N, \ \ N > n$). Similarly, we introduce the stochastic gradient descendent (SDG) method and how this system is used to train CNN by the computation of gradients in $\mathbb{R}^n$. In addition, the characteristics of every layer is explained from a mathematical point of view.

The core of the research which covers the part related with **computer science** is divided into two main chapters.

Research. Here all the steps carried out during the investigation are explained and detailed. The functioning of the algorithms and the considerations taken are also exposed following a temporary order. The results of SVM, CNN and all their different variations are reported along with the mechanisms used to extract the features.

Annex. Contains the files (classes, functions, scripts, constants,...) used to develop the

steps in *Research*. This set of files is accordingly referenced through the development of *Research*.

1.4. Modules related with the project

The modules related with this investigation that are recommended to study in order to correctly understand the contents of this thesis are enumerated as follows. These modules are taught in the University of Granada.

The modules related with the **mathematical** part are:

- Geometría I
- Geometría II
- Álgebra I
- Ánalisis Matemático I
- Ánalisis Matemático II
- Ánalisis Funcional
- Probabilidad
- Inferencia Estadística

The modules related with **computer science** are:

- Fundamentos de la Programación
- Metodología de la Programación
- Estructura de Datos
- Visión por Computador
- Inteligencia de Negocio
- Algorítmica

1.5. Main Sources

The main sources consulted during this research are highlighted as follows. To see the full list of resources please consult bibliography.

- [1] Y. Y. Tan, *Wavelet Theory Approach to Pattern Recognition*. Series in Machine Perception and Artificial Intelligence — Vol. 74 (2009).
- [4] Z. Zhao, R. Anand & M. Wang, *Maximum Relevance and Minimum Redundancy Feature Selection Methods for a Marketing Machine Learning Platform*. IEEE

(2020).

- [6] M. Mei, *Principal Component Analysis*. The University of Chicago (2009).
- [10] J. Wu, *Introduction to Convolutional Neural Networks*. Nanjing University, China (2017).
- [13] F. Falah, Y. Prasad, *Multi-class Support Vector Machine (SVM) classifiers – An Application in Hypothyroid detection and Classification* . Sixth International Conference on Bio-Inspired Computing: Theories and Applications. IEEE (2011).

1.6. Objectives

The established objectives have three different characters and they encompass medical, computing and mathematical goals. These are detailed below.

1.6.1. Medical Objective

The medical objective of the research aim to give an answer to the question of whether it is possible to predict early stages in Alzheimer's disease and therefore give an according and more preventive treatment to those patients with a high likelihood of developing more advanced cognitive impairments. As we will see in future sections we can give a positive response: not only it is possible to predict early stages but also it is possible to do it with a precision higher than 99 %.

1.6.2. Computing Objectives

The objectives involving the computing part are a bit wider and they can be summarized in the following bullet points:

1. First, we want to decide which approach performs better in order to classify normalized biomedical images: SVM or CNN. As we will see, the response will be SVM which corroborates the intuition that leaded this investigation.
2. Secondly, we want to compare the performance of PCA against mRMR when selecting wavelet coefficients from normalized images. Although mRMR is a more recent approach and hence it aroused some interest in this investigation, PCA gave better results.
3. Thirdly, we also aim to develop smart systems of extraction that can lead us to use as much information as possible when processing such a big set of images.

1.6.3. Mathematical Objective

Last but not least, the mathematical objective is particularly interesting. The goal is to collect and develop the theories behind the tools used in the computing part. Never before has a single investigation covered in such a way the principles and methods most relevant today. They are presented in a manner that anyone with a strong mathematical basis can understand and learn how all the algorithms work. This could be an optimal starting point for anyone aiming to learn machine learning at a theoretical level, even if the reader is experienced in this discipline.

Chapter 2

Mathematical Concepts and Tools

In this section I will expose the tools that support the experiments conducted. The objective is to use this section as a guide to understand the mathematical instruments and theories that make it possible to build a proper machine learning model for image processing, examinating the key points that make a significant impact on the performance, and therefore the results obtained. This angle goes from preprocessing and feature extraction to the most advanced deep learning features.

2.1. The Wavelet Transform and Feature Extraction

In every classification problem the data representation and its neatness become a turning point when looking for a better performance and results. The purpose is not only to have a clean and balanced dataset but also to choose and extract the right content from it.

Following this philosophy, we are going to introduce one of the most powerful instruments used in the past decades to extract information for image processing: the Wavelet transform (WT).

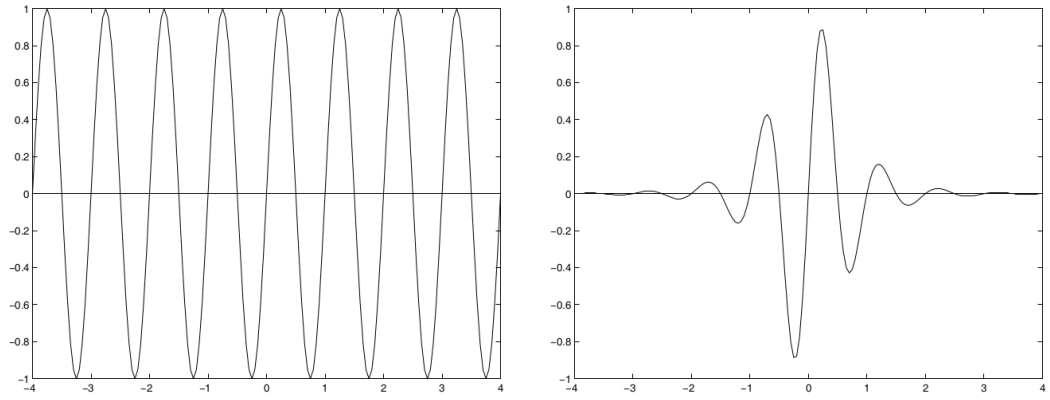
Wavelets can be thought of as kind of a supercharged Fourier transform. Using a Fourier transform, you can decompose a signal into a sum of sines and cosines where these sines and cosines form and orthogonal basis for the space of functions that we want to represent.

This idea of sines and cosines can be generalized using wavelets to other orthogonal functions that might provide a better representation of certain types of signals. Today, we can affirm that wavelets has changed the way we compress and represent signals in the

digital era.

2.1.1. Introduction

Wavelet analysis is a relatively recent development of applied mathematics in 1980s. The thinking behind wavelets is that signals can be locally characterized in both the time domain and frequency domain simultaneously. According to this property, wavelet analysis can be efficiently applied to analyze and process the non-stationary signals.



(a) (b)

Figure 2.1: simple examples of wave and wavelet. (a) wave (b) wavelet

We consider two typical mechanical movements: non-damped simple harmonic vibration and damped oscillation.

A wave can be produced by a non-damped simple harmonic vibration, which is illustrated in figure 2.1(a). This type of signal is well processed using Fourier analysis. By contrast, a wavelet can be formed by a damped oscillation, as shown in figure 2.1(b). Two conditions have to be satisfied in order to consider a signal as a wavelet:

1. The wavelet must be oscillatory.
2. Its amplitudes are nonzero only during a short interval.

Fourier analysis has its own deficiency. In fact, it has two major problems. First, Fourier analysis can not characterize the signals locally in time domain and secondly, Fourier expansion can approximate the stationary signals well, but can not do so for the non-stationary ones.

To contest the above deficiencies, we should find other basic function $\psi(x)$ to replace the basic function $sin(x)$. This new function should satisfy the following conditions:

- Similar to $sin(x)$, any complicated signal $f(x)$ can be constructed by the linear combination of $\psi(jx-k)\,(j,k\in\mathbb{Z})$, which are produced by the dilations and translations

of the basic function $\psi(x)$.

- The expansion coefficients of a signal using the basic function $\psi(x)$ can reflect the locations of the transient or localized image components in the time domain.
- This new basic function $\psi(x)$ and its family can "fit"transient signal $f(x)$ much better than Fourier basic wave $sin(x)$. In other words, they can minimize the error between the approximation of the signal $f(x)$ and $f(x)$ itself.

We already found such a basic function $\psi(x)$ as early as 1910. It is the well-known square function Haar. This basic function $\psi(x)$ can be written in form of

$$\psi(x) = \begin{cases} 1 & x \in [0, \frac{1}{2}) \\ -1 & x \in [\frac{1}{2}, 1) \end{cases} \tag{2.1}$$

It has been mathematically proved that $\{\psi(2^j x - k) \mid j, k \in \mathbb{Z}\}$ can constitute an orthogonal basis of the finite energy signal space $L^2(\mathbb{R})$. They can also establish an orthonormal basis of $L^2(\mathbb{R})$ through a normalization:

$$\psi_{j,k}(x) := 2^{j/2}\psi(2^j x - k),\ (j, k \in \mathbb{Z}) \tag{2.2}$$

Thus, any finite energy signal $f(x)$ can be represented by

$$f(x) = \sum_{j\in\mathbb{Z}} \sum_{k\in\mathbb{Z}} c_{j,k}\psi_{j,k}(x), \tag{2.3}$$

where

$$c_{j,k} := \int_{\mathbb{R}} f(x)\psi_{j,k}(x)dx$$

2.1.2. The Continuous Wavelet Transform

In Fourier transform

$$\mathcal{F}f(\xi) = \int_{\mathbb{R}} f(\xi - x)\overline{e^{i\xi x}}dx \tag{2.4}$$

When replacing $e^{i\xi x}$, the dilation of the wave function e^{ix} by $\psi(\frac{t-b}{a})$, the translation and dilation of the basic wavelet $\psi(x)$, then, the transform is referred as a continuous wavelet transform, which is defined as follows:

2.1.1 Definition. *A function $\psi \in L^2(\mathbb{R})$ is called an admissible wavelet or a basic wavelet if it satisfies the following "admissibility" condition:*

$$C_\psi := \int_{\mathbb{R}} \frac{|\hat{\psi}(\xi)|^2}{|\xi|} d\xi < \infty \text{ ,where } \hat{\psi} \text{ is the Fourier transform of } \psi \tag{2.5}$$

The continuous wavelet transform with kernel ψ is defined by

$$(W_\psi f)(a,b) := |a|^{-\frac{1}{2}} \int_{\mathbb{R}} f(t)\overline{\psi\left(\frac{t-b}{a}\right)}\, dt,\ f \in L^2(\mathbb{R}) \tag{2.6}$$

where $a, b \in \mathbb{R}$ and $a \neq 0$ are the dilation parameter and the translation parameter respectively.

According to the admissibility condition (2.5), if $\psi \in L^1(\mathbb{R})$, it can be inferred that

- $\hat{\psi}(0) = 0$
- $\hat{\psi}(\xi) \to 0$ when $(|\xi| \to \infty)$

This indicates that the function ψ is a bandpass filter. The characteristics of the localized components of a signal $f(t)$ can be described by the continuous wavelet transform.

- Because of the damp of $\psi(x)$ at infinity, the localized characteristics of f near x = b is described by (2.6). If we assume that the $\psi(x)$ is always zero out of $[-1, 1]$, it will be clearer. Then, for all $x \notin [b - |a|, b + |a|]$, we have:

  $$\psi(\frac{t-b}{a}) = 0$$

  Therefore,

  $$\begin{aligned}(W_\psi f)(a,b) &= |a|^{-\frac{1}{2}} \int_{\mathbb{R}} f(t)\overline{\psi\left(\frac{t-b}{a}\right)}\, dt \\ &= |a|^{-\frac{1}{2}} \int_{b-|a|}^{b+|a|} f(t)\overline{\psi\left(\frac{t-b}{a}\right)}\, dt\end{aligned}$$

  Which means that $(W_\psi f)(a,b)$ is completely determined by the behaviours of f in $[b-|a|, b+|a|]$ with the center b (Fig. 2.2). It is said that $(W_\psi f)(a,b)$ describes only the localized characteristics of f in $[b - |a|, b + |a|]$. The smaller a, the better the localized characteristic of f, which can also be found in Fig. 2.2

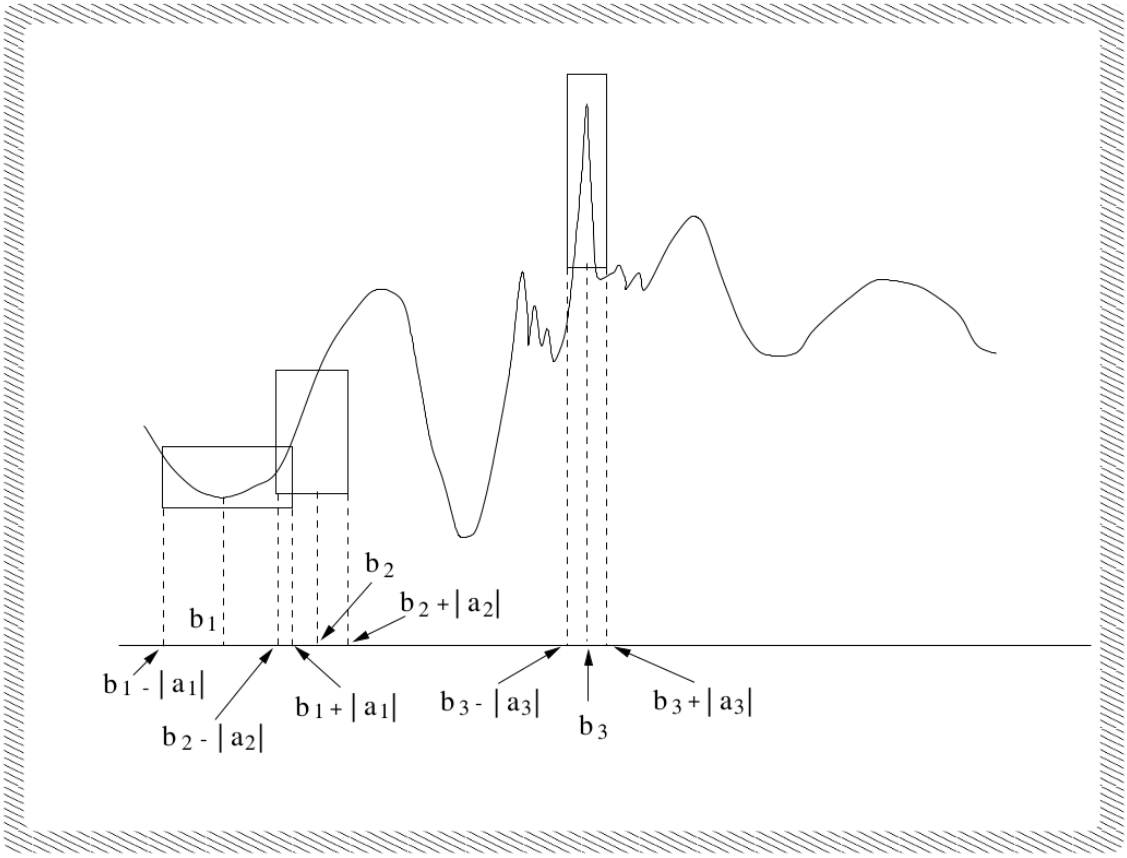


Figure 2.2: Time-frequency location

- On the other hand, by the basic properties of Fourier transform, we have:

$$\begin{aligned}(W_\psi f)(a,b) &= \frac{|a|^{-\frac{1}{2}}}{2\pi}\int_{\mathbb{R}} \hat{f}(\xi)(\psi(\frac{\psi-b}{a}))\hat{\psi}(\xi)\ d\xi \\ &= \frac{|a|^{-\frac{1}{2}}}{2\pi}\int_{\mathbb{R}} \hat{f}(\xi)|a|e^{-ib\xi}\hat{\psi}(a\xi)\ d\xi \\ &= \frac{|a|^{\frac{1}{2}}}{2\pi}\int_{\mathbb{R}} \hat{f}(\xi)e^{-ib\xi}\hat{\psi}(a\xi)\ d\xi\end{aligned}$$

  There is only a phase difference between $e^{-ib\xi}\hat{\psi}(a\xi)$ and $\hat{\psi}(a\xi)$. The bandpass property of ψ ensures that the energy of $\psi(a\xi)$ concentrates on two bands, while the bandwidth depends on the scale parameter a with positive ratio. Hence, the location of $\hat{f}$ is illustrated by $(W_\psi f)(a,b)$, and $\hat{f}$ gets worse localization with a smaller a.

ψ can be viewed as a basic window function. Through this window, we can not observe the integrated f or $\hat{f}$, whereas the local performances of f and $\hat{f}$ are very clear. Moving the translation parameter b, each part of f and $\hat{f}$ can be traveled. Meanwhile, by adjusting a, we can observe f and $\hat{f}$ with different localization. The later is the so-called focus property

of the wavelet. For the convenience of our discussion, under meaning of statistics, we define the center of basic window ψ as

$$x_{\psi}^{*} := \frac{1}{||\psi||_2^2} \int_{\mathbb{R}} t|\psi(t)|^2 dt \tag{2.7}$$

Moreover, we define the radius of the window, which is the statistic width of the energy of ψ to its center, as

$$\Delta_{\psi} := \frac{1}{||\psi||_2} \left[\int_{\mathbb{R}} (t - x^{*})^2 |\psi(t)|^2 dt \right]^{1/2}, \tag{2.8}$$

where $||\psi||_2$ is the norm of ψ, i.e.,

$$||\psi||_2 := \left(\int_{\mathbb{R}} |\psi(x)|^2 dx \right)^{1/2}.$$

The window of ψ (also called the time window of ψ in general) is

$$[x_{\psi}^{*} - \Delta_{\psi}, x_{\psi}^{*} + \Delta_{\psi}]$$

In the same way, the window of $\hat{\psi}$ (also called the frequency window of ψ) is

$$[x_{\hat{\psi}}^{*} - \Delta_{\hat{\psi}}, x_{\hat{\psi}}^{*} + \Delta_{\hat{\psi}}]$$

The following square region

$$[x_{\psi}^{*} - \Delta_{\psi}, x_{\psi}^{*} + \Delta_{\psi}] \times [x_{\hat{\psi}}^{*} - \Delta_{\hat{\psi}}, x_{\hat{\psi}}^{*} + \Delta_{\hat{\psi}}]$$

is referred to the time-frequency window of ψ. For $a, b \in \mathbb{R}, a \neq 0$, a set of wavelets can be generated by the basic wavelet ψ as follows:

$$\psi_{a,b}(t) := |a|^{-\frac{1}{2}} \psi(\frac{t-b}{a}). \tag{2.9}$$

Thus, the time-frequency window of $\psi_{a,b}$ is

$$[b + ax_{\psi}^{*} - |a|\Delta_{\psi}, b + ax_{\psi}^{*} + |a|\Delta_{\psi}] \times \left[\frac{x_{\hat{\psi}}^{*}}{a} - \frac{1}{|a|}\Delta_{\hat{\psi}}, \frac{x_{\hat{\psi}}^{*}}{a} + \frac{1}{|a|}\Delta_{\hat{\psi}} \right] \tag{2.10}$$

The procedure of reasoning is below:

$$\begin{aligned} x^*_{\psi_{a,b}} &= \frac{1}{||\psi||_2^2}\int_{\mathbb{R}} t|a|^{-1}|\psi(\frac{t-b}{a})|^2 dt \\ &= \frac{|a|^{-1}}{||\psi||_2^2}\int_{\mathbb{R}} (ax+b)|\psi(x)|^2|a|dx \\ &= \frac{1}{||\psi||_2^2}\left[a\int_{\mathbb{R}} x|\psi(x)|^2dx + b||\psi||_2^2\right] \\ &= ax^*_\psi + b; \end{aligned}$$

$$\begin{aligned} \Delta_{\psi_{a,b}} &= \frac{1}{||\psi||_2}\left[\int_{\mathbb{R}}(t-ax^*_\psi-b)^2|a|^{-1}|\psi(\frac{t-b}{a})|^2dt\right]^{1/2} \\ &= \frac{1}{||\psi||_2}\left[\int_{\mathbb{R}}(ax-ax^*_\psi)^2|\psi(x)|^2|a|^{-1}|a|dx\right]^{1/2} \\ &= \frac{|a|}{||\psi||_2}\left[\int_{\mathbb{R}}(x-x^*_\psi)^2|\psi(x)|^2dx\right]^{1/2} \\ &= |a|\Delta_\psi; \end{aligned}$$

Also,

$$\hat{\psi}_{a,b}(\xi) = \int_{\mathbb{R}} |a|^{-\frac{1}{2}}\psi(\frac{t-b}{a})e^{-i\xi t}dt = |a|^{\frac{1}{2}}e^{-ib\xi}\hat{\psi}(a\xi);$$

$$||\hat{\psi}_{a,b}||_2^2 = |a|\int_{\mathbb{R}}|\hat{\psi}(a\xi)|^2d\xi = ||\hat{\psi}||_2^2$$

$$\begin{aligned} x^*_{\hat{\psi}_{a,b}} &= \frac{1}{||\hat{\psi}_{a,b}||_2^2}\int_{\mathbb{R}}\xi|a||\hat{\psi}(a\xi)|^2d\xi \\ &= \frac{1}{a}\frac{1}{||\hat{\psi}_{a,b}||_2^2}\int_{\mathbb{R}}\xi|\hat{\psi}(\xi)|^2d\xi \\ &= \frac{1}{a}x^*_{\hat{\psi}} \end{aligned}$$

$$\begin{aligned} \Delta_{\hat{\psi}_{a,b}} &= \frac{1}{||\hat{\psi}||_2}\left[\int_{\mathbb{R}}(\xi-\frac{1}{|a|}x^*_{\hat{\psi}})^2|a||\hat{\psi}(a\xi)|^2d\xi\right]^{1/2} \\ &= \frac{1}{a}\frac{1}{||\hat{\psi}||_2}\left[\int_{\mathbb{R}}(|a|\xi-x^*_{\hat{\psi}})^2|a||\hat{\psi}(a\xi)|^2d\xi\right]^{1/2} \\ &= \frac{1}{a}\frac{1}{||\hat{\psi}||_2}\left[\int_{\mathbb{R}}(\xi-x^*_{\hat{\psi}})^2|\hat{\psi}(\xi)|^2d\xi\right]^{1/2} \\ &= \frac{1}{|a|}\Delta_{\hat{\psi}}. \end{aligned}$$

From (2.10) we can easily find out, when $|a|$ changes to be smaller, the time window of ψ becomes more narrow, whereas the frequency window becomes wider. The are of its time-frequency window is a constant, which is irrelevant to a and b.

$$(2\Delta_{\psi_{a,b}})(2\Delta_{\hat{\psi}_{a,b}}) = 4\Delta_{\psi}\Delta_{\hat{\psi}}.$$

From the above discussion, we find an intrinsic fact: for a basic wavelet ψ, it is impossible to achieve perfect localization in both the time domain and the frequency domain simultaneously. Is there any ψ that could make the area of the time-frequency to be small enough? Unfortunately, the following theorem, the famous Heisenberg uncertainty principle, gave a negative answer to this question.

2.1.2 Theorem. *(Heisenberg uncertainty principle) Let $\psi \in L^2(\mathbb{R})$ satisfy $x\psi(x)$ and $\xi\hat{\psi}(\xi) \in L^2(\mathbb{R})$. Then*

$$\Delta_{\psi}\Delta_{\hat{\psi}} \geq \frac{1}{2}.$$

Furthermore, the equality in the above equation holds if and only if

$$\psi(x) = ce^{iax}g_{\alpha}(x-b),$$

where $c \neq 0, \alpha > 0, a, b \in \mathbb{R}$ and $g_{\alpha}(x)$ is the Gaussian function defined by

$$g_{\alpha}(x) := \frac{1}{2\sqrt{\pi\alpha}}e^{-\frac{x^2}{4\alpha}} \tag{2.11}$$

Proof We assume that the window centers of ψ and $\hat{\psi}$ are 0 without losing generality. Based on the basic knowledge of Fourier analysis, we have

$$\begin{aligned}
(\Delta_{\psi}\Delta_{\hat{\psi}})^2 &= \frac{(\int_{\mathbb{R}} t^2|\psi(t)|^2dt)(\int_{\mathbb{R}} \xi^2|\hat{\psi}(\xi)|^2d\xi)}{||\psi||_2^2||\hat{\psi}||_2^2} = \frac{(\int_{\mathbb{R}} t^2|\psi(t)|^2dt)(\int_{\mathbb{R}} |\hat{\psi}'(\xi)|^2d\xi)}{||\psi||_2^2||\hat{\psi}||_2^2} \\
&= \frac{||x\psi(x)||_2^2||\hat{\psi}'(x)||_2^2}{||\psi||_2^2||\hat{\psi}||_2^2} = \frac{2\pi||x\psi(x)||_2^2||\psi'(x)||_2^2}{2\pi||\psi||_2^2||\psi||_2^2} \\
&\geq \frac{||x\psi(x)\psi'(x)||_2^2}{||\psi||_2^4} \geq \frac{1}{||\psi||_2^4}\left|Re\int_{\mathbb{R}} x\psi(x)\overline{\psi'(x)}dx\right|^2 \\
&= \frac{1}{||\psi||_2^4}\left|\frac{1}{2}\int_{\mathbb{R}} x\frac{d}{dx}|\psi(x)|^2dx\right|^2 = \frac{1}{||\psi||_2^4}\left(\frac{1}{2}\int_{\mathbb{R}} |\psi(x)|^2dx\right)^2 \\
&= \frac{1}{4}.
\end{aligned}$$

Thus, we have

$$\Delta_\psi \Delta_{\hat{\psi}} \geq \frac{1}{2}$$

Further, by the conditions, which preserve the equal-sign in the Holder inequality, we know that the equalities in above reasoning will be tenable, if and only if there is a constant α such that

$$\begin{cases} -Re(x\psi(x)\overline{\psi'(x)} = |x\psi(x)\overline{\psi'(x)}|, \\ |x\psi(x)| = 2\alpha|\psi'(x)|, \\ ||\psi||_2 \neq 0 \end{cases}$$

First, by the second equality, we have

$$x\psi(x) = 2\alpha\psi'(x)e^{i\theta(x)}$$

,

where $\theta(x)$ is a real-valued function. Second, by the first equality, we have

$$-x\psi(x)\overline{\psi'(x)} \geq 0$$

Then,

$$-2\alpha|\psi'(x)|^2 e^{i\theta(x)} \geq 0$$

It infers $e^{i\theta(x)} = -1$. Thus, we have

$$x\psi(x) = -2\alpha\psi'(x)$$

By resolving this ordinary differential equation, we can obtain

$$\psi(x) = ce^{\frac{-x^2}{4\alpha}}$$

Finally, by the third equality, we know that $c \neq 0$. It should be mentioned, at the beginning of our proof it was assumed that the window centers of ψ and $\hat{\psi}$ are 0. Otherwise, consider

$$\tilde{\psi}(x) := e^{-iax}\psi(x+b)$$

,

where $a := x^*_{\hat{\psi}}, b := x^*_{\psi}$. The centers of the time window and the frequency window of $\tilde{\psi}(x)$ are 0. It means that the above reasoning is available to $\tilde{\psi}(x)$. It is easy to deduce that

$$\Delta_{\psi} = \Delta_{\tilde{\psi}},\ \Delta_{\hat{\psi}} = \Delta_{\hat{\tilde{\psi}}}$$

Hence, $\Delta_{\psi}\Delta_{\hat{\psi}} \geq \frac{1}{2}$. The equality holds if and only if

$$\tilde{\psi}(x) = ce^{-\frac{x^2}{4\alpha}}$$

i.e.

$$\psi(x) = ce^{i(x-b)a}e^{-\frac{(x-b)^2}{4\alpha}}$$

,

where $c \neq 0$. Replacing $\frac{ce^{iab}}{2\sqrt{2\pi\alpha}}$ with c we have

$$\psi(x) = ce^{iax}g_{\alpha}(x-b)$$

where $c \neq 0, \alpha > 0, a, b \in \mathbb{R}$ and $g_{\alpha}(x)$ is the Gaussian function defined by (2.11). This establishes the theorem.

■

From this theorem we can extract the following fact: no matter the wavelet ψ we choose, it can not achieve perfect localization in both the time domain and the frequency domain simultaneously. When the time window also narrow, the frequency window must be wide.

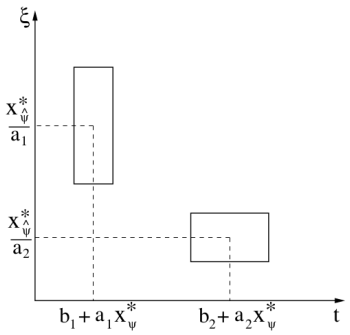


Figure 2.3: Time-frequency location

By the contrary, when the time window is wide, the frequency window must be narrow (Fig. 2.3). The narrow time window and the wide frequency window are good for the high frequency part of the signal, whereas the wide time window and the narrow frequency window are good for its low frequency.

As the same as Fourier transform, wavelet transform is invertible. The inverse wavelet transform can be viewed as the reconstruction of the original signal. First of all, we need to prove the following theorem, which specifies that wavelet transform keeps the law of energy-conservation.

2.1.3 Theorem. *Let ψ be a basic wavelet. Then*

$$\int_{\mathbb{R}}\int_{\mathbb{R}} W_\psi f(a,b)\overline{W_\psi g(a,b)}\frac{dadb}{a^2} = C_\psi\langle f,g\rangle$$

holds for any $f,g \in L^2(\mathbb{R})$ where C_ψ is a constant defined by (2.5), $\langle f,g\rangle$ is the inner product of f and g. Particularly, for $g=f$ we have

$$\int_{\mathbb{R}}\int_{\mathbb{R}} |W_\psi f(a,b)|^2\frac{dadb}{a^2} = C_\psi||f||_2^2$$

Proof Obviously, the second equality is a specific instance of the first equality at $g=f$. Thus, only the first equality needs to be proved. It is easy to know that

$$(\psi(\frac{t-b}{a}))\hat{\psi}(\xi) = |a|e^{-ib\xi}\hat{\psi}(a\xi).$$

Denote that

$$\begin{cases} F(x) := \hat{f}(x)\overline{\hat{\psi}(ax)} \\ G(x) := \hat{g}(x)\overline{\hat{\psi}(ax)} \end{cases}$$

Then,

$$\begin{aligned}
(W_\psi f)(a,b) &= \langle f(x), |a|^{-\frac{1}{2}}\psi(\frac{x-b}{a})\rangle \\
&= \frac{1}{2\pi}\langle \hat{f}(\xi), |a|^{\frac{1}{2}}e^{-b\xi}\hat{\psi}(a\xi)\rangle \\
&= \frac{1}{2\pi}|a|^{\frac{1}{2}}\int_{\mathbb{R}} \hat{f}(\xi)\overline{\hat{\psi}(a\xi)}e^{b\xi}d\xi \\
&= \frac{1}{2\pi}|a|^{\frac{1}{2}}\int_{\mathbb{R}} F(\xi)e^{b\xi}d\xi \\
&= \frac{1}{2\pi}|a|^{\frac{1}{2}}\hat{F}(-b).
\end{aligned}$$

In the same way,

$$(W_\psi g)(a,b) = \frac{1}{2\pi}|a|^{\frac{1}{2}}\hat{G}(-b)$$

Therefore,

$$\int_{\mathbb{R}}(W_\psi f)(a,b)\overline{(W_\psi g)(a,b)}db = (\frac{1}{2\pi})^2|a|\int_{\mathbb{R}}\hat{F}(-b)\overline{\hat{G}(-b)}db$$
$$= (\frac{1}{2\pi})^2|a|\int_{\mathbb{R}}\hat{F}(b)\overline{\hat{G}(b)}db = \frac{1}{2\pi}|a|\int_{\mathbb{R}}F(x)\overline{G(x)}dx$$

Furthermore,

$$\int_{\mathbb{R}}\int_{\mathbb{R}}(W_\psi f)(a,b)\overline{(W_\psi g)(a,b)}\frac{dadb}{a^2} = \int_{\mathbb{R}}\frac{1}{2\pi}|a|\int_{\mathbb{R}}F(x)\overline{G(x)}dx\frac{da}{a^2}$$
$$= \frac{1}{2\pi}\int_{\mathbb{R}}\int_{\mathbb{R}}\frac{|\psi(ax)|^2}{|a|}\hat{f}(x)\overline{\hat{g}(x)}dxda = \frac{1}{2\pi}\int_{\mathbb{R}}\left(\int_{\mathbb{R}}\frac{|\psi(ax)|^2}{|ax|}d(ax)\right)\hat{f}(x)\overline{\hat{g}(x)}dx$$
$$= \frac{1}{2\pi}\left(\int_{\mathbb{R}}\frac{|\psi(\xi)|^2}{|\xi|}d\xi\right)\int_{\mathbb{R}}\hat{f}(x)\overline{\hat{g}(x)}dx = \frac{1}{2\pi}C_\psi\langle\hat{f},\hat{g}\rangle = C_\psi\langle f,g\rangle$$

This completes the proof.

■

According to the above theorem, the inverse wavelet transform can be formally inferred. The formal inverse wavelet transform, by which we can reconstruct the original signal from the wavelet transform $W_\psi f(a,b)$ is:

$$f(x) = C_\psi^{-1}\int_{\mathbb{R}}\int_{\mathbb{R}}W_\psi f(a,b)|a|^{-\frac{1}{2}}\psi(\frac{x-b}{a})\frac{dadb}{a^2} \tag{2.12}$$

We will study the origin of the formula above and the exact inverse formula as follows.

2.1.4 Theorem. *Let ψ be a basic wavelet. Then, for any $f \in L^2(\mathbb{R})$, the inverse wavelet transform holds in the sense of $L^2(\mathbb{R})$-norm, namely:*

$$\left\| f(x) - C_\psi^{-1}\int_{|a|\geq A}da\int_{|b|\leq B}(W_\psi f)(a,b)|a|^{-\frac{1}{2}}\psi(\frac{x-b}{a})\frac{dadb}{a^2}\right\|_2 \longrightarrow 0,$$

as $A \longrightarrow 0, B \longrightarrow \infty$, where C_ψ is a constant defined by (2.5)

Proof

For any $A, B > 0$, we prove the following equality at first:

$$\begin{aligned}&\langle \int\int_{|a|\geq A,|b|\leq B} (W_\psi f)(a,b)|a|^{-\frac{1}{2}}\psi(\frac{x-b}{a})\frac{dadb}{a^2}, g(x)\rangle \\ &= \int\int_{|a|\geq A,|b|\leq B} (W_\psi f)(a,b)\overline{(W_\psi g)(a,b)}\frac{dadb}{a^2}\end{aligned}$$

In fact,

$$\begin{aligned}&\int_{\mathbb{R}}\int\int_{|a|\geq A,|b|\leq B}\left|(W_\psi f)(a,b)|a|^{-\frac{1}{2}}\psi(\frac{x-b}{a})g(x)\right|\frac{dadb}{a^2}dx \\ &\leq \int\int_{|a|\geq A,|b|\leq B}|(W_\psi f)(a,b)||a|^{-\frac{1}{2}}(\int_{\mathbb{R}}|\psi(\frac{x-b}{a})|^2dx)^{\frac{1}{2}}||g||_2\frac{dadb}{a^2}dx \\ &= ||\psi||_2||g||_2\int\int_{|a|\geq A,|b|\leq B}|(W_\psi f)(a,b)|\frac{dadb}{a^2} \\ &\leq ||\psi||_2||g||_2\bigg(\int\int_{|a|\geq A,|b|\leq B}|(W_\psi f)(a,b)|^2\frac{dadb}{a^2}\bigg)^{\frac{1}{2}}\bigg(\int\int_{|a|\geq A,|b|\leq B}\frac{dadb}{a^2}\bigg)^{\frac{1}{2}} \\ &= ||\psi||_2||g||_2C_\psi^{\frac{1}{2}}||f||_2\bigg(4B\int_A^\infty\frac{da}{a^2}\bigg)^{\frac{1}{2}} = ||\psi||_2||g||_2||f||_2\bigg(\frac{4B}{A}C_\psi\bigg)^{\frac{1}{2}} < \infty\end{aligned}$$

where C_ψ is the constant defined by (2.5). Using the Fubini theorem in real-analysis [Rudin, 1974], we have

$$\begin{aligned}&\langle \int\int_{|a|\geq A,|b|\leq B} (W_\psi f)(a,b)|a|^{-\frac{1}{2}}\psi(\frac{x-b}{a})\frac{dadb}{a^2}, g(x)\rangle \\ &= \int_{\mathbb{R}}\int\int_{|a|\geq A,|b|\leq B}(W_\psi f)(a,b)|a|^{-\frac{1}{2}}\psi(\frac{x-b}{a})\overline{g(x)}\frac{dadb}{a^2}dx \\ &= \int\int_{|a|\geq A,|b|\leq B}(W_\psi f)(a,b)|a|^{-\frac{1}{2}}\int_{\mathbb{R}}\psi(\frac{x-b}{a})\overline{g(x)}dx\frac{dadb}{a^2} \\ &= \int\int_{|a|\geq A,|b|\leq B}(W_\psi f)(a,b)\overline{(W_\psi g)(a,b)}\frac{dadb}{a^2}\end{aligned}$$

This is the inequality that we intended to prove. Therefore,

$$
\begin{aligned}
&\left\| f(x) - C_\psi^{-1} \int_{|a|\geq A} da \int_{|b|\leq B} (W_\psi f)(a,b)|a|^{-\frac{1}{2}} \psi(\frac{x-b}{a}) \frac{dadb}{a^2} \right\|_2 \\
&= C_\psi^{-1} \sup_{||g||_2=1} \left| C_\psi \langle f, g\rangle - \langle \int\int_{|a|\geq A, |b|\leq B} (W_\psi f)(a,b)|a|^{-\frac{1}{2}} \psi(\frac{x-b}{a}) \frac{dadb}{a^2}, g(x)\rangle \right| \\
&= C_\psi^{-1} \sup_{||g||_2=1} \left| C_\psi \langle f, g\rangle - \int\int_{|a|\geq A, |b|\leq B} (W_\psi f)(a,b) \overline{(W_\psi g)(a,b)} \frac{dadb}{a^2} \right| \\
&= C_\psi^{-1} \sup_{||g||_2=1} \left| \int\int_{|a|\geq A \text{ or } |b|\leq B} (W_\psi f)(a,b) \overline{(W_\psi g)(a,b)} \frac{dadb}{a^2} \right| \\
&\leq C_\psi^{-1} \sup_{||g||_2=1} \left(\int\int_{\mathbb{R}^2} |(W_\psi g)(a,b)|^2 \frac{dadb}{a^2} \right)^{\frac{1}{2}} \left(\int\int_{|a|\geq A \text{ or } |b|\leq B} |(W_\psi f)(a,b)|^2 \frac{dadb}{a^2} \right)^{\frac{1}{2}} \\
&= C_\psi^{-1} \sup_{||g||_2=1} C_\psi^{\frac{1}{2}} ||g||_2 \left(\int\int_{|a|\geq A \text{ or } |b|\leq B} |(W_\psi f)(a,b)|^2 \frac{dadb}{a^2} \right)^{\frac{1}{2}} \\
&= C_\psi^{-\frac{1}{2}} \left(\int\int_{|a|\geq A \text{ or } |b|\leq B} |(W_\psi f)(a,b)|^2 \frac{dadb}{a^2} \right)^{\frac{1}{2}} \longrightarrow 0, \ (A \to 0, B \to \infty)
\end{aligned}
$$

■

It is important to determine whether formula (2.12) is tenable since the result of the previous theorem is not so clear as this formula. By the result of the theorem, we can infer the following conclusion: as $A \to 0, B \to \infty$,

$$
f_{A,B}(x) := C_\psi^{-1} \int_{|a|\geq A} da \int_{|b|\leq B} (W_\psi f)(a,b)|a|^{-\frac{1}{2}} \psi(\frac{x-b}{a}) \frac{dadb}{a^2}
$$

converges to $f(x)$ in measure ([Rudin, 1974]). According to Riesz theorem ([Rudin, 1974]), there are two sequences $A_k \to 0+$ and $B_k \to +\infty$ such that $f_{A_k,B_k}(x) \to f(x), \ (k \to \infty)$, a.e. $x \in \mathbb{R}$. That is,

$$
f(x) = \lim_{k\to\infty} C_\psi^{-1} \int_{|a|\geq A_k} da \int_{|b|\leq B_k} (W_\psi f)(a,b)|a|^{-\frac{1}{2}} \psi(\frac{x-b}{a}) \frac{dadb}{a^2}
$$

If the right interval of (2.12) exists, (2.12) holds a.e. $x \in \mathbb{R}$. Based on this conclusion, we have

2.1.5 Theorem. *Let ψ be a basic wavelet. Then for any $f \in L^2(\mathbb{R})$, (2.12) holds* a.e. *$x \in \mathbb{R}$ if*

$$
\int_{\mathbb{R}} \int_{\mathbb{R}} \left| (W_\psi f)(a,b)|a|^{-\frac{1}{2}} \psi(\frac{x-b}{a}) \right| \frac{dadb}{a^2} < \infty \tag{2.13}
$$

where C_ψ is a constant, which can be defined by (2.5)

The following theorem gives a sufficient condition such that (2.13) holds.

2.1.6 Theorem. *Let $\psi \in L^1(\mathbb{R})$ be a basic wavelet. For any $f \in L^2(\mathbb{R})$, if there exists a non-negative measurable function $F(a)$, such that*

$$|W_\psi f(a,b)| \le F(a), \text{ and } \int_{\mathbb{R}} \frac{1}{|a|^{3/2}} F(a)da < \infty,$$

then, (2.13) holds. Consequently, the inverse wavelet transform (2.12) holds a.e. $x \in \mathbb{R}$. Furthermore, if $\psi(x)$ is continuous, the right part of (2.12) is a continuous function on $\mathbb{R}$

Proof It is known that

$$\begin{aligned}
&\int_{\mathbb{R}}\int_{\mathbb{R}} \left|(W_\psi f)(a,b)|a|^{-\frac{1}{2}}\psi(\frac{x-b}{a})\right| \frac{dadb}{a^2} \\
&\le \int_{\mathbb{R}}\int_{\mathbb{R}} F(a)\left|\psi(\frac{x-b}{a})\right| \frac{1}{|a|^{5/2}} dadb \\
&= \int_{\mathbb{R}} F(a)\frac{1}{|a|^{5/2}}\left(\int_{\mathbb{R}} \left|\psi(\frac{x-b}{a})\right| db\right) da \\
&= ||\psi||_1 \int_{\mathbb{R}} \frac{1}{|a|^{3/2}} F(a)da < \infty
\end{aligned}$$

Thus, (2.13) holds. If $\psi(x)$ is continuous, we denote

$$w_x(a,b) := (W_\psi f)(a, x-b)|a|^{-\frac{1}{2}}\psi(\frac{b}{a})\frac{1}{a^2}$$

Then

$$|w_x(a,b)| \le F(a)\left|\psi(\frac{b}{a})\right| \frac{1}{|a|^{5/2}} \in L^1(\mathbb{R})$$

$\forall x_n, x \in \mathbb{R},\ x_n \to x\ (n \to \infty)$, by the Lebesgue dominated convergence theorem (see [Rudin, 1974]), we have:

$$\lim_{n\to\infty} \int_{\mathbb{R}}\int_{\mathbb{R}} w_{x_n}(a,b)dadb = \int_{\mathbb{R}}\int_{\mathbb{R}} w_x(a,b)dadb$$

It is worth noting that

$$\lim_{n\to\infty}\int_{\mathbb{R}}\int_{\mathbb{R}}(W_\psi f)(a,b)|a|^{-\frac{1}{2}}\psi(\frac{x_n-b}{a})\frac{dadb}{a^2}$$
$$=\lim_{n\to\infty}\int_{\mathbb{R}}\int_{\mathbb{R}}w_{x_n}(a,b)dadb=\int_{\mathbb{R}}\int_{\mathbb{R}}w_x(a,b)dadb$$

Therefore, the right part of (2.12) which is $\int_{\mathbb{R}}\int_{\mathbb{R}}w_x(a,b)dadb$ is a continuous function on $\mathbb{R}$.

■

Let us appreciate: Let ψ be a basic wavelet, $\alpha>0$ and $f\in L^2(\mathbb{R})$. If there exists constants $C,\delta>0$, such that

$$|W_\psi f(a,b)|\leq C|a|^{\alpha+\frac{1}{2}},\ (\forall b\in\mathbb{R}),$$

for $|a|<\delta$, the conditions of the previous theorem are satisfied. In fact, it will be clear if we let

$$F(a):=\begin{cases}C|a|^{\alpha+\frac{1}{2}},\ |a|<\delta;\\ ||f||_2||\psi||_2,\ |a|\geq\delta;\end{cases}$$

The admissibility condition (2.5) is very important to ensure the existence of the inverse wavelet transform. As a conclusion, it is feasible to apply the wavelet transform to signal analysis, image processing and pattern recognition. It is easy to see that this condition is rather weak and is almost equivalent to $\hat{\psi}(0)=0$, or

$$\int_{\mathbb{R}}\psi(x)dx=0 \tag{2.14}$$

Actually, it is not difficult to proof that if $\psi(x)\in L^2(\mathbb{R})$ and there exists $\alpha>0$, such that $(1+|x|)^\alpha\psi(x)\in L^1(\mathbb{R})$, then (2.5) must be equivalent to (2.14). In general, the wavelets applied in practice have a good damping and satisfy the condition $(1+|x|)^\alpha\psi(x)\in L^1(\mathbb{R}),\ (\alpha>0)$. As a result, in engineering, the wavelet is often defined as the function in $L^2(\mathbb{R})$ which satisfies the condition (2.14). In signal analysis, only the positive frequency is need to be considered, that is $a>0$.

2.1.3. The Continuous Wavelet Transform as a Filter

2.1.7 Definition. *Let ψ be a basic wavelet and we denote*

$$\tilde{\psi}(x) := \overline{\psi(-x)}. \tag{2.15}$$

We define the scale wavelet transform as follows:

$$W_s f(x) := W_s^{\tilde{\psi}} f(x) := (f * \tilde{\psi})(x) \tag{2.16}$$

By the definition of the wavelet transform, for $a > 0$, we have

$$\begin{aligned}(W_\psi f)(a,b) &:= \int_{\mathbb{R}} f(t) a^{-\frac{1}{2}} \overline{\psi(\frac{t-b}{a})} dt = \int_{\mathbb{R}} f(t) a^{-\frac{1}{2}} \tilde{\psi}(\frac{t-b}{a}) dt \\ &= a^{\frac{1}{2}} (f * \tilde{\psi}_a)(b)\end{aligned}$$

where $\tilde{\psi}_a(x) := \frac{1}{a}\tilde{\psi}(\frac{x}{a})$. It specifies that the wavelet transform is actually a convolution, which is also called a filter in engineering. Although Heisenberg uncertainty principle has specified that the area of the time-frequency domain can not be arbitrarily small, the localization in the time-frequency domain still can be perfect, if ψ and its Fourier transform $\tilde{\psi}$ have compact support simultaneously. Unfortunately, from the following theorem, we can find that this condition can not be satisfied.

2.1.8 Theorem. *If $f \in L^2(\mathbb{R})$ is a non-zero function, f and its Fourier transform $\hat{f}$ can not be compactly supported simultaneously.*

Proof If $\hat{f}$ is compactly supported, supp $\hat{f} \subset [-B, B]$, then

$$f(z) := \frac{1}{2\pi} \int_{-B}^{B} \hat{f}(\xi) e^{i\xi z} d\xi$$

is an analytic function on complex plane $\mathbb{C}$. Because of the zero-isolation of non-zero analytic functions, f can not be compactly supported.

■

Now that we have known that the Fourier transform $\hat{\psi}$ of a compactly supported basic wavelet ψ can not be compactly supported, we wish its damping property would be good enough. This is equivalent to that the smoothness of ψ would be good enough. The

following theorem, which is a fundamental fact in the theory of Sobolev spaces and whose detailed proof can be found in [Gilbarg and Trundinger, 1977] ensures this fact.

2.1.9 Theorem. *Let m be a non-negative integer and $H^m(\mathbb{R})$ be the Sobolev space of order m defined by*

$$H^m(\mathbb{R}) := \{f | f, f', ..., f^{(m)} \in L^2(\mathbb{R})\}$$

Then, $f \in H^m(\mathbb{R})$ if and only if

$$\int_{\mathbb{R}} (1 + |x|^m)|\hat{f}(x)|dx < \infty \tag{2.17}$$

Following the definition of $H^m(\mathbb{R})$, $f \in H^m(\mathbb{R})$ indicates that f has certainly differentiableness and smoothness (or regularity). Nevertheless, (2.17) indicates that $\hat{f}$ has damping of order m. Therefore, when we choose a basic wavelet ψ as a filter function, in order to ensure that ψ is good for localized analysis in the time-frequency, we must consider both its damping (or compact support) and its regularity.

Another important property of filters is the linear phase. It is defined as follows.

2.1.10 Definition. *$f \in L^2(\mathbb{R})$ is said to have a linear phase if its Fourier transform satisfies*

$$\hat{f}(\xi) = \epsilon|\hat{f}(\xi)|e^{-ia\xi},$$

where $\epsilon = 1$ or $\epsilon = -1$ and a is a real constant. f is said to have generalized linear phase if there exists a real function $F(\xi)$ and two real constants a and b, such that

$$\hat{f}(\xi) = F(\xi)e^{-1(a\xi+b)}.$$

Obviously, if f has linear phase, it also has generalized linear phase. The generalized linear phase degenerates to the linear phase if and only if $e^{-ib} = \pm 1$ and the real function $F(\xi)$ keeps its sign, i.e., identically positive or identically negative. It can be inferred that the generalized linear phase of ψ is an attribute in Fourier transform domain. This generalized linear phase is equivalent to a symmetry of ψ itself.

2.1.11 Theorem. *$f \in L^2(\mathbb{R})$ has generalized phase if and only if f is skew-symmetric at $a \in \mathbb{R}$, i.e., there exists a constant $b \in \mathbb{R}$ such that*

$$e^{ib}f(a + x) = \overline{e^{ib}f(a - x)},\ x \in \mathbb{R}$$

In particular, for a real function f, it has generalized linear phase if and only if it is symmetric or antisymmetric at $a \in \mathbb{R}$. More precisely,

$$f(a+x) = f(a-x) \;\text{ or }\; f(a+x) = -f(a-x), \;\; x \in \mathbb{R}$$

Proof f has a generalized phase, if and only if two constants a and b exist, such that

$$\hat{f}(\xi)e^{i(a\xi+b)} = F(\xi)$$

is a real function. That is

$$\hat{f}(\xi)e^{i(a\xi+b)} = \overline{\hat{f}(\xi)e^{i(a\xi+b)}}, \;\; (\xi \in \mathbb{R})$$

which is equivalent to

$$\int_{\mathbb{R}} \hat{f}(\xi)e^{i(a\xi+b)}e^{i\xi x}d\xi = \overline{\int_{\mathbb{R}} \hat{f}(\xi)e^{i(a\xi+b)}e^{-i\xi x}d\xi}$$

i.e.,

$$e^{ib}f(a+x) = \overline{e^{ib}f(a-x)}, \;\; (x \in \mathbb{R})$$

If f is a real function, e^{i2b} must be a real function. It means $e^{i2b} = \pm 1$. Thus

$$f(a+x) = f(a-x) \;\text{ or }\; f(a+x) = -f(a-x), \;\; x \in \mathbb{R}$$

It specifies that f is symmetric or antysimmetric on a.

■

Summarily, as a filter, in order to be good for the localized analysis in the time domain, the basic wavelet ψ should have good damping or have compact support. On the other hand, ψ should have good regularity or smoothness, to obtain a good property for the localized analysis in the frequency domain. At last, ψ should be skew-symmetric to avoid the distortion.

2.1.4. Multiresolution Analysis (MRA)

At the beginning of the history of the wavelets, it was difficult to construct a wavelet basis of $L^2(\mathbb{R})$ and also to find a function ψ with good regularity and localization to ensure that $\{\psi_{j,k}(x) := 2^{j/2}\psi(2^j x - k)\}_{j\in\mathbb{Z},k\in\mathbb{Z}}$ is an orthonormal basis of $L^2(\mathbb{R})$. In 1980s such functions were found along with a standard scheme to construct wavelet. This standard scheme called Multiresolution Analysis (MRA) has been proved to construct almost all useful wavelet bases.

The objective is to find a function ψ to ensure that $\{\psi_{j,k}(x) := 2^{j/2}\psi(2^j x-k)\}_{j\in\mathbb{Z},k\in\mathbb{Z}}$ is an orthonormal base of $L^2(\mathbb{R})$. The variable $k \in \mathbb{Z}$ is called the translation factor, whereas the variable $j \in \mathbb{Z}$ is called the dilation factor. The decomposition of $L^2(\mathbb{R})$ in frequency when j is fixed is as follows

$$L^2(\mathbb{R}) = ... \oplus W_{j-1} \oplus W_j \oplus W_{j+1} \oplus ...$$

where $W_j := \overline{\text{span}\{\psi_{j,k}|k \in \mathbb{Z}\}}$ and $\{\psi_{j,k}\}_{k\in\mathbb{Z}}$ is the orthonormal base of W_j. From left to right, the frequency of W_j changes from low to high. We denote that

$$V_j = ... \oplus W_{j-1} \oplus W_j, \quad (j \in \mathbb{Z})$$

where V_j refers to the function set with lower frequency, and satisfies

- $... \subset V_{-1} \subset V_0 \subset V_1 \subset ...$
- $\bigcap_{j\in\mathbb{Z}} V_j = \{0\}, \ \overline{\bigcup_{j\in\mathbb{Z}} V_j} = L^2(\mathbb{R})$
- $f(x) \in V_j \Longleftrightarrow f(2x) \in V_{j+1}$

We can also write $W_j = V_{j+1} - V_j$. Therefore $\{W_j\}_{j\in\mathbb{Z}}$ can also be represented by $\{V_j\}_{k\in\mathbb{Z}}$. The next question that we can name is whether a function ϕ with low frequency such that $\{\phi(\cdot-k)\}_{k\in\mathbb{Z}}$ is an orthonormal basis of V_0 exists or not. The answer is affirmative and therefore we can find ψ such

$$\psi \in V_1 - V_0$$

By $\phi \in V_0 \subset V_1$, we obtain the two-scale equation as follows:

$$\phi(x) = 2\sum_{k\in\mathbb{Z}} h_k \phi(2x - k), \ \text{ where } \ \{h_k\} \in l^2(\mathbb{Z})$$

It is worth noting that $\{\phi_{j,k}(x) := 2^{j/2}\phi(2^j x - k)\}_{k\in\mathbb{Z}}$ is an orthonormal basis of V_j.

This is the framework of MRA used to construct wavelets. Let us definite it mathematically.

2.1.12 Definition. *Let H be a Hilbert space. Then a sequence in H, $\{e_j\}_{j=1}^{\infty}$ is said to be a Riesz basis of H, if the following conditions are satisfied:*

1. $\overline{span}\{e_j\}_{j=1}^{\infty} = H$, *i.e.,* $\forall x \in H$ *and* $\forall \epsilon > 0$, *there exists* $n \in \mathbb{N}$ *and a sequence* $\{c_j\}_{j=1}^{n}$ *such that* $||x - \sum_{j=1}^{n} c_j e_j|| < \epsilon$;
2. *There exists constants A and B, such that*

$$A\sum_{j=1}^{\infty}|c_j|^2 \leq \left\|\sum_{j=1}^{\infty} c_j e_j\right\|^2 \leq B\sum_{j=1}^{\infty}|c_j|^2, \quad \forall\{c_j\}_{j=1}^{\infty} \in l^2.$$

In particular, if $A = B = 1$, $\{e_j\}_{j=1}^{\infty}$ is said to be an orthonormal basis.

Let us now define the main concept in this section.

2.1.13 Definition. *A closed subspace $\{V_j\}_{-\infty}^{\infty}$ of $L^2(\mathbb{R})$ is said to be multiresolution analysis (MRA), if*

1. $... \subset V_{-1} \subset V_0 \subset V_1 \subset ...$
2. $\bigcap_{j\in\mathbb{Z}} V_j = 0$, $\overline{\bigcup_{j\in\mathbb{Z}} V_j} = L^2(\mathbb{R})$
3. $f(x) \in V_j \Longleftrightarrow f(2x) \in V_{j+1}$
4. $\exists\phi \in V_0$ *such that* $\{\phi(\cdot - k)\}_{k\in\mathbb{Z}}$ *is a Riesz basis of space V_0. ϕ is called the scaling function of the MRA.*

V_j is a translation-invariant with $2^{-j}\mathbb{Z}$. Also, $\{\phi_{j,k}\}_{k\in\mathbb{Z}}$ is a Riesz basis of V_j and $\phi_{j,k}(x) := 2^{j/2}\phi(2^j x - k)$.

The key point of MRA is how the scale function ϕ can be constructed. This can be achieved by solving the two-scale equation:

$$\phi(x) = 2\sum_{k\in\mathbb{Z}} h_k \phi(2x - k) \tag{2.18}$$

which is equivalent, applying Fourier transform to both sides to:

$$\text{There exists } \; m_0(\xi) = \sum_{k\in\mathbb{Z}} h_k e^{-ik\xi} \in L^2(\mathbb{T}), \; \text{ such that } \hat{\phi}(\xi) = m_0(\frac{\xi}{2})\hat{\phi}(\frac{\xi}{2}) \tag{2.19}$$

The most important problems in MRA are:

- The existence of the solution for the two-scale equation (2.18) or (2.19).
- Whether the solutions of the two-scale equation can generate a MRA.
- The orthonormality and biorthonormality of the solutions of the two-scale equation.

The orthonormality and biorthonormality can strictly be defined below:

2.1.14 Definition. *Suppose that* $\tilde{\phi}, \phi \in L^2(\mathbb{R})$.

1. *If*
$$\langle \tilde{\phi}(\cdot - k), \phi(\cdot) \rangle = \delta_k \quad (\forall k \in \mathbb{Z}),$$
then $\{\tilde{\phi}, \phi\}$ *is said to be biorthonormal. Furthermore, if* $\{\phi, \phi\}$ *is biorthonormal, we can say* ϕ *is orthonormal.*

2. *If* $\{\tilde{\phi}, \phi\}$ *is biorthonormal, and*
$$\tilde{V}_j := \{\tilde{\phi}_{j,k} | k \in \mathbb{Z}\}, \quad V_j := \{\phi_{j,k} | k \in \mathbb{Z}\}, \quad (j \in \mathbb{Z})$$
forms MRAs in $L^2(\mathbb{R})$ *with scale functions* $\tilde{\phi}$ *and* ϕ *respectively, we say that* $\{\tilde{\phi}, \phi\}$ *generate a pair of biorthonormal MRAs. In particular, if* $\{\phi, \phi\}$ *forms a pair of orthonormal MRAs,* ϕ *is said to generate an orthonormal MRA.*

Although the purpose of this section is not to tackle the process and theory that builds the functions meeting the conditions above, it is worth mentioning the S. Mallat algorithm. This algorithm provides a recursive scheme to calculate the coefficients of the biorthonormal wavelet expansion of a signal from one layer to the next layer which can be applied to image processing or pattern recognition.

2.1.5. Two-Dimensional MRA and Mallat Algorithm

In this section we will extend the principle of the 1-D MRA to 2-D by replacing $L^2(\mathbb{R})$ with $L^2(\mathbb{R}^2)$.

2.1.15 Definition. *Let* $\{V_j\}_j, j \in \mathbb{Z}$ *be a sequence of closed subspaces in* $L^2(\mathbb{R})$. *The 2-D MRA can be constructed by tensor product space* $\{V_j^2\}_{j \in \mathbb{Z}}$ *if and only if* $\{V_j\}_{j \in \mathbb{Z}}$ *is a 1-D MRA in* $L^2(\mathbb{R})$ *and*

$$V_j^2 = V_j \otimes V_j \tag{2.20}$$

The scaling function $\Phi(x, y)$ *for 2-D MRA has the form of*

$$\Phi(x, y) = \varphi(x)\varphi(y),$$

where φ is the real scaling function of the 1-D MRA $\{V_j\}_{j\in\mathbb{Z}}$. For each $j \in \mathbb{Z}$, the orthonormal bases of V_j^2 can be produced by the function system of

$$\{\Phi_{j,k_1,k_2}|(k_1,k_2) \in \mathbb{Z}^2, \Phi_{j,k_1,k_2} = \varphi_{j,k_1}(x)\varphi_{j,k_2}(y)\}$$

Such MRA $\{V_j^2\}_{j\in\mathbb{Z}}$ in space $L^2(\mathbb{R})$ is called divisible MRA.

The wavelet space is defined as $W_j^2 = (V_j^2)^\perp$ i.e. $V_j^2 \oplus W_j^2 = V_{j-1}^2$. The wavelet function consists then of three basic wavelet functions: ψ^1, ψ^2 and ψ^3. The orthonormal bases of the wavelet space W_j^2 can be obtained from ψ^1, ψ^2 and ψ^3 by performing a dilation by 2^j and a dyadic dilation of $k/2^j$.

2.1.16 Theorem. *Let $\{V_j^2\}_{j\in\mathbb{Z}}$ be a divisible MRA in $L^2(\mathbb{R}^2)$: $V_j^2 = V_j \otimes V_j$, where $\{V_j\}_{j\in\mathbb{R}}$ is a 1-D MRA in the space $L^2(\mathbb{R})$ with scaling function φ and wavelet function ψ. We define the following three functions:*

$$\begin{cases} \psi^1(x,y) = \varphi(x)\psi(y) \\ \psi^2(x,y) = \psi(x)\varphi(y) \\ \psi^3(x,y) = \psi(x)\psi(y) \end{cases}$$

For any $j \in \mathbb{Z}$ the orthonormal bases of the space W_j^2 can be obtained from the following function system:

$$\begin{cases} \Psi^1_{j,k,m} = \varphi_{j,k}(x)\psi_{j,m}(y) \\ \Psi^2_{j,k,m} = \psi_{j,k}(x)\varphi_{j,m}(y) \\ \Psi^3_{j,k,m} = \psi_{j,k}(x)\psi_{j,m}(y) \end{cases}$$

Therefore, the function system

$$\{\Phi^e_{j,k,m}|e = 1,2,3;\ j,k,m \in \mathbb{Z}\} \tag{2.21}$$

becomes a set of orthonormal bases of $L^2(\mathbb{R}^2)$

Proof From (2.20) a very significant formula can be produced

$$\begin{aligned} V_{j-1}^2 &= V_{j-1} \otimes V_{j-1} = (V_j \oplus W_j) \otimes (V_j \oplus W_j) \\ &= (V_j \otimes V_j) \oplus (V_j \otimes W_j) \oplus (W_j \otimes V_j) \oplus (W_j \otimes W_j) \\ &= V_j^2 \oplus [(V_j \otimes W_j) \oplus (W_j \otimes V_j) \oplus (W_j \otimes W_j)] \end{aligned} \tag{2.22}$$

where $[(V_j \otimes W_j) \oplus (W_j \otimes V_j) \oplus (W_j \otimes W_j)] = W_j^2$

Since $\{\varphi_{j,k}|k \in \mathbb{Z}\}$ is an orthonormal base for V_j, the set $\{\psi_{j,k}|k \in \mathbb{Z}\}$ becomes the orthonormal bases of the space W_j. Therefore,

- $\{\Phi^1_{j,k,m}|m \in \mathbb{Z}\}$ is an orthonormal base of $V_j \otimes W_j$;
- $\{\Phi^2_{j,k,m}|m \in \mathbb{Z}\}$ is an orthonormal base of $W_j \otimes V_j$;
- $\{\Phi^3_{j,k,m}|m \in \mathbb{Z}\}$ is an orthonormal base of $W_j \otimes W_j$;

We can then affirm that (2.22) indicates that the function represented by (2.21) has constituted the orthonormal bases of the W_j^2.

■

Let P_j, D_j^1, Q_j^2 and Q_j^3 be the projection operators from $L^2(\mathbb{R}^2)$ to its subspaces $(V_j \otimes V_j), (V_j \otimes W_j), (W_j \otimes V_j), (W_j \otimes W_j)$ respectively. Considering that $f(x,y) \in V_{j_1}^2$ is a document image (which has a limited resolution, namely: j_1 is a certain integer). We can write:

$$\begin{aligned} f(x,y) &= P_{j_1} f(x,y) \\ &= \sum_{k_1 \in \mathbb{Z}} \sum_{k_2 \in \mathbb{Z}} c_{j_1,k_1,k_2} \Phi_{j_1,k_1,k_2} \\ &= \sum_{k_1 \in \mathbb{Z}} \sum_{k_2 \in \mathbb{Z}} c_{j_1,k_1,k_2} \varphi_{j_1,k_1}(x) \varphi_{j_1,k_2}(y) \\ &= P_{j_1+1} f + Q^1_{j_1+1} f + Q^2_{j_1+1} f + Q^3_{j_1+1} f \end{aligned} \tag{2.23}$$

where $c_{j_1,k_1,k_2} = \langle P_{j_1+1} f(x,y), \varphi_{j_1,k_1} \varphi_{j_1,k_2}(y) \rangle$ and:

- $P_{j_1} f \in V_{j_1} \otimes V_{j_1}$;
- $P_{j_1+1} f \in V_{j_1+1} \otimes V_{j_1+1}$;
- $Q^1_{j_1+1} f \in V_{j_1+1} \otimes W_{j_1+1}$;
- $Q^2_{j_1+1} f \in W_{j_1+1} \otimes V_{j_1+1}$;
- $Q^3_{j_1+1} f \in W_{j_1+1} \otimes W_{j_1+1}$;

In addition, $P_{j_1+1} f$ and $Q^\beta_{j_1+1}$, $\beta = 1, 2, 3$ can be computed by

$$\begin{aligned} P_{j_1+1} f &= \sum_{m_1 \in \mathbb{Z}} \sum_{m_2 \in \mathbb{Z}} c_{j_1+1,m_1,m_2} \Phi_{j_1+1,m_1,m_2} \\ &= \sum_{m_1 \in \mathbb{Z}} \sum_{m_2 \in \mathbb{Z}} c_{j_1+1,m_1,m_2} \varphi_{j_1+1,m_1}(x) \varphi_{j_1+1,m_2}(y) \end{aligned}$$

and

$$Q^{\beta}_{j_1+1} f = \sum_{m_1 \in \mathbb{Z}} \sum_{m_2 \in \mathbb{Z}} d^{\beta}_{j_1+1,m_1,m_2} \Psi^{\beta}_{j_1+1,m_1,m_2}$$

Therefore, (2.23) can be written as follows:

$$\begin{aligned} f(x,y) &= P_{j_1+1} f + Q^1_{j_1+1} f + Q^2_{j_1+1} f + Q^3_{j_1+1} f \\ &= \sum_{m_1 \in \mathbb{Z}} \sum_{m_2 \in \mathbb{Z}} c_{j_1+1,m_1,m_2} \varphi_{j_1+1,m_1}(x) \varphi_{j_1+1,m_2}(y) \\ &\quad + \sum_{m_1 \in \mathbb{Z}} \sum_{m_2 \in \mathbb{Z}} d^1_{j_1+1,m_1,m_2} \varphi_{j_1+1,m_1}(x) \psi_{j_1+1,m_2}(y) \\ &\quad + \sum_{m_1 \in \mathbb{Z}} \sum_{m_2 \in \mathbb{Z}} d^2_{j_1+1,m_1,m_2} \psi_{j_1+1,m_1}(x) \varphi_{j_1+1,m_2}(y) \\ &\quad + \sum_{m_1 \in \mathbb{Z}} \sum_{m_2 \in \mathbb{Z}} d^3_{j_1+1,m_1,m_2} \psi_{j_1+1,m_1}(x) \psi_{j_1+1,m_2}(y) \end{aligned}$$

An iterative algorithm called *Mallat algorithm* is presented below:

$$\begin{cases} c_{j_1+1,m_1,m_2} = \sum_{k_1 \in \mathbb{Z}} \sum_{k_2 \in \mathbb{Z}} h_{k_1-2m_1} h_{k_2-2m_2} c_{j_1,k_1,k_2} \\ d^1_{j_1+1,m_1,m_2} = \sum_{k_1 \in \mathbb{Z}} \sum_{k_2 \in \mathbb{Z}} h_{k_1-2m_1} g_{k_2-2m_2} c_{j_1,k_1,k_2} \\ d^2_{j_1+1,m_1,m_2} = \sum_{k_1 \in \mathbb{Z}} \sum_{k_2 \in \mathbb{Z}} g_{k_1-2m_1} h_{k_2-2m_2} c_{j_1,k_1,k_2} \\ d^3_{j_1+1,m_1,m_2} = \sum_{k_1 \in \mathbb{Z}} \sum_{k_2 \in \mathbb{Z}} g_{k_1-2m_1} g_{k_2-2m_2} c_{j_1,k_1,k_2} \end{cases} \tag{2.24}$$

Let $H_r = (H_{k_1,m_1}), H_c = (H_{k_2,m_2}), G_r = (G_{k_1,m_1})$ and $G_c = (G_{k_2,m_2})$ be matrices where r indicates an operation on the rows and c indicates an operation o the columns. This way, (2.24) can be compacted in the following form:

$$\begin{cases} C_{j_1+1} = H_r H_c C_{j_1} \\ Q^1_{j_1+1} = H_r G_c C_{j_1} \\ Q^2_{j_1+1} = G_r H_c C_{j_1} \\ Q^3_{j_1+1} = G_r G_c C_{j_1} \end{cases} \tag{2.25}$$

$P_{j_1+1} f$ will be equally decomposed and $P_{j_1+2} f$ and $Q^{\beta}_{j_1+2}$ will be produced. After $j_2 - j_1$ steps we arrive at:

$$f(x,y) = P_{j_2} f(x,y) + \sum_{j=j_1+1}^{j_2} \sum_{\beta=1}^{3} Q^{\beta}_j f(x,y) \tag{2.26}$$

In practice, a document image can be transformed into four sub-images by applying the Mallat algorithm. According to (2.22), these images posses the following properties:

• LL sub-image: both horizontal and vertical directions have low-frequencies. It corresponds to $(V_j \otimes V_j)$, and its othonormal basis is $\{\Phi_{j,k,m}|k,m \in \mathbb{Z}\}$.The coefficients associated with this sub-image are c_j which are the approximation coefficients.

• LH sub-image: the horizontal direction has high-frequencies and the vertical one has low-frequencies. It corresponds to $(V_j \otimes W_j)$, and its orthonormal basis is $\{\Psi^1_{j,k,m}|k,m \in \mathbb{Z}\}$. The coefficients associated with this sub-image are d^1_j which are the horizontal coefficients.

• HL sub-image: the horizontal direction has low-frequencies and the vertical one has high-frequencies. It corresponds to $(W_j \otimes V_j)$, and its orthonormal basis is $\{\Psi^2_{j,k,m}|k,m \in \mathbb{Z}\}$. The coefficients associated with this sub-image are d^2_j which are the vertical coefficients.

• HH sub-image: both horizontal and vertical directions have high-frequencies. It corresponds to $(W_j \otimes W_j)$, and its orthonormal basis is $\{\Psi^3_{j,k,m}|k,m \in \mathbb{Z}\}$. The coefficients associated with this sub-image are d^3_j which are the diagonal coefficients.

These extracted coefficients are the ones that we will consequently use as features when developing our machine learning models.

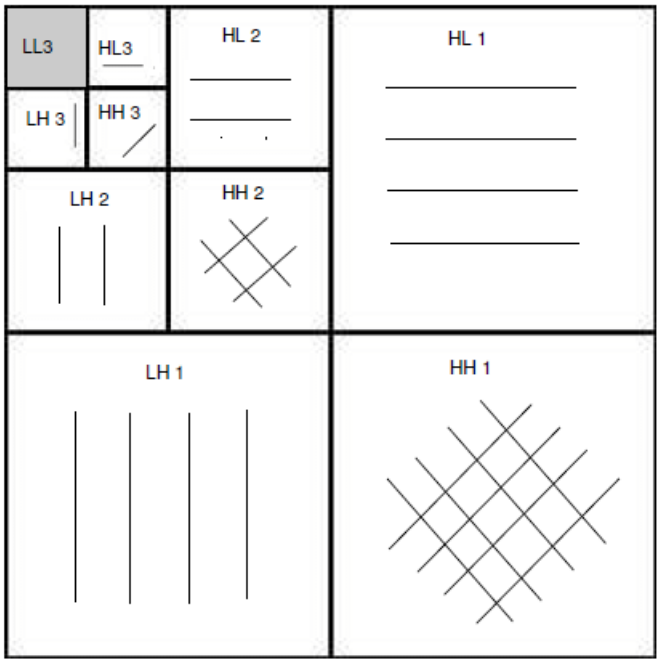


Figure 2.4: 2-D Wavelet Transform of an image

2.2. Wavelet Coefficients and Feature Selection

Once we have obtained a complete and dense set of features from our images, the unavoidable question is: how can we process the big size of the resulting data using a machine with limited resources? To answer this question, we need to introduce the concept of feature selection.

Feature selection is a process that leads to select a subset of features from the set of all available ones taking into account, using different criteria, their global relevance. It provides several advantages:

- It can reduce the computational cost significantly.
- It can improve the classification score by reducing the noise.
- The function types can be identified and monitored by more interpretable features.

In this section, we are going to expose two algorithms that we will use to reduce the amount of coefficients obtained from the Wavelet transform.

2.2.1. Minimum Redundancy Maximum Relevance (mRMR)

Minimum Redundancy Maximum Relevance (mRMR) is a feature selection approach that tends to select features with a high correlation with the class (output) and low correlation between themselves.

Given a selected feature set S and assuming there are m features ($|S| < m$), the feature importance of X_i, $i \in \{1, 2, ..., m\}$ based on the mRMR criterion can be expressed as:

$$f^{mRMR}(X_i) = I(Y, X_i) - \frac{1}{|S|} \sum_{X_s \in S} I(X_s, X_i) \tag{2.27}$$

where X_i denotes a feature currently not selected ($X_i \notin S$) and Y is the response variable. The mutual information between two random variables X and Y can be stated formally as follows:

$$I(X, Y) = \int_{\Omega_Y} \int_{\Omega_X} p(x, y) log\left(\frac{p(x, y)}{p(x)p(y)}\right) dxdy \tag{2.28}$$

where Ω_Y and Ω_X are the sample spaces corresponding to Y and X, $p(x, y)$ is the joint probability density and $p(x)$ is the marginal density function.

The discrete version of the formula above is defined as follows:

$$I(X,Y) = \sum_{y \in \Omega_Y} \sum_{x \in \Omega_X} p(x,y) log\left(\frac{p(x,y)}{p(x)p(y)}\right) \tag{2.29}$$

Also, the maximum relevance criterion of the features with respect to the class variable should be supplemented by the use of a minimum redundancy among features. There is a high likelihood that the dependency between features could be increased in case that only relevance is implemented. This way, minimum redundancy should be implemented without disturbing its relevance. To get minimum redundancy between features the following formula is used:

$$W_I(S) = \frac{1}{|S|^2} \sum_{X_i, X_j \in S} I(X_i, X_j) \tag{2.30}$$

This criterion combining minimum redundancy and the correlation with the output class is what we call mRMR. The simplest form of optimising relevance and redundancy to obtain a good subset of features is:

$$max\{\phi(V_I(S), W_I(S))\} \tag{2.31}$$

where $\phi = (V_I(S) - W_I(S))$ and:

$$V_I(S) = \frac{1}{|S|} \sum_{X_i \in S} I(Y, X_i)$$

Practically speaking, the implementation of the mRMR algorithm is high resource consuming. For the research that concerns us, this method will be applied recursively on different subsets of features instead of the full feature set directly. We will discuss its details in upcoming sections.

2.2.2. Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is another technique to reduce the number of variables in our data. However, unlike mRMR, this algorithm does not preserve the original features but it applies dimensionality reduction. Apart from the necessity of decreasing the number of variables for obvious efficiency and resource reasons, one of the primary problem associated with high-dimensionality in the machine learning field is model overfitting, which reduces the ability to generalize beyond the examples in the training set. Overall, we could say that high-dimensional data is often overcomplete, i.e., many dimensions are redundant and can be explained by a combination of other dimensions.

Developing this idea, when dealing with vectors for statistical purposes, the actual variation of the vector data may occur in only a few dimensions. The objective of PCA is therefore to transform the original random vector into variables called principal components. These components are all orthogonal and are ordered so that the first few explain most of the variation of the random vector. In order to achieve such goal we need to find an orthogonal basis that aligns itself with the data and thus explains a substantial amount of variation in just a few dimensions.

2.2.1 Definition. *An inner product space is a vector space over* $\mathbb{R}$ *or* $\mathbb{C}$ *(from now on* $\mathbb{K}$*) that has an inner product operation satisfying the following properties:*

- $\langle x, y\rangle = \overline{\langle y, x\rangle}$
- $\alpha\langle x, y\rangle = \langle \alpha x, y\rangle$
- $\langle x + z, y\rangle = \langle x, y\rangle + \langle z, y\rangle$
- $\langle x, x\rangle > 0, \ \ \forall x \neq 0$

2.2.2 Definition. *An adjoint of a linear map* $A : V \to V$ *is another linear map* $A^* : V \to V$ *that satisfies:*

$$\langle Ax, y\rangle = \langle x, A^* y\rangle, \ \ \forall x, y \in V \tag{2.32}$$

2.2.3 Proposition. *If* A *is a linear map,* $A : V \to V$*, then the adjoint* A^* *exists.*

Proof Let ϕ be a linear map $\phi : V \to \mathbb{K}$ for some inner product space V. Let $\{e_1, e_2, ..., e_n\}$ be an orthonormal basis in V. Note that

$$\begin{aligned}\phi(u) &= \phi(\langle u, e_1\rangle e_1 + \langle u, e_2\rangle e_2 + ... + \langle u, e_n\rangle e_n) \\ &= \phi(e_1)\langle u, e_1\rangle + \phi(e_2)\langle u, e_2\rangle + ... + \phi(e_n)\langle u, e_n\rangle\end{aligned}$$

$$
\begin{aligned}
&= \langle \phi(e_1)u, e_1\rangle + \langle \phi(e_2)u, e_2\rangle + ... + \langle \phi(e_n)u, e_n\rangle \\
&= \langle u, \overline{\phi(e_1)}e_1\rangle + \langle u, \overline{\phi(e_2)}e_2\rangle + ... + \langle u, \overline{\phi(e_n)}e_n\rangle \\
&= \langle u, \overline{\phi(e_1)}e_1 + \overline{\phi(e_2)}e_2 + ... + \overline{\phi(e_n)}e_n\rangle
\end{aligned}
$$

Let $v \in V$ be given. The inner product $\langle Au, v\rangle$ is a linear map that takes a vector $u \in V$ to an element in $\mathbb{K}$. Let $\phi^*(u) = \langle Au, u\rangle$. From above we can state that

$$\langle Au, v\rangle = \langle u, \overline{\phi^*(e_1)}e_1 + \overline{\phi^*(e_2)}e_2 + ... + \overline{\phi^*(e_n)}e_n\rangle$$

We can then set

$$A^*v = \overline{\phi^*(e_1)}e_1 + \overline{\phi^*(e_2)}e_2 + ... + \overline{\phi^*(e_n)}e_n$$

showing that

$$\langle Au, u\rangle = \langle u, A^*v\rangle$$

Which leads us to affirm that A^* exists.

■

2.2.4 Definition. *The linear map A on a real inner product space is self-adjoint if $A = A^*$*

2.2.5 Proposition. *Suppose that linear map A has a matrix T with respect to an orthonormal basis in a real inner product space V. Then T^* which is the matrix of A^* with respect to the same basis, is the transpose of T.*

Proof Let $\{e_1, e_2, ..., e_n\}$ be the orthonormal basis of V. Because A is a linear map, matrix T is composed of n columns and there are scalars $a_1, a_2, ..., a_n$ such that

$$Ae_i = a_1e_1 + a_2e_2 + ... + a_ne_n$$

Using that the base is orthonormal for $j = 1, 2, ..., n$ we can write:

$$
\begin{aligned}
\langle Ae_i, e_j\rangle &= \langle a_1e_1 + a_2e_2 + ... + a_ne_n, e_j\rangle \\
&= \langle a_1e_1, e_j\rangle + \langle a_2e_2, e_j\rangle + ... + \langle a_ne_n, e_j\rangle \\
&= a_1\langle e_1, e_j\rangle + a_2\langle e_2, e_j\rangle + ... + a_n\langle e_n, e_j\rangle \\
&= a_j
\end{aligned}
$$

Thus,

$$Ae_1 = \langle Ae_i, e_1\rangle e_1 + \langle Ae_i, e_2\rangle e_2 + ... + \langle Ae_i, e_n\rangle e_n$$

Note that in matrix $T_{ji}, j, i = 1, 2, ..., n$ the row j is $\langle Ae_i, e_j \rangle$. Repeating this process for A^* we obtain that T^*_{ji} is likewise:

$$\begin{aligned} \langle A^* e_i, e_j \rangle &= \langle e_i, Ae_j \rangle \\ &= \overline{\langle Ae_j, e_i \rangle} \\ &= \langle Ae_j, e_i \rangle \end{aligned}$$

where we use in the last step that we are in a real inner product space. We can affirm then that $T^*_{ji} = T_{ij}$, concluding that T is the transpose of T^*.

■

From the previous proposition we can fathom that a self-adjoint operator matrix is equal to its transpose. Given a symmetric matrix or self-adjoint operator, we will introduce the Spectral Theorem that will give us a way to find orthonormal eigenvectors. This is what we are seeking for PCA.

2.2.6 Lemma. *If $A : V \to V$ is a self-adjoint linear map, then A has a real eigenvalue.*

Proof If T is the matrix of A (with respect to some basis) we consider the matrix $(T - Iz)$. By the fundamental theorem of algebra we know that the characteristic equation $p(z) = det(T - Iz), z \in \mathbb{C}$ has a complex solution $\lambda \in \mathbb{C}$. Therefore, there is a $v \in V$ such that:

$$\begin{aligned} (T - \lambda I)v &= 0 \\ Tv &= \lambda Iv \\ Tv &= \lambda v \\ Av &= \lambda v \end{aligned}$$

So λ is a eigenvalue. We can also affirm that λ is real:

$$\begin{aligned} &\lambda |v|^2 = \langle Tv, v \rangle = \langle v, Tv \rangle = \overline{\langle Tv, v \rangle} = \overline{\lambda |v|^2} = \overline{\lambda} |v|^2 \\ &\implies \lambda = \overline{\lambda} \end{aligned}$$

■

2.2.7 Theorem. *(The Real Spectral Theorem) Let V be a real inner product space and let $A : V \to V$ be a linear map. Then there is an orthonormal basis in V consisting of eigenvectors of A if and only if A is self-adjoint.*

Proof Let $\{e_1, e_2, ..., e_n\}$ be an orthonormal eigenbasis in V. Then, we can write with respect to that basis,

$$\begin{aligned} Ae_1 &= \lambda_1 e_1 \\ Ae_2 &= \lambda_2 e_2 \\ &... \\ Ae_n &= \lambda_n e_n \end{aligned}$$

Clearly A has a diagonal matrix, which is equal to its transpose and therefore A is self-adjoint:

$$\begin{pmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_n \end{pmatrix}$$

The first implication is proved. For the second implication, we need to introduce first a special subspace U. If A is self-adjoint, we know from (2.2.6) that the matrix of A has a real eigenvalue. Let u be the corresponding eigenvector (we can scale u so that the norm is 1). Let U be the set of all scalar multiples of u. U is a subspace of V with $dim(V) = 1$. Let $U^{\perp} = \{v \in V | \langle u, v \rangle = 0\}$. Given $v_0 \in U^{\perp}$, note that:

$$\langle u, Av_0 \rangle = \langle Au, v_0 \rangle = \langle \lambda u, v_0 \rangle = \lambda \langle u, v_0 \rangle = 0$$

This way, $v_0 \in U^{\perp}$ implies $A_{v_0} \in U^{\perp}$. Let S be the linear map on V defined by $S = A|_{U^{\perp}}$. Let $x, y \in U^{\perp}$:

$$\langle Sx, y \rangle = \langle Ax, y \rangle = \langle x, Ay \rangle = \langle x, Sy \rangle$$

Which proves that S is self-adjoint. Now we are going to develop the following induction: for dimension n greater than 1 the theorem holds for vector spaces of dimension less than n. For $dim = 1$, the theorem holds because all transformations by linear maps on a real one-dimensional inner product space are simply scalar multiplications by real numbers. Now let us suppose that the theorem holds for the n-dimensional subspace $U^{\perp}$. Let $\{u_1, u_2, ..., u_n\}$ be the orthonormal eigenbasis for $U^{\perp}$. Then $\{u, u_1, u_2, ..., u_n\}$ is an

orthonormal eigenbasis for V which is $n+1$ dimensional.

■

2.2.8 Corollary. *Let T be the self-adjoint matrix of a linear map of a real inner product space V. Then T can be decomposed into $T = E\Lambda E^T = \lambda_1 e_1 e_1^T + \lambda_2 e_2 e_2^T + ... + \lambda_n e_n e_n^T$ where E is a diagonalizing column with orthonormal columns.*

Proof We know there is an orthonormal eigenbasis in V since the linear map is self-adjoint. Let this basis be $\{e_1, e_2, ..., e_n\}$. Let E be the matrix with the basis vectors on the columns,

$$E = \begin{pmatrix} e_1 & e_2 & \dots & e_n \end{pmatrix}$$

and let

$$\Lambda = \begin{pmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_n \end{pmatrix}$$

Since $Te_i = \lambda_i e_i$ it is clear that $TE = E\Lambda$. Therefore, $T = E\Lambda E^{-1}$. Since $\langle e_i, e_i \rangle = 1$ and $\langle e_i, e_j \rangle = 0$ for $i \neq j$, we have $EE^T = I$ and thus $E^{-1} = E^T$. We have then a diagonalizing matrix E with orthonormal columns such that $T = E\Lambda E^T$.

$$\begin{aligned} E\Lambda E^T &= \begin{pmatrix} e_1 & e_2 & \dots & e_n \end{pmatrix} \begin{pmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_n \end{pmatrix} \begin{pmatrix} e_1^T \\ e_2^T \\ \vdots \\ e_n^T \end{pmatrix} \\ &= \begin{pmatrix} e_1 & e_2 & \dots & e_n \end{pmatrix} \begin{pmatrix} \lambda_1 e_1^T \\ \lambda_2 e_2^T \\ \vdots \\ \lambda_n e_n^T \end{pmatrix} \\ &= \lambda_1 e_1 e_1^T + \lambda_2 e_2 e_2^T + ... + \lambda_n e_n e_n^T \end{aligned} \tag{2.33}$$

■

Our goal is to find vectors in a real inner product space that are statistically uncorrelated i.e. they are orthogonal. Thanks to the Real Spectral Theorem we can use the

covariance matrix which is symmetric and therefore is the matrix of a self-adjoint linear map. This theorem guarantees that we will find an orthogonal basis of eigenvectors. These eigenvectors will have corresponding eigenvalues of great significance as we will see.

Let us first remember some basic concepts about random variables:

2.2.9 Definition. *Let $E(X)$ be the expected value of a random variable X. Let $cov[X,Y]$ be the covariance of two random variables X and Y, defined by:*

$$cov[X,Y] = E(XY) - E(X)E(Y)$$

2.2.10 Definition. *Let $var[X]$ the variance of a random variable X where*

$$var[X] = cov[X,X]$$

2.2.11 Definition. *With regards to a basis, $X = (X_1, X_2, ..., X_n)$ is a vector in an n-dimensional real inner product space (equipped with a dot product) determined by n random scalars.*

2.2.12 Definition. *For $k = 1, 2, ..., n$, $Z_k = (Z_{k_1}, Z_{k_2}, ..., Z_{k_n})$ is the vector satisfying the following properties:*

- $Z_k X = Z_{k_1} X_1 + Z_{k_2} X_2 + ... + Z_{k_n} X_n$
- *$var[Z_k X]$ is maximized under the constraint that $\langle Z_k, Z_k \rangle = 1$*
- *Z_{k+1} is calculated after Z_k and is uncorrelated to Z_k*

2.2.13 Definition. *Let $X = (X_1, X_2, ..., X_n)$. The covariance matrix Σ of X is the matrix with entries $\Sigma_{ij} = cov[X_i, X_j]$*

2.2.14 Proposition. *Let Σ be the covariance matrix for the elements of X. Then $var[Z_k X] = Z_k \Sigma Z_k^T$.*

Proof Note that $cov[Z_k X, X]$ is the row matrix M where $M_{1j} = cov[Z_k X, X_j]$. This is:

$$M = \Big(cov[(Z_{k_1} X_1 + Z_{k_2} X_2 + ... + Z_{k_n} X_n), X_1] \quad \dots \quad cov[(Z_{k_1} X_1 + Z_{k_2} X_2 + ... + Z_{k_n} X_n), X_n] \Big)$$

Now, using that Z_k is constant:

$$M = \Big(\sum_{i=1}^{n} Z_{k_i} cov[X_i, X_1] \quad \dots \quad \sum_{i=1}^{n} Z_{k_i} cov[X_i, X_n] \Big)$$

$$= \begin{pmatrix} Z_{k_1} & \dots & Z_{k_n} \end{pmatrix} \begin{pmatrix} cov[X_1, X_1] & \dots & cov[X_1, X_n] \\ \vdots & \ddots & \vdots \\ cov[X_1, X_n] & \dots & cov[X_n, X_n] \end{pmatrix}$$

$$= Z_k\Sigma$$

Following a similar argument, $cov[X, Z_kX] = \Sigma Z_k^T$, concluding that $var[Z_kX] = cov[Z_kX, Z_kX] = Z_k\Sigma Z_k^T$.

■

What this proposition is saying in principle is that in order to get good properties for (2.3.1) we need to maximize $Z_k\Sigma Z_k^T$. However, until now Σ has distinct eigenvalues. Having eigenvalues of multiplicities greater than one is theoretically possible, which can lead to have principal components not uniquely defined. In (Jolliffe, 27), such occurrences are said to be uncommon and the assumption usually holds.

2.2.15 Theorem. *(Principal Components) Let Σ be the covariance matrix for X. If Σ has distinct eigenvalues, then for $k = 1, 2, ..., n$, Z_k^T is an eigenvector corresponding to the kth largest eigenvalue of Σ.*

Proof Z_1 is derived first regarding (2.3.1). The maximization of $var[Z_kX] = Z_k\Sigma Z_k^T$ is set up as a Lagrange maximization problem. Let β be the Lagrange multiplier:

$$L = Z_1\Sigma Z_1^T - \beta(Z_kZ_k^T - 1)$$

Differentiating both sides with respect to Z_1 yields:

$$\Sigma Z_1^T - \beta Z_1^T = 0 \Longrightarrow \Sigma Z_1^T = \beta Z_1^T$$

This way β is an eigenvalue of Σ. The fact that β is the largest eigenvalue follows from the observation that the variance that is being maximized is $Z_1\Sigma Z_1^T = Z_1\beta Z_1^T = \beta Z_1 Z_1^T = \beta$ so β is not only maximized as an eigenvalue but also equal to the largest variance.

Regarding Z_2, to satisfy the third property in (2.3.1) we need to maximize $Z_2\Sigma Z_2^T$ with the restriction that Z_2X is uncorrelated with Z_1X i.e. $cov[Z_1X, Z_2X] = 0$. We can note that:

$$cov[Z_1X, Z_2X] = Z_1\Sigma Z_2^T = Z_2\Sigma Z_1^T = Z_1\beta Z_2^T = \beta Z_1 Z_2^T = \beta Z_2 Z_1^T = 0$$

Knowing that $Z_2Z_1^T = 0$ we can set up the Lagrange maximization problem to be the following, where γ and δ are the Lagrange multipliers:

$$L = Z_2\Sigma Z_2^T - \delta(Z_kZ_k^T - 1) - \gamma(Z_2Z_1^T)$$

If we differentiate both sides with respect to Z_2 yields:

$$\begin{aligned} \Sigma Z_2^T - \delta Z_2^T - \gamma Z_1^T &= 0 \\ Z_1 \Sigma Z_2^T - \delta Z_1 Z_2^T - \gamma Z_1 Z_1^T &= 0 \\ 0 - 0 - \gamma &= 0 \\ \gamma &= 0 \end{aligned}$$

Therefore, $\Sigma Z_2^T = \delta Z_2^T$ and δ is an eigenvalue with Z_2^T its corresponding eigenvector. It is also maximized, but not greater than or equal to β as β was maximized first and Σ has distinct eigenvalues. Repeating this process for $Z_k X$ and $k \geq 3$, we obtain eigenvectors Z_k^T corresponding with the kth largest eigenvalue.

■

The previous theorem shows that the random vector X can be transformed into numerous ordered principal components and that the eigenvalue corresponding to a given eigenvector is actually the variance of $Z_k X$. Now we can introduce the main definition of this section,

2.2.16 Definition. *Principal components are the variables $Z_k X$ where Z_k are the transposes of the eigenvectors Z_k^T from theorem (2.2.15)*

2.2.17 Corollary. *Let Ω be the matrix that has Z_k, the kth eigenvector of Σ, as the kth column. Then $\Sigma = \lambda_1 Z_1^T Z_1 + \lambda_2 Z_2^T Z_2 + ... + \lambda_n Z_n^T Z_n$.*

Proof Let Λ be the diagonal matrix with eigenvalues from the maximization process in (2.2.15). Because Σ has eigenvalues and eigenvectors:

$$\Sigma\Omega = \Omega\Lambda \Longrightarrow \Sigma = \Omega\Lambda\Omega^T$$

Finally, $\Sigma = \lambda_1 Z_1^T Z_1 + \lambda_2 Z_2^T Z_2 + ... + \lambda_n Z_n^T Z_n$ by (2.33).

■

This corollary expands the information of theorem (2.2.15) and decomposes the covariance matrix into parts $\lambda_i Z_i^T Z_i,\ i = 1, ..., n$ that gives more detail into how the variation is spread among individual principal components. The reduction of dimension via PCA can not be fully intelligible when there is no clear cutoff for a principal component. In these cases dimensions are reduced until a set percentage of variation is accounted for (e.g. 80 %).

2.2.18 Theorem. *Let T be an symmetric matrix of order n that has distinct eigenvalues. Then T has exactly n eigenvalues.*

Proof The Spectral Theorem guarantees at least n eigenvectors, which gives n eigenvalues. For any eigenvector $v \in V$, we know that:

$$Tv = \lambda v$$
$$(\lambda I - T)v = 0$$

$(\lambda I - T)$ is a singular matrix that has a determinant of zero since $v \neq 0$. Thanks to lemma (2.2.6) we can affirm that the determinant is a polynomial function of λ, which vanishes at at most n points. T has therefore n eigenvalues.

■

The importance of the theorem above is that by simply finding eigenvalues and corresponding eigenvectors of Σ and ranking the eigenvalues by size, we are indeed performing PCA. As shown in theorem ((2.2.15) the eigenvalues of Σ are already the largest variances of some Z_k, so no maximization procedure needs to be performed.

2.3. Model Building and Classification

We have now our data extracted and selected. The next step is to build a Supervised Machine Learning Model that allows us to classify our images into six different Alzheimer's stages. Supervised learning is where you have input variables X and an output variable Y and you use an algorithm to learn the mapping function f from the input to the output.

$$Y = f(X)$$

The goal is to approximate the mapping function so well that when you have new input data X you can predict the output variables Y for that data. More specifically, our algorithms are included inside the group of classification problems as the output variable is a category and not a real value.

For this section, two different approaches will be covered: SVM (Support Vector Machine) and CNN (Convolutional Neural Network) which has shown excellent performance in many computer vision and machine learning problems. As we shall see below, the advantage of CNN is that the network learns to optimize the filters (or kernels) through automated learning, whereas in traditional algorithms like SVM these filters are hand-engineered. This means that for CNN, the pre-processing steps conducted in the previous sections can be omitted.

In practice, we will use CNN in two different ways: we will build a custom model from scratch and we will remake the input and output layer in a pre-trained model (Transfer Learning). The objective is to give a general understanding of two of the main algorithms used in the past decade for image processing. The mathematical theory behind these algorithms and specially behind CNN is vast so the following subsections will not try to explain and proof the results of the theoretical part corresponding to every algorithm but to make it possible for a reader to understand how a CNN runs at a mathematical level.

2.3.1. Support Vector Machine (SVM)

Support Vector Machine (SVM) is based on the idea of decision hyperplanes that determines boundaries in input space or high dimensional feature space. The objective is to construct linear functions from a set of labeled training data to split the positive samples from the negative samples. Intuitively, the farther distance between the linear separator and the closest negative and positive samples the better. When dealing with nonlinear problems, the input is normally mapped into a high dimensional feature space. A linear classifier is then constructed on top of this high dimensional space.

Linear Separable SVMs Classifier

Let us first consider a binary classification problem with $N \in \mathbb{N}$ training samples. Each sample can be expressed as a tuple (x_i, y_i) where $i = 1, 2, ..., N$, $x_i = (x_{i1}, x_{i2}, ..., x_{in})$ and $y_i \in \{-1, 1\}$. $n \in \mathbb{N}$ corresponds in principle with the number of features. The decision boundary of a linear classifier can be written as follows:

$$w^T x + b = 0 \tag{2.34}$$

where w is the weight vector and b is a bias term.The distance between the decision boundary and the closest data point determines the margin of the classifier. Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we maximize the margin of the classifier. If the training data are linearly separable then there exists a pair (w, b) such that:

$$\begin{aligned} w^T x_i + b \geq 1 \ \text{ if } \ y_i = 1, \\ w^T x_i + b \leq 1 \ \text{ if } \ y_i = -1, \end{aligned} \tag{2.35}$$

The linear classifier is defined as:

$$f(x) = sign(w^T x + b) \tag{2.36}$$

Therefore, the functional margin of x_i with respect to an hyperplane (w, b) is defined as follows.

$$\gamma_i = y_i(w^T x_i + b) \tag{2.37}$$

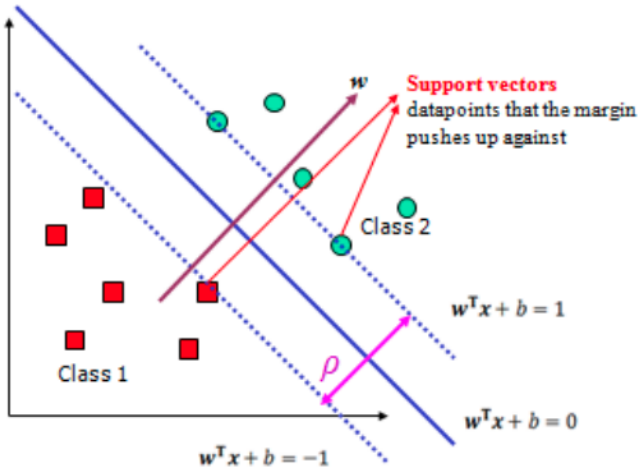


Figure 2.5: Decision boundary and margin of SVM

The objective is to maximize the geometric margin ρ in order to find w and b as shown in Fig. (2.5), which is a convex quadratic optimization problem with quadratic function subject to linear constraints.

$$\rho = \frac{2}{||w||} \quad \text{is maximized} \tag{2.38}$$

$\forall(x_i, y_i), \quad i = 1, ..., N$ such that $y_i(w^T x_i + b) \geq 1$. The above problem can be easily reformulated as a minimization using the following function:

$$\Phi(w) = ||w|| = w^T w \quad \text{is minimized} \tag{2.39}$$

$\forall(x_i, y_i), \quad i = 1, ..., N$ such that $y_i(w^T x_i + b) \geq 1$ This minimization can be solved using a dual problem where a Lagrange multiplier α_i is linked with every inequality constraint $(y_i(w^T x_i + b) \geq 1)$ in the primal problem. This is, we need to find $\alpha_1, \alpha_2, ..., \alpha_N$ such that:

$$Q(\alpha_1, \alpha_2, ..., \alpha_N) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j x_i^T x_j, \tag{2.40}$$

is maximized, with constraints:

$$\sum_{i=1}^{N} y_i \alpha_i = 0, \quad \alpha_i \geq 0$$

The solution of the dual problem α_i must satisfy the condition $\alpha_i[y_i(w^T x_i - b) - 1] =$

$0, \;\; i = 1, 2, ..., N$. The solution to the primal is:

$$w = \sum_{i=1}^{N} \alpha_i y_i x_i, \quad b = y_\beta - \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i y_i x_i^T x_j, \quad \forall \alpha_i > 0, \; \beta \in 1, 2, ..., N \tag{2.41}$$

Most of α_i are zero for samples which are not support vectors. Each non-zero α_i specifies that the equivalent x_i is a support vector. The classification function can therefore be written as shown below:

$$f(x) = \sum_{i=1}^{N} \alpha_i y_i x_i^T x + b \tag{2.42}$$

Linear None Separable SVMs Classifier

When the data is not linearly separable the standard approach is to allow the fat decision boundary to make a few mistakes, paying a cost for each misclassified sample. To implemet this, slack variables ξ_i are introduced. The SVM problem can be rewritten as follows:

Find w, b and $\xi_i \geq 0$ such that

$$\Phi(w) = w^T w + C \sum_{i=1}^{N} \xi_i \quad \text{is minimized} \tag{2.43}$$

$\forall (x_i, y_i), \;\; i = 1, ..., N$ such that $y_i(w^T x_i + b) \geq 1 - \xi_i$ where C is a regularization term used to control overfitting. C determines how important ξ should be. A smaller C emphasizes the importance of ξ and a larger C diminishes its importance. Small values of C will result in a wider margin, at the cost of some misclassifications; large values of C will give you the Hard Margin classifier and tolerates zero constraint violation.

The dual problem is the same as the previous example. We need to find $\alpha_1, \alpha_2, ..., \alpha_n$ such that

$$Q(\alpha_1, \alpha_2, ..., \alpha_N) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j x_i^T x_j \quad \text{is maximized} \tag{2.44}$$

with the constraints:

$$\sum_{i=1}^{N} \alpha_i y_i = 0, \;\; C \geq \alpha_i \geq 0, \;\; \forall \alpha_i, \; i = 1, ..., N \tag{2.45}$$

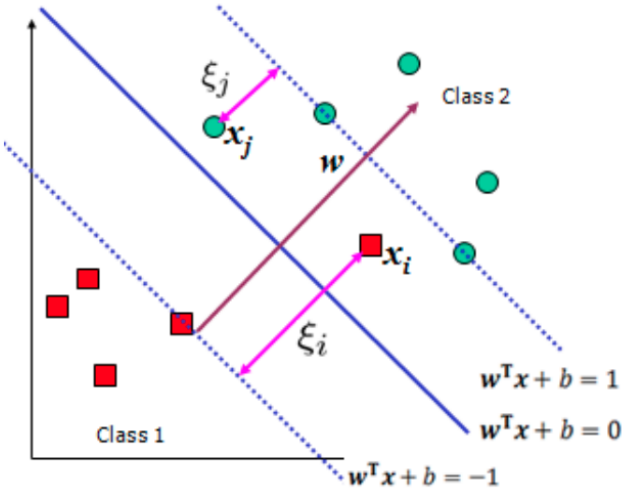


Figure 2.6: Slack variables for linearly non-separable data

The x_i with non-zero α_i are again the support vectors. The solution of the dual problem is of the form:

$$w = \sum_{i=1}^{N} \alpha_i y_i x_i, \quad b = y_j(1 - \xi_j) - \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i y_i x_i^T x_j, \quad \alpha_i > 0 \; \forall i = 1, ..., N \tag{2.46}$$

The solution must satisfy the condition $\alpha_i[y_i(w^T x_i - b) - 1 + \xi_i] = 0$ for $i = 1, ..., N$. As in the previous case, we do not need to compute w explicitly for classification function as shown below:

$$f(x) = \sum_{i=1}^{N} \alpha_i y_i x_i^T x + b \tag{2.47}$$

Nonlinear SVMs Classifier

When the training data are not linearly separable, the original input space is mapped into a high dimensional space which is known as a feature space. SVM has an easy way of performing this mapping called the kernel kick. The SVMs linear classifier depends on a dot product between data point vectors. Assuming that we transform the data to some space H via a mapping function ϕ such that the data appear in the form $\phi(x_i)^T \phi(x_j)$. Linear operation in H is alike to non-linear operation in input space. For example, if $x = (x_1, x_2)$ we can set $\phi(x) = (x_1^2, x_2^2, \sqrt{2}x_1 x_2)$ as shown in Fig. (2.7).

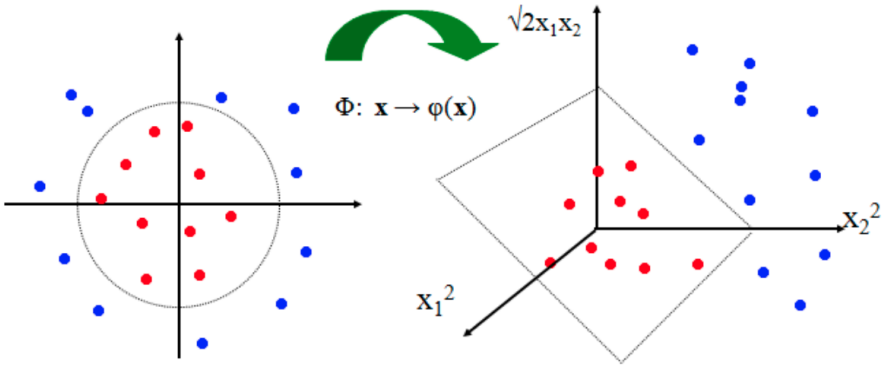


Figure 2.7: Nonlinear SVMs decision boundary

Let $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ as kernel function, then designing SVM is to find $\alpha_1, ..., \alpha_N$ such that

$$Q(\alpha_1, ..., \alpha_N) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \quad \text{is maximized} \tag{2.48}$$

with the constrains:

$$\sum_{i=1}^{N} \alpha_i y_i = 0, \quad C \geq \alpha_i \geq 0, \quad \forall \alpha_i \tag{2.49}$$

Then the classifier is:

$$f(x) = \sum_{i=1}^{N} \alpha_i y_i K(x_i, x_j) + b \tag{2.50}$$

The most used families of kernels are:

- Linear kernel: $K(x_i, x_j) = x_i^T x_j$
- Polynomial kernel with degree d: $K(x_i, x_j) = (x_i^T x_j + 1)^d$
- Radial basis function with kernel $\sigma > 0$: $K(x_i, x_j) = exp(-||x_i - x_j||^2 / 2\sigma^2)$
- Sigmoid with kernel $\delta > 0$: $K(x_i, x_j) = tanh(\delta x_i^T x_j + r)$

For our experiment, the kernel that provided the best results is radial basis function (RBF) as we will see in following sections.

Multiclass SVMs

When the output can be more than one class and it must be divided into M mutually exclusive classes the issue becomes more complex. There are many ways to solve multiclass classification problems for SVM. The most common strategy which is the one that we will use in the conducting experiments is One-Against-One (OAO).

This approach constructs $M \times (M-1)/2$ binary classifiers using all the binary pair-wise combinations of the M classes. This way, each classifier is qualified by using the examples of the first class as positive and the examples of the second class as negative examples. The method used to find the result class is to select the class voted by the majority of the classifiers.

Another common approach is to construct M binary SVM classifiers where the ith SVM is trained such that the examples in the ith class are labeled as positive samples and all the rest as negative samples. To obtain a prediction, a test sample is obtained from all M SVMs and is labeled according to the maximum output. This approach is called One-Against-All (OAA).

2.3.2. Convolutional Neuronal Networks (CNN)

A CNN is one special type of multilayer feedforward artificial neural networks, which is a deep learning technique that that has been widely used in developing applications for computer vision, natural language processing, data mining, and computer games. CNN are biologically inspired variants of multilayer perceptron neural network (MLP) that has been designed to emulate the multilayered visual cortex of human vision system.

The Architecture

A CNN usually takes an order 3 tensor as its input, e.g., an image with H rows, W columns, and 3 channels (R, G and B color channels). Higher order tensor inputs, however, can be handled by CNN in a similar fashion. The input then sequentially goes through a series of processing. One processing step is usually called a layer, which could be of many different types as we will see in later subsections. An abstract description of the CNN stricture is shown as follows:

$$\boldsymbol{x}^1 \longrightarrow \boxed{\boldsymbol{w}^1} \longrightarrow \boldsymbol{x}^2 \longrightarrow \cdots \longrightarrow \boldsymbol{x}^{L-1} \longrightarrow \boxed{\boldsymbol{w}^{L-1}} \longrightarrow \boldsymbol{x}^L \longrightarrow \boxed{\boldsymbol{w}^L} \longrightarrow z$$

Figure 2.8: Abstract CNN structure

The input data represented as x^1 in Fig. (2.8) is usually an image (order 3 tensor). It goes through the processing in the first layer, whose parameters are denoted as tensor w^1 in the first box. The output of layer w^n is x^{n+1} which acts as the input of the next processing layer w^{n+1} being $n = 1, ..., L-1$. One additional layer is added for backward error propagation, a method that learns good parameter values in CNN. The last layer z is a loss layer: supposing that t is the corresponding target vector value for the input x^1, then a cost function can be used to measure the discrepancy between the CNN prediction x^L and the target t. A simple cost function could be:

$$z = \frac{1}{2}||t - x^L||^2 \tag{2.51}$$

Supposing that the problem at hand is an image classification problem with C classes, a commonly used strategy is to output x^L as a C dimensional vector where the ith entry encodes the probability of x^1 comes from the ith class. A softmax transformation can be used in $(L-1)$th layer to make x^L a probability mass function. In a classification problem (our case) the cross entropy loss is often used.

2.3.1 Definition. *The cross-entropy loss function is defined as follows:*

$$CE(t, x^L) = -\sum_n t_n log(x_n^L), \;\; n = 1, ..., C \tag{2.52}$$

where t is the ground-truth C dimensional vector and x^L is the score for each class.

The loss layer is only useful when training the CNN parameters using a set of training examples. When all the parameters of a CNN model $w^1, ..., w^{L-1}$ have been learnt, the CNN model only runs forward for predictions.

2.3.2 Definition. *Given an arbitrary set X, a totally ordered set Y and a function $f : X \to Y$, the argmax over some subset S of X is defined by:*

$$\arg\max_{x \in S} := \{x \in S : f(s) \leq f(x), \; \forall s \in S\} \tag{2.53}$$

Therefore, if $p_i, \; i = 1, ..., n$ is the ith class we can output the CNN class prediction p associated with x^L as:

$$p = p_j \text{ where } j \text{ is such that } x_j^L = \arg\max_{x_i^L}, \; i = 1, ..., C \tag{2.54}$$

Let us now have a look at how we learn the model parameters. As in many other learning systems, the parameters of a CNN model are optimized to minimize the loss z so it matches the ground-truth labels. The method used to modify the parameters is called stochastic gradient descent (SDG). Given a loss function z, $z : \mathbb{R}^C \to \mathbb{R}$ the SDG method modifies the tensor w^i as follows:

$$(w^i)_{t+1} = (w^i)_t - \eta \frac{\partial z}{\partial (w^i)_t} \tag{2.55}$$

where t is a time index and η is the step size with $\eta > 0$. If we denote $g = \frac{\partial z}{\partial (w^i)^t}$, then in order to minimize the loss function we should update w^i along the opposite direction of the gradient as shown in Fig. (2.9).

From a single training sample x^1 we can make the loss smaller but only for this particular sample. This could make the loss of some other training samples become larger. Therefore, we need to repeat the process for every training example that we have so the parameters are built based on all the training data. When all training examples have been used to update the parameters, we say one epoch has been processed. We can then perform SDG again on the resulting epoch and repeat until the average loss increases.

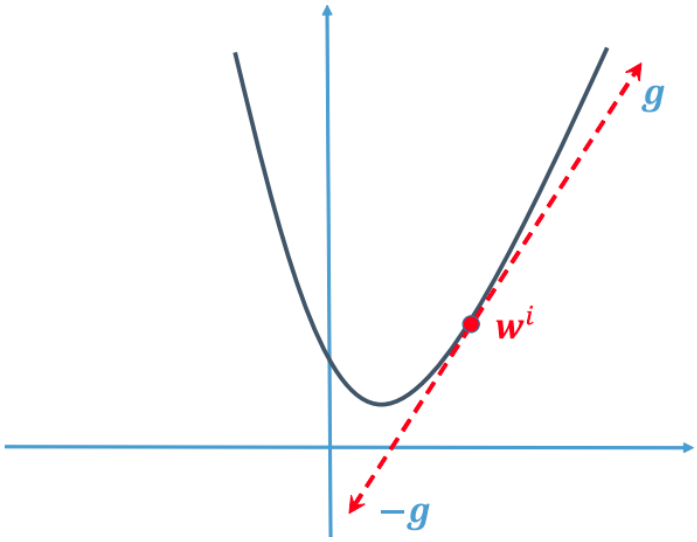


Figure 2.9: Gradient descent method

In practice, the calculation of w^i in the step $t+1$ is performed using error back propagation. This method involves the computation of two set of gradients for every step, from the last layer to the first one (backwards):

- $\frac{\partial z}{\partial (w^i)}$ as shown in equation (2.55).
- $\frac{\partial z}{\partial (x^i)}$ which is the gradient of the layer's input x^i.

The reason why these two gradients are calculated together is that having computed the terms $\frac{\partial z}{\partial (w^{i+1})}$ and $\frac{\partial z}{\partial (x^{i+1})}$ we can compute $\frac{\partial z}{\partial (w^i)}$ and $\frac{\partial z}{\partial (x^i)}$ using the chain rule:

- $\frac{\partial z}{\partial (w^i)^T_{t+1}} = \frac{\partial z}{\partial (x^{i+1})^T_{t+1}} \frac{\partial (x^{i+1})_{t+1}}{\partial (w^i)^T_t}$
- $\frac{\partial z}{\partial (x^i)^T_{t+1}} = \frac{\partial z}{\partial (x^{i+1})^T_{t+1}} \frac{\partial (x^{i+1})_{t+1}}{\partial (x^i)^T_t}$.

This operation requires only a matrix reshaping and an additional transpose operation, which is much easier than computing $\frac{\partial z}{\partial (w^i)^T_{t+1}}$ and $\frac{\partial z}{\partial (x^i)^T_{t+1}}$ directly. Of course, we first need to compute $\frac{\partial z}{\partial (w^L)^T_{t+1}}$ and $\frac{\partial z}{\partial (x^L)^T_{t+1}}$. Before jumping into the different types of layers, let us have a look at the notation used.

2.3.3 Definition. *Assuming $\alpha_1, \alpha_2, ..., \alpha_n \in \mathbb{N}$, the tensor $y \in \mathbb{R}^{\alpha_1, \alpha_2, ..., \alpha_n}$ is a vector of n directions where its elements $y_{i_1, i_2, ..., i_n} \in \mathbb{R}$ are indexed by $i_k \in \{1, 2, ..., \alpha_k\}$ having $1 \leq k \leq n$.*

In the ith layer, the input is $x^i \in \mathbb{R}^{H_i \times W_i \times D_i}$ where $H_i, W_i, D_i \in \mathbb{N}$. Clearly, for $i = 1$ H_1 and W_i are the dimensions of the image passed as input and $D_i = 3$. The triplet (h^i, w^i, d^i) refers then to one element in x^i.

The ReLU Layer

The Rectified Linear Unit (ReLU) layer does not change the size of the input, this is, if w^i is a ReLU layer x^i and x^{i+1} share the same size. This layer performs a truncation individually for every element in the input:

$$x^{i+1}_{h,w,d} = max\{0, x^i_{h,w,d}\} \tag{2.56}$$

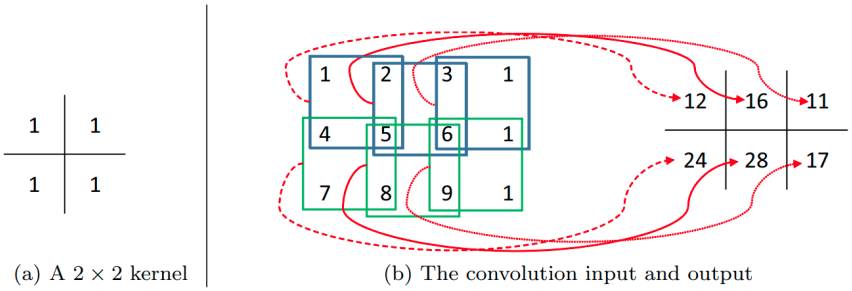
where $0 \leq h < H^i = H^{i+1}$, $0 \leq w < W^i = W^{i+1}$ and $0 \leq d < D^i = D^{i+1}$. The parameters in this layer do not need to be learnt since there is no parameter inside a ReLU layer. Note that we can express the gradient as follows:

$$\left[\frac{\partial z}{\partial x^i}\right]_{h,w,d} = \begin{cases} \left[\frac{\partial z}{\partial x^{i+1}}\right]_{h,w,d} & \text{if } x^i_{h,w,d} > 0 \\ 0 \text{ otherwise} \end{cases} \tag{2.57}$$

Although the function $max(0, x)$ is not differentiable at $x = 0$ it is not an issuein practice. The purpose of ReLU is to increase the nonlinearity of the CNN. Intuitively, this property is useful for recognizing complex patterns and objects. For example, $x^i_{h,w,d}$ may be positive if a region inside the input image has a certain pattern and $x^i_{h,w,d}$ is negative or 0 when the region does not exhibit these patterns.

The Convolution Layer

Supposing an input image whose size is 3×4 and the convolution kernel. A convolution kernel of size 2×2 will compute the product between the numbers at the same location for each 2×2 submatrix as shown in Fig. (2.10).



(a) A 2×2 kernel (b) The convolution input and output

Figure 2.10: Example of convolution operation

For order 3 tensors, the convolution operation is performed similarly. If the input of the convolutional ith layer is an order 3 tensor with size $H^i \times W^i \times D^i$, a convolutional kernel is also a 3 tensor with size $H_c \times W_c \times D^i$, $H_c \leq H^i$, $W_c \leq W^i$. When we overlap the kernel

on top of the input tensor, we compute the products of the corresponding elements in all the D^i channels and sum them to get the convolution result at the specific location.

In a convolution layer, multiple convolutions are usually used. If we use D kernels and each kernel is of spatial span $H \times W$ by denoting f as the set of our kernels we have $f \in \mathbb{R}^{H \times W \times D^i \times D}$. Using a mathematical reasoning, the convolutional procedure can be expressed as an equation:

$$x^{i+1}_{h^{i+1},w^{i+1},d} = \sum_{h=0}^{H} \sum_{w=0}^{W} \sum_{d^i=0}^{D^i} f_{h,w,d^i,d} \times x^i_{h^{i+1}+h,w^{i+1}+w,d^i} \tag{2.58}$$

The convolution can be used to amplify vertical, horizontal or other angle edges. These can be emphasized if a ReLU layer is set next. As all the spatial locations share the same convolutional kernel, the number of parameters that need to be saved for a convolutional layer is reduced.

Fully Connected Layer

A fully connected layer refers to a layer where the computation of any element in the ouput x^{i+1} requires all of the elements in the input x^i unlike regular convolutional layers where one element is usually computed using only a small number of elements in its input. A fully connected layer is sometimes useful at the end of a CNN model when the output of the current layer contains distributed representations for the input image and we want to use all these features together to build features with stronger capabilities in the next layer.

Pooling Layer

Let $x^i \in \mathbb{R}^{H^i \times W^i \times D^i}$ be the input of the ith layer. Considering that the ith layer is a pooling layer, and let $H, W \in \mathbb{N}$ such that H divides H^i and W divides W^i, the output of pooling will be an order 3 tensor of size $H^{i+1} \times W^{i+1} \times D^{i+1}$ with:

$$H^{i+1} = \frac{H^i}{H}, \quad W^{i+1} = \frac{W^i}{W}, \quad D^{i+1} = D^i \tag{2.59}$$

The pooling layer operates upon each channel independently. Within each channel the matrix with $H^i \times W^i$ elements are divided into $H^{i+1} \times W^{i+1}$ nonoverlaping subregions where each subregion is $H \times W$ in size. The pooling operation requires no parameters since the spatial extent of the pooling $(H \times W)$ is specified in the design of the CNN structure.

There are two widely used types of operations that maps a subregion into a specific

value. These are max pooling and average pooling. As the names suggest, we can mathematically write them as:

$$\begin{aligned} \text{max}: \quad & x^{i+1}_{h^{i+1},w^{j+1},d} = \max_{0\leq h<H, 0\leq w<W} x^{i}_{h^{i+1}\times H+h, w^{i+1}\times W+w,d} \\ \text{average}: \quad & x^{i+1}_{h^{i+1},w^{j+1},d} = \frac{1}{HW} \sum_{0\leq h<H, 0\leq w<W} x^{i}_{h^{i+1}\times H+h, w^{i+1}\times W+w,d} \end{aligned} \tag{2.60}$$

Pooling layers are normally used to reduce the the size of the input into tensors that contain only the most relevant information. This step is also useful to decrease the chance of overfitting.

Chapter 3

Research

Once we have studied and explored the foundation behind the mechanisms that we need to use, it is time now to address our objective: to build a machine learning model to classify patients in six different stages of Alzheimer's disease. As in every ML model, the process to develop a final and effective solution comprises several steps. This process is not linear, this is, usually more than one approaches are tested and consequently the pipeline is modified over the time.

This section will cover the most relevant experiments carried out during the research. One of key points is to decide which algorithm performs better: SVM or CNN. Since an additional experiment will be done using transfer learning with one of the most common and popular pre-trained models VGG16 we are going to confront the power of a super trained model against the wavelet transform. Another key point is to learn the best strategy to use in terms of data handling. As an example of this, we will address the question of whether it is better to process a single slice with a higher level of detail, or to process simultaneously more than one slice with a lower level of it. In addition, we need to determine which are the classes that we will care the most. It is clear that the sooner the disease is detected the better the prognosis will be for the patient so it will be also relevant for us to test the model's performance detecting Alzheimer's early stages. To make it more didactic, the different approaches in every algorithm will be exposed from lowest to highest depending on the goodness of their results.

The data source used can be found in `http://adni.loni.usc.edu`. The full project is available on `https://github.com/juliopradom/alzheimer-classifier`.

3.1. Problem Exploration

As mentioned before, the set of images provided are classified in six groups:

- AD (Alzheimer Disease): the images in this group correspond to patients diagnosed with Alzheimer.
- CN (Cognitively Normal): corresponds to healthy individuals (control).
- MCI (Mild Cognitive Impairment): causes a slight but noticeable and measurable decline in cognitive abilities.
- EMCI (Early Mild Cognitive Impairment): an early stage of MCI with milder episodic memory impairment.
- LMCI (Late Mild Cognitive Impairment): a more advanced stage of MCI previous to AD
- SMC (Significant Memory Concern): patients with SMC are characterized by self-report significant memory concern, quantified by using the Cognitive Change Index and the Clinical Dementia Rating (CDR) of zero. SMC participants score within the normal range for cognition, and the informant does not equate the expressed concern with progressive memory impairment.

SMCs have been shown to be correlated with a higher likelihood of progression, thereby minimizing the stratification of risk among normal controls and addressing the gap between healthy elderly controls and MCI. With these classes, we can then build a progression diagram in Alzheimer's disease as shown in Fig. (3.1).

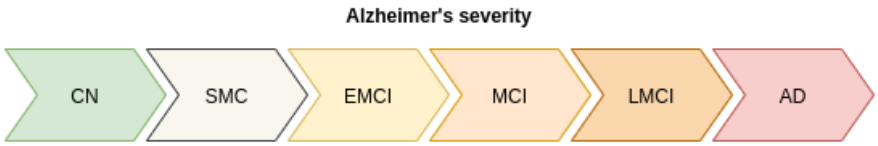


Figure 3.1: Stratification of classes according to their severity

Unlike other experiments and investigations, by achieving the goals set for this classifier we will not only discern between healthy patients and patients with dementia but also provide a more accurate diagnosis about the stage of this dementia. This could be useful to detect the disease earlier and prescribe a more effective and personalized treatment to those individuals located in intermediate stages.

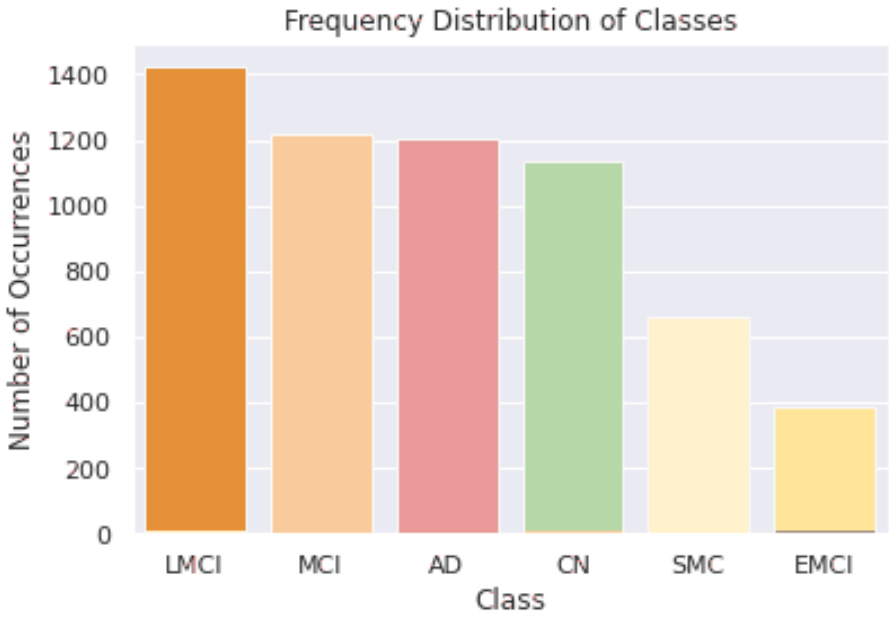


Figure 3.2: Frequency distribution of classes

Let us take a look at the distribution of the different classes within our data. By looking at Fig. (3.2) we can see the number of samples per category. LMCI, MCI, AD and CN have approximately the same number of samples (~ 1200) whereas SMC and EMCI have a significantly less amount of them. We will pay particular attention to these minority classes given that, as commented at the beginning of this chapter, we need to specially focus on Alzheimer's early stages.

Each of the samples is composed of nine different images corresponding to nine different slices from a MRI image. These slices have the numbers 55, 56, 61, 72, 82, 90, 104, 106 and 114. They all have three different colour channels (RGB) and they are already resized to 656×875. This means, every image is an array with shape $(656, 875, 3)$. In Fig.(3.3) we can see the nine images for our patient 1 in AD.

In practice, the management of all this information is very resource-consuming. Particularly, being limited by a 16 GB RAM will only allow us to access a partial amount of all the available data. This will be a limiting factor in our investigation. Smart approaches must be taken to overcome these difficulties and they will be covered also in the following sections.

Having analysed and described our data, we are going to open two lines of research. First, we are going to proceed to extract and select features to apply SVM. Secondly, we will use custom CNN and CNN with transfer learning to train our model without pre-processing.

I encourage the reader to understand the contents described in chapter 2 in order to make the most of the following sections.

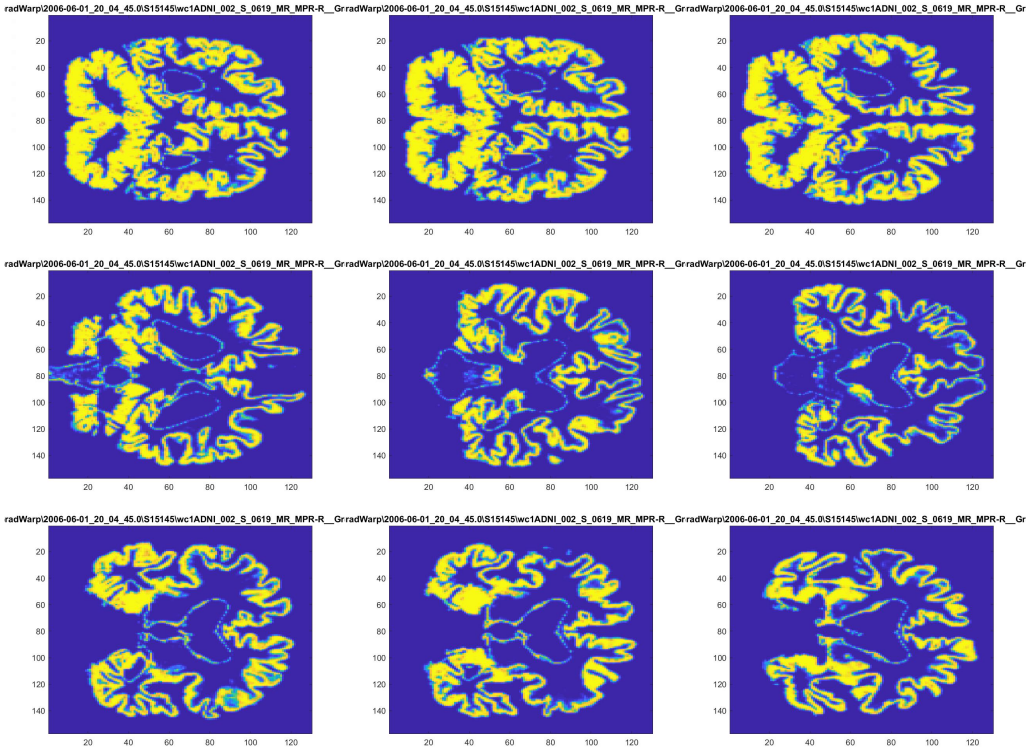


Figure 3.3: Example of images for a given individual

3.2. Objective

Before tackling our objective let us better agree on what our real objective is. For our particular case, we will focus on precision and recall. First, some basic observations about notation:

- TP (True Positives): number of samples predicted as class x when they actually belong to class x.
- TN (True Negatives): number of samples not predicted as class x when they actually don't belong to class x.
- FP (False Positives): number of samples predicted as class x when they actually don't belong to class x.
- FN (False Negatives): number of samples not predicted as class x when they actually belong to class x.

The reason why we are using the words "positive" and "negative" is that these terms originally come from binary problems where only two possible outputs are considered. Here, we are using these terms to point multi-classes. Given these four terms, let us define precision and recall:

$$Precision = \frac{TP}{TP + FP} \tag{3.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{3.2}$$

That is, precision is the fraction of events where we correctly declared x out of all instances where the algorithm declared x. Conversely, recall is the fraction of events where we correctly declared x out of all of the cases where the true of state of the world is x. Roughly, precision stipulates how good the algorithm is when it affirms that a sample corresponds to class x whereas recall stipulates the percentage of real samples of x that we are not missing. Since we want both measurements to be considered, we will introduce another definition.

$$F1 = \frac{2 * (Recall * Precision)}{(Recall + Precision)} \tag{3.3}$$

F1 is the weighted average of precision and recall. This score will be calculated using the average F1 of every class. The global F1 is therefore the measurement that we will use to compare the performance of the different algorithms. However, let us not forget that another relevant indicator that we will use in practice when facing similar scores is the specific performance (F1) of the algorithms identifying SMC and EMCI.

3.3. SVM Research

The first approach to our model will be to build a Support Vector Machine classifier (see section 2.3.1) using wavelet coefficients (see section 2.1) extracted from our images. By using wavelet coefficients the identification of relevant anomalies and slight changes within our samples will be easily detected, which is actually what we are seeking to achieve to distinguish between one class or another.

The coefficients extracted for a given patient can be used to construct a vector in a high-dimensional space. The set of these vectors labelled with their corresponding classes will help us to determine an hyperplane able to separate every group from the other. In practice, one hyperplane will be created for every class as shown in section 2.3.1.

3.3.1. Feature Extraction

To make working with our samples easier let us first create a handler class *Patient* (see section 5.1). This class will contain the logic needed to save image's data into different formats and extract the wavelet coefficients. The class allows to access the coefficients of the approximation image at every level (See Fig. 2.4). However, for our experiment we will only access the coefficients of the approximation image at level three and four for memory reason.

This task to extract and save the coefficients is performed by *coeff_getter.py* script (see section 5.2) which uses *Patient* for such purpose. It is mandatory to first convert the images from RGB space to grey so the wavelet module used can extract the coefficients using a single channel. In particular, we will perform ten extractions: one for every slice available and one combining all the slices consecutively. Extracting one slice means obtaining the wavelet coefficients at the third level for every patient at this specific slice, which gives a total of 6028 arrays of 10120 coefficients each. Extracting all the slices means obtaining the wavelet coefficients at the fourth level for all the available slices and join them as one, which gives again a total of 6028 arrays of $9 \times 2867 = 25803$ coefficients each. In Table 3.1 we can see some of the coefficients that we could extract per image.

level5	level4	level3	level2	level1
918	2867	10120	37856	145971

Tabla 3.1: wavelet coefficients per image for some approximation level

Nevertheless, as commented before going beyond the third level becomes computationally impossible for the machine used in our research as the size of the arrays handled exceeds the capacity of our memory.

3.3.2. Feature Selection

After the coefficients are successfully extracted let us remember that we have arrays of 10120 and 25803 coefficients respectively. In order to generate an input small and meaningful enough to train and work with our models we need to apply feature selection. For such purpose, we will perform two different experiments using the two methods exposed in section 2.2: mRMR and PCA.

The code and logic used to perform this task can be found in *coeff_selector.py* script (see section 5.3). For the first experiment we make use of mRMR to select the best 100 coefficients. This algorithm is extremely resource-consuming and it is not possible to pass the whole set of arrays at once. To overcome this issue a recursive version of this algorithm was designed and built. The functioning of this recursive version is shown if Fig. 3.4.

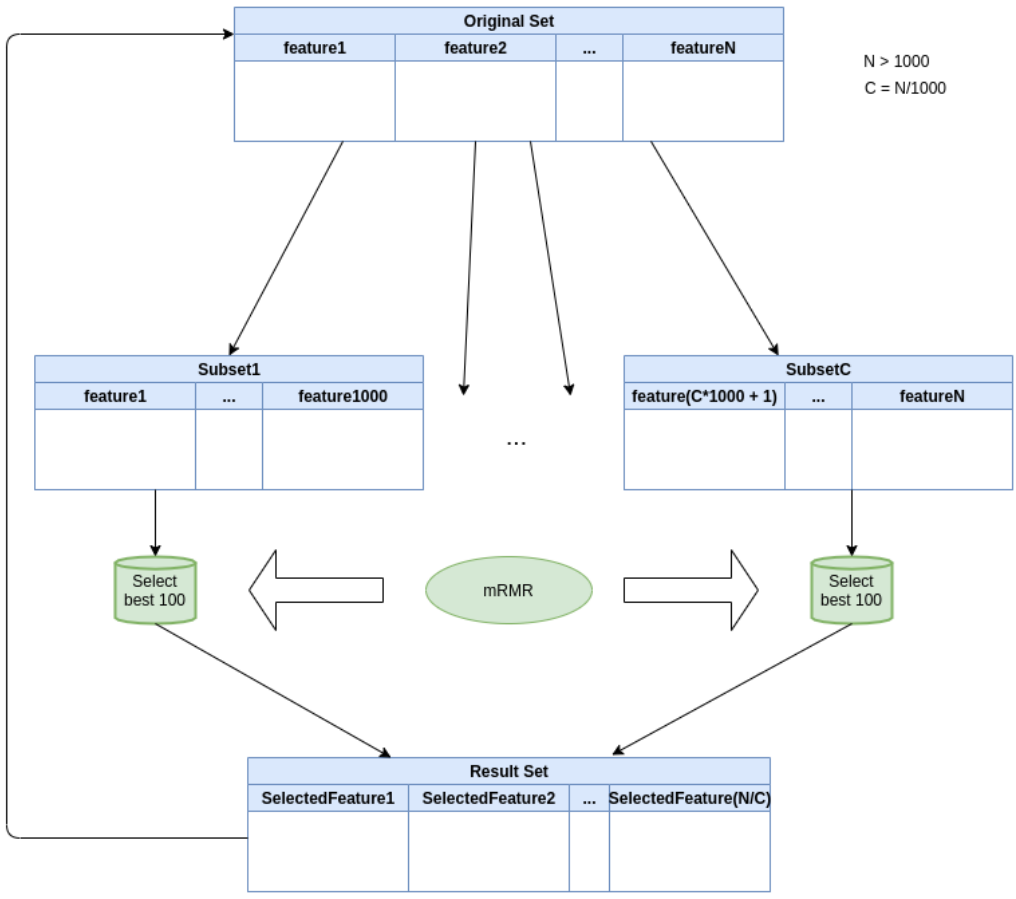


Figure 3.4: Recursive version of mRMR algorithm

In short, we apply mRMR to a maximum set of 1000 features. The original set is divided into multiples subsets of 1000 features from which the most relevant columns

are kept and joined together to construct a new input set. It is worth noting that the application of mRMR will give a set of existent features as a result. Before starting the process, the data is standardized and rescaled so the mean of observed values is 0 and the standard deviation is 1. This can be useful to avoid different scales between our images and to take a step further to ensure normalization.

The same process is repeated but using PCA this time. Contrary to mRMR, by using PCA we obtain 100 new features as a result of dimensionality reduction, this is, we keep the information of all the original features in a smaller set. No particular consideration needs to be taken to perform PCA as the module used is powerful enough to handle an input of our size.

3.3.3. SVM Construction

Now that we have our coefficients extracted and selected we need to find the best hyperparameters to tune our SVM model (see section 2.3.1). The code behind the SVM manipulation and construction is hold in class *SVM* (see section 5.4). After applying grid search using slice 55 on the set of hypermarameters shown in table 3.2 considering that C is the regularization parameter and *gamma* is the kernel coefficient, the most promising hyperparameters using the output of mRMR are kernel = rbf, $C = 1000$ and $gamma = 10^{-2}$. Likewise, using the output of PCA we obtain the parameters kernel = rbf, $C = 10$ and $gamma =$ scale.

kernel	C	gamma
linear, rbf, sigmoid	1, 10, 100, 1000	10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, scale

Tabla 3.2: Hyperparameter space considered for SVM

For all the experiments conducted below the set of patients is split into training and test set. These sets will remain unchanged so we can contrast the results more effectively. More specifically, the training set covers 75 % (4521) of the available samples and test set covers the rest 25 % (1507) where the number of representatives for each class follows the original distribution in both sets. In addition, the number 104729 is used as random seed for shuffling.

Let us now jump into the results for mRMR, where our main objective keeps being maximizing F1 score. By looking at table 3.3 we can see that we obtained very decent results. Classes in the middle are perfectly recognized and predicted whereas control and sick patients present a noticeable worse score. The experiment with the best results is the one that used all the available coefficients at level four. This result is interesting since it

shows that processing all the information as a whole is in principle more effective than using single slices for classification, even if the level of detail accessed is lower.

F1	CN	SMC	EMCI	MCI	LMCI	AD	AVG
55	0.8090	1	1	1	1	0.8148	**0.9373**
56	0.8007	1	1	1	1	0.8133	**0.9361**
61	0.8083	1	1	1	1	0.8122	**0.9367**
72	0.7958	1	1	1	1	0.8040	**0.9333**
82	0.8091	1	1	1	1	0.8180	**0.9379**
90	0.7917	1	1	1	1	0.7980	**0.9316**
104	0.7825	1	1	1	1	0.7933	**0.9293**
106	0.7813	1	1	1	1	0.7706	**0.9253**
114	0.8035	1	1	1	1	0.8101	**0.9356**
All	0.8909	1	1	1	1	0.9032	**0.9657**

Tabla 3.3: F1 mRMR results

In Fig. 3.5 we can see more clearly the different scores between slices. Obviously, after this experiment we have reached a F1 score of 0.9657. Regarding the models built using single slices we can see that slice 82 slightly stands out from the rest. Below we will discuss if this trend is confirmed.

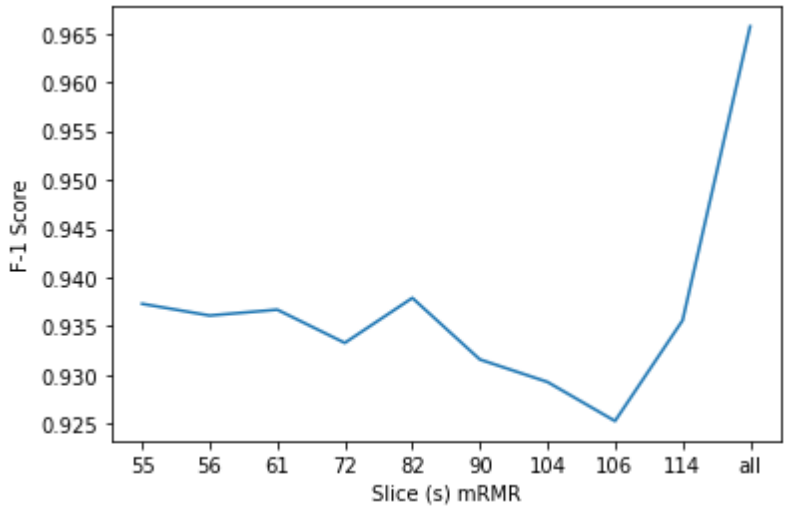


Figure 3.5: F1 mRMR scores

The same experiment was reproduced using now PCA as shown in table 3.4. It is undeniable how good the classifier performed using PCA for every slice. This time the slices with the best results were slices 104 and 106. Since the method performed to extract the coefficients (PCA) transformed and resized all the available information we could build the following hypothesis considering also the conclusions from mRMR: slice 82 is the most informative slice when looking at specific regions of the image but slices 104 and 106 are more relevant when viewing the image as a whole. Again, the experiment with the best outcome keeps being the all-slices classifier with an outstanding F1 score of 0.9979.

F1	CN	SMC	EMCI	MCI	LMCI	AD	AVG
55	0.9912	0.9939	1	0.9984	0.9930	0.9917	**0.9947**
56	0.9930	0.9939	1	0.9984	0.9930	0.9933	**0.9953**
61	0.9947	0.9939	0.9896	0.9984	0.9903	0.9950	**0.9936**
72	0.9947	0.9939	0.9948	1	0.9930	0.9950	**0.9952**
82	0.9912	1	1	1	0.9958	0.9933	**0.9967**
90	0.9947	1	0.9948	1	0.9944	0.9967	**0.9968**
104	0.9947	1	1	0.9984	0.9958	0.9950	**0.9973**
106	0.9947	1	1	0.9984	0.9958	0.9950	**0.9973**
114	0.9947	1	0.9948	1	0.9944	0.9967	**0.9968**
All	0.9947	1	1	1	0.9958	0.9967	**0.9979**
$*$	0.9930	1	1	1	0.9958	0.9950	**0.9973**

Tabla 3.4: F1 PCA results

An extra experiment was conducted also using PCA and was noted as "$*$". This test retrieves the output of every slice and selects the most common value, simulating a decision system between different experts as shown in Fig. 3.6.

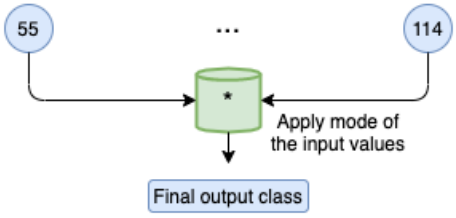


Figure 3.6: $*$ classifier functioning

However, the all-slices classifier performs better, confirming the hypothesis opened after discussing the results from mRMR: processing all the information is more effective than focusing on specific areas.

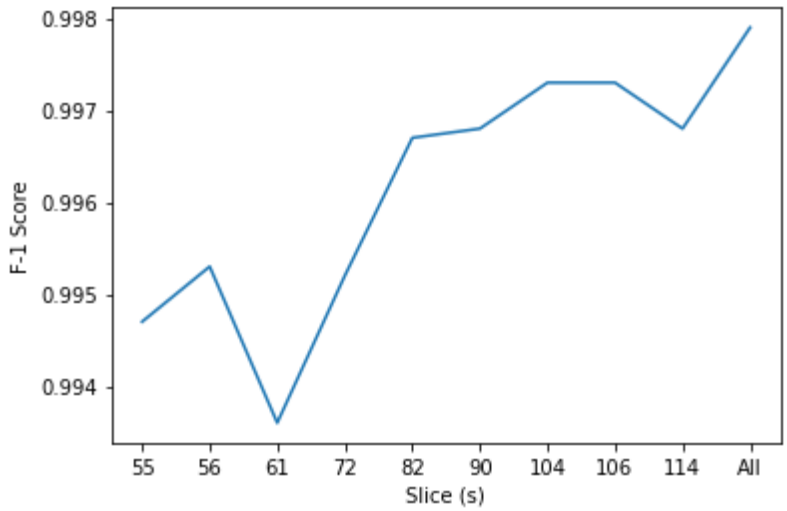


Figure 3.7: F1 PCA scores

So far, our classifier is able to recognize SMC, EMCI and MCI perfectly. We will contrast the score obtained from the all-slices classifier against the outcome of CNN in the following section. Overall, by using PCA we have achieved an increment of more than 3 % in F1 score as shown in Fig. 3.8, being also noticeable how this approach provides a most homogeneous performance for every slice.

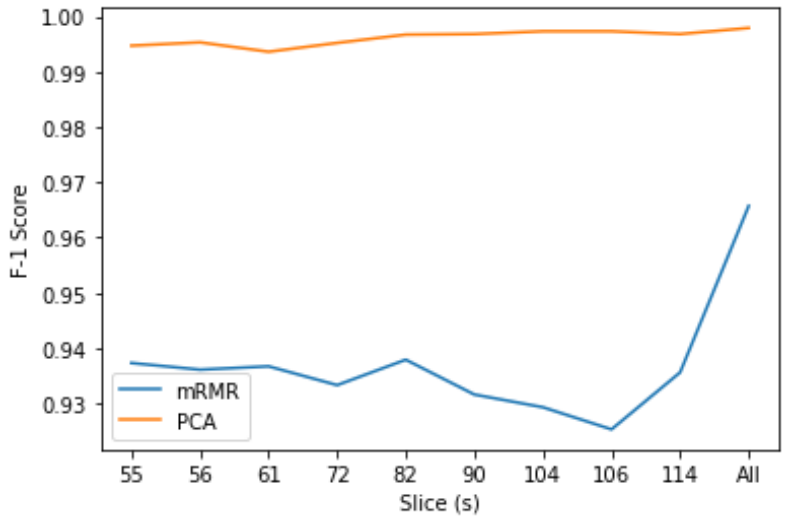


Figure 3.8: F1 mRMR-PCA scores comparison

3.4. CNN Research

The second part in our investigation is to train a convolutional neural network (see section 2.3.2) using directly the images available as input. Along with a custom CNN we will also test VGG16 modifying the input and output layers. For both experiments, the number of epochs is set to ten and the images are resized to 224×224 according to VGG16's requirements. The objective is to make both experiments as similar as possible.

By using CNN, the pre-processing steps covered in SVM can be skipped since CNN is able to learn features from the image itself. Also, we can keep the RGB channels making our input a tensor of order four. To be able to compare the performance of CNN with SVM, the same training and test set is considered. For custom and transfer learning CNN the script *raw_array_getter.py* (see section 5.5) is used to resize and save the arrays of each slice's set while the class *cnn.py* handles the logic for the CNN models (see section 5.6).

3.4.1. Custom CNN

To build our CNN, we use a common and effective structure consisted of the following sequence of layers:

1. One convolutional layer with 32 filters, a kernel size of (3,3) and "relu" as activation function. The number of filters determines the number of kernels to convolve with the input volume. Each of these operations produces a 2D activation map. Layers early in the network architecture (i.e., closer to the actual input image) usually learn fewer convolutional filters while layers deeper in the network (i.e., closer to the output predictions) will learn more filters. This layer will expect an input of size (224, 224, 3) so it can fit our images.
2. One max pooling layer to reduce the spatial dimensions of the output volume.
3. Another similar convolutional layer but using 64 filters this time.
4. Another max pooling layer to reduce the spatial dimensions of the output volume in the second convolutional layer.
5. One flatten layer to connect the multidimensional data from convolution to dense layers.
6. Two dense layers with "relu" and "softmax" activation functions respectively. We use "softmax" to convert the scores to a normalized probability distribution.

Copying the experiments conducted in SVM, we build a model for every slice as shown in table 3.5. The pattern observed in the previous section is also repeated when using CNN. Slice 82 is the most informative and SMC, EMCI, MCI and LMCI are perfectly

matched with a noticeable but not pronounce difficulty identifying CN and AD.

F1	CN	SMC	EMCI	MCI	LMCI	AD	AVG
55	0.9577	1	1	1	1	0.9601	**0.9862**
56	0.9420	1	1	1	1	0.9482	**0.9817**
61	0.9388	1	1	1	1	0.9477	**0.9811**
72	0.9410	1	1	1	1	0.9393	**0.9801**
82	0.9568	1	1	1	1	0.9609	**0.9863**
90	0.9486	1	1	1	1	0.9488	**0.9829**
104	0.9461	1	1	1	1	0.9479	**0.9823**
106	0.9226	1	1	1	1	0.9165	**0.9732**
114	0.9495	1	1	1	1	0.9545	**0.9840**
$*$	0.9665	1	1	1	1	0.9685	**0.9892**

Tabla 3.5: F1 custom CNN results

The experiment "$*$" (see previous section) is also included and performs better than considering the decision of every slice separately, reaching a maximum F1 score of 0.9892. Note that for CNN it is not possible to build the all-slices classifier because the input of our model would be too large for the machine in which this research is carried out.

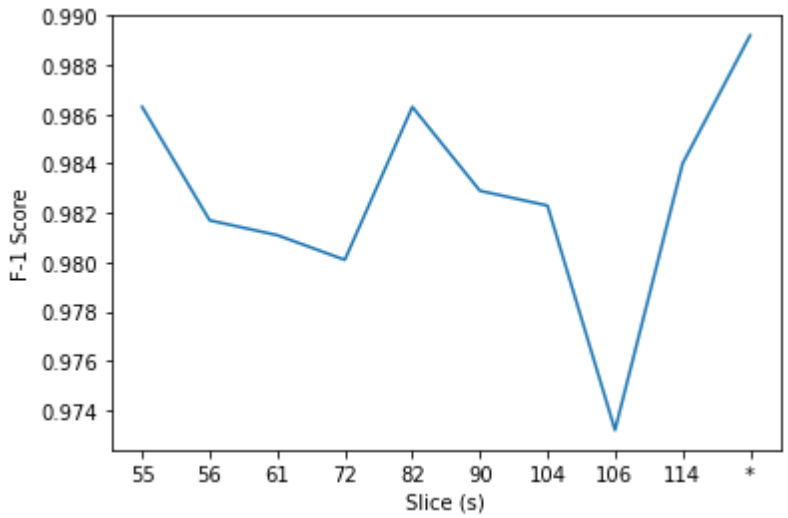


Figure 3.9: F1 custom CNN scores

Although the scores obtained using custom CNN are more than decent it still performs worse than our SVM classifier based on PCA (let us remember that we make the classifier reach 0.9973). In Fig. 3.9 we can see the final score for each slice and custom CNN classifier.

3.4.2. Transfer Learning and CNN

For our last experiment we will use VGG16 pre-trained model with a custom output (dense) layer using again "softmax" as activation function. This model needs a longer time to get trained since the number of layers and weights are significantly higher than our custom CNN. Jumping directly into the results in table 3.6 we can see that the previous pattern is repeated except that slice 82 is not the best classifier but slice 114, probably due to easier recognition patterns present in slice 114 for VGG16 . The highest score obtained belongs to "∗" classifier as expected, with a total F1 score of 0.9464.

F1	CN	SMC	EMCI	MCI	LMCI	AD	AVG
55	0.8380	0.9970	0.9948	0.9901	0.9972	0.8313	**0.9414**
56	0.6815	0.9970	1	0.9867	1	0.7967	**0.9103**
61	0.8106	0.9970	0.9949	0.9696	1	0.7762	**0.9247**
72	0.8432	0.9970	0.9843	0.9838	1	0.8604	**0.9447**
82	0.8396	0.9970	0.9897	0.9854	1	0.8571	**0.9448**
90	0.7960	0.9970	0.9949	0.9799	1	0.7223	**0.9150**
104	0.8407	0.9970	0.9896	0.9885	0.9972	0.8419	**0.9425**
106	0.7938	0.9970	0.8977	0.9885	0.9767	0.8397	**0.9156**
114	0.8444	0.9970	0.9949	0.9901	1	0.8484	**0.9458**
∗	0.8459	0.9970	0.9949	0.9901	1	0.8503	**0.9464**

Tabla 3.6: F1 transfer learning CNN results

It would be interesting to compare the results obtained from VGG16 with the ones outputted from other transfer learning models like ResNet50, Inceptionv3 or EfficientNe. Nevertheless, this extension goes beyond the objective of this investigation. With respect to VGG16, a more pronounce difference in terms of performance it is appreciable between the slices (see Fig. 3.10).

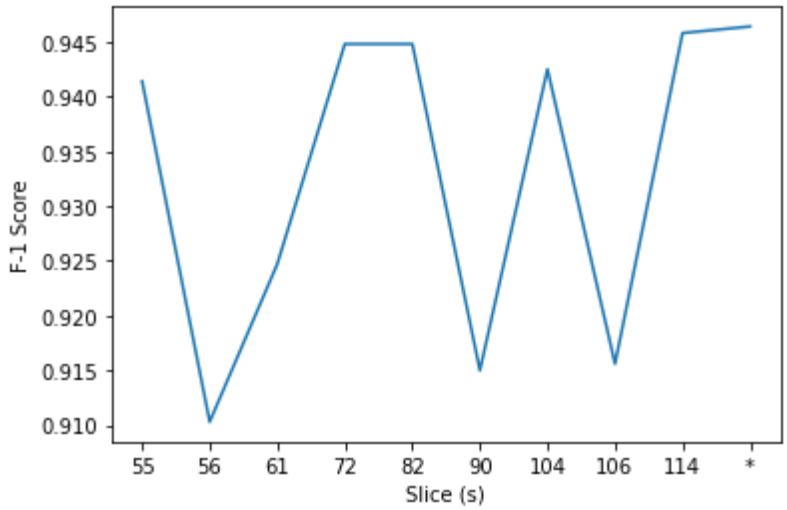


Figure 3.10: F1 custom CNN scores

Overall, the classifier performs very well. However, once we have covered all the possible approaches, transfer learning proves not to be the best option between the available CNN alternatives as shown in Fig. 3.11.

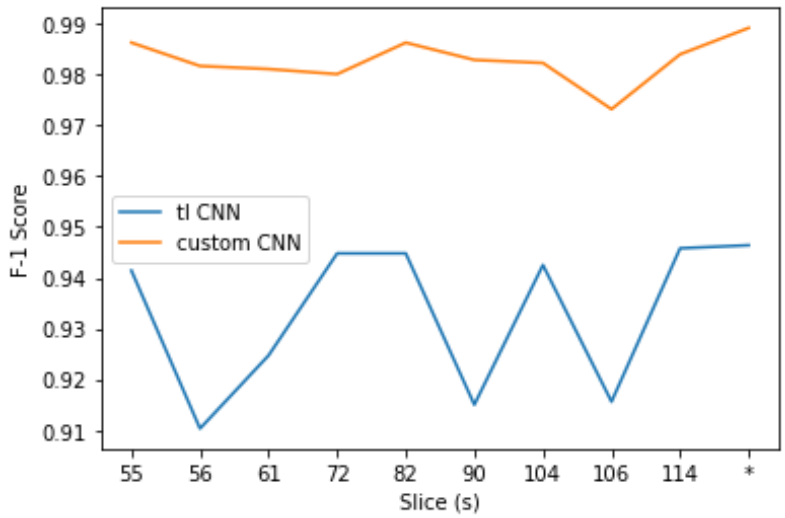


Figure 3.11: F1 tl-custom CNN scores comparison

Chapter 4

Conclusions and Future Investigations

Lastly, in this section the conclusions reached through the experimentation are exposed. Likewise, some relevant proposals are included to continue the investigation conducted in this research.

4.1. Conclusions

The development of this work was motivated not only by medical purposes but also the existing dilemma about the usage of support vector machine and convolutional neural networks. More and more, medicine and artificial intelligence tend to go together and prove to obtain outstanding results when it comes to enhancing people's live. The limits of this symbiosis are still uncertain although it seems undeniable that such combination has already started a new era in science early in the 21th century.

Following our main objectives, the conclusions obtained are highlighted below:

- It is possible to detect early stages in Alzheimer's disease and this prediction can be as precise as the prediction of dementia itself. Significant Memory Concern (SMC) and Early Mild Cognitive Impairment (EMCI) have proved to have an effect on the brain that can be measured and spotted. Patients with early symptoms of dementia can be localized and preventive treatments can be applied.
- If the MRI images reach a high level of normalization and enough samples are accessible it is possible to build an SVM classifier able to predict Alzheimer's stages with an accuracy higher than 99 %. As mention through the research the key point of this investigation is the high quality of the dataset and the normalization applied to the data beforehand.

- In MRI images some slices are more informative than others i.e., there are parts in the brain that contain more information and can be used more precisely to provide a diagnosis of the patient. Between the available slices in our dataset, slice 82 demonstrated to give the better results. This slice is located in the coronal plane, confirming the conclusions exposed by Luis Balderas in his thesis [14].
- In order to give a more accurate diagnosis it is better to process all the information available in the brain rather than considering located regions only. Even if the disease has a more noticeable impact on specific regions, the knowledge distributed around the brain mass makes a difference when seeking for the best results.
- Both SVM and CNN approaches perform extremely well. Nevertheless, SVM stands out above CNN. A possible explanation to this fact is the normalization and regularity of the data. Since the available images are already resized, and the classifier is built using delimited and located image zone's variations the edge identification power of CNN does not beat the capacity of SVM to allocate samples in $\mathbb{R}^n$, $n \in \mathbb{N}$ and group them using the partitions generated by its trained hyperplane.
- The Mallat algorithm exposed in section 2.1.5 can be used to access the wavelet coefficients at deeper levels of the approximation image LL. These coefficients are still very informative, exposing the power of the wavelet transform even in today's image classification tasks. Using the wavelet coefficients from the approximation image at level four gave an outstanding F1 score of 0.9979. This classifier that used all the available coefficients from the set of slices handled performed better than slice-isolated classifiers accessing wavelet coefficients at level three.
- PCA works better than regular feature selection algorithms when facing image classification problems where data has certain continuity properties. Features are highly correlated with each other and present small variations. Applying feature selection could lead to miss wider anomalies that would be detected using a dimensionality reduction system.

4.2. Future Investigations

Disease's diagnosis using MRI is an open and extensive line of research. To continue the steps followed in this investigation I suggest these possible lines:

- Using a machine with better specifications will allow to access a higher level of detail in the images and therefore anomalies can be potentially better detected. Also, by using a larger RAM memory, it would be possible to build a multi-image CNN classifier using the aggregation of all the available slices.
- Investigate whether accessing different types of wavelet coefficients (diagonal, horizontal or vertical) can lead to a better outcome in F1 score or not.
- Research on the structure of CNN models to develop smarter and more suitable networks using different distribution and types of convolutional layers.
- Develop new paradigms of research to process 3D images and investigate the possible use and applications of the 3D wavelet transform.

Chapter 5

Annex

In this chapter the code developed throughout the research in chapter 3 will be exposed. The following snippets correspond to Python code so the reader should be familiarized with this language for a full understanding of its logic.

5.1. patient.py

```
import cv2
import pywt
import numpy as np

from .constants import GROUPS, SLICES, PATH_TO_IMAGE_FOLDER, IMAGE_FORMAT

class Patient:

    def __init__(
        self,
        patient_number,
        group,
        path_to_folder=PATH_TO_IMAGE_FOLDER,
        normalize_colour=True,
        resize_image=False,
        resize_shape=(224, 224, 3)
    ):
        """ Constructor: initialize an instance of the class.

        Args:
            patient_number (str): number of the patient
            group (str): group to which the patient belongs
            path_to_folder (str): relative path to images folder
            normalize_colour (str): if set to True the image will be
                                    converted to BGR2GRAY
```

```
26              resize_image (str): if set to True the image will be resized to 224x224
27          """
28          if group.lower() not in GROUPS:
29              raise Exception(f"Error: group={group} not valid")
30          group = group.upper()
31          images = {}
32          for slice_number in SLICES:
33              image_file_name = IMAGE_FORMAT.format(group=group,
34                                                    patient_number=patient_number,
35                                                    slice_number=slice_number
36                                                )
37              path_to_slice = f"{path_to_folder}/{group}/{image_file_name}"
38              try:
39                  image = cv2.imread(path_to_slice)
40                  if resize_image:
41                      image.resize(resize_shape)
42                  if normalize_colour:
43                      image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
44                  image = np.float32(image)
45                  image /= 255
46                  images[str(slice_number)] = image
47              except Exception as e:
48                  raise Exception(f"Error: could not load slice={slice_number}"
49                                  f"for patient={patient_number}, detail={e}")
50          self.images = images
51          self.normalized_colour = normalize_colour
52          self.group = group.lower()
53
54      def get_full_images_array(self):
55          """ Gets the all the flatten images array concatenated in a single 1D array
56          Returns:
57              np.array
58          """
59
60          final_image = self.images[str(SLICES[0])]
61          for slice_number in SLICES[1:]:
62              final_image = np.append(
63                                  final_image,
64                                  self.get_single_image_array(slice_number),
65                                  axis=0
66                              )
67          return final_image
68
69      def get_single_image_array(self, slice_number):
70          """ Gets the flatten array corresponding to the slice specified in the
71              parameter
72          Args:
73              slice_number (int): number of the slice to access
74          Returns:
```

```
            np.array
        """
        return self.images[str(slice_number)]

    def get_slice_wavelet_coefficients(self, wavelet_name, level, slice_number):
        """ Gets the wavelet coefficients associated with the slice_number
            specified in the parameter
        Args:
            wavelet_name (str): wavelet type to use
            level (int): level (depth) of the approximation layer to access
            slice_number (int): number of the slice to access
        Returns:
            list
        """
        if not self.normalized_colour:
            raise Exception("Error: patient's images are not normalized")
        # Gets only approx layer in the specified level
        raw_coeffs = pywt.wavedec2(
                            data=self.get_single_image_array(slice_number),
                            wavelet=wavelet_name,
                            level=level
                        )
        target_coeffs = raw_coeffs[0]
        coeffs = []
        for i in range(len(target_coeffs)):
            for j in range(len(target_coeffs[0])):
                coeffs.append(target_coeffs[i][j])

        return coeffs

    def get_wavelet_coefficients(self, wavelet_name, level):
        """ Gets all the wavelet coefficients associated with the patient
        Args:
            wavelet_name (str): wavelet type to use
            level (int): level (depth) of the approximation layer to access
        Returns:
            list
        """

        if not self.normalized_colour:
            raise Exception("Error: patient's images are not normalized")
        coefficients_list = []
        for slice_number in self.images:
            coefficients_item = self.get_slice_wavelet_coefficients(
                                    wavelet_name,
                                    level,
                                    slice_number
                                )
            coefficients_list = np.concatenate([coefficients_list, coefficients_item])
```

```
124         return coefficients_list
```

Listing 1: patient.py

5.2. coeff_getter.py

```
1   from ..common.patient import Patient
2   from ..common.constants import PATH_TO_COEFFICIENTS_FOLDER, INDEXES
3   import numpy as np
4   import pandas as pd
5   import pickle
6   import typer
7
8   WAVELET_NAME = "bior3.3"
9   MAX_SAMPLES_NUMBER = 8000
10
11
12  def get_coefficients(
13      patient_slice: int=None,
14      wavelet_level: int=4,
15      result_file: str=None,
16      save: bool=True
17  ):
18      """ Extracts Wavelet coefficients for a specific (or all) slices from the approx
19          image in the passed level. The function will append the coefficients for
20          all the patients
21
22          Args:
23              patient_slice (int): number of the slice to process or "all" if all
24                                   slices must be processed together
25              wavelet_level (int): level of the approx image
26              result_file (str): file to store the results
27              save (bool): True if results must be saved
28          Returns:
29              np-array: array of tuples. Each tuple contains the coefficients of one
30                        patient plus the class it belongs to
31
32      """
33      for patient_class in INDEXES:
34          coeff_list = [None]*3000
35          coeff_index = 0
36          class_name, values = list(patient_class.items())[0]
37          print(f"loading {class_name} patient coefficients...")
38          for i in range(values["first"], values["last"]):
39              print(f" patient {i}")
40              try:
```

```
                new_patient = Patient(i, class_name)
                if patient_slice:
                    coeffs = new_patient.get_slice_wavelet_coefficients(
                        wavelet_name=WAVELET_NAME,
                        level=wavelet_level,
                        slice_number=patient_slice
                    )
                else:
                    coeffs = new_patient.get_wavelet_coefficients(
                        wavelet_name=WAVELET_NAME,
                        level=wavelet_level
                    )
            except Exception as e:
                print(type(e))
                print(f" Couldn't load patient {i}")
                continue

            coeff_list[coeff_index] = (coeffs, class_name)
            coeff_index += 1

        coeff_list = [coeff for coeff in coeff_list if coeff]
        class_name_string = f"class_{class_name}"
        if not result_file:
            slice_string = f"slice_{patient_slice}" if patient_slice else "slice_all_"
            wavelet_level_string = f"level_{wavelet_level}"
            formated_result_file = f"{PATH_TO_COEFFICIENTS_FOLDER}/"
                                   f"{class_name_string}_{slice_string}"
                                   f"_{wavelet_level_string}.npy"
        else:
            formated_result_file = f"{PATH_TO_COEFFICIENTS_FOLDER}/"
                                   f"{class_name_string}_{result_file}"
        coeff_array = np.array(coeff_list, dtype=object)

        if save:
            np.save(formated_result_file, coeff_array)
            print(f"File {formated_result_file} saved")
        return coeff_array


def main(
    patient_slice: int=typer.Argument(None),
    wavelet_level: int=typer.Argument(None),
    result_file: str=typer.Argument(None),
    save: bool=typer.Argument(True)
):

    get_coefficients(patient_slice,
                     wavelet_level,
                     result_file,
```

```
                    save
                    )

if __name__ == "__main__":
    typer.run(main)
```

Listing 2: coeff_getter.py

5.3. coeff_selector.py

```
import pandas as pd
import numpy as np
import typer

from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from ..common.constants import PATH_TO_COEFFICIENTS_FOLDER,
PATH_TO_SELECTED_COEFFICIENTS_FOLDER, GROUPS, SLICES
from pymrmre import mrmr

available_algorithms = ["pca", "mrmr"]
MAX_COEFFS_BATCH = 1000


def mrmr_selector(df, df_target, max_coeffs_batch, total_features=100):
    """ Recursively selects the best {total_features} features from the DataFrame passed
        as argument, processing only {max_coeffs_batch} columns per iteration.

        Args:
            df (DataFrame): contains the columns with values that we want to select
            df_target (DataFrame): contains the class associated with each row in df
            max_coeffs_batch (int): number of columns to process per iteration
            total_features (int): max_number of features to select

        Returns:
            selected {total_features} (int)

    """
    def select_features(df, df_target, total_features):
        result = mrmr.mrmr_ensemble(
            features=df,
            targets=df_target,
            solution_length=total_features
        )
        return result.iloc[0][0]

```

```
    features_list = df.columns
    number_of_coeffs = len(df.columns)
    number_subsets = number_of_coeffs // max_coeffs_batch
    module_subsets = False if number_of_coeffs % max_coeffs_batch == 0 else True
    selected_features = []
    # If we are handling the final subset, select and return
    if number_subsets == 0 or (number_subsets == 1 and not module_subsets):
        print("Processing last subset...")
        result = select_features(df, df_target, total_features)
        selected_features.extend(
            [feature for feature in result if feature != "group"]
        )
        return selected_features
    # Otherwise, get selected features for every subset
    else:
        print("Processing subsets...")
        for i in range(0, number_subsets):
            print(f" subset {i}")
            subset = df[features_list[i*MAX_COEFFS_BATCH:(i+1)*MAX_COEFFS_BATCH]]
            result = select_features(subset, df_target, total_features)
            selected_features.extend(
                [feature for feature in result if feature != "group"]
            )
        if module_subsets:
            print(f" subset {number_subsets}")
            subset = df[features_list[(number_subsets-1)*MAX_COEFFS_BATCH:]]
            result = select_features(subset, df_target, total_features)
            selected_features.extend(
                [feature for feature in result if feature != "group"]
            )

    selected_features = mrmr_selector(
        df[selected_features],
        df_target,
        max_coeffs_batch
    )
    return selected_features


def pca_reductor(df, total_features=100):
    """ Applies dimensionality reduction to fit a maximum of {total_features}
        features

        Args:
            df (DataFrame): contains the columns with values that we want to reduct
            total_features (int): max_number of features to select

        Returns:
            reducted df (DataFrame)
```

```
86
87      """
88      pca = PCA(n_components = 100, svd_solver = 'full')
89      df = pca.fit_transform(df)
90      return df
91
92
93  def transform_and_select_coefficients(
94      patient_slice: str="all",
95      wavelet_level: int=4,
96      algorithm: str="pca",
97      standarize: bool=True,
98      save=True
99  ):
100     """ Apply feature selection and returns a DataFrame with the result
101         features
102
103         Args:
104             patient_slice (str): slice to process
105             wavelet_level (int): level of the wavelet coefficients accessed
106             algorithm (str): algorithm to use
107             standarize (bool): True if data must be standardized
108             save (bool): True if resulting DataFrame must be saved
109
110         Returns:
111             reducted df (DataFrame)
112
113     """
114     # Check if algorithm is available
115     if algorithm not in available_algorithms:
116         raise Exception(f"algorithm={algorithm} not valid")
117
118     # Load coefficients
119     all_coefficients = None
120     all_targets = None
121     group_int = 1
122     if patient_slice == "all":
123         print("loading coefficients for all slices...")
124         for group in GROUPS:
125             partial_coefficients = None
126             for slice_number in SLICES:
127                 file = f"{PATH_TO_COEFFICIENTS_FOLDER}/"
128                        f"class_{group}_slice_{slice_number}_level_{wavelet_level}.npy"
129                 coefficients = np.load(file, allow_pickle=True)
130                 slice_partial_coefficients = [coeff[0] for coeff in coefficients]
131                 if partial_coefficients is not None:
132                     partial_coefficients = [partial_coefficients[i]
133                         + slice_partial_coefficients[i]
134                         for i in range(0, len(partial_coefficients))]
```

```
                else:
                    partial_coefficients = slice_partial_coefficients

            if all_coefficients is not None:
                all_coefficients = np.concatenate(
                    [all_coefficients, partial_coefficients]
                )
            else:
                all_coefficients = partial_coefficients
            partial_targets = [group_int]*len(partial_coefficients)
            if all_targets is not None:
                all_targets = np.concatenate([all_targets, partial_targets])
            else:
                all_targets = partial_targets
            group_int += 1
    else:
        print(f"loading coefficients for slice={patient_slice}...")
        for group in GROUPS:
            file = f"{PATH_TO_COEFFICIENTS_FOLDER}/"
                   f"class_{group}_slice_{patient_slice}_level_{wavelet_level}.npy"
            coefficients = np.load(file, allow_pickle=True)
            partial_coefficients = [coeff[0] for coeff in coefficients]
            if all_coefficients is not None:
                all_coefficients = np.concatenate(
                    [all_coefficients, partial_coefficients]
                )
            else:
                all_coefficients = partial_coefficients
            partial_targets = [group_int]*len(partial_coefficients)
            if all_targets is not None:
                all_targets = np.concatenate([all_targets, partial_targets])
            else:
                all_targets = partial_targets
            group_int += 1

    # Make the mean of the distribution 0
    if standarize:
        print("Standardizing...")
        scaler = StandardScaler()
        all_coefficients = scaler.fit_transform(all_coefficients)

    # Create a more readable feature list
    print("Adding features...")
    features_list = [f"feature{i}" for i in range(0, len(all_coefficients[0]))]
    print("length coeff")
    print(len(features_list))
    # Creating DataFrame for algorithm's input
    print("Transforming into DataFrame...")
    df_coefficients = pd.DataFrame(data=all_coefficients, columns=features_list)
```

```
184     df_target = pd.DataFrame(data=all_targets, columns=["group"])
185 
186     # Memory concern
187     del all_coefficients
188     del all_targets
189 
190     if algorithm == "mrmr":
191         print("Applying MRMR...")
192         selected_features = mrmr_selector(df_coefficients, df_target, MAX_COEFFS_BATCH)
193         df_coefficients = df_coefficients[selected_features]
194 
195     else:
196         print("Applying PCA...")
197         df_coefficients = pca_reductor(df_coefficients)
198         df_coefficients = pd.DataFrame(data=df_coefficients)
199         print(df_coefficients)
200 
201     df_concat = pd.concat([df_coefficients, df_target], axis=1)
202     if save:
203         df_concat.to_csv(
204             f"{PATH_TO_SELECTED_COEFFICIENTS_FOLDER}/"
205             f"selected_features_{algorithm}_slice_{patient_slice}"
206             f"_wavelet_{wavelet_level}.csv"
207         )
208     return df_concat
209 
210 
211 def main(
212     patient_slice: str=typer.Argument(None),
213     wavelet_level: int=typer.Argument(None),
214     algorithm: str=typer.Argument(None),
215     save: bool=typer.Argument(True)
216 ):
217 
218     transform_and_select_coefficients(
219         patient_slice,
220         wavelet_level,
221         algorithm,
222         save
223     )
224 
225 if __name__ == "__main__":
226     typer.run(main)
```

Listing 3: coeff_selector.py

5.4. svm.py

```
import pandas as pd
from sklearn import metrics
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from statistics import mode

from .. common.constants import PATH_TO_SELECTED_COEFFICIENTS_FOLDER

class SVM:

    parameters = {'kernel':('linear', 'rbf', 'sigmoid'), 'C': [1, 10, 100, 1000],
                  'gamma': [1e-1, 1e-2, 1e-3, 1e-4, "scale"]}

    def __init__(
        self,
        kernel=None,
        c=None,
        gamma=None,
        selection_algorithm="pca",
        wavelet_level=4,
        path_to_folder=PATH_TO_SELECTED_COEFFICIENTS_FOLDER,
        target_column="group",
        target_score="f1",
        slices=["all"],
        random_state=104729,
        models=None,
    ):
        """ Constructor: initialize an instance of the class.

        Args:
            patient_number (str): kernel of the SVM
            c (int): regularization parameter
            gamma (str): kernel coefficient
            selection_algorithm (str): algorithm used to select features
            wavelet_level (int): wavelet level used
            path_to_folder (str): path to coefficient files in local directory
            target_column (str): target column in saved csv
            target_score (str): target score to use for gridseacrh
            slices (list): list of slices to use to build the models (>1 if multiexpert)
            random_state (int): random state parm for train_test_split
            models (list): in case models are loaded from an external source,
                        list of models
        """
        df_coefficients = {}
```

```
47          df_target = {}
48          for slice_number in slices:
49              path_to_coefficients_file = f"{path_to_folder}/selected_features_"
50                                          f"{selection_algorithm}_slice_{slice_number}_"
51                                          f"wavelet_{wavelet_level}.csv"
52              # Drop residual columns
53              df = pd.read_csv(path_to_coefficients_file)
54              df = df.drop(list(df.filter(regex='Unnamed')), axis=1)
55              df_coefficients[str(slice_number)] = df.drop(target_column, axis=1)
56              df_target[str(slice_number)] = df.drop(
57                  df.columns.difference([target_column]),
58                  1
59              )
60
61          if target_score == "f1":
62              score = metrics.make_scorer(metrics.f1_score, average = "weighted")
63          elif target_score == "precision":
64              score = metrics.make_scorer(metrics.precision_score, average = "weighted")
65          elif target_score == "recall":
66              score = metrics.make_scorer(metrics.recall_score, average = "weighted")
67          else:
68              raise Exception(f"Error: target_score={target_score} not available")
69
70          self.df_coefficients = df_coefficients
71          self.df_target = df_target
72          self.score = score
73          self.target_column = target_column
74          self.kernel = kernel
75          self.c = c
76          self.gamma = gamma
77          self.models = {} if not models else models
78          self.random_state = random_state
79
80      def find_best_paremeters(
81          self,
82          print_results=True,
83          train_size=0.75,
84          slice_number=None
85      ):
86          """Finds the best parameters to build and train the model
87
88             Args:
89                 print_results (bool): True if results should be printed
90                 train_size (float): 0-1 percentage of sample to use to
91                                     train the model
92          """
93          if not slice_number:
94              slice_number = list(self.df_coefficients.keys())[0]
95
```

```
        X_train, X_test, Y_train, Y_test = train_test_split(
            self.df_coefficients[str(slice_number)],
            self.df_target[str(slice_number)],
            train_size=train_size,
            random_state=self.random_state,
            stratify=self.df_target[str(slice_number)]
        )
        print("Getting scores...")
        svc = SVC()
        clf = GridSearchCV(estimator=svc, param_grid=self.parameters, scoring=self.score)
        clf.fit(X_train, Y_train.values.ravel())
        means = clf.cv_results_['mean_test_score']
        stds = clf.cv_results_['std_test_score']
        y_true, y_pred = Y_test, clf.predict(X_test)


        if print_results:
            print("Best parameters set found on development set:")
            print()
            print(clf.best_params_)
            print()
            print("Grid scores on development set:")
            print()
            for mean, std, params in zip(means, stds, clf.cv_results_['params']):
                print("%0.3f (+/-%0.03f) for %r"
                      % (mean, std * 2, params))
            print()
            print("Detailed classification report:")
            print()
            print("The model is trained on the full development set.")
            print("The scores are computed on the full evaluation set.")
            print()
            print(classification_report(y_true, y_pred, digits=4))
            print()

        self.c = clf.best_params_["C"]
        self.gamma = clf.best_params_["gamma"]
        self.kernel = clf.best_params_["kernel"]

    def train_model(self, train_size=0.75, slice_number=None):
        """Builds and trains the model using a specific slice in the class

           Args:
               train_size (float): 0-1 percentage of sample to use to
                                   train the model
                slice_number (str). slice number to use to build the model
        """

        if not slice_number:
```

```
145             slice_number = list(self.df_coefficients.keys())[0]
146
147         X_train, X_test, Y_train, Y_test = train_test_split(
148             self.df_coefficients[str(slice_number)],
149             self.df_target[str(slice_number)],
150             train_size=train_size,
151             random_state=self.random_state,
152             stratify=self.df_target[str(slice_number)]
153         )
154         print("training model...")
155         model = SVC(kernel=self.kernel, C=self.c, gamma=self.gamma)
156         model.fit(X_train, Y_train.values.ravel())
157         y_hat = [x for x in model.predict(X_test)]
158         print(classification_report(Y_test, y_hat, digits=4))
159         self.models[str(slice_number)] = model
160
161     def train_multiexpert_model(self, train_size=0.75):
162         """Builds and trains the model as a multiexpert (multimage)
163            model, using all the slices available in the class
164
165            Args:
166                train_size (float): 0-1 percentage of sample to use to
167                                    train the model
168         """
169
170         def get_average_pred(models, X_test):
171             for i in range(0, len(models.keys())):
172                 y_hats[i] = models
173
174         print("training models...")
175         models = {}
176         Y_train, Y_test = train_test_split(
177             self.df_target[list(self.df_target.keys())[0]],
178             train_size=train_size,
179             random_state=self.random_state,
180             stratify=self.df_target[list(self.df_target.keys())[0]]
181         )
182         y_hats = []
183         for slice_number in self.df_coefficients.keys():
184             X_train, X_test = train_test_split(
185                 self.df_coefficients[str(slice_number)],
186                 train_size=train_size,
187                 random_state=self.random_state,
188                 stratify=self.df_target[str(slice_number)]
189             )
190             model = SVC(kernel=self.kernel, C=self.c, gamma=self.gamma)
191             model.fit(X_train, Y_train.values.ravel())
192             models[str(slice_number)] = model
193             y_hats.append([x for x in model.predict(X_test)])
```

```
        y_hat_final = []
        for i in range(0, len(y_hats[0])):
            y_hat_final.append(mode([item[i] for item in y_hats]))

        print(classification_report(Y_test, y_hat_final, digits=4))
        self.models = models


    def make_prediction(self, X_to_predict, slice_number=None):
        """Use the models available to make predictions based on the
           input passed as parameter

           Args:
               X_to_predict (array-like): transformed array to pass as input
                                          to the model
           Returns:

        """
        if not slice_number:
            y_hats = []
            for model in self.models.keys():
                y_hats.append(
                    [x for x in self.models[str(model)].predict(X_to_predict)]
                )

            y_hat_final = []
            for i in range(0, len(y_hats[0])):
                y_hat_final.append(mode([item[i] for item in y_hats]))
            return y_hat_final
        else:
            y_hats = [x for x in self.models[str(slice_number)].predict(X_to_predict)]
            return y_hats
```

Listing 4: svm.py

5.5. raw_array_getter.py

```
import numpy as np
import typer

from ..common.patient import Patient
from ..common.constants import INDEXES, PATH_TO_RAW_RESHAPED_IMAGES_FOLDER

def get_raw_images(
    patient_slice: int,
```

```
    result_file: str=None,
    save: bool=True,
):
    """ Extracts image array for a specific slice. The function will append
        the arrays for all the patients

        Args:
            patient_slice (int): number of the slice to process
            result_file (str): file to store the results
            save (bool): True if array must be saved

        Returns:
            np-array: array of tuples. Each tuple contains the values of one patient
                      plus the class it belongs to

    """
    array_list = [None]*8000
    array_index = 0
    for patient_class in INDEXES:
        class_name, values = list(patient_class.items())[0]
        print(f"loading {class_name} patient arrays...")
        for i in range(values["first"], values["last"]):
            print(f" patient {i}")
            try:
                new_patient = Patient(
                    i,
                    class_name,
                    normalize_colour=False,
                    resize_image=True,
                    path_to_folder="./alzheimer_classifier/images"
                )
                if patient_slice:
                    array = new_patient.get_single_image_array(
                        slice_number=patient_slice
                    )
                else:
                    array = new_patient.get_single_image_array(
                        slice_number=patient_slice
                    )
            except Exception as e:
                print(e)
                print(f" Couldn't load patient {i}")
                continue

            array_list[array_index] = (array, class_name)
            array_index += 1

    array_list = [item for item in array_list if item]
    if not result_file:
```

```
        slice_string = f"slice_{patient_slice}"
        formated_result_file = f"{PATH_TO_RAW_RESHAPED_IMAGES_FOLDER}/"
                               f"{slice_string}.npy"
    else:
        formated_result_file = f"{PATH_TO_RAW_RESHAPED_IMAGES_FOLDER}/"
                               f"{result_file}.npy"
    array_object = np.array(array_list, dtype=object)
    if save:
        np.save(formated_result_file, array_object)
        print(f"File {formated_result_file} saved")
    return array_object


def main(
    patient_slice: int=typer.Argument(None),
    result_file: str=typer.Argument(None),
    save: bool=typer.Argument(True)
):

    get_raw_images(
        patient_slice,
        result_file,
        save
    )

if __name__ == "__main__":
    typer.run(main)
```

Listing 5: raw_array_getter.py

5.6. cnn.py

```
import numpy as np
from tensorflow.keras import datasets, layers, metrics, models
from tensorflow.keras.applications import VGG16
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from statistics import mode

from .. common.constants import PATH_TO_RAW_RESHAPED_IMAGES_FOLDER, PATH_TO_CNN_FOLDER,
GROUPS, SLICES

class CNN:

    def __init__(
        self,
```

```
        path_to_raw_folder=PATH_TO_RAW_RESHAPED_IMAGES_FOLDER,
        path_to_cnn_folder=PATH_TO_CNN_FOLDER,
        target_column="group",
        target_score="f1",
        slices=[],
        input_shape=(224,224,3),
        random_state=104729,
        class_number=6,
        model_list=None
    ):
        """ Constructor: initialize an instance of the class.

        Args:
            path_to_raw_folder (str): path to array files in local system
            path_to_cnn_folder (str): path to current directory in local system
            target_column (str): target column in saved csv
            target_score (str): target score to use for training
            slices (list): list of slices to use to build the models (>1 if multiexpert)
            input_shape (tuple): size of the images to process
            random_state (int): random state parm for train_test_split
            class_number (int): number of existing classes
            model_list (list): in case models are loaded from an external source,
                               list of models

        """
        image_arrays = {}
        image_targets = {}
        for slice_number in slices:
            path_to_arrays_file = f"{path_to_raw_folder}/"
                                  f"slice_{slice_number.split('_')[0]}.npy"
            # Drop residual columns
            arrays = np.load(path_to_arrays_file, allow_pickle=True)
            hey = np.array([array[0] for array in arrays])
            print(hey.shape)
            image_arrays[str(slice_number)] = np.array([array[0] for array in arrays])
            image_targets[str(slice_number)] = np.array(
                [GROUPS.index(array[1]) for array in arrays]
            )


        if target_score == "f1":
            scores = [metrics.Precision(), metrics.Recall()]
        elif target_score == "precision":
            scores = [metrics.Precision()]
        elif target_score == "recall":
            scores = [metrics.Recall()]
        else:
            raise Exception(f"Error: target_score={target_score} not available")

```

```
        self.path_to_cnn_folder = path_to_cnn_folder
        self.input_shape = input_shape
        self.class_number = class_number
        self.image_arrays = image_arrays
        self.image_targets = image_targets
        self.scores = scores
        self.target_column = target_column
        models_dict = {}
        if model_list:
            for model_slice in model_list:
                models_dict[str(model_slice)] = models.load_model(
                    f"{self.path_to_cnn_folder}/{model_slice}.h5"
                )

        self.models_dict = models_dict
        self.random_state = random_state


    def train_model(self, train_size=0.75, slice_number=None):
        """Builds and trains the model using a specific slice in the class

           Args:
               train_size (float): 0-1 percentage of sample to use to
                                   train the model
               slice_number (str): slice number to use to build the model
        """

        if not slice_number:
            slice_number = list(self.image_arrays.keys())[0]

        cnn = models.Sequential([
            layers.Conv2D(
                filters=32,
                kernel_size=(3, 3),
                activation='relu',
                input_shape=self.input_shape
            ),
            layers.MaxPooling2D((2, 2)),

            layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
            layers.MaxPooling2D((2, 2)),

            layers.Flatten(),
            layers.Dense(64, activation='relu'),
            layers.Dense(self.class_number, activation='softmax')
        ])


        cnn.compile(optimizer='adam',
```

```
113             loss='sparse_categorical_crossentropy',
114             metrics=["accuracy"])
115
116         X_train, X_test, Y_train, Y_test = train_test_split(
117             self.image_arrays[str(slice_number)],
118             self.image_targets[str(slice_number)],
119             train_size=train_size,
120             random_state=self.random_state,
121             stratify=self.image_targets[str(slice_number)]
122         )
123         print("Training model...")
124         print(Y_train)
125         cnn.fit(X_train, Y_train, epochs=10)
126         cnn.evaluate(X_test, Y_test)
127         y_pred = cnn.predict(X_test)
128         y_pred_classes = [np.argmax(element) for element in y_pred]
129
130         print()
131         print("Detailed classification report:")
132         print()
133         print("The model is trained on the full development set.")
134         print("The scores are computed on the full evaluation set.")
135         print()
136         print(classification_report(Y_test, y_pred_classes, digits=4))
137         print()
138
139         print("Saving model...")
140         cnn.save(f"{self.path_to_cnn_folder}/{slice_number}.h5")
141         self.models_dict[str(slice_number)] = cnn
142
143     def train_model_tl(self, train_size=0.75, slice_number=None):
144         """Builds and trains the model using a specific slice in the class
145            and VGG16 model
146
147            Args:
148                train_size (float): 0-1 percentage of sample to use to
149                                    train the model
150                slice_number (str): slice number to use to build the model
151         """
152
153         if not slice_number:
154             slice_number = list(self.image_arrays.keys())[0]
155
156
157         # add preprocessing layer to the front of VGG
158         vgg = VGG16(
159             input_shape=list(self.input_shape),
160             weights='imagenet',
161             include_top=False
```

```
        )

        # don't train existing weights
        for layer in vgg.layers:
          layer.trainable = False


        # our layers - you can add more if you want
        x = layers.Flatten()(vgg.output)
        # x = Dense(1000, activation='relu')(x)
        prediction = layers.Dense(6, activation='softmax')(x)

        # create a model object
        cnn = models.Model(inputs=vgg.input, outputs=prediction)

        # view the structure of the model
        cnn.summary()

        # tell the model what cost and optimization method to use
        cnn.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=["accuracy"])

        X_train, X_test, Y_train, Y_test = train_test_split(
            self.image_arrays[str(slice_number)],
            self.image_targets[str(slice_number)],
            train_size=train_size,
            random_state=self.random_state,
            stratify=self.image_targets[str(slice_number)]
        )
        print("Training model...")
        print(Y_train)
        cnn.fit(X_train, Y_train, epochs=10)
        cnn.evaluate(X_test, Y_test)
        y_pred = cnn.predict(X_test)
        y_pred_classes = [np.argmax(element) for element in y_pred]

        print()
        print("Detailed classification report:")
        print()
        print("The model is trained on the full development set.")
        print("The scores are computed on the full evaluation set.")
        print()
        print(classification_report(Y_test, y_pred_classes, digits=4))
        print()

        print("Saving model...")
        cnn.save(f"{self.path_to_cnn_folder}/{slice_number}_tl.h5")
        self.models_dict[str(slice_number)] = cnn
```

```
211
212
213
214     def make_prediction(self, X_to_predict=None, slice_number="all", save=False):
215         """Use the models available to make predictions based on the
216            input passed as parameter
217
218            Args:
219                X_to_predict (array-like): transformed array to pass as input
220                                           to the model
221                slice_number (str): all if all models will be use
222                save (bool): True to save the results
223            Returns:
224
225         """
226         if not X_to_predict:
227             X_to_predict = train_test_split(
228                 self.image_arrays[str(slice_number)],
229                 train_size=0.75,
230                 random_state=self.random_state,
231                 stratify=self.image_targets[str(slice_number)]
232             )[1]
233
234         y_hat_final = []
235         if slice_number == "all":
236             y_hats = []
237             for model in self.models.keys():
238                 y_hats_array = [x for x in
239                     self.models_dict[str(model)].predict(X_to_predict)]
240                 y_hats_argm = [np.argmax(element) for element in y_hats_array]
241                 y_hats.append(y_hats_argm)
242
243             y_hat_final = []
244             for i in range(0, len(y_hats[0])):
245                 y_hat_final.append(mode([item[i] for item in y_hats]))
246         else:
247             y_hats = [x for x in
248                 self.models_dict[str(slice_number)].predict(X_to_predict)]
249             y_hat_final = [np.argmax(element) for element in y_hats]
250
251         if save:
252             array_object = np.array(y_hat_final)
253             np.save(
254                 f"./alzheimer_classifier/cnn/slice_{slice_number}_prediction.npy",
255                 array_object
256             )
257
258         return y_hat_final
259
```

```
260     def get_scores_model(self, train_size=0.75, slice_number=None):
261         """Builds and trains the model using a specific slice in the class
262            and VGG16 model
263 
264            Args:
265                train_size (float): 0-1 percentage of sample to use to
266                                    train the model
267                slice_number (str): slice number to use to build the model
268         """
269 
270         if not slice_number:
271             slice_number = list(self.image_arrays.keys())[0]
272 
273 
274         X_train, X_test, Y_train, Y_test = train_test_split(
275             self.image_arrays[str(slice_number)],
276             self.image_targets[str(slice_number)],
277             train_size=train_size,
278             random_state=self.random_state,
279             stratify=self.image_targets[str(slice_number)]
280         )
281         print("Training model...")
282         y_hats = [x for x in self.models_dict[str(slice_number)].predict(X_test)]
283         y_hats = [np.argmax(element) for element in y_hats]
284         y_hat_final = y_hats
285 
286         print()
287         print("Detailed classification report:")
288         print()
289         print("The model is trained on the full development set.")
290         print("The scores are computed on the full evaluation set.")
291         print()
292         print(classification_report(Y_test, y_hat_final, digits=4))
293         print()
```

Listing 6: cnn.py

Bibliografía

[1] Y. Y. Tan, *Wavelet Theory Approach to Pattern Recognition*. Series in Machine Perception and Artificial Intelligence — Vol. 74 (2009).

[2] D. G. Costa & L. F. Guedes, *A Discrete Wavelet Transform (DWT)-Based Energy-Efficient Selective Retransmission Mechanism for Wireless Image Sensor Network, 2nd Edition*. J. Sens. Actuator Netw. (2012).

[3] S. L. Brunton & J. N. Kutz, *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control, 1st Edition*. J. Sens. Actuator Netw. (2019).

[4] Z. Zhao, R. Anand & M. Wang, *Maximum Relevance and Minimum Redundancy Feature Selection Methods for a Marketing Machine Learning Platform*. IEEE (2020).

[5] M. Billah & S. Waheed, *Minimum Redundancy Maximum Relevance (mRMR) Based Feature Selection from Endoscopic Images for Automatic Gastrointestinal Polyp Detection*. Springer Science+Business Media, LLC, part of Springer Nature (2020).

[6] M. Mei, *Principal Component Analysis*. The University of Chicago (2009).

[7] I. Jolliffe, *Principal Component Analysis, Second Edition*. Springer Series in Statistics (2002).

[8] M. Teow, *Understanding Convolutional Neural Networks Using A Minimal Model for Handwritten Digit Recognition*. IEEE 2nd International Conference on Automatic Control and Intelligent Systems (2017).

[9] J. Wu, *A Guide to Convolutional Neural Networks for Computer Vision*. Synthesis Lectures on Computer Vision, Gérard Medioni & Sven Dickinson Series Editors, Morgan & Claypool Publishers (2018).

[10] J. Wu, *Introduction to Convolutional Neural Networks*. Nanjing University, China (2017).

[11] L. P. Lebedev, M. J. Cloud, *Tensor Analysis*. River Edge, NJ : World Scientific Pub. (2003).

[12] I. Steinwart, A. Christmann, *Support Vector Machines*. Springer Science+Business Media, LLC (2008).

[13] F. Falah, Y. Prasad, *Multi-class Support Vector Machine (SVM) classifiers – An Application in Hypothyroid detection and Classification* . Sixth International Conference on Bio-Inspired Computing: Theories and Applications. IEEE (2011).

[14] L. Balderas, *Desarrollo de sistemas inteligentes para la clasificación automática de la enfermedad del Parkinson utilizando imágenes MRI*. Facultad de Ciencias y Escuela Superior de Ingeniería Informática y Telecomunicaciones, Universidad de Granada (2019).

[15] Y. Zhang, Z. Dong, P. Phillips, S. Wang, G. Ji, J. Yang, & T.-F. Yuan, *Detection of subjects and brain regions related to Alzheimer's disease using 3D MRI scans based on eigenbrain and machine learning* . Frontiers in Computational Neuroscience, 9 (2015).

[16] I. Beheshti, H.Demirel , *Feature-ranking-based Alzheimer's disease classification from structural MRI* . Magnetic Resonance Imaging (2015), doi:10.1016/j.mri.2015.11.00