SAPIENZA
UNIVERSITÀ DI ROMA

Exploring Authorial Identity through Data Mining Techniques

Analyzing Artistic Traits for Authorship Attribution

Faculty of Mathematics, Physics and Natural Sciences
Master degree in Applied Mathematics

Stefano Magrini Alunno
ID number 1851728

Advisor
Prof. Grabriella Anna Puppo

Academic Year 2023/2024

Thesis defended on 00/00/00
in front of a Board of Examiners composed by:

Prof. uno (chairman)
Prof. due
Prof. tre
Prof. ...
Prof. ...
Prof. ...
Prof. ...

Exploring Authorial Identity through Data Mining Techniques
Master thesis. Sapienza University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Version: December 30, 2024

Author's email: stefanomagrini99@gmail.com

Mater artium necessitas

Contents

List of Figures

List of Tables

Acronyms

CADA Continuous Automatic Drawings' Analysis. 37

CFT Continue Fourier Transform. 23, 34

CPU Central Processing Unit. 54, 55, 65, 68, 77

CUDA Compute Unified Device Architecture. 5, 55, 59, 66, 68, 77

CZT chirp Z-transform. 27, 28

DADA Discrete Automatic Drawings' Analysis. 36, 37

DFT Discrete Fourier Transform. iii, 22–29, 34

EM Expectation-Maximisation. 11, 16, 20

FCM Fuzzy C-Means Clustering. iii, v, 5, 11–16, 19–22, 30, 32, 33, 47, 49–55, 62, 64, 68, 71, 73, 77

FFT Fast Fourier Transform. 5, 27, 29, 30, 34, 35, 43, 44, 46, 59

GMM Gaussian Mixture Models. 5, 20, 21

GPGPU General-Purpose computing on Graphics Processing Units. 54, 55

GPU Graphics Processing Unit. 5, 54, 55, 59, 65–68, 77

HPC High Performance Computing. 55

HSL Hue Saturation Lightness. 41

KKT Karush–Kuhn–Tucker. 18, 19

PCA Principal Component Analysis. 64, 69–73

PPI pixels per inch. 4, 5, 38, 39, 47, 57, 68, 70–73

RGB Red Green Blue. 39–41

SM Stream Multi-Processing. 54, 55

SVD Singular Value Decomposition. 42, 43

Chapter 1

Introduction

1.1 Attribution of handwriting works

The comparison of graphic works plays a very important role in several fields, including art history, digital forensics and intellectual property protection. By analysing the characteristics of graphic works, it is possible to identify the author, verify the authenticity of a work or detect possible counterfeits. In art history, for example, stylistic and technical analysis of handwritten notes or sketches can provide valuable insights into the creative processes of renowned authors. Similarly, in digital forensics, the comparison of graphic works can help detect counterfeit documents or identify alterations to legal documents.
Beyond these practical applications, the ability to compare graphic works also opens up possibilities for understanding more general patterns. For example, it can help uncover stylistic influences between artists or identify recurring patterns within a collection. In the context of machine learning and data analysis, graphical comparison serves as a basis for developing algorithms capable of processing complex visual data, which is increasingly important in an era dominated by digital media.
However, the process is not without its challenges. The presence of noise, variations in resolution and the diversity of graphical styles make it difficult to establish a robust and reliable framework for comparison. This thesis aims to address these issues by developing methods to improve the accuracy and adaptability of graphical work comparisons.

The practical applications of comparing graphic works cover a wide range of fields, each of which benefits from customised analysis techniques:

- **Authorship attribution**: Determining the author of a handwritten document or artistic work is important in fields such as art history, where verifying the authenticity of an artist can have a significant impact on the cultural and financial value of the work.
- **Falsification Detection**: In digital forensics and legal investigations, identifying alterations to documents or detecting forgeries in graphic works plays a key role in ensuring authenticity and legality.

- **Intellectual Property Protection**: The ability to compare graphic works is critical for enforcing copyright laws and resolving disputes over original creations.
- **Historical Analysis**: In the study of historical documents and manuscripts, graphic comparison helps to trace stylistic influences, identify authors, and reconstruct fragmented works.
- **Digital Archiving and Restoration**: Automated comparison methods help to cluster, catalogue and restore large collections of graphic works, ensuring their preservation for future generations.

These applications demonstrate the versatility and importance of robust graphical comparison methods. Each context presents unique challenges, such as the need to handle different resolutions, styles and noise levels, which this thesis aims to address through innovative methods.

Despite its importance, the comparison of graphic works faces several challenges and limitations that have hindered progress in the field:

- **Distortions and impurities**: Graphic works, especially handwritten or historical documents, often contain noise such as background patterns, stains or scanning distortions. These contaminants can distort the analysis and reduce the reliability of the comparison results.
- **Variability in resolutions and formats**: Works are often digitised at different resolutions and stored in different formats, making it difficult to standardise data for analysis. This variability makes it difficult to extract meaningful features.
- **High dimensionality and computational cost**: Graphic works are represented as high-dimensional data, especially when detailed features or pixel-level analysis are involved. This increases computational costs and limits the feasibility of large-scale comparisons.
- **Limited robustness of clustering techniques**: Traditional clustering methods, such as hard KMeans, struggle with noisy and overlapping data distributions, leading to suboptimal results in many real-world scenarios.
- **Lack of standardised datasets**: The lack of well-curated and representative datasets for testing and validating comparative methods makes it difficult to benchmark algorithms and ensure their generalisability.
- **Subjective Preprocessing Steps**: Many preprocessing techniques depend on manual adjustments or heuristics, which can introduce bias and limit the reproducibility of the analysis.

These issues highlight the need for advanced methods that can adapt to noise, handle different data representations, and provide reliable results in a range of scenarios. This thesis directly addresses these challenges by refining preprocessing techniques, introducing fuzzy clustering for improved robustness, and exploring scalable solutions for high-dimensional data.

1.2 Background of the project

In my thesis [5], an attempt was made to adapt an authorship attribution method similar to that proposed in [1], which uses the n-gram model advanced by Martin [6]. This method has several distinctive features:

- The n-gram model was originally designed to emulate natural language rather than images.
- Applying this idea to attributing authorship to images introduces a high degree of complexity, especially in preventing falsifications.

Another remarkable aspect of the attribution process is the comparison formula defined in [1] and later adopted in [5]. This formula defines a comparison function between discrete distributions: the unknown work is compared to all known works, and the results of this function are analysed to determine the most likely author.
However, the nature of this comparison formula, as presented in [5], was not well suited to graphic works. This necessitated a pre-processing phase in which images were converted into matrices of black and white pixels, which simplified the representation of the data but introduced limitations in the handling of more complex graphical features.

One of the main problems encountered in [5] is the significant loss of information caused by the pre-processing phase. A graphic work had to be processed by eliminating shades or editing entire noisy regions. The comparison formula, by its very nature, emphasises details and the presence of high noise is a serious obstacle. For this reason, in [5], we chose to work with manuscripts produced on a tablet, thus ensuring a controlled environment free of impurities. The results were remarkable: almost all the works analysed were correctly attributed.
However, it was observed that this methodology has significant limitations when applied to works of a different nature. The success of the experiment is largely attributable to the fact that writing, by its very nature, is an image composed of small regions that are either very light or very dark. This drastically reduced the negative effects of pre-processing, such as the destruction or creation of information. With less controlled data, such as images from real sheets instead of a tablet, numerous problems could have been caused, compromising the effectiveness of the method.

In this thesis, we investigate the possibility of creating a variant for colour images, thus eliminating the main problem identified in [5]: pre-processing. The aim is to develop a new theory that, unlike in [5] and [1], does not require the discretization of the data. This is a significant step in terms of application, as a colourful image is expected to provide a higher level of matching accuracy.
However, this idea presents some fundamental challenges. The formula for comparing works, as defined in [1], is not directly compatible with colourful images, and the inherent continuous nature of colours may cause the model to consider the works all equally distinct. In addition, the n-gram model of Martin is well suited to natural language words, but less effective for images, which are more susceptible to noise and require an appropriate metric to interpret them.

To realise the idea of attributing works without pre-processing, several methodologies were explored:

1. **Represent the work as a surface in colour space**: Although interesting, this proposal presented significant difficulties in defining an effective way of comparing two works.
2. **Using the Wasserstein distance to compare distributions**: This method proved to be extremely computationally expensive and inefficient, as it did not give sufficient weight to the details of the work, an important element in attribution.
3. **Discretizing the union of distributions by clustering**: This approach finally showed the greatest potential and formed the basis for the development of this thesis.

The central question around which this thesis revolves is: is it possible to attribute graphical works using a dynamic discretisation of space? In other words, the comparison function defined in [1] is seen as an approximate integral over boxes. In fact, by using a matrix with black and white pixels, we have divided the space into boxes of equal size. This method, already used in [5], allows an efficient approximation when the number of boxes is not too large compared to the sparsity of the n-grams. In this thesis, an alternative is proposed: replacing boxes with clustering algorithms. These algorithms successfully handle high-dimensional sparsity problems by offering a more adaptive discretisation that serves as the basis for redefining the comparison formula.

This thesis introduced significant changes not only in the comparison methodology, but also in the pre-processing phase. It was no longer acceptable to work with a 'perfect' dataset; it was necessary to use a real, and therefore inevitably 'dirty' dataset. After much difficulty, it was possible to collect a dataset of 113 university note sheets. However, the quality of the images was insufficient for an accurate analysis, making pre-processing indispensable, which, although minimal, could still have compromised the project. This pre-processing phase was limited to image cleaning and greyscale conversion, an operation that, for university notes, should not have a significant impact.
Another major change affected the image synthesis phase. In [5], images were transformed into a list of n-grams with their respective occurrence in the work. However, as this aspect was no longer central to the proposed methodology, it was preferred to simply provide a list of the extracted tiles.

1.3 Key points

As already pointed out, the collection of the dataset presented considerable difficulties, making manual collection necessary. For this research, dozens of university notebooks were made available, from which 113 pages were selected and scanned. The final result comprises 420 uncompressed image fragments, totalling 1.1 Gb, with a resolution of 400 pixels per inch (PPI). The dataset includes one main author

(`Author 1`), representing more than half of the works, and three other secondary authors whose purpose is to complicate attribution.
The lack of a professional dataset and the very nature of the notebooks required careful pre-processing. Indeed, university notebooks have a background with a grid of squares that can confuse the algorithm, leading it to mistake this structure for human handwriting. Removing the squares without compromising the handwriting details was one of the main challenges faced.
Once a method for comparing the works had been defined, it was necessary to implement the algorithms required for the calculations. As no frameworks were available to directly support the more computationally onerous operations on Graphics Processing Unit (GPU), a customised solution was opted for using Python and Compute Unified Device Architecture (CUDA), producing tools capable of handling both high and low-level data and calculations.
In-depth analyses were conducted to produce accurate results, taking into account the physical and temporal limitations of computational resources. Each step was carefully examined, testing various techniques and parameters to optimise the process.
This thesis introduces clustering FCM as a generalisation of the work carried out in [5], demonstrating its effective implementation and application, and laying the groundwork for further developments in the automatic comparison of graphical works.

In summary, in [5], image analysis consists of three main steps:

- **Pre-processing**: Images are transformed into matrices of black and white pixels.
- **Synthesis**: The pixel matrices are converted into a list of tiles with their respective occurrences.
- **Comparison**: The formula defined in [1] is used to compare works.

In this thesis, the three phases have been redefined as follows:

- **Pre-processing**: The images are converted to greyscale with a specific resolution (PPI) and then cleaned of pollutants by cutting and removing squares.
- **Synthesis**: An ordered list of tiles with their respective occurrences is no longer generated, but only an unordered list of tiles with repetitions.
- **Comparison**: A new comparison formula is developed that dynamically discretizes the space of tiles by clustering.

This paper illustrates the main sources in the Literature Review section, providing the intuitive basis for applying them to the context at hand. The n-gram model of Martin [6] and its use in [1] and [5] will be introduced. Furthermore, clustering concepts and related algorithms, including KMeans, FCM and Gaussian Mixture Models (GMM), will be discussed and compared. Finally, the algorithm Fast Fourier Transform (FFT) for removing squares in images will be presented.

After having introduced the fundamental concepts, these will be explored in detail from both an implementation and theoretical point of view in the Methodology chapter. Here, the new comparison formula will be shown and design choices will be evaluated by means of synthetic examples.
Once the code has been developed and the theoretical foundations clarified, the direct application on the dataset will be described in the chapter Results, where the parameters will be refined and detailed qualitative results given.

Chapter 2

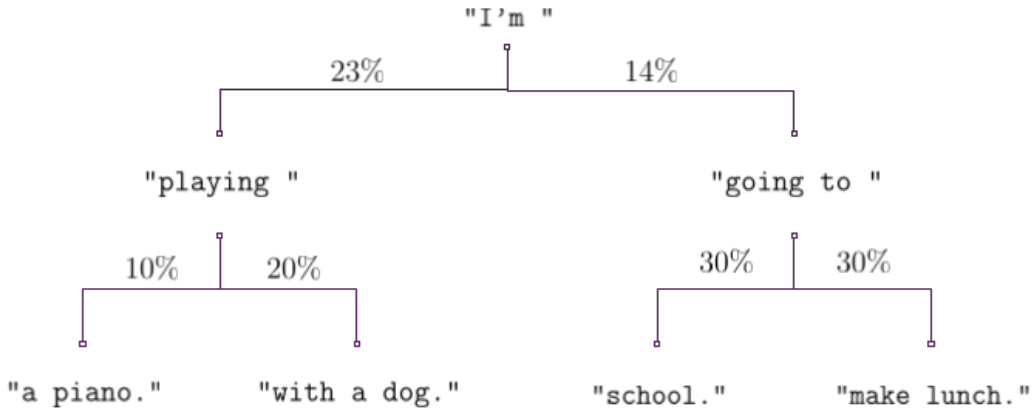
Literature Review

This chapter reviews key literature that forms the theoretical foundation of this research. Specifically, the n-gram language model, fuzzy clustering, and Fourier transforms will be discussed. Additionally, specific details of these topics will be examined to demonstrate their practical applications within this project.
Relevant definitions and properties needed to understand these applications will be introduced, alongside examples and comparisons to provide a more comprehensive and nuanced understanding of the theory.

2.1 n-gram language model

The n-gram language model analyzes a sequence of symbols by examining sets of frames, known as n-grams. This model is designed to construct a natural language model based on the assumption that each new symbol is statistically influenced by the preceding symbols. For example, the phrase '`I am playing` ' could be followed by '`a piano`' or '`with a dog`'. The n-gram model assumes that the continuation of this phrase depends solely on a finite window of preceding words or characters, and assigns '`a piano`' a certain probability of being the most natural continuation. However, as language modeling techniques have advanced, the n-gram model has largely been replaced by more sophisticated models, such as transformers[1].
The n-gram language model, first introduced by Shannon and described in Martin [6], was later applied in Basile et al. [1] for the purpose of author attribution. In this case, the goal was not to generate natural text but to identify the author of a given text. The research found that the best performance was achieved using 8-grams, and attribution was done by comparing each known author's work with the target text.
An additional development of this model extended its use to a completely different field: images. In the research thesis [5], this attribution method was employed to determine the authorship of a manuscript by treating the grams as square pixel tiles.

[1] For more information on transformers, see https://en.wikipedia.org/wiki/Transformer_(deep_learning_architecture)



2.1.1 Insights

As previously mentioned, the n-gram model predicts the next characters or words to be placed based on the preceding ones. For example, if the focus is on sequences of n printable characters, the model is trained on a dataset of text files, empirically deriving the probabilities $\mathbb{P}[w_n|w_1,\dots,w_{n-1}]$, where w_i represents the i-th character of the n-gram.
In this way, a machine can construct a sentence from a few characters: $w_1,\dots,w_{n-1}$. The algorithm iteratively selects the most probable next character w_n given the current window of $n-1$ characters, then updates the window to $w_2,\dots,w_n$ to predict w_{n+1}, and so on. This step-by-step process resembles a Markov chain, where each state corresponds to the current sequence of characters, and transitions between states occur with probabilities derived from the dataset.
The model assumes that the next character depends only by the last $n-1$ characters so:

$$\mathbb{P}[w_{n+k}|w_k\dots w_{n+k-1}] = \mathbb{P}[w_{n+k}|w_{k+1}\dots w_{n+k-1}] \quad \forall k \tag{2.1}$$

From this assumption we derive the following property:

Proposition 2.1.1. *Let $w_1,\dots,w_{n-1},w_n\dots,w_{n+k}$ be a sequence of characters, where n represents the size of the n-gram window used by the model. Specifically, the model assumes that the probability of the n-th character depends only on the preceding $n-1$ characters for all n-grams. We have that:*

$$\mathbb{P}[w_n,\dots,w_{n+k}|w_1,\dots,w_{n-1}] = \prod_{i=0}^{k}\mathbb{P}[w_{n+i}|w_{i+1}\dots w_{n+i-1}] \tag{2.2}$$

Proof. This property is proved by using Bayes' theorem and the assumption of the model eq. (2.1):

$$\begin{aligned}\mathbb{P}[w_n,\dots,w_{n+k}|w_1,\dots,w_{n-1}] &= \prod_{i=0}^{k}\mathbb{P}[w_{n+i}|w_1\dots w_{n+i-1}] \\ &= \prod_{i=0}^{k}\mathbb{P}[w_{n+i}|w_{i+1}\dots w_{n+i-1}]\end{aligned}$$

□

The model is built on the assumption that considering n-grams larger than n is unnecessary. However, this limitation can introduce significant weaknesses in natural language processing, as it ignores information outside the n-character window. For instance, in a mystery novel, understanding the entire plot and its intricate details is crucial, something the model might overlook. Despite this, the n-gram model can still be quite effective in simpler contexts like everyday conversation. For example, after the input '`Hi! How are `', the model might predict '`you?`' as a natural continuation.
As mentioned earlier, this model was repurposed for a different goal by Basile et al. [1]. While the n-gram model might struggle to generate fully coherent natural language without encountering semantic inconsistencies or syntactic errors, it remains useful for capturing an author's writing style. This is achieved by training the model exclusively on texts by that author. The approach is to extract all n-grams from the target work and compare their distribution to that of known works by other authors. The proposed formula for comparing work A with work B is as follows:

$$d(A,B) = \frac{1}{|D_A| + |D_B|} \sum_x \left(\frac{f_A(x) - f_B(x)}{f_A(x) + f_B(x)} \right)^2 \tag{2.3}$$

where $f_A(x)$ represents the frequency of the n-gram x in the work A and $|D_A|$ is the variety of observed n-grams.
As highlighted in [5], two relevant properties of this method of comparing distributions can be mentioned:

- Each addendum of the summation takes on a value between 0 and 1.
- The external factor not only normalises the result so that it remains between 0 and 1, but also calls up the Jaccard index[2], improving the arithmetic mean of the summation:

  $$\frac{1}{|D_A| + |D_B|} = (1 + J_{D_A,D_B})^{-1} \frac{1}{|D_A \cup D_B|}$$

  where $J_{D_A,D_B} := \frac{|D_A \cap D_B|}{|D_A \cup D_B|}$.

  In this way, we can see the comparison formula as a product between two types of index:

  $$d(A,B) = (1 + J_{D_A,D_B})^{-1} \times \frac{1}{|D_A \cup D_B|} \sum_{x \in D_A \cup D_B} \left(\frac{f_A(x) - f_B(x)}{f_A(x) + f_B(x)} \right)^2$$

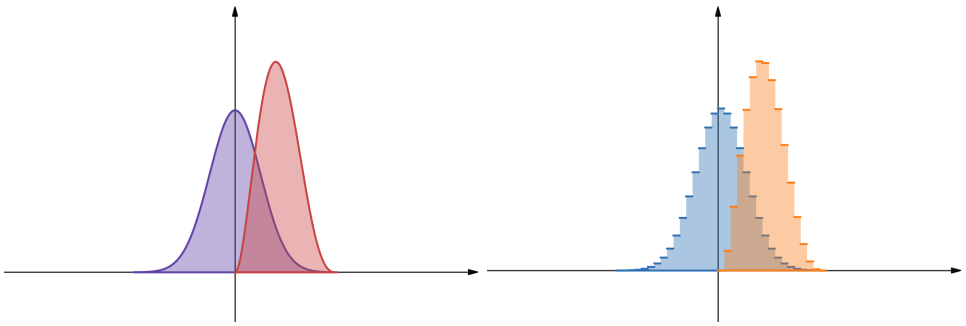
2.1.2 Implications

We conclude this section by discussing the implications of this theory. As seen in eq. (2.1), no distinction is made between one n-gram and another. This is because n-grams of length 3, or slightly longer, are typically used, which are insufficient to cover a full word. However, when using 7-grams, it becomes possible to consider the relationships between synonyms and distances between n-grams. For instance,

[2] For more about Jaccard index, see https://en.wikipedia.org/wiki/Jaccard_index

we would expect some level of correlation between '`paper`' and '`article`' as they are synonyms.
In written texts, this rarely impacts the outcome, as there are relatively few synonyms for each term within the full vocabulary, and especially few long chains of synonyms. For example, it is unlikely that many synonyms for '`paper`' begin with the string '`p`' or '`pe`'.
As noted in [5], the research deliberately set appropriate image *resolution* and *posterization* to avoid discussions about the similarity between tiles. By controlling these parameters, the focus remained on the attribution process rather than getting lost in complex considerations of tile 'synonyms' and their potential concatenations.

We further examine the properties of the comparison formula defined in eq. (2.3), particularly focusing on the problem of sparsity in n-grams.
Let N represent the samples drawn from two absolutely continuous distributions $\mathcal{A}$ and $\mathcal{B}$ defined on $\mathbb{R}$. We partition $\mathbb{R}$ into intervals (boxes) of amplitude $1 \,/\, C$, thus transforming the original distributions into their discretized versions, $\mathcal{A}(C)$ and $\mathcal{B}(C)$, along with their respective samples.



Now, let's consider eq. (2.3) and analyze the behavior of the estimated distance as the number of data points and the size of the boxes vary. By fixing the number of data points and decreasing the size of the boxes, the estimated distance gradually approaches 1. This phenomenon occurs because there exists a critical dimension for $1 \,/\, C$, such that with a sufficiently high probability, each sample becomes isolated within its own box. Consequently, this drives J_{D_A,D_B} toward 0, making each term in the summation equal to 1.

In [5], the continuous nature of the n-tiles was addressed by discretizing the continuous space in which they were defined, fixing a resolution and applying posterization. Posterization is a process that reduces the number of discrete levels of color or intensity in an image, effectively mapping a continuous range of values into a smaller set of intervals. For example, in a color image, subtle variations in shades of red might all be grouped into a single pure red, removing the nuance of intermediate shades. However, this approach results in a significant loss of information. Therefore, this paper will explore a method to achieve a cleaner and more dynamic discretization.

2.2 Fuzzy clustering

Fuzzy clustering, also known as soft KMeans, is a data analysis technique that allows data points to be assigned to more than one cluster. To understand this, it is important to define the concepts of a cluster, a centroid, and the distinction between hard and soft clustering. A cluster is a group of data points that share similar characteristics or are close to one another in a defined feature space. For example, in a two-dimensional space, a cluster might represent a dense grouping of points around a specific region. A centroid is the representative point of a cluster, often located at its geometric center or computed as the mean of all the points within the cluster.
In hard clustering, each data point is assigned exclusively to a single cluster, resulting in binary membership (e.g., a data point either belongs to a cluster or it does not). A common example is traditional KMeans clustering. In contrast, soft clustering allows data points to have degrees of membership in multiple clusters. Instead of binary membership (0 or 1), fuzzy clustering employs a continuous measure of membership that ranges from 0 to 1. This is represented by a similarity function, $\mu_x(C)$, which quantifies the degree to which a data point x belongs to a cluster C.
Fuzzy clustering originated from the work of Dunn, who introduced this concept as an extension of the KMeans clustering algorithm in 1973. Dunn emphasized the increased accuracy and robustness of fuzzy clustering compared to hard clustering, particularly in handling anomalous data (outliers) [3].
The robustness of fuzzy clustering comes from its use of continuous membership degrees, which allows data points to partially belong to multiple clusters. This reduces the impact of outliers by assigning them smaller membership values across clusters, diluting their influence. In addition, fuzzy clustering better models overlapping clusters by capturing the uncertainty in their boundaries, a common feature in real-world data.
The algorithm proposed by [3] is known as the FCM and employs the Expectation-Maximisation (EM) technique to define the cluster membership function. This method is an iterative approach for estimating statistical parameters, such as cluster centroids. The EM process consists of two main steps:

1. Expectation (E): In this phase, a likelihood function is formulated. This function expresses the probability that a data point belongs to each cluster, based on its distance or similarity to the centroids.

2. Maximisation (M): In the maximization phase, an optimization procedure is used to identify the model parameters that maximize the likelihood, specifically the centroids of the clusters.

In summary, fuzzy clustering provides greater flexibility compared to hard clustering, allowing for a more accurate and detailed representation of the data, particularly in the presence of heterogeneous data or outliers.

2.2.1 Insights

To fully understand the FCM algorithm, we must first define the fundamental concepts on which it is based.

Data points: These are the observations or entities in our dataset, each characterized by a set of attributes or features. Data points can be viewed as points in a multidimensional space, where each dimension corresponds to a specific attribute. To represent these points mathematically, each attribute is assigned a numerical value, resulting in a vector in $\mathbb{R}^d$, where d is the number of attributes. For example, if an observation is described by three attributes height, weight and age, it can be represented as a vector $(h, w, a) \in \mathbb{R}^3$, where h,w,a are numerical values corresponding to these attributes. This transformation from abstract attribute spaces to real-valued vectors enables the application of mathematical and computational techniques, such as clustering.

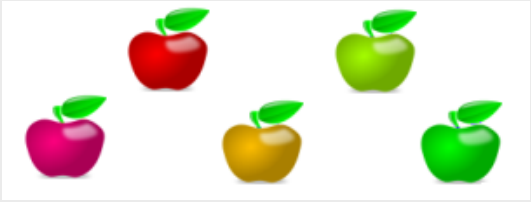
Clusters: Once we have defined the data points, we can organize them into groups called clusters. A cluster is a collection of data points that share similar characteristics. The goal of the clustering algorithm is to group data points so that those within the same cluster are more similar to each other than to those in different clusters.

Cluster Membership Function: Each data point can be associated with one or more clusters through the cluster membership function, denoted as μ_x, where x represents a specific data point and C represents a cluster. This function assigns each data point a degree of membership in each cluster, represented by a value between 0 and 1. For example, if $\mu_x(C) = 1$, it indicates that data point x belongs completely to cluster C. Conversely, if $\mu_x(C) = 0.5$, it suggests that x has some degree of uncertainty or ambiguity in its membership to cluster C.

Notation. Denote:

- the dataset with
  $\mathcal{S} = (x_i)_{i=1:N}$ where $x_i \in \mathbb{R}^K$
- the set of cluster centroids with
  $\mathcal{C} = C_1, \ldots, C_M$ where $C_j \in \mathbb{R}^K$
- the measure of the membership of a data point x_i in the cluster with centroids C_j with
  $\mu_{x_i}(C_j) \in [0, 1]$
- the weight matrix with
  $U = (u_{ij})$ where $u_{ij} = \mu_{x_i}(C_j)$
- the quadratic Euclidean distance matrix with
  $D^2 = (d_{ij}^2)$ where $d_{ij}^2 = \|x_i - C_j\|^2$

Exempli Gratia. To understand the meaning of data points and clusters, let us consider the following example:
Imagine we need to classify the apples transported by a lorry into two color categories: green and red. However, within this set, there might also be yellow apples. The k-means algorithm would treat a yellow apple as an anomaly compared to the red and green apples. In contrast, fuzzy clustering quantifies this anomaly by asserting that a yellow apple is a blend of red and green.
The colours of the apples can be represented in a one-dimensional space. In this space, we observe a distribution of data points showing green and red at the extremes, with yellow apples located in the center.
Clusters, in the context of fuzzy logic, are sets defined by centroids representing specific shades of colour, which are not necessarily present in the set of data points. For example, if there are red and green apples, there are actually two centroids representing specific shades of red and green, even though there are no apples with such colour shades.



Remark. The FCM algorithm recognizes that real-world data may contain errors or uncertainties. Instead of rigidly assigning each data point to a single cluster, it introduces a measure of uncertainty in the assignment. This approach results in assigning a fuzzy degree of membership to each point concerning each cluster, rather than relying on a binary classification.

Definition 2.2.1 (membership measure)**.** Given a point x and a cluster C, the membership measure $\mu_x(C)$ is a probability distribution proportional to the inverse of the square of the Euclidean distance between the point x and the centroid of the cluster C, i.e. $\frac{1}{\|x-C\|^2}$.
In other words:

$$\forall x \left(\sum_{C \in \mathcal{C}} \mu_x(C) = 1 \right) \quad \text{and} \quad \forall x \, \exists \lambda_x \, \forall C \left(\mu_x(C) = \frac{\lambda_x}{\|x - C\|^2} \right)$$

Consequently:

$$\forall x \, \exists \lambda_x \left(1 = \sum_{C \in \mathcal{C}} \mu_x(C) = \sum_{C \in \mathcal{C}} \frac{\lambda_x}{\|x - C\|^2} = \lambda_x \sum_{C \in \mathcal{C}} \frac{1}{\|x - C\|^2} \right)$$

So for all x it holds that $\lambda_x = \frac{1}{\sum_{C \in \mathcal{C}} \frac{1}{\|x-C\|^2}}$

Finally:

$$\begin{aligned} u_{ij} = \mu_{x_i}(C_j) = \frac{\lambda_{x_i}}{\|x_i - C_j\|^2} &= \frac{\frac{1}{\sum_k \frac{1}{\|x_i - C_k\|^2}}}{\|x_i - C_j\|^2} \\ = \frac{\frac{1}{\|x_i - C_j\|^2}}{\sum_k \frac{1}{\|x_i - C_k\|^2}} = \frac{\frac{1}{d_{ij}^2}}{\sum_k \frac{1}{d_{ik}^2}} &= \frac{1}{\sum_k \frac{d_{ij}^2}{d_{ik}^2}} \end{aligned}$$

The objective function As already mentioned, the algorithm FCM can be seen as a derivation of KMeans. To better understand this step, we will explain what KMeans is.
The KMeans algorithm is one of the most widely used clustering methods in data analysis. It divides a data set into a predefined number M of clusters, each represented by a centroid. The algorithm iteratively refines the cluster assignments and centroids to minimise the intra-cluster variance. In other words, the goal of the KMeans algorithm is to minimise the sum of the variances of all clusters, which is defined as the sum of the squared distances between each point in the cluster and the cluster centroid.
Formally, the goal of the KMeans algorithm is to solve this problem over $\mathcal{C} = \{C_1, \ldots, C_M\}$:

$$\text{argmin}_{\mathcal{C}} \left[\sum_{C \in \mathcal{C}} \sum_{x \in C} \|x - C\|^2 \right]$$

where with $x \in C$ we means that x is in the cluster of the centroid C.
The KMeans algorithm assigns each data point to a single cluster C_j in a "hard" way, meaning that each point belongs exclusively to one cluster. The FCM algorithm generalises this approach by replacing the binary assignment of points with a continuous membership function, $\mu_x(C)$, which quantifies the degree to which a data point belongs to each cluster.

In the case of fuzzy clustering, the membership of a point in a cluster is expressed by a fuzzy measure instead of a Boolean logic. The objective function of the KMeans method can be rewritten equivalently with membership measures:

$$\sum_{C\in\mathcal{C}}\sum_{x\in\mathcal{S}}\mathcal{X}_C(x)\|x-C\|^2$$

where $\mathcal{X}_C(x)$ represents the membership measure of point x in cluster C (1 if $x \in C$, 0 otherwise). This objective function reflects the weighting of the squared distances by the membership measure of each point in the clusters.

Definition 2.2.2 (Loss function and cluster weight)**.** We define the fuzzy clustering loss function from the KMeans algorithm as follows:

$$L_{\mathcal{S}}(\mathcal{C}) := \sum_{C\in\mathcal{C}} p_{\mathcal{S}}(C) := \sum_{C\in\mathcal{C}}\sum_{x\in\mathcal{S}}\mu_x(C)^2\|x-C\|^2 \tag{2.4}$$

where $p_{\mathcal{S}}(C)$ is the weighted sum of the squared distances between the points in the cluster C and its centroid. This loss function is coercive on the centroids, which means that it tends to infinity when the centroids recede to infinity, and has a finite lower bound when the centroids are bound to a compact set in the data space. Formally, for any norm defined over $\vec{\mathcal{C}}$, a vector representation of $\mathcal{C}$ with $M \times K$ components, it hold that:

$$\lim_{\|\vec{\mathcal{C}}\|\to\infty} L_{\mathcal{S}}(\mathcal{C}) = \infty$$

and

$$L_{\mathcal{S}}(\mathcal{C}) \geq 0$$

For this reason we know that there is a global minimum although the uniqueness of local minima is not guaranteed.[a]

[a] for more about coercitive functions, see
https://www1.mat.uniroma1.it/people/lanzara/OTT0708/Ottimizzazione0708.pdf

EM As previously mentioned, the KMeans and FCM algorithms can be seen as applications of the EM technique. Specifically, in the case of the KMeans algorithm, the procedure can be summarized as follows:

- **E-step (Expectation)**: Each data point is assigned to the nearest centroid. For example, a yellow apple would be assigned to a red apple rather than a green apple. This corresponds to a "hard" assignment, where each data point is exclusively linked to one cluster. This step defines the membership measure and the corresponding optimization problem:
  $\text{argmin}_{\mathcal{C}}\left[\sum_{C\in\mathcal{C}}\sum_{x\in C}\|x-C\|^2\right]$
- **M-step (Maximization)**: With fixed memberships, the centroids are recomputed by solving the optimization problem. Specifically, it can be shown that:
  $C = \text{mean}\,[x|x \in C]$

Both steps alternate iteratively: the **E-step** minimises the loss function $\sum_{C\in\mathcal{C}}\sum_{x\in C}\|x-C\|^2$ for fixed centroids, and the **M-step** minimises the same loss function for fixed memberships. Thus, the KMeans algorithm minimises the loss function through an alternating optimization approach.
The FCM algorithm is similar:

- **E-step (Expectation)**: The membership matrix U is computed as defined in definition 2.2.1, assigning each data point a degree of membership between 0 and 1. This corresponds to a "soft" assignment, where each point can belong to multiple clusters with varying degrees of membership. This step defines the membership measure and the corresponding optimization problem: $\text{argmin}_{\mathcal{C}}\left[\sum_{C\in\mathcal{C}}\sum_{x\in\mathcal{S}}\mu_x(C)^2\|x-C\|^2\right]$
- **M-step (Maximization)**: With fixed memberships, the centroids are recomputed by solving the optimization problem to minimize the objective function.

Now we compute the solution of objective function of FCM algorithm during '**M-step**' phase.

Theorem 2.1 (*Update formula 'M'*)

Given the matrix U, the loss function of eq. (2.4) reaches its minimum when the centroids are given by:

$$C_j^{\texttt{new}} = \frac{\sum_{i=1}^N u_{ij}^2 x_i}{\sum_{i=1}^N u_{ij}^2} \qquad \forall j \tag{2.5}$$

Proof. Given the matrix U, the loss function is differentiable, convex and coercive. Consequently, to find the global minimum of the function, it is sufficient to find the points where the gradient is null.
Calculating the partial derivatives of the gradient with respect to the centroids C_j, we obtain:

$$\frac{\partial}{\partial C_j}L_{\mathcal{S}}(\mathcal{C}) = \frac{\partial}{\partial C_j}p_{\mathcal{S}}(C_j) = \sum_{i=1}^N \frac{\partial}{\partial C_j}u_{ij}^2\|x_i - C_j\|^2 = 2\sum_{i=1}^N u_{ij}^2\,(C_j - x_i)$$

The gradient is null when all partial derivatives are null, which implies:

$$\sum_{i=1}^N u_{ij}^2 C_j^{\text{new}} = \sum_{i=1}^N u_{ij}^2 x_i \quad \forall j$$

Since C_j^{new} does not depend on the index i, the thesis is proved. □

Remark. As noted above, the KMeans uses the EM technique to minimize an objective function by choosing centroids and memberships optimally. Similarly, FCM also adopts this logic, but replaces hard assignments with soft assignments. This is formalized in the following theorem.

Theorem 2.2 (*Update formula 'E'*)

*Let $\mu^\star$ be the solution of the optimization problem in the '**E-step**' for fixed centroids $\mathcal{C}$:*

$$\operatorname{argmin}_\mu \left[\sum_{C\in\mathcal{C}} \sum_{x\in\mathcal{S}} \mu_x(C)^2 \|x - C\|^2 \right].$$

Updating the membership matrix U provides the optimal solution:

$$u_{ij} = \mu^\star_{x_i}(C_j).$$

Proof. Since the membership degrees $\mu_x(C)$ are independent for different clusters $C \in \mathcal{C}$, the optimization problem can be decoupled into independent subproblems for each cluster.

$$\operatorname{argmin}_\mu \left[\sum_{C\in\mathcal{C}^\star} \sum_{x\in\mathcal{S}} \mu_x(C)^2 \|x - C\|^2 \right] = \sum_{C\in\mathcal{C}} \operatorname{argmin}_\mu \left[\sum_{x\in\mathcal{S}} \mu_x(C)^2 \|x - C\|^2 \right]$$

The optimization problem $\operatorname{argmin}_\mu \left[\sum_{x\in\mathcal{S}} \mu_x(C)^2 \|x - C\|^2\right]$ is a convex quadratic problem (QP) because the loss function is quadratic, and the feasible region is a convex set (probability simplex). The solution can thus be found using the Karush–Kuhn–Tucker (KKT) conditions.

In general, these kinds of problems are solved mechanically through the conditions KKT, stated below:
Consider the following nonlinear optimisation problem in standard form:

$$\operatorname{argmin}_x f(x)$$

where $x \in X$, $g_i(x) \leq 0$, $h_j(x) = 0$, with X a convex subset of $\mathbb{R}^n$, g_i and h_j differentiable functions with $i = 1, \dots m$ and $j = 1, \dots, l$.
Suppose that $x^\star \in \mathbb{R}^n$ is a local optimum, then there exists constants η_i and λ_j with $i = 1, \dots m$ and $j = 1, \dots, l$ such that:

$$\begin{aligned}
&\partial f(x^\star) + \sum_j \lambda_j \partial h_j(x^\star) + \sum_i \eta_i \partial g_i(x^\star) = 0 \\
&h_j(x^\star) = 0 \ \forall j \\
&g_i(x^\star) \leq 0 \ \forall i \\
&\eta_i \geq 0 \ \forall i \\
&\sum_i \eta_i g_i(x^\star) = 0
\end{aligned}$$

In this context:

- X is the set of probabilities, that is, the values that $(\mu_x(C))_C$ can have
- $g_i((\mu_x(C))_C) = -\mu_x(C_j)$ and $m = M$
- $h_1((\mu_x(C))_C) = 1 - \sum_j \mu_x(C_j)$ and $l = 1$
- $f((\mu_x(C))_C) = \sum_{x\in\mathcal{S}} \mu_x(C)^2 \|x - C\|^2$

The local optimal solution, if it exists, will have to meet the conditions KKT for that, substituting and partial derivatives gives:

$$
\begin{aligned}
&2\mu_x(C_j)\left\|x - C_j\right\|^2 - \lambda_1 - \eta_j = 0 \quad \forall j \\
&\sum_j \mu_{x_i}(C_j) = 1 \ \forall j \\
&\mu_x(C_j) \geq 0 \ \forall i \\
&\eta_i \geq 0 \ \forall i \\
&\sum_i \eta_i \mu_x(C_j) = 0
\end{aligned}
$$

From this conclusion, we observe that, for any j, it can be $\eta_j = 0$ since it must be non-negative and the linear combination with $(\mu_x(C))_C$ (non-negative values) must be 0. Thus:

$$\mu_x(C_j) = \frac{\lambda_1/2}{\|x - C\|^2}$$

And following the demonstration in definition 2.2.1, we have that:
$\mu_x(C) = \frac{1/\|x-C_j\|^2}{\sum_k 1/\|x-C_k\|^2}$

We obtain an admissible solution that satisfies all conditions and is therefore a local optimal solution. However, since this is a convex quadratic problem, we have that the local minimal solution is also optimal. □

Probabilistic perspective As mentioned above, the '**E-step**' is a phase designed to construct a likelihood function, which then during the '**M-step**' is maximised. Can a probabilistic perspective be given to the algorithm FCM?
The calculation of the centroid update formula in fuzzy clustering is closely tied to the concept of parameter estimation through the maximum likelihood method. Specifically, it involves maximizing the joint density of the data and model parameters, conditional on the assignment of points to clusters. The density to be maximized is given by:

$$f_U(\mathcal{S}|\mathcal{C}) \propto \prod_{i,j} \exp\left[-u_{ij}^2 \frac{\|x_i - C_j\|^2}{2}\right]$$

where $f_U(\mathcal{S}|\mathcal{C})$ is the joint density of the points conditioned on the set of centroids where u_{ij} is the membership measure of point x_i in cluster C_j.
The density of the single sample x_i is:

$$f_U^i(x_i|\mathcal{C}) \propto \prod_{j} \exp\left[-u_{ij}^2 \frac{\|x_i - C_j\|^2}{2}\right]$$

where $f_U^i(x_i|\mathcal{C})$ is the density of the observed datum x_i conditioned on the centroids. The goal is to maximise the likelihood density in order to obtain accurate estimates of the model parameters, which in the context of fuzzy clustering are represented by

the cluster centroids. For this reason, the algorithm is classified as EM because the phase '**E**' sets the densities for maximum likelihood when the matrix U is defined and the phase '**M**' solves the problem by finding the centroids that best explain the model.

The algorithm 1 computes the optimal centroids using a defined stopping criterion. In the case of KMeans, a robust stopping criterion is when the **E-step** no longer updates the memberships, as this implies that the centroids will also stop changing. For FCM, however, defining a good stopping criterion can be more challenging due to the continuous nature of the memberships. Common criteria include a fixed number of iterations, monitoring the update value of the loss function, or tracking the norm of the centroid updates. In this paper, we adopt the latter approach, as it is computationally simpler and faster to calculate than evaluating the changes in the membership matrix, while still providing a reliable indication of convergence.

Algorithm 1 Fuzzy Clustering
INPUT
• $\mathcal{S}$: set of data $x_1, \cdots, x_N$
• $\mathcal{C}$: centroids $C_1, \cdots, C_j$

function FuzzyClustering($\mathcal{S}, \mathcal{C}$, `stop`)
  while Loss update less then `stop` **do**
    $D^2 \leftarrow (d_{ij}^2)_{ij}$ with $d_{ij}^2 = \|x_i - C_j\|^2$
    $U^2 \leftarrow (u_{ij}^2)_{ij}$ with $u_{ij} = \left(\sum_k D_{ij}^2 / D_{ik}^2\right)^{-1}$ ▷ see theorem 2.2
    $L \leftarrow \sum_i \sum_j u_{ij}^2 d_{ij}^2$ ▷ see definition 2.2.2
    for $j \leftarrow 1$ to M **do** ▷ see theorem 2.1
      $C_j^{\text{new}} \leftarrow \sum_{i=1}^N x_i u_{ij}^2 / \sum_{i=1}^N u_{ij}^2$
    end for
  end while
  return $\mathcal{C}$, U
end function

2.2.2 Implications

This algorithm is particularly effective for clustering problems that involve noise. Two other well-known methods operate on different paradigms: KMeans and GMM.

KMeans is a fundamental clustering algorithm that partitions data points into disjoint groups, where the centroids represent the mean of each group. Its primary goal is to minimise the sum of variances within these groups, resulting in well-defined clusters.

$$\text{argmin}_{\mathcal{C},\mathcal{X}} \left[\sum_{C \in \mathcal{C}} \sum_{x \in \mathcal{S}} \mathcal{X}_C(x) \|x - C\|^2 \right]$$

where $\mathcal{X}_C(x)$ is 1 if x belongs to the cluster of centroid C or 0 otherwise, and each x belongs to an only one cluster.

GMM is a more sophisticated algorithm based on the assumption that the data points are generated by a specific statistical process:

1 Select with probability p_j the j-th cluster.

2 Generate a random data point from $\mathcal{N}(C_j, \Sigma_j)$.

This method not only tries to find clusters and label data points, but also proposes normal distributions with averages $\mathcal{C}$ and covariances Σ, assigning probabilities indicating the weight of each cluster. It is possible to label the data points using the Mahalanobis distance[3].

Next, we examine how these methods behave in the presence of noisy data. Specifically, 6 random centroids were chosen in a two-dimensional space. Around each centroid, data points were generated according to a Gaussian distribution, with the centroid serving as the mean and a distinct covariance matrix defining the spread. For each cluster, 20, 20, 20, 30, 30 and 40 data points were generated, respectively. In addition, uniformly generated noisy data points were added, constituting 30% of the total (i.e. 48 noisy data points). A representation of this data can be seen in fig. 2.2. As can be seen, noise can create *outliers*, i.e. distant points that would be better ignored to achieve good clustering.

We are going to see the results of different clusterisation over the same data: KMeans, GMM, FCM.

KMeans We cluster the data using KMeans with the correct number of centroids, i.e. 6. In fig. A.1 we can see that the noise has created an additional cluster and two of the original clusters have merged into one.

GMM Next, we apply clustering using a GMM with 6 centroids. In fig. A.2, it can be seen that the cluster corresponding to noise has a very low weight, with a probability of 0.07. Furthermore, the use of covariance matrices allows for a more precise determination of the centroid positions and provides additional information about the shape and spread of the clusters. This enables the identification of substructures or dependencies among clusters, which can be interpreted as a form of hierarchical organization. It is important to note that in GMM, clusters are modeled as Gaussian distributions rather than discrete entities, meaning that what we observe are formal probability distributions rather than "real" clusters with strict boundaries.
Finally, let clustering be applied with FCM using 6 centroids. In fig. A.3 we observe that the centroids are less affected by noise, making it possible to identify which data are potentially noisy or shared between several clusters.

We have observed that in the presence of noise, the algorithm FCM can be very useful due to its flexibility in handling overlapping clusters and noisy data. On the other hand, the algorithm GMM, while providing valuable insights about

[3]For more information on the Mahalanobis distance, see https://en.wikipedia.org/wiki/Mahalanobis_distance

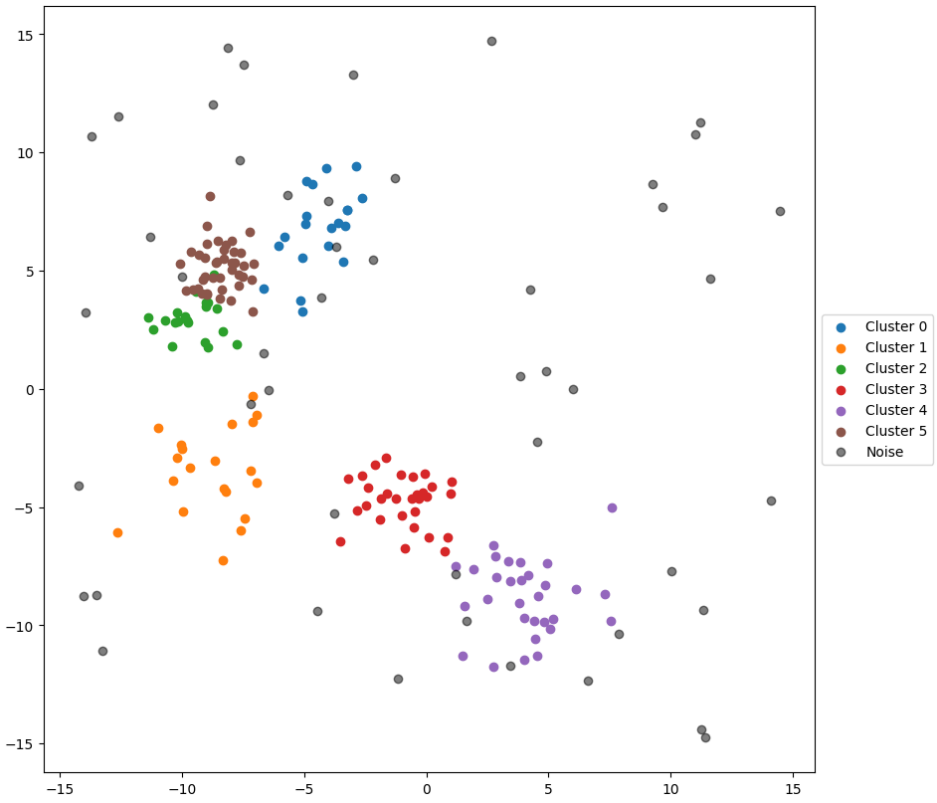


Figure 2.2. Data points are coloured according to their label, while noise is shown in grey.

the nature of the data through its probabilistic framework, is computationally expensive, especially for large datasets or high-dimensional spaces. In this thesis, we prioritize computational efficiency and scalability. Therefore, we use the FCM algorithm to reduce the amount of information to be processed, leveraging its ability to model uncertainty in cluster assignments, and the KMeans algorithm for efficient initialisation of centroids.

2.3 DFT

The DFT is a powerful tool for analyzing signals in the frequency domain, often used in applications such as signal processing, image analysis, and data compression. Its ability to transform spatial or temporal data into frequency components makes it particularly effective for isolating patterns or removing noise. In this thesis, the DFT will be used to compress and clean images during the pre-processing phase, which will be discussed in detail in chapter 3.
This procedure, known as spectral analysis, makes it possible to study signals, waves, vibrations, sounds and images. In addition to these areas, DFT also has major scientific applications, such as the precise estimation of sunspot cycles, helping to predict and control phenomena such as geomagnetic storms[4].

[4]For more about geomagnetic storms and them correlation with sunspot, see https://en.

2.3.1 Insights

The DFT is a discrete analogue of the Continue Fourier Transform (CFT), which represents a function in terms of its frequency components. More rigorously, given a function $f \in L^1(\mathbb{R})$, its Fourier transform is a functional operator, denoted by $\mathcal{F}$, which associates f with its frequency representation:

$$\mathcal{F}(f) : \omega \mapsto \int_{\mathbb{R}} e^{-i2\pi\omega x} f(x)\, dx$$

By contrast, the Fourier series applies specifically to periodic functions, representing them as an infinite sum of sine and cosine waves with discrete frequencies. In this context, the DFT can be viewed as a finite approximation of the Fourier series, applied to sampled data over a finite interval.
The **Fourier inversion theorem** states that, if both f and $\mathcal{F}(f)$ belong to $L^1(\mathbb{R})$ (i.e. they are both integrable), then for almost any $x \in \mathbb{R}$ it is possible to recover $f(x)$ via the following inverse relation:

$$f(x) = \int_{\mathbb{R}} \mathcal{F}\,(f)(\omega) e^{i2\pi x\omega}\, d\omega$$

In other words, the Fourier transform and its inverse allow switching back and forth between the time (or space) and frequency domains.

The discrete version of this transform, namely DFT, is used to analyse signals sampled at regular intervals. Thus, while CFT works on continuous signals, DFT applies to finite and discrete signals, making it suitable for digital signal processing. To better understand how DFT gives information about the amplitudes and phases of the different frequencies that make up a sequence, it is useful to introduce the Fourier coefficients. These coefficients make it possible to decompose a data sequence into sinusoids associated with different frequencies.

Remark. The Fourier coefficients of the sequence $(x_n)_{n=0}^{N-1}$ is a sequence of complex numbers $(X_k)_{k=0}^{N-1}$ such that:

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{i2\pi \frac{k}{N} n} \tag{2.6}$$

In the follow examples, we want show an intuitive relation between Fourier coefficients and them time series. It is very important to interpretate a result of a DFT.

`wikipedia.org/wiki/Geomagnetic_storm`

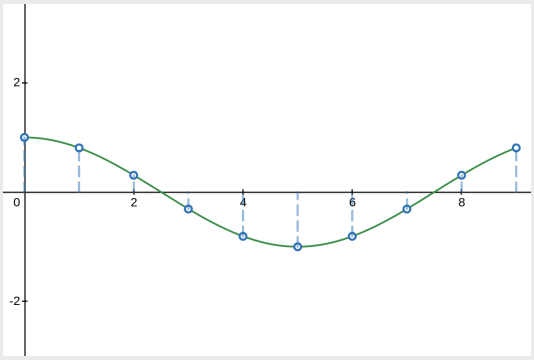
Exempli Gratia (Amplitude). In this example we understand how the Fourier coefficients determine the amplitude of a frequency.
Take a time series consisting of N complex numbers, described by the following coefficients:

$$X_0 = 0,\ X_1 = A,\ X_2 = 0\ \cdots\ X_{N-2} = 0,\ X_{N-1} = A$$

From these coefficients we obtain the time series:

$$x_n = \frac{1}{N} A \left(e^{i2\pi \frac{n}{N}} + e^{i2\pi \frac{n}{N}(N-1)} \right) = \frac{2A}{N} \cos\left(2\pi \frac{n}{N} \right)$$

We note that the series is only a sinusoidal function, with amplitude $\frac{2A}{N}$ and frequency $\frac{2\pi}{N}$.



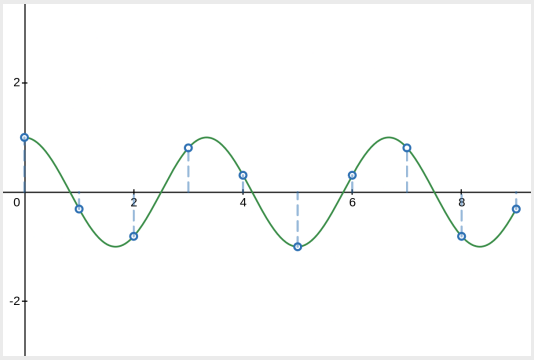
Exempli Gratia (Frequency). In the previous example the significant frequency is $2\pi\frac{n}{N}$, in this example we study how to refer to a different frequency.
Take a time series consisting of N complex numbers, described by the following coefficients:

$$X_p = A,\ X_{N-p} = A$$

where all other values are zero.
The generated time series will be:

$$x_n = \frac{1}{N} A \left(e^{i2\pi\frac{n}{N}p} + e^{i2\pi\frac{n}{N}(N-p)} \right) = \frac{2A}{N} \cos\left(2\pi\frac{n}{N}p\right)$$

As can be seen, the coefficients refer to the amplitude of a certain frequency indicated by the index of Fourier coefficient.



If we have $X_1 = A, X_{N-1} = A$, $X_2 = B, X_{N-2} = B$, then we will obtain time series:

$$x_n = \frac{2A}{N} \cos\left(2\pi\frac{n}{N}\right) + \frac{2B}{N} \cos\left(2\pi\frac{n}{N}2\right)$$

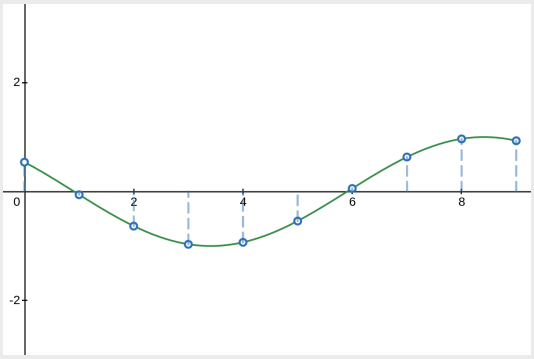
Exempli Gratia (Phase). In the previous example we observed that the index of the Fourier coefficient refers to a specific frequency. Now we are going to see how the Fourier coefficients derive the phase of a frequency in a time series.
Take a time series consisting of N complex numbers, described by the following coefficients:

$$X_0 = 0,\ X_1 = Ae^{i\theta},\ X_2 = 0,\ \dots,\ X_{N-2} = 0,\ X_{N-1} = Ae^{-i\theta}$$

The resulting time series will be:

$$x_n = \frac{2A}{N}\cos\left(\frac{2\pi}{N}n + \theta\right)$$

Here, the modulus of the coefficient describes the amplitude of the frequency, while its argument θ describes its phase.



We observe also that to ensure that the series is real, the opposite coefficient must be the conjunct of X_1.

We now ask ourselves whether it is possible to estimate the Fourier coefficients from a time series.

Proposition 2.3.1. *The matrix* $\frac{1}{\sqrt{N}}\left(e^{i2\pi\frac{k}{N}n}\right)_{n,k=0}^{N-1}$ *is unitary.*

Proof. We prove the thesis by studying the inner product between two rows of the matrix:

$$\frac{1}{N}\sum_{k=0}^{N-1} e^{i2\pi\frac{k}{N}n}\cdot e^{-i2\pi\frac{k}{N}m} = \frac{1}{N}\sum_{k=0}^{N-1} e^{i2\pi\frac{k}{N}(n-m)}$$

If $n = m$, the result of the sum is 1. Else if $n \neq m$, we obtain:

$$\frac{1}{N}\sum_{k=0}^{N-1} e^{i2\pi\frac{k}{N}(n-m)} = \frac{1}{N}\frac{e^{i2\pi\frac{N}{N}(n-m)} - 1}{e^{i2\pi\frac{1}{N}(n-m)} - 1} = 0$$

This proves that the matrix is unitary. □

Proving that the matrix $\frac{1}{\sqrt{N}}(e^{i2\pi\frac{k}{N}n})_{n,k=0}^{N-1}$ is unitary is extremely relevant to the computation of Fourier coefficients.

Theorem 2.3 (*DFT*)

The Fourier coefficients of the time series $(x_n)_{n=0}^{N-1}$ *are given by the formula:*

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi \frac{n}{N} k} \tag{2.7}$$

Proof. Define U as matrix $\frac{1}{\sqrt{N}}(e^{i2\pi \frac{k}{N} n})_{n,k=0}^{N-1}$
By definition of the discrete Fourier transform, we can write $\vec{X} = \frac{1}{\sqrt{N}} U\vec{x}$, where $\vec{X}$ is the vector of Fourier coefficients and $\vec{x}$ is the vector of the original time series.
As shown in proposition 2.3.1, the matrix U is unitary, so it admits an inverse, which is its transposed conjugate U^H.
Therefore, we can invert the transformation and obtain:

$$\vec{x} = \sqrt{N} U^H \vec{X}$$

This relation allows us to get the Fourier coefficients from the time series. □

Definition 2.3.1 (Fourier coefficient)**.** We define Fourier coefficient of the time series $(x_n)_{n=0}^{N-1}$ is the sequence of complex numbers $(X_k)_{k=0}^{N-1}$:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi \frac{n}{N} k}$$

2.3.2 Implications

The possibility of calculating Fourier coefficients directly by means of a simple matrix-vector product not only makes the analysis very informative, but also easily achievable.

FFT The computational cost of the matrix-vector product for computing Fourier coefficients is $O(N^2)$, which is not excessive in itself. However, there are better algorithms that compute Fourier coefficients and that reduce this cost. An example is chirp Z-transform (CZT) algorithm of Rabiner et al. [8] with computational cost $O(N \log N)$. An other example is the algorithm of Cooley J. W. [2] showed in algorithm 2, which use a technique of *divide et impera* to compute Fourier coefficients.

Remark. The algorithm 2 uses an important property of FFT: when $N = N_1 \times N_2$ with $N_1, N_2 > 1$ we can *divide* the input sequence, *impera* over them and, finally, merge results to obtain the desiderate Fourier coefficients.
Let $x = (x_i)_{i=0}^{N-1}$ a time series, we compute these Fourier coefficients:

- X^0 are the Fourier coefficients of $(x_{N_2 k})_k$ with N_1 values.
- X^1 are the Fourier coefficients of $(x_{N_2 k+1})_k$ with N_1 values.

- $\cdots$
- X^{N_2-1} are the Fourier coefficients of $(x_{N_2k+N_2-1})_k$ with N_1 values.

Now we use them to compute X:

$$\begin{aligned} X_k &= \sum_{n=0}^{N-1} x_n e^{-i2\pi\frac{n}{N}k} = \sum_{p=0}^{N_2-1}\sum_{l} x_{N_2l+p} e^{-i2\pi\frac{N_2l+p}{N}k} \\ &= \sum_{p=0}^{N_2-1}\sum_{l} x_{N_2l+p} e^{-i2\pi\frac{N_2l}{N}k} e^{-i2\pi\frac{p}{N}k} \\ &= \sum_{p=0}^{N_2-1} e^{-i2\pi\frac{p}{N}k} X_k^p \end{aligned}$$

In particular, the algorithm 2 uses the case of $N_2 = 2$ and the follow observation:

$$\begin{aligned} X_{k+N/2} &= \sum_{n=0}^{N-1} x_n e^{-i2\pi\frac{n}{N}(k+N/2)} = \sum_{n=0}^{N-1} x_n e^{-i2\pi\frac{n}{N}k - i2\pi\frac{n}{2}} \\ &= \sum_{l} x_{2l} e^{-i2\pi\frac{2l}{N}k - i2\pi l} + \sum_{l} x_{2l} e^{-i2\pi\frac{2l+1}{N}k - i2\pi\frac{2l+1}{2}} \\ &= X_k^0 - e^{-i2\pi\frac{1}{N}k} X_k^1 \end{aligned}$$

The computational cost of algorithm 2 is $O(N \log N)$ if the base case (when N is not even, or in general is a prime) has cost $O(N \log N)$.

Algorithm 2 Cooley J. W. [2] & CZT algorithms.
INPUT
x: time series $x_1, \ldots, x_N$
OUTPUT
X: Fourier coefficients of x

```
1:  function FFT(x)
2:      if N is odd then return CZT(x)                ▷ base case
3:      end if
4:      even ← FFT([x_0, x_2, ··· , x_{N-2}])
5:      odd ← FFT([x_1, x_3, ··· , x_{N-1}])
6:      X is an array of size N
7:      for k ∈ 0, ..., N/2 − 1 do
8:          t ← exp(−2πik/N) · odd[k]
9:          X[k] ← even[k] + t
10:         X[k + N/2] ← even[k] − t
11:     end for
12:     return X
13: end function
```

Periodogram The periodogram is a practical tool that graphically represents the amplitude of the different frequencies composing a time series, providing insight into the frequency spectrum of the signal. By visualising the power associated with each frequency, it highlights the dominant components of the spectrum, making it particularly useful for distinguishing between noise and significant frequencies. This connects directly to the theory of the DFT, as the periodogram is derived from the squared modulus of the Fourier coefficients, representing the signal's energy distribution across frequencies.
To illustrate this, we apply the DFT to the `sunspots` dataset from the R package, which contains the number of sunspots observed each month from 1749 to 1984. Sunspots are known to correlate with the Sun's magnetic activity, which is believed to exhibit periodic cycles. Using the periodogram, we aim to identify these cycles, particularly the dominant periodic components in the data, and evaluate their significance compared to noise.

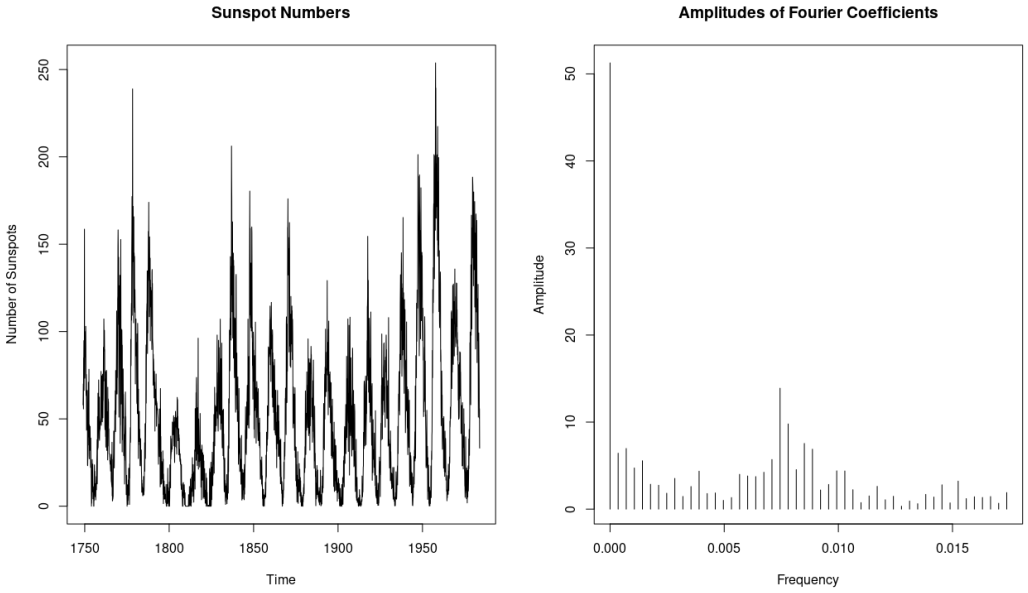


Figure 2.3. On the left, the time series of the number of sunspots recorded each month from 1749 to 1984 ($N = 2820$). This representation shows the variation of sunspots over time, highlighting any cycles. On the right, the periodogram obtained by applying the DFT to the time series. Each point represents the squared modulus of a Fourier coefficient, which measures the energy associated with a given frequency. We can see a peak at frequency 0, corresponding to the sum of the values of the time series ($X_0 = \sum_{n=0}^{N-1} x_n$).

Looking at the periodogram obtained with FFT, we observe that some frequencies are much more pronounced than others. In particular, the frequency $\approx 0.0075\,\text{Hz}$ (corresponding to about a cycle of 135 months) indicates a cycle of about 11 years, which confirms what is already known about the cyclic nature of solar activity.
This example demonstrates how the DFT is a powerful tool for detecting and analysing periodic patterns in observational data. The use of the periodogram not only makes it possible to identify the main frequencies, but also to evaluate their relative importance respect to the background noise.

2.4 Application

In this section, we will develop specific applications of the topics discussed in this chapter to see how they can be used in the context of this thesis. The discussion will focus on the use of fuzzy clustering to address the problem of continuity in colour space, the practical implementation of the FCM algorithm, and the use of FFT for image analysis.

2.4.1 Comparing works, the idea of clustering

As we have seen, the tiles used to compute eq. (2.3) can be represented as sequences of real numbers. However, their continuous nature introduces a problem in the computation of comparison values, since it reduces the n-grams to unique and unrepeatable objects (**sparsity problem**). This problem renders eq. (2.3) ineffective, so that any pair of different works is assigned a comparison value of 1.
One solution proposed in [5] was the use of **posterization**, i.e., reducing the variety of colors. In this way, distinct n-grams could become equal after posterization, improving the comparison. However, this method introduced significant impurities and a loss of shade information.
In this thesis, a way is proposed that avoids or reduces the need for strong posterization (e.g., reduction to only two colors, `b/w`, as in [5]). The proposed generalization is based on the dynamic use of clustering, comparing two approaches:

- In [5]: The space is clustered **a priori**, labeling the tiles with predefined boxes. The centroids take fixed values (0 and 1), corresponding to the colors. For example, in the case of 1D: the 2-gram $(0.6, 0.8)$ will be the centroid $(1, 1)$, while $(0.2, 0.8)$ will be the centroid $(0, 1)$.
- In this thesis: The space is clustered **a posteriori**, labeling the tiles using a clustering algorithm, such as FCM.

In this chapter, we will use KMeans, since its application is more similar to the box system and more intuitive than FCM. The key idea is to apply clustering on the union of tiles extracted from two works.
Doing so is expected to provide more accurate results than posterization. However, it will be necessary to provide a definition of comparative value in the case of KMeans.

Exempli Gratia (Distance between distributions reformulated with KMeans). In this paragraph we will adopt the theoretically easy clustering KMeans. Let us take 10 000 samples from $\mathcal{N}(-1, 0.25)$ and 40 000 from $\mathcal{N}(+1, 1)$, the two distributions will be denoted by $\mathcal{A}$ and $\mathcal{B}$ respectively.
The two sets of samples will be merged and clustered with KMeans using 32 centroids, resulting in a predictor $\mathcal{P}$. This predictor maps each data point x to the centroid c of the cluster it belongs to, i.e., $P(x) = c$ if x belongs to the cluster with centroid c. Each cluster will be viewed as a region with a certain measure that will be the mean square of the distances of each datum in the cluster from its centroid. Given a cluster of centroid c, we want to estimate:

$$\mu(c) = \sqrt{\mathbb{E}_{x\sim\mathcal{L}}\left[\|x-c\|^2 \middle| \mathcal{P}(x) = c\right]}$$

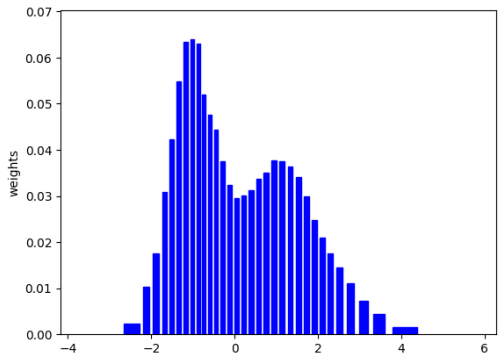
where $\mathcal{L}$ is the law of the two merged samples.
We can see, in the image at left, that the regions has a little measure $\mu(c)$ where the density of $\mathcal{L}$ is higher. Furthermore, the image shows the weights of each cluster c defined as $\mathbb{P}_{x\sim\mathcal{L}}[\mathcal{P}(x) = c]$
We now study the two sets of samples separately over this clustering. The densities will be weighted on the new measure μ, so the density on the centroid c of measure $\mu(c)$ respectively for the distribution $\mathcal{A}$ will be:

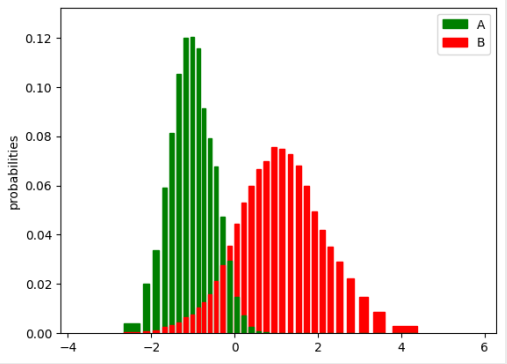
$$d_{\mathcal{A}}(c) := \frac{p_{\mathcal{A}}(c)}{\mu(c)} := \frac{\mathbb{P}_{x\sim\mathcal{A}}[\mathcal{P}(x) = c]}{\mu(c)}$$

similarly for $\mathcal{B}$.
In the figure at right we show the cluster membership probabilities of the two distributions $\mathcal{A}$ and $\mathcal{B}$.
Let us try to calculate the comparison value formulated eq. (2.3) considering the measure:

$$\begin{aligned} d_{\text{KMeans}}(A,B) &= (1+J_{D_A,D_B})^{-1}\frac{1}{\sum_{c\in D_A\cup D_B}\mu(c)}\sum_c \mu(c)\left(\frac{p_A(c)-p_B(c)}{p_A(c)+p_B(c)}\right)^2 \\ &= (1+J_{D_A,D_B})^{-1}\frac{1}{\mu(D_A\cup D_B)}\int d\mu(c)\left(\frac{p_A(c)-p_B(c)}{p_A(c)+p_B(c)}\right)^2 \end{aligned}$$

where D_A is the support for the discretisation of $\mathcal{A}$ and similarly for D_B.
The result for 32 centroids is 0.54.





In the example, an intuitive definition of a comparison value between works is proposed using clustering. In such contexts, clustering proves to be an effective tool for handling the complexity and sparsity of high-dimensional data. This is particularly relevant for counteracting the curse of dimensionality, a phenomenon where the volume of the data space is inherently vast compared to the number of available data points. In such scenarios, rigid box-based discretisations often fail because they divide the space into a fixed grid, which becomes inefficient as the dimensionality increases. The result is an exponential increase in the number of boxes, most of which remain empty or sparsely populated, leading to poor generalization.
By contrast, dynamic clustering the data with methods such as KMeans or FCM allows an adaptive representation that aligns with the data distributions. Instead of imposing fixed boundaries, the centroids are positioned to capture meaningful structures in the data, improving both computational efficiency and empirical performance. This flexibility makes dynamic clustering a practical and effective choice for high-dimensional problems.

Exempli Gratia (Curse of Dimensionality). In this example, we compute the comparison value between the distribution $\mathcal{N}\left(\vec{0}, \mathbb{1}_{\mathtt{d}}\right)$ in $\mathbb{R}^K$ and itself for different values of K.
Specifically, for each K, two samples A and B, each containing 128 points, are taken from the same distribution. The samples are compared using two approaches: a static clustering approach (box-based) and a dynamic clustering approach (KMeans). In the box-based method, the space is divided into 2^K boxes (2 boxes per axis), while in the KMeans method, the number of centroids is fixed at 2. The table below shows the average comparison values as K varies:

Dimensions	1	2	4	8	16
Comparison with clustering	0.00	0.00	0.00	0.00	0.00
Comparison without clustering	0.00	0.01	0.04	0.57	1.00

Now consider two sets A and B of samples drawn from $\mathcal{N}(\vec{0}, \mathbb{1})$ and $\mathcal{N}(\vec{1}/\sqrt{K}, \mathbb{1})$ in $\mathbb{R}^K$. These Gaussian distributions are equidistant regardless of the dimensionality K, so we expect stable results. The table below reports the results of this comparison:

Dimensions	1	2	4	8	16
Comparison with clustering	0.07	0.07	0.05	0.03	0.02
Comparison without clustering	0.08	0.08	0.11	0.62	1.00

The number of centroids is a critical parameter. As with boxes, too many centroids can overfit, increasing the comparison value, while too few may fail to capture the data's structure efficiently. Additionally, in high-dimensional spaces, box-based clustering becomes computationally prohibitive compared to dynamic clustering, with computation times exceeding those of KMeans by a factor of over 100. The dynamic shape of KMeans clusters explains this difference: a single dynamically generated cluster can cover regions spanning thousands of boxes, drastically reducing computational cost while preserving accuracy.

2.4.2 Fuzzy Clustering as a Noise Filtering Method

In this thesis, a variant of the algorithm FCM will be used to compare two datasets. This variant introduces a weighting factor w, which represents the importance or contribution of each data point in the clustering process. The weight ww can be interpreted as the effective cardinality of the point: for instance, a point with $w = 2$ is treated as if it were two identical points, while $w = 0.5$ corresponds to half a point. This generalization allows for cleaner comparisons between sets of samples with different cardinality, simulating a balanced dataset by appropriately scaling the influence of each point.
Additionally, this variant includes specific adjustments to account for machine error in the computation of centroids (see algorithm 1). These adjustments improve robustness in high-dimensional spaces, with large datasets, or when using many centroids, mitigating numerical precision issues for stable clustering.

Data's weight We introduce a vector w indicating the weight of the data as a positive real value. As stated in theorems 2.1 and 2.2 and definition 2.2.2, the following equations summarize the key components of the fuzzy clustering algorithm discussed so far:

$$\begin{aligned} C_j^{\text{new}} &= \frac{\sum_{i=1}^{N} u_{ij}^2 x_i}{\sum_{i=1}^{N} u_{ij}^2 w_i} \quad \forall j \\ L &= \sum_i \sum_j u_{ij}^2 \left\| x_i - C_j \right\|^2 \\ u_{ij} &= \frac{1}{\sum_k \frac{d_{ij}^2}{d_{ik}^2}} \quad \forall i, j \\ d_{ij} &= \left\| x_i - C_j \right\| \quad \forall i, j \end{aligned}$$

Since it is a weighted average over u_{ij}^2, if a datum has a higher weight then it should increase its influence, thus having the following results:

$$\begin{aligned} C_j^{\text{new}} &= \frac{\sum_{i=1}^{N} u_{ij}^2 w_i x_i}{\sum_{i=1}^{N} u_{ij}^2 w_i} \quad \forall j \\ L &= \sum_i \sum_j w_i u_{ij}^2 \left\| x_i - C_j \right\|^2 \\ u_{ij} &= \frac{1}{\sum_k \frac{d_{ij}^2}{d_{ik}^2}} \quad \forall i, j \\ d_{ij} &= \left\| x_i - C_j \right\| \quad \forall i, j \end{aligned}$$

machine error The use of FCM in this thesis involves millions of data, and it is possible that the classical algorithm will find itself making serious machine errors that must be kept under control. In algorithm 1, might exist is a value $D_{ik}^2 \approx 0$ that may be null or so small that when D_{ij}^2 / D_{ik}^2 is computed, it is infinity or a number so large that when added to other numbers it overshadows all other data in the sum.

For this reason algorithm 3 proposes a more robust approach. We also remark that the computational cost in a sequential algorithm will be $O(NMK)$. However scalability allows us to reduce this cost to $O(\log(KM))$ by exploiting the independence of each cycle and scalable reductions (details in chapter 3).

N is the number of data and M is the number of centroids, K is the size of the tiles

Algorithm 3 Membership update stable computation.
INPUT
$\mathcal{S}$: set of data $x_1, \ldots, x_N$
$\mathcal{C}$: centroids $c_1, \ldots, c_M$

procedure MembershipUpdateStable($\mathcal{S}, \mathcal{C}$)
  $D^2 \leftarrow (d_{ij}^2)_{ij}$ with $d_{ij}^2 = \|x_i - C_j\|^2$
  for $i \leftarrow 0$ to N **do**
    $l \leftarrow \min_k\{D_{ik}^2\}$
    if $l = 0$ **then**
      where $D_{ij}^2 = 0$ **do** $u_{ij} \leftarrow 1$
      where $D_{ij}^2 \neq 0$ **do** $u_{ij} \leftarrow 0$
    else
      $u_{ij} \leftarrow \frac{l}{D_{ij}^2} \quad \forall j$
    end if
    $S_i \leftarrow \sum_j u_{ij}$
    $u_{ij} \leftarrow u_{ij}/S_i \quad \forall j$
  end for
end procedure

2.4.3 Analysis of images with DFT

It was seen in the introductory section of DFT that the algorithm FFT can only be applied to time series or, more generally, to a sequence of complex numbers. However, it is also possible to extend this concept to the analysis of the spectrum of a matrix with periodic behaviour.
Consider a matrix $x \in \mathbb{C}^{N \times M}$ defined as follows:

$$x_{n,m} = \cos(\omega_r n + \omega_c m)$$

It can be shown that its DFT results in a matrix of the same shape as x, where the only nonzero component is in the row corresponding to ω_r and in the column corresponding to ω_c.
This is analogous to the application of CFT on $\mathbb{R}^2$. In particular, we would like to obtain the following inverse relation to reconstruct x from its Fourier coefficients:

$$x_{n,m} = \frac{1}{NM} \sum_{r,c} X_{r,c} e^{i2\pi\left(\frac{rn}{N} + \frac{cm}{M}\right)} \tag{2.8}$$

Where the Fourier coefficients $X_{r,c}$ are given by:

$$X_{r,c} = \sum_{n,m} x_{n,m} e^{-i2\pi\left(\frac{nr}{N} + \frac{mc}{M}\right)} \tag{2.9}$$

At the algorithmic level, the two-dimensional FFT is obtained by applying the algorithm to the columns first and to the rows of the original matrix x. In particular:

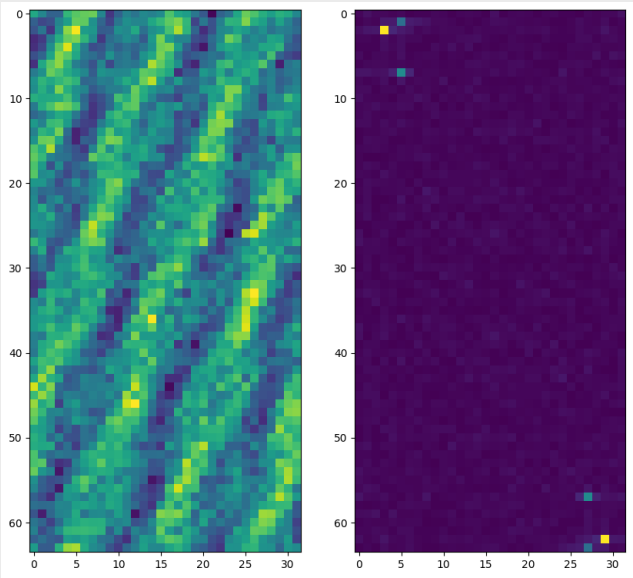
$$y_{r,m} = \sum_n x_{n,m} e^{-i2\pi \frac{nr}{N}} \qquad \text{over each column apply FFT}$$

$$X_{r,c} = \sum_m y_{r,m} e^{-i2\pi \frac{mc}{M}} \qquad \text{over each row apply FFT}$$

Exempli Gratia. We analyse a matrix composed of several overlapping frequencies and Gaussian noise with variance 1.

$$\begin{aligned} x_{n,m} &= 2\cos\left(2\pi(2n+3m)+3\right) \\ &\quad + 0.8\cos\left(2\pi(n+5m)+2\right) \\ &\quad + \cos\left(2\pi(7n+5m)\right) + \mathcal{N}_{n,m} \end{aligned}$$

In the figure, the original data is shown on the left as a 2D matrix visualized using the `viridis` color scale, while the Fourier coefficients computed from this matrix are displayed on the right.



The two-dimensional FFT can be used to analyse periodic patterns in matrices that represent images or signals. Typical applications include filtering, image compression and pattern recognition, where the spectral decomposition allows significant components to be distinguished from the noise.

Chapter 3

Methodology

This research aims to analyse the attribution of graphic works, taking its inspiration from the methodologies used for the attribution of literary works [see 5]. In particular, it examines the application of N-grams, i.e. sequences of N adjacent symbols, as a method of representing works.
To fully understand the representation of graphic works, it is appropriate to start from the method used for literary works. For example, in the expression `"Hello world!"`, a 3-gram can be represented by `"llo"`, which corresponds to a sequence of 3 consecutive characters. Importantly, spaces and punctuation are also considered characters, so `"o w"` and `"ld!"` are also valid 3-grams.
Research in this field is extensive and has led to various applications. Representations based on N-grams are used not only in literary attribution, but also in natural language models in which these analyses are integrated with the use of recurrent neural networks. However, little is known so far about the application of this technique to graphic works, and this constitutes the main focus of this research.

Discrete Automatic Drawings' Analysis (DADA) In the thesis research about the application of N-gram analysis on calligraphic works, [see 5], a graphic work is considered as matrix of pixels and an N-gram was defined as a square sub-matrix of size N, also called 'tile'.
The research focused on the analysis of images that have been reduced to matrices, in which the only colours present are black and white (see fig. 3.1). This process, known as 'posterisation', aims to reduce the amount of colours present in the work (known as 'depth'). This methodological choice was motivated by the need to manage the size of the alphabet; indeed, while in a literary work there may exist up to 32^N distinct N-grams (considering the 26 letters of the alphabet plus 6 punctuation symbols), in a pictorial work there may be 256^{3N^2} distinct tiles with side N. This considerable difference caused significant challenges in image analysis, rendering the application of tools designed for literary works on graphic works ineffective.
The research concluded with promising results, showing that the automatic analysis of handwriting samples allows very precise author attributions. However, it turned out that applying the same analysis to pictorial works such as panels and drawings does not produce satisfactory results. It was conjectured that this is due to the intrinsic nature of pictorial works, where the presence of colour plays an important

Figure 3.1. Example of a conversion from a coloured image to an image with only black and white (bw) pixels. The image is transformed to greyscale by reducing the saturation, finally half of the lightest pixels are rendered white and the remaining are rendered black.

role, and posterisation, instead of helping attribution, generates noise and new information that make the representation of data unreliable.

Continuous Automatic Drawings' Analysis (CADA) Following the same line of research [5, my bachelor thesis], we have proposed a re-adaptation of the methodology in order to investigate what might be the most effective strategy for the automatic analysis of images characterised by a large alphabet of colours. The aim is to construct a CADA that can analyse colours in their natural space, i.e. in a continuous space. This will allow us to overcome the discrete constraint of DADA and obtain more accurate and detailed results.
Image analysis in DADA is a process in five stages: acquisition, pre-processing, synthesis, comparison and attribution. In this paper we are going to focus on the first four phases, detailing the process up to image comparison, while attribution will be briefly discussed as the concluding phase.
The acquisition phase digitises two-dimensional works of art by scanning or photography. Scanning, in particular, allows a high resolution and accurate representation of the original document, while photography allows greater flexibility when the size of the work or physical conditions make scanning complex. The result of this phase is a digital image that accurately represents the work to be analysed.
The pre-processing phase prepares the image for analysis by removing irrelevant elements through pixel-wise and work-size transformations. Pixel-wise techniques, such as greyscale conversion, act on each individual pixel. Work-size transformations, on the other hand, act on the image as a whole, e.g. with compression techniques or normalisation of colour scales.
The synthesis phase deconstructs the image into a list of tiles of size $N \times N$, generating a sequence of discrete elements representing portions of the image. This process, aligned with the idea of n-gram language models, allows stylistic features to be analysed in a $\mathbb{R}^{N^2}$ space, preparing the data format for later comparison.

In the comparison phase, we are going to address the real computational and theoretical challenge. Clustering techniques will be used to dynamically discretise the space of tiles and organise them according to similarity criteria. This allows us both to efficiently process the large number of tiles in a very high dimension and also to compare the target work with known works in the database.
Finally, in the attribution phase, the similarities between the target work and known works are examined. The aim is to identify the author of the target work by studying the closest works in terms of stylistic and compositional characteristics. This phase focuses on identifying the works that show the greatest stylistic affinity with the target work, suggesting possible attributions.
This research not only aims to tackle technical and computational challenges, but also to explore new perspectives in the analysis of artworks, thus opening up new horizons in the field of art criticism and art attribution.

3.1 Data Set

3.1.1 Description

In this study, the data set consists of a number of graphic works for which the attribution is well known and for which a uniform resolution expressed in PPI is known. It was decided to use calligraphic works, as evidenced in [5]. The use of samples of handwriting represents a promising first way for artistic attribution.
The dataset used in [5] consisted of authentic and unaltered handwriting samples. In this study, the dataset consists of university notes, which do not have a sufficiently high standard and therefore present some significant challenges:

- **The squares**: Note sheets are often written in squared notebooks. The background dirties the writer's handwriting. This visual interference can complicate the attribution process.
- **Use a variety of writing tools**: University notes can be written using a variety of writing tools, such as highlighters, markers, pencils or white-out. These different writing tools produce different lines and colours, which can affect the attribution of authorship.
- **Different graphical contexts**: Notes can contain a variety of elements such as mathematical formulae, graphs and erasures. These features represent heterogeneous graphical contexts that add complexity to the analysis.
- **Scanning contamination**: The quality of the page scan can introduce blemishes into the record. These blemishes, such as stains or blurring, can affect the clarity and legibility of the works.

In addition, the use of a pen with a thickness of 0.4 mm on the sheet was considered a reasonable first choice since in [5] the stroke had the same thickness. This particular aspect is also important in ensuring a certain uniformity and consistency in the visual attributes of the works under examination, thus facilitating a more accurate attribution of authorship.
The dataset is organised into 4 authors, and the included images are as follows:

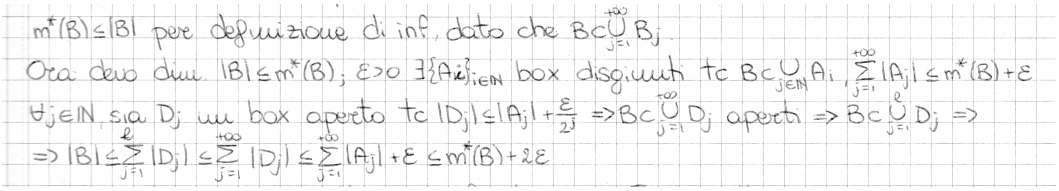
- The images are saved in the png format.
- Each image is composed of three colour channels Red Green Blue (RGB), each with a depth of 8 bits.
- The original sheets were scanned at a resolution of 400 PPI.
- The thickness of the pen used to write on the original sheets is 0.4 mm.

The images were then cut to remove large impurities or large white spaces. Finally leaving a total of 420 images.

AuthorID	Num of items	Size	Mean side length
1	270	554 MB	1.43 Kpx
2	43	133 MB	1.8 Kpx
3	56	227 MB	2.0 Kpx
4	51	247 MB	2.2 Kpx
Total	420	1.2 GB	1.7 Kpx

Table 3.1. Each item is a cleaned area of the original images. The size represent the total number of pixels. The last column is the geometric mean of the side length, computed as follows: $\sqrt{\frac{\text{Num of pixels}}{\text{Num of items}}}$

In the dataset creation, full clipboard images were collected and then cut to exclude non-pen strokes, highlights, and erasures. In addition, some cuts are particularly small and in a few cases are slightly overlapping.



3.2 Pre processing

Digitising images is a fundamental step in preparing the data set for analysis. To fully understand how these transformations work, it is important to understand the process by which a work of art is captured by a camera and then digitised in an electronic device.

Definition of image: from the real world to the virtual world Artists of antiquity used various techniques, such as the camera obscura and the principles of projective geometry, to represent landscapes realistically. During the Renaissance, they further refined these methods with tools such as the camera lucida, showing advanced mathematical understanding in works of art such as Raphael's School of Athens.[1]

[1] for more about camera obscura, see
https://www.academia.edu/28360703

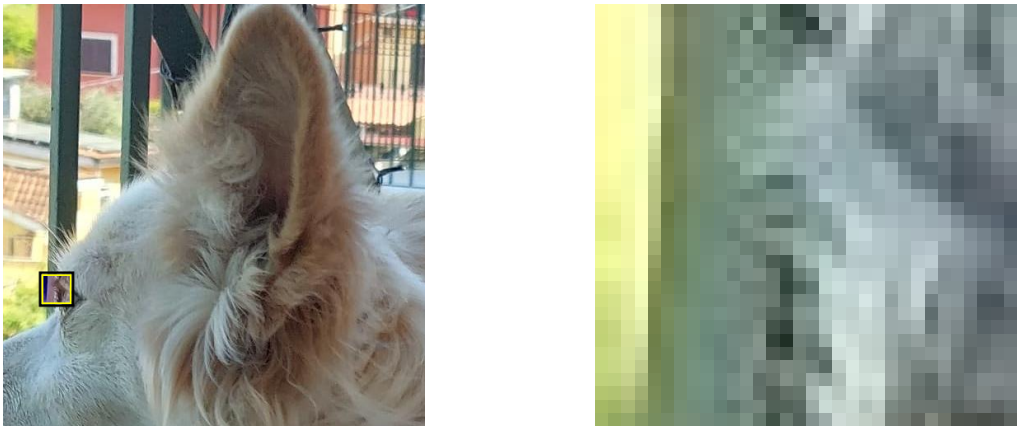
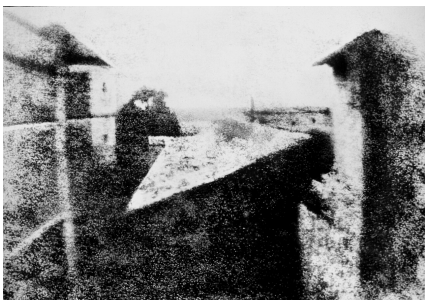
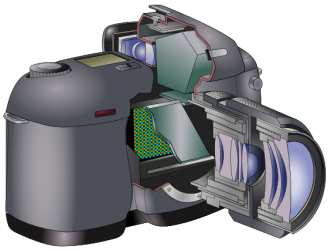


Figure 3.2. Image detail.

The invention of photography in 1826, in conjunction with the art movement of Realism, marked a fundamental moment in the history of visual art. Photography, with its ability to represent reality objectively, became a tool increasingly used by artists. The advancement of knowledge in optics and chemistry led to the creation of analogue photography.[2]



(a) 'View from the Window at Le Gras', Joseph Nicéphore Niépce, c.1826, Heliography, Harry Ransom Center, University of Texas at Austin, USA[7].



(b) Illustration of digital camera[9].

During the 1980s, with the advent of computer technology, digital photography became a reality. The basic operation remained similar to previous techniques: instead of being printed on a photosensitive surface, the image was captured by a grid of optical sensors and digitised as a matrix of colours, with each component called a pixel.
The concept of colour reproduction is not new and as early as 1855 James Clerk Maxwell worked on this subject, introducing the colour model RGB, which is still widely used today (see [4]). This model encodes each colour with three real numbers

[2] for more about history of photography, see
https://marcocrupi.it/2011/03/come-la-fotografia-cambio-larte-storia.html

in an interval between 0 and 1. However, for computing purposes, it was agreed to represent the 3 values of a colour with integers between 0 and 255. In this way, a digital image becomes a matrix of triples, and each pixel is represented with the three colour channels RGB.

3.2.1 Grayscale reduction

The quantisation of colours is expressed in 3 channels RGB, each with an integer value between 0 and 255. The choice of these 3 colours is not arbitrary; it derives from a physiological feature of human vision. Our perception of colour is determined by photoreceptor cells in the eye called cones, of which there are three types: S-cones, M-cones and L-cones. These cones are sensitive to wavelengths that approximately correspond to blue, green and red light respectively.
This model of colour representation is therefore deeply rooted in the way our visual system works, rather than in the physical reality of light. For example, when we see yellow, it is the result of the simultaneous activation of both M and L cones, which our brain interprets as a pure yellow colour. This perception is not a direct consequence of the physical properties of light, but rather a result of our brain's processing. Similarly, the colour magenta does not have a single wavelength in the visible spectrum, but is perceived when our brain combines stimuli from both red and blue wavelengths. These examples illustrate that colour perception is a subjective phenomenon in which the brain combines signals in a way that does not always reflect a direct physical counterpart.
This subjective nature of colour perception means that our representation of colour space is influenced more by neurological processes than by physical reality. While physically we might think of colour as occupying a continuous, structured space (e.g. a cone or cylinder in RGB representation), the actual experience of colour varies between individuals. For example, people with colour blindness may perceive certain colours differently due to differences in their cones. In addition, cultural differences can also affect perception: the Himba people of Namibia, for example, can distinguish between shades of green that many others cannot, while they may have difficulty distinguishing green from blue[3].
Given this subjective interpretation, the classification of colours often goes beyond a purely scientific approach. In contexts such as art, where perception and expression are key, it is often more useful to refer to colour spaces such as Hue Saturation Lightness (HSL), which represent hue, saturation and lightness, rather than the more physically oriented RGB. These spaces provide a more intuitive way of describing how colours relate to each other in terms of what we actually perceive.
Greyscale reduction builds on this understanding of colour perception. It involves reducing the image to the brightness component only, effectively removing hue and saturation. There are several techniques to achieve this. One common method is to take a weighted average of the three RGB channels, with the weights chosen to reflect the varying sensitivity of the human eye to different colours. Another approach is to average between the maximum and minimum intensity channels. Whatever the method, the aim is to create a representation of the image that retains luminance

[3]For more about Himba people, see https://psycnet.apa.org/record/2000-05104-005

information while discarding colour, thus simplifying the visual data while retaining the essence of light intensity.

3.2.2 Removing squares

One of the main challenges in processing and collecting the dataset concerns the cleanliness of the images, which can have significant impurities, such as the presence of squares on the sheets. It is essential to remove or mitigate these elements so that they blend in with the background and do not interfere with the analysis. Initially, the use of static techniques such as thresholding or the application of specific kernels to recognise the squares was considered. However, such techniques risked compromising the author's writing by erasing details. For this reason, it was decided to use compression techniques to detect patterns, since squares constitute a highly visible pattern that is more easily distinguishable by the machine than human handwriting.

Singular Value Decomposition (SVD) A first test involved examining the predominant patterns by means of a decomposition SVD. The image can be viewed as a matrix $H \times W$ of real values and can be decomposed into the product:

$$U\Sigma V^H = M$$

where M represents the image, U and V are unitary matrices ($UU^H = I$, same for V), V^H is the Hermitian of V, and Σ is a matrix $H \times W$ with only non-negative values along the diagonal, called singular values. It is possible to obtain a less detailed version of M by cancelling some singular values of Σ, thus ignoring the contribution of certain patterns. It was observed that the predominant pattern (associated with the largest singular value) concerns squares, but its removal does not completely solve the problem, requiring the elimination of further patterns. The result obtained is encouraging, but entails a loss of information from human handwriting, as it too is present among the major patterns identified.
This issue arises from the nature of the SVD decomposition. Since the method considers the image as a linear transformation, removing the dominant component is

equivalent to removing the strongest eigenvector of the transformation. However, M is not inherently a linear transformation, and even small isometric transformations of the image (e.g. rotations) can affect the result.
An example is shown in section 3.2.2 to illustrate this issue. The first row shows an ideal scan with perfectly vertical and horizontal lines. The second row shows the effect of a 1° rotation, where the grid remains visible despite the application of SVD compression. Finally, the third row shows stronger compression, highlighting that the grid remains while the handwriting fades.

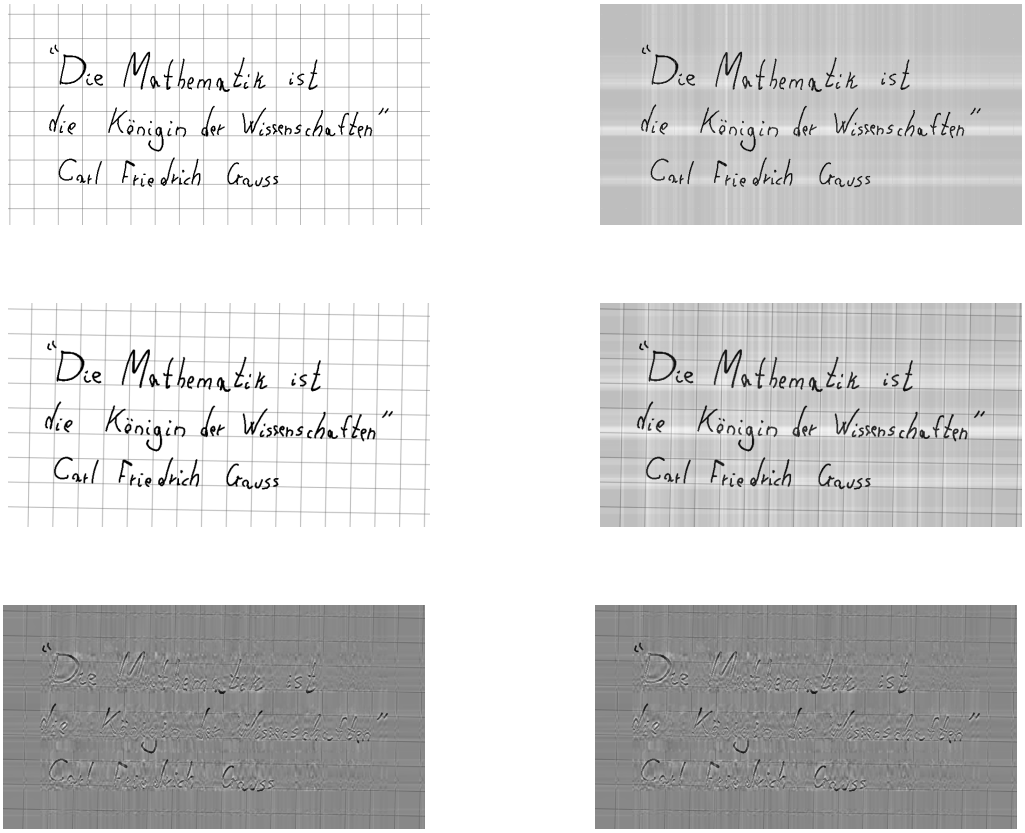


Figure 3.5. The first row shows an ideal scan with perfectly vertical and horizontal lines. The second row illustrates an imperfect scan with a 1° rotation, where the grid remains visible. The third row shows stronger compression, where the grid persists, but the handwriting fades.

FFT An alternative idea is to perform a harmonic analysis of the images using FFT. The squares are repeated with a specific frequency along the two dimensions of the sheet and thus one can exploit their regular nature compared to human handwriting, which is less repetitive. Thus, one exploits both the difference between the regular patterns of the squares and the irregular strokes of the handwriting, as well as a greater capacity for compression than a simple pixel-by-pixel analysis.
To verify this idea, we run a FFT on test images: a virtual sheet with only squares and the same sheet with handwriting added to it. In the case of the unmarked sheet, frequencies with larger amplitudes are mainly found aligned along the x and y axes

(fig. 3.6a). By adding handwriting, the primary frequencies characterizing the grid remain visible, even though additional frequencies introduced by the handwriting may partially overlap them (fig. 3.6b). To remove the squares, we filter out the frequencies of squares and reconstruct the image (fig. 3.6c).
Empirically, by analysing 10 randomly selected real images from the dataset, it was observed that the squares were successfully removed without damaging the lettering, cancelling out the amplitudes corresponding to the first 5% percentile of the most significant frequencies.
After reconstructing the image using the FFT, a significant number of pixels took on intermediate shades of grey that weren't present before. This effect is likely due to the removal of specific frequency components during the filtering process. The reconstruction introduces a real-valued matrix, where some components have extreme positive or negative values. When the image is normalized to the $[0, 1]$ range, the majority of pixel intensities are compressed into a narrower range of grey values, blending the background and the lettering. This normalization amplifies the loss of contrast, making it harder to distinguish important features from the unwanted noise.
To reduce the effects of this distortion, we adopted a different colour normalisation approach to keep the grey values as close as possible to the original image. The main idea was to identify extremely light and extremely dark grey values in the original image and to expand the range of grey values in the reconstructed image to match them.
It was observed that, in the original image, pixels with a grey level below 0.2 correspond to the lettering, while those with a grey level above 0.8 represent the white background of the page. Using these thresholds, we count the number of dark pixels c_d and light pixels c_l in the original image. In the cleaned image, we then identify the c_d darkest pixels and the c_l lightest pixels to determine the new grey levels: h_d, below which all pixels are considered dark, and h_l, above which all pixels are considered light.
To enhance the result of the FFT filtering, we adjust the grey levels of the cleaned image using a linear transformation. This transformation ensures that all pixels with a grey value below h_d became black (0), and those above h_l become white (1):

$$\texttt{new gray value} = \frac{\texttt{gray value} - h_d}{h_l - h_d}$$

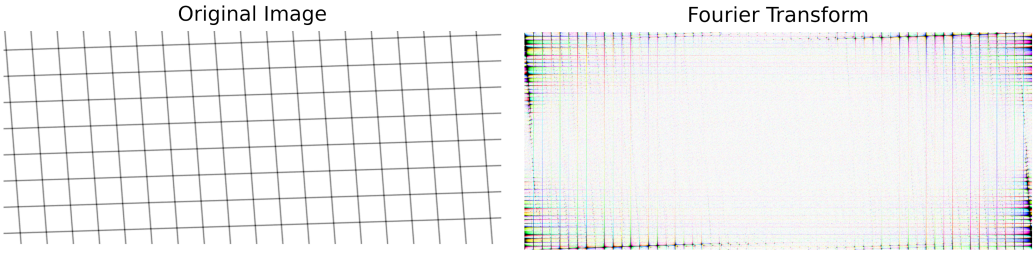
Finally, a clamp operation is applied to ensure all values remain within the range $[0, 1]$:

$$\texttt{final gray value} = \begin{cases} 0 & \text{if } \texttt{new gray value} < 0, \\ 1 & \text{if } \texttt{new gray value} > 1, \\ \texttt{new gray value} & \text{otherwise} \end{cases}$$

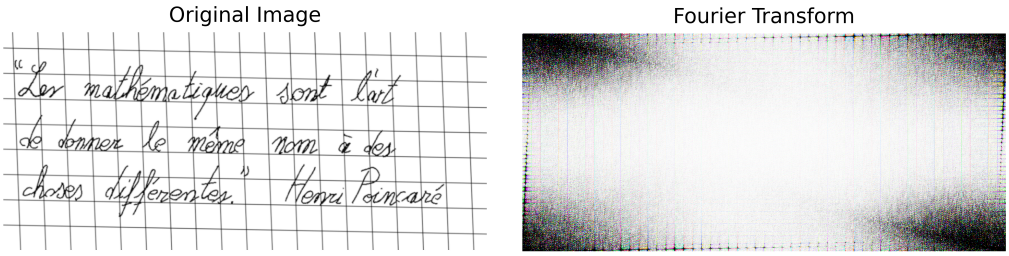
This operation, called `clamp`, ensures that the contrast between the lettering and the background is maximized, improving the overall quality of the cleaned image.

3.3 Synthesis

The tessellation phase extracts tiles $N \times N$ from each image. Each image is rep-



(a) An imperfect scan can show a slightly rotated grid, as seen in the left image. On the right, the Fourier coefficients of the scanned image are displayed. The most significant frequencies, with high amplitudes, are highlighted in black.



(b) In this case, handwriting was superimposed. The frequencies of the grid remain visible despite the added handwriting.

(c) The cleaned handwriting, free from the grid, no longer shows the Fourier coefficients corresponding to the squares, as recurring patterns are absent.

Figure 3.6. Fourier coefficients of grayscale images are organized into matrices of complex numbers. Each component at coordinates n, m has an amplitude and a phase that correspond to the respective frequencies $2\pi\frac{n}{N}$ and $2\pi\frac{m}{M}$, where N and M are the dimensions of the image. The phase is represented by hue, while amplitude is represented by brightness: darker pixels indicate higher amplitude.

Algorithm 4 Cleaning procedure using FFT.
INPUT
$\mathcal{I}$: Image as matrix $W \times H$ in gray scale
p: Fraction (or percentile) of the highest magnitudes to be considered significant
a: thresholds $(\min, \max)$
OUTPUT $\mathcal{I}_{\text{new}}$: Image as matrix $W \times H$ in gray scale

procedure CLEANFFT($\mathcal{I}, p, a$)
  $\hat{\mathcal{I}} \leftarrow \text{normalize}(\mathcal{I})$ ▷ Normalize the gray level of image's pixels
  $F_{\hat{\mathcal{I}}} \leftarrow \text{fft2d}\left(\hat{\mathcal{I}}\right)$ ▷ $F_{\hat{\mathcal{I}}}$ is a Matrix-Like object with complex values

  Sort all couples of frequencies using as role the magnitude
  $f = (f_1, \ldots, f_{W \times H}) \leftarrow \text{argsort}\left(\text{flatten}\left(F_{\hat{\mathcal{I}}}\right), \text{rule=abs}, \text{descending=True}\right)$

  Remove significant frequencies
  $f_{\text{best}} \leftarrow f\left[\text{last } \lfloor p \cdot \text{length} \rfloor \text{ values}\right]$ ▷ components with high magnitude
  for $k \in f_{\text{best}}$ **do**
    $F_{\hat{\mathcal{I}}}[k] = 0.0$
  end for
  $\hat{\mathcal{I}} \leftarrow \text{ifft2d}\left(F_{\hat{\mathcal{I}}}\right)$ ▷ rebuild normalized image

  Adjust the result of FFT
  $c_d \leftarrow$ **count** $\{\mathcal{I} \leq a.\min\}$ ▷ number of pixels with gray value $\leq a.\min$
  $c_l \leftarrow$ **count** $\{\mathcal{I} \geq a.\max\}$ ▷ number of pixels with gray value $\geq a.\max$
  $h_d \leftarrow \min_{c_d}^{-\text{th}} \hat{\mathcal{I}}$ ▷ get the $c_d^{-\text{th}}$ darkest gray value
  $h_l \leftarrow \max_{c_l}^{-\text{th}} \hat{\mathcal{I}}$ ▷ get the $c_d^{-\text{th}}$ lightest gray value
  $\mathcal{I}_{\text{new}} \leftarrow \frac{\hat{\mathcal{I}} - h_d}{h_l - h_d}$
  clamp operation:
  where $\mathcal{I}_{\text{new}} > 1$, $\mathcal{I}_{\text{new}} = 1$ **do**
  where $\mathcal{I}_{\text{new}} < 0$, $\mathcal{I}_{\text{new}} = 0$ **do**
  return $\mathcal{I}_{\text{new}}$
end procedure

resented as a matrix of grey pixels in $[0,1]^{h \times w}$, where h is the height and w the width of the image. At the end of this procedure, each image is represented as a list of tiles, making it difficult to reconstruct the original image. This process, referred to as *'synthesis'*, introduces a layer of complexity that helps protect against falsification.
Falsifying an artwork under this scheme presents several challenges:

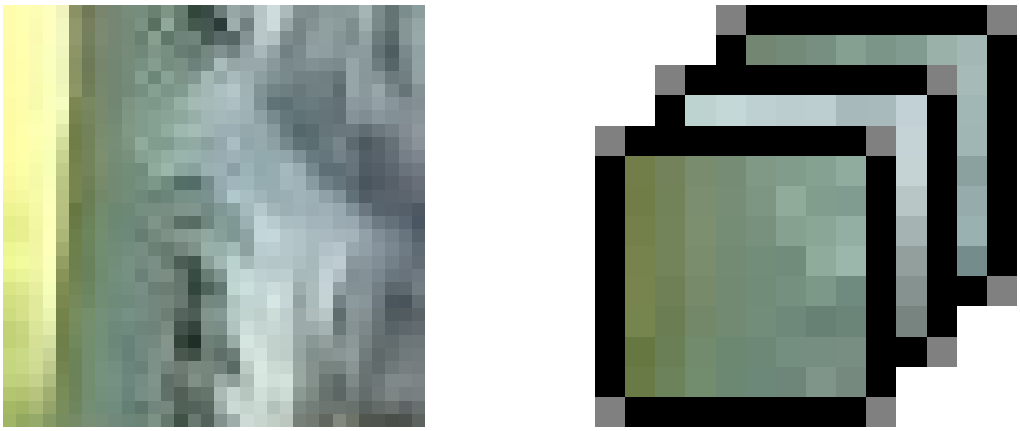
- Reconstructing an image from its tiles would require an extremely complex program, as brute-force approaches are computationally infeasible due to time constraints.
- A forger would not only need to generate a set of plausible tiles but also ensure that these tiles can be reconstructed into a coherent image, a task far more complex than simple reconstruction.

- Even if the forger perturbs an image repeatedly to reduce the distance from a target author, they must still preserve the overall sense of the artwork, which the tiles alone do not guarantee.

Furthermore, the small size of the tiles makes this process particularly robust. The tiles are large enough to capture the author's unique handwriting traits but small enough to make manual reproduction extremely difficult. For example, with a resolution of 400 PPI, each pixel measures approximately $63,5\,\mu$m. Given that the pen stroke width in the dataset is around 0.4 mm, tiles of size 6×6 are sufficient to capture an entire stroke.

Algorithm 5 Algorithm for tile extraction.
INPUT
M: Matrix of gray values with shape $h \times w$

```
1:  function TilesExtraction(M)
2:      L ← empty list of tiles        ▷ will have (h − N + 1) × (w − N + 1) tiles
3:      for row ← 0 to h − N, col ← 0 to w − N do
4:          Copy tiles in coordinates [row, row + N] × [col, col + N] of matrix M
5:          declare v as matrix N × N                          ▷ N is size of tiles
6:          for i ← 1 to N, j ← 1 to N do
7:              v[i][j] ← M[row + i][col + j]
8:          end for
9:          call Append(L, v)                          ▷ Append the tile v to list L
10:     end for
11: end function
```



(a) Image detail 32×32. **(b)** List of tiles 8×8.

Figure 3.7. The synthesis process convert an image in a list of its tiles.

3.4 Comparison

In order to compare two works, we decided to use the clustering FCM, which is less sensitive to potential noise. This is particularly important in our case, as the

comparison between two works leverages areas with lower densities, where noise could significantly affect the results.
In the previous study ([5]), it was observed that the formula for comparing works was highly influenced by tiles with fewer occurrences. In our current context, which operates in a continuous space, these tiles correspond to regions with lower densities. In this thesis, I developed an algorithm for comparing works and demonstrated its application through an example to better investigate its characteristics.

Comparison value using KMeans As already mentioned in chapter 2, we compared two works using their synthesized tiles. Specifically, we extracted two sets of tiles, $\mathcal{A}, \mathcal{B}$, from the two works, where each tile is represented as a vector in $\mathbb{R}^K$. In [5], we employed predefined boxes to cluster the tiles. Each box had the same measure but contained a different number of data points (i.e., tiles). To analyze these clusters, we used a clustering algorithm, KMeans, as described in chapter 2. Each cluster represents a region in the space $\mathbb{R}^K$ with its own measure and weights respect $\mathcal{A}, \mathcal{B}$.
The clustering procedure using KMeans can be summarized as follows:

1. Compute the KMeans clustering of $\mathcal{L} := \mathcal{A} \cup \mathcal{B}$. This gives the predictor $\mathcal{P}$, where $\mathcal{P}(x)$ is the centroid of the data point x.
2. Compute the measure of each cluster: $\mu(c) = \sqrt{\mathbb{E}_{x\sim\mathcal{L}}\left[|x-c\|^2 \middle| \mathcal{P}(x) = c\right]}^K$ where c is the centroid of the cluster.
3. Compute the weight of each cluster with respect to $\mathcal{A}$: $p_{\mathcal{A}}(c) = \mathbb{P}_{x\sim\mathcal{A}}\left[\mathcal{P}(x) = c\right]$
4. Compute the weight of each cluster with respect to $\mathcal{B}$: $p_{\mathcal{B}}(c) = \mathbb{P}_{x\sim\mathcal{B}}\left[\mathcal{P}(x) = c\right]$
5. Compute the Jaccard Index:

$$
\begin{aligned}
D_{\mathcal{A}} &:= \texttt{clusters containing tiles from } \mathcal{A} \\
D_{\mathcal{B}} &:= \texttt{clusters containing tiles from } \mathcal{B} \\
|D_{\mathcal{A}} \cup D_{\mathcal{B}}| &= M \quad \texttt{(total number of clusters)} \\
|D_{\mathcal{A}} \cap D_{\mathcal{B}}| &= \texttt{clusters containing tiles from both } \mathcal{A} \texttt{ and } \mathcal{B} \\
J_{\mathcal{A},\mathcal{B}} &:= \frac{|D_{\mathcal{A}} \cap D_{\mathcal{B}}|}{|D_{\mathcal{A}} \cup D_{\mathcal{B}}|}
\end{aligned}
$$

6. Compute the measure of all clusters: $\mu(D_{\mathcal{A}} \cup D_{\mathcal{B}}) = \sum_{c\in D_{\mathcal{A}}\cup D_{\mathcal{B}}} \mu(c)$
7. Compute the comparison value between the two works:

$$
d_{\text{KMeans}}(\mathcal{A}, \mathcal{B}) = (1 + J_{\mathcal{A},\mathcal{B}})^{-1} \frac{1}{\mu(D_{\mathcal{A}} \cup D_{\mathcal{B}})} \sum_c \mu(c) \left(\frac{p_{\mathcal{A}}(c) - p_{\mathcal{B}}(c)}{p_{\mathcal{A}}(c) + p_{\mathcal{B}}(c)}\right)^2
$$

3.4.1 Comparison value using FCM

Next, we will apply FCM to the fused distribution $\mathcal{L}$. Using fuzzy clustering, each data point will be assigned a degree of membership to multiple clusters, allowing for a more flexible representation of the data. For each cluster with centroid c, we will compute its size $\mu(c)$ and the weights of the set of samples, $p_{\mathcal{A}}(c)$ and $p_{\mathcal{B}}(c)$. Finally, we will redefine the Jaccard index between $D_{\mathcal{A}}$ and $D_{\mathcal{B}}$ within a fuzzy logic framework to better capture the similarities between the two sets.
The fuzzy nature of FCM requires a redefinition of the cluster measure. Unlike KMeans, where each data point is assigned to a single cluster by the predictor $\mathcal{P}$, in FCM each data point has a degree of membership in multiple clusters. To extend the concept of cluster size, we define the measure of a cluster as a quantity proportional to the K dimensional hypervolume of a hypersphere with radius equal to the mean square distance of the data from its centroid. For KMeans this is expressed as:

$$\mu(c) = \sqrt{\mathbb{E}_{x\sim\mathcal{L}}\left[\|x-c\|^2 \middle| \mathcal{P}(x)=c\right]}^K = \sqrt{\frac{1}{|\{x|\mathcal{P}(x)=c\}|}\sum_{x:\mathcal{P}(x)=c}\|x-c\|^2}^K$$

In KMeans, the algorithm aims to minimise the loss function:

$$\sum_c \sum_{x:\mathcal{P}(x)=c} \|x-c\|^2$$

We get a different perspective of this loss function:

$$\begin{aligned}
\sum_c \sum_{x:\mathcal{P}(x)=c} \|x-c\|^2 &= \sum_c |\{x:\mathcal{P}(x)=c\}| \frac{1}{|\{x:\mathcal{P}(x)=c\}|} \sum_{x:\mathcal{P}(x)=c} \|x-c\|^2 \\
&= N \sum_c \frac{|\{x:\mathcal{P}(x)=c\}|}{N} \mathbb{E}_{x\sim\mathcal{L}}\left[\|x-c\|^2 \middle| \mathcal{P}(x)=c\right] \\
&= N \sum_c \mathbb{P}_{x\sim\mathcal{L}}\left[\mathcal{P}(x)=c\right] \mathbb{E}_{x\sim\mathcal{L}}\left[\|x-c\|^2 \middle| \mathcal{P}(x)=c\right] \\
&= N \sum_c p(c)E(c)
\end{aligned}$$

where $p(c) := \mathbb{P}_{x\sim\mathcal{L}}\left[\mathcal{P}(x)=c\right]$ and $E(c) := \mathbb{E}_{x\sim\mathcal{L}}\left[\|x-c\|^2 \middle| \mathcal{P}(x)=c\right]$

Extending this idea, we generalize the concept using the objective function of FCM, as defined in definition 2.2.2. This allows the introduction of fuzzy membership.

Definition 3.4.1 (cluster's values)**.** Applying FCM to the data set $\mathcal{S}$ with weights w and tiles in $\mathbb{R}^K$, we obtain the centroids $\mathcal{C}$ and the memberships $\mu_c(x)$.
We call the representability of cluster with centroid c respect all other clusters the value: $p(c) = \frac{\sum_x w_x \mu_x(c)}{\sum_x w_x}$
We call the concentration of cluster with centroid c respect the membership measure the value: $\sigma(c) = \frac{\sum_x w_x \mu_x(c)^2}{\sum_x w_x \mu_x(c)}$

Remark. $\sigma(c) \in (0, 1]$ because $\mu_x(c) \leq 1$ and so: $\sum_x w_x \mu_x(c)^2 \leq \sum_x w_x \mu_x(c)$

We call the mean square radius of the cluster with centroid c the value:
$E(x) = \frac{\sum_x w_x \mu_x(c)^2 \|x-c\|^2}{\sum_x w_x \mu_x(c)^2}$
The loss function of algorithm FCM is can rewritted as:

$$\mathcal{L}_{\mathcal{S}}(\mathcal{C}) = \sum_x w_x \sum_c p(c)\sigma(c)E(c)$$

We denote by $\mu(c)$ the measure of the cluster c as:

$$\mu(c) = \sigma(c)E(c)^{K/2} = \frac{\sum_x w_x \mu_x(c)^2}{\sum_x w_x \mu_x(c)} \sqrt{\frac{\sum_{x\in\mathcal{S}} w_x \mu_x(c)^2 \|x-c\|^2}{\sum_{x\in\mathcal{S}} w_x \mu_x(c)^2}}^K \tag{3.1}$$

Similarly, we can define the values of the sets of tiles on the centroids:

Definition 3.4.2. Given a merged data set $\mathcal{S} = \mathcal{A} + \mathcal{B}$, where tiles belong in $\mathbb{R}^K$, and weights $w_{\mathcal{A}}$ and $w_{\mathcal{B}}$, the application of FCM yields the set of centroids $\mathcal{C}$ and the membership degrees $\mu_c(x)$ for each data point $x \in \mathcal{S}$.
The representability of the set $\mathcal{A}$ with respect to a centroid c is defined as

$$p_{\mathcal{A}}(c) = \frac{\sum_{x\in\mathcal{A}} w_x \mu_x(c)}{\sum_{x\in\mathcal{A}} w_x}$$

Similarly, the representability of the set $\mathcal{B}$ with respect to a centroid c is defined as

$$p_{\mathcal{B}}(c) = \frac{\sum_{x\in\mathcal{B}} w_x \mu_x(c)}{\sum_{x\in\mathcal{B}} w_x}$$

In this way, we have defined both the cluster measures and the weights of each cluster for the sets of tiles $\mathcal{A}$ and $\mathcal{B}$. This completes part of the comparison formula, leaving the inclusion of the Jaccard index $J_{\mathcal{A},\mathcal{B}}$ as the next step:

$$\frac{1}{\mu(D_{\mathcal{A}} \cup D_{\mathcal{B}})} \sum_c \mu(c) \left(\frac{p_{\mathcal{A}}(c) - p_{\mathcal{B}}(c)}{p_{\mathcal{A}}(c) + p_{\mathcal{B}}(c)}\right)^2$$

fuzzy logic and the Jaccard Index Defining the Jaccard index in fuzzy logic is a non-trivial task because the cardinality of a set is not defined in the same way as in Boolean logic. Unlike Boolean logic, where membership has a binary truth value (true or false), in fuzzy logic, membership degrees are continuous.

Even if FCM is requested to produce M centroids, not all clusters necessarily "exist". In Boolean logic, a cluster exists if at least one data point belongs to it. In fuzzy logic, however, no data point fully belongs to a specific cluster, and a data point can belong to multiple clusters simultaneously to varying degrees. Thus, the truth value of the statement "x `belongs to the cluster with centroid` c" is $\mu_x(c)$. Extending this concept, the truth value of the existence of a cluster with centroid c must also be defined. While there is no unique answer, we seek an acceptable and consistent definition for this operation.
Let $D_{\mathcal{A}}$ denote the set of clusters containing at least one data point from $\mathcal{A}$, and $\mu_c(D_{\mathcal{A}})$ represent the degree to which the cluster with centroid c exists within $\mathcal{A}$. Similarly, $|D_{\mathcal{A}}|$ represents the cardinality of the set of clusters, but in fuzzy logic, this cardinality is not an integer and can take continuous values. Below, we provide formal definitions for these concepts.

Definition 3.4.3. (Jaccard index) In fuzzy logic, the truth value of a logical disjunction is defined as the maximum of the truth values of its individual propositions. Based on this, the degree to which the cluster with centroid c exists in A is given by:

$$\mu_c(D_{\mathcal{A}}) = \max_{x \in \mathcal{A}} \mu_x(c)$$

The cardinality of a set is defined as the sum of the membership degrees of all data points to the set. Thus, the cardinality of $D_{\mathcal{A}}$ is:

$$|D_{\mathcal{A}}| = \sum_c \mu_c(D_{\mathcal{A}})$$

Consequently, the cardinality of the union of $D_{\mathcal{A}}$ and $\mathcal{B}$ is:

$$|D_{\mathcal{A}} \cup D_{\mathcal{B}}| = \sum_c \mu_c(D_{\mathcal{A}} \cup D_{\mathcal{B}}) = \sum_c \max\{\mu_c(D_{\mathcal{A}}), \mu_c(D_{\mathcal{B}})\}$$

Similarly, the cardinality of the intersection of $D_{\mathcal{A}}$ and $D_{\mathcal{B}}$ is derived from the logical conjunction, which uses the minimum:

$$|D_{\mathcal{A}} \cap D_{\mathcal{B}}| = \sum_c \mu_c(D_{\mathcal{A}} \cap D_{\mathcal{B}}) = \sum_c \min\{\mu_c(D_{\mathcal{A}}), \mu_c(D_{\mathcal{B}})\}$$

Based on these definitions, the Jaccard index for fuzzy logic is given by:

$$J_{D_{\mathcal{A}}, D_{\mathcal{B}}} := \frac{|D_{\mathcal{A}} \cap D_{\mathcal{B}}|}{|D_{\mathcal{A}} \cup D_{\mathcal{B}}|} = \frac{\sum_c \min\left\{\max_{x \in \mathcal{A}}\left(\mu_x\left(c\right)\right), \max_{x \in \mathcal{B}}\left(\mu_x\left(c\right)\right)\right\}}{\sum_c \max\left\{\max_{x \in \mathcal{A}}\left(\mu_x\left(c\right)\right), \max_{x \in \mathcal{B}}\left(\mu_x\left(c\right)\right)\right\}}$$

The definitions provided in definitions 3.4.2 and 3.4.3 and eq. (3.1) are sufficient to formulate the distance between two proposed samplings based on FCM:

$$d_{\mathrm{FCM}}\left(\mathcal{A}, \mathcal{B}\right) = \left(1 + J_{\mathcal{A},\mathcal{B}}\right)^{-1} \frac{1}{\mu(D_{\mathcal{A}} \cup D_{\mathcal{B}})} \sum_c \mu(c) \left(\frac{p_{\mathcal{A}}(c) - p_{\mathcal{B}}(c)}{p_{\mathcal{A}}(c) + p_{\mathcal{B}}(c)}\right)^2$$

3.4.2 Algorithm

The algorithm initially involves applying FCM to the fused dataset of the two works, in order to define the size and discretisation of the space of the tiles.

- $\mu(c) = \frac{\sum_x w_x \mu_x(c)^2}{\sum_x w_x \mu_x(c)} \sqrt{\frac{\sum_{x \in \mathcal{S}} w_x \mu_x(c)^2 \|x-c\|^2}{\sum_{x \in \mathcal{S}} w_x \mu_x(c)^2}}^K$
- $p_{\mathcal{A}}(c) = \frac{\sum_{x \in \mathcal{A}} w_x \mu_x(c)}{\sum_{x \in \mathcal{A}} w_x}$ and similarly $p_B(c) = \frac{\sum_{x \in \mathcal{B}} w_x \mu_x(c)}{\sum_{x \in \mathcal{B}} w_x}$
- $J_{D_{\mathcal{A}}, D_{\mathcal{B}}}$ as in definition 3.4.3

The distance between works will be defined as:

$$d_{FCM}(A, B) = (1 + J_{D_A, D_B})^{-1} \frac{\sum_c \mu(c) \left(\frac{p_{\mathcal{A}}(c) - p_{\mathcal{B}}(c)}{p_{\mathcal{A}}(c) + p_{\mathcal{B}}(c)} \right)^2}{\sum_c \mu(c)} \tag{3.2}$$

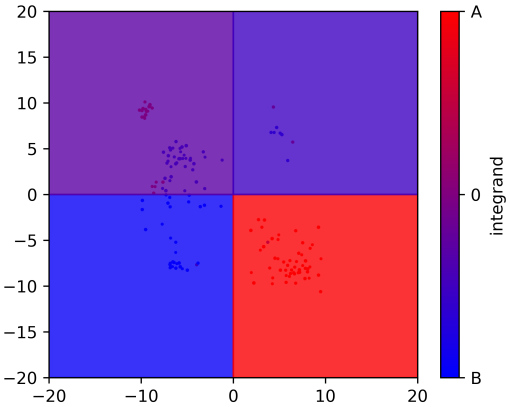
A simple application for a synthetic dataset is shown below, where we compare three methods for clustering and comparing two distributions $\mathcal{A}$ and $\mathcal{B}$. The three methods are:

- **Box-clustering**: the space is partitioned into fixed, predefined boxes.
- KMeans: the space is divided into circular clusters, where each cluster is defined by its centroid and radius.
- **FCM**: the space is divided into fuzzy circular clusters, where each data point has a degree of membership to multiple clusters.

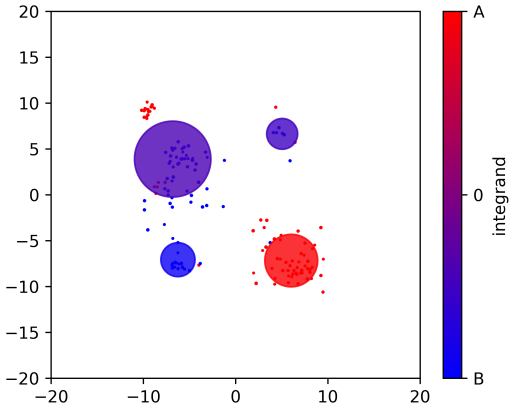
To illustrate these approaches, we generate two synthetic datasets $\mathcal{A}$ and $\mathcal{B}$ in a two-dimensional space. Each dataset consists of 150 data points randomly sampled from six Gaussian distributions, with the addition of uniform noise. We cluster the merged dataset into 4 clusters using each of the three methods.
In fig. 3.8a, we show the box-clustering method, where the space is divided into a grid of fixed-size boxes, and each box contains a certain number of points from $\mathcal{A}$ and $\mathcal{B}$. In fig. 3.8b, we apply KMeans, where the clusters are represented as circular regions centered at their respective centroids. Finally, in fig. 3.8c, we use FCM to produce fuzzy clusters, where the degree of membership of each data point is visualized using a color map.
In order to see the different perspective, we compare the different values obtained from box-clustering, KMeans and FCM.

Values	Box-Clustering	KMeans	FCM
$\lvert D_{\mathcal{A}} \cap D_{\mathcal{B}} \rvert$	4	4	2.889
$\lvert D_{\mathcal{A}} \cup D_{\mathcal{B}} \rvert$	4	4	3.980
$J_{D_{\mathcal{A}}, D_{\mathcal{B}}}$	1	1	0.7259
$(1 + J_{D_{\mathcal{A}}, D_{\mathcal{B}}})^{-1}$	0.5	0.5	0.5794
Integral	0.5368	0.4418	0.4980
$d(\mathcal{A}, \mathcal{B})$	0.268	0.221	0.289

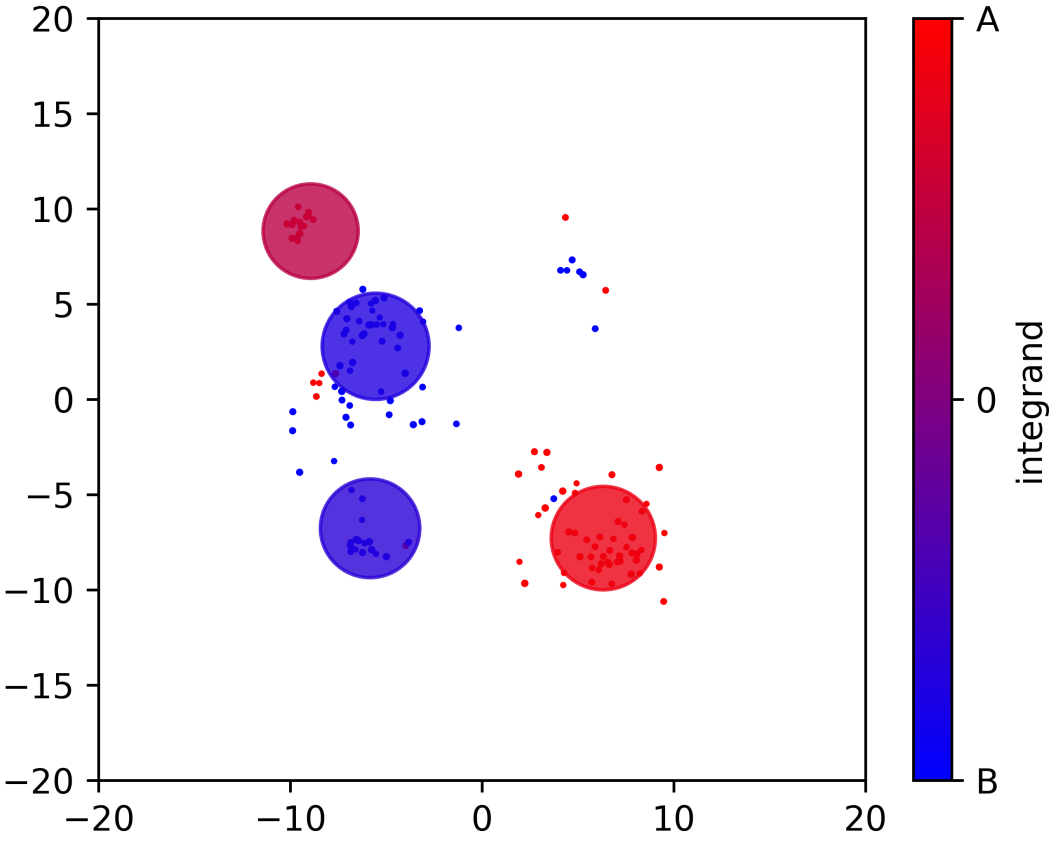
Table 3.2. Comparing the values obtained from the 3 algorithms, one can see how FCM manages to provide a similar result using fuzzy logic.



(a) Box clustering using 4 boxes. Computed distance: 0.268.



(b) KMeans clustering using 4 centroids. Computed distance: 0.221.



(c) FCM clustering using 4 centroids. Computed distance 0.289.

Figure 3.8. As can be seen, clusters are assigned a colour that is mapped to $[-1, 1]$ (-1 is blue, 1 is red). In particular, this colour is associated with the number $\frac{p_{\mathcal{A}}(c)-p_{\mathcal{B}}(c)}{p_{\mathcal{A}}(c)+p_{\mathcal{B}}(c)}$ for each cluster of centroid c. Furthermore, it is observed that KMeans and FCM have clusters represented by a circle with area $\mu(c)$ for the two clustering methods.

Exempli Gratia (Stability of FCM). Consider a high-dimensional example comparing two sets of samples drawn from two Gaussian distributions with the gradual addition of uniform noise.
The goal is to evaluate how noise affects the results obtained from FCM compared to KMeans.
Specifically, we draw two sets of samples from the Gaussian distributions $\mathcal{N}\left(-\vec{1}, \mathbb{1}\right)$ and $\mathcal{N}\left(\vec{1}, \mathbb{1}\right)$ in $\mathbb{R}^{16}$. Uniform noise with distribution $Unif[-5,5]^{16}$ is then added to the data.
Each dataset consists of 500 samples, combining points from the Gaussian distributions and the noise. The table below shows how the distances between the two distributions decrease as the noise density increases.

Noise Density	KMeans Distance	FCM Distance
0%	1.000	0.2975
1%	0.5669	0.2427
2%	0.4858	0.2411
5%	0.4076	0.2247
10%	0.0491	0.1921
20%	0.0054	0.1302

The results demonstrate that FCM is significantly more robust than KMeans, as it maintains a discernible distance between the two Gaussian distributions even in the presence of high noise density.

GPU The algorithm 3 shown is a sequential solution for FCM. It works by iterating through the data and centroids to compute the membership matrix. However, this approach has a computational cost of $O(NCk)$.
To improve computational efficiency, the use of GPU boosting can be employed. This kind of operation is known as General-Purpose computing on Graphics Processing Units (GPGPU). Exploiting the parallel computing power offered by a GPU can greatly accelerate the process of comparison.
The GPU is an electronic component present in every computer, able to perform a large number of operations in parallel. Originally designed to handle the graphical interface in video games, the GPU is capable of handling billions of pixels on any computer screen at speeds that the Central Processing Unit (CPU) cannot achieve. This processor is made up of thousands of threads, organised hierarchically at the hardware level to maximise performance:

i. Stream Multi-Processing (SM): Runs a kernel and consists of numerous warps;

ii. warp: Runs the kernel of its stream and has shared memory between its threads, usually 32;

iii. thread: Executes the kernel of its warp by synchronising with the other threads of the same warp and has its own reserved memory in its registers.

The memory that a GPU can access is divided into different categories, namely *global*, *shared*, *cache* and *register* memory. Access to these memories by the processor's threads depends on the hierarchy of the threads themselves. For example,

the *global* memory is accessible by every thread, while the *shared* memory is only accessible by threads in the same warp.
In a high-end computer, it is common to find a GPU with dozens of SM, each containing 64 or 128 cores. For example, if a process wants to use 1024 threads on a single SM, the GPU will divide these threads into 32 warps (each containing 32 threads). Each warp is executed in chunks of 64 threads at a time, meaning that 2 warps can be executed simultaneously per clock cycle, while the remaining warps are scheduled for execution in subsequent cycles. This total of tens of thousands of threads can execute the exact same code in parallel, permitting extremely fast processing of images and other operations requiring a high degree of parallelism.
Since the 2000s, the use of GPU has extended to the field of scientific computing, introducing important concepts such as scalability and High Performance Computing (HPC). Since 2020, dedicated GPU are available on the market for artificial intelligence operations.
In Python, there are useful frameworks for the utilisation of GPU, such as *torch* and *TensorFlow*, which are widely employed in the field of computer vision. However, also languages such as `C++` offer dialects that allow these powerful computing units to be exploited. In this paper, the CUDA dialect will be used. [4] [5]

The integration of GPGPU techniques would allow the workload to be distributed over several cores of the GPU, thus reducing the time needed for clustering operations. This method is particularly advantageous when handling large amounts of data, as the GPU can perform many operations in parallel, speed up computation to be 2000 times faster than the CPU could have done.
The sum of N numbers can be performed in with computational cost $O(log(N))$. This is because in parallel the GPU threads sum one half of the vector over another at the same instant and then repeat until they get a single component that will have only one number. This operation is called *reduction* and we can see it in the algorithm 6. This is just one detail of how the GPU can reduce the asymptotic computational cost of an algorithm. Suffice it to say that thanks to reductions and strong parallelism, it is possible to multiply two $N \times K$ and $K \times M$ matrices with cost $O(log(K))$ instead of $O(NMK)$. In clustering many operations can be parallelised and FCM in particular requires many sums and linear operations.
The limitations of GPU are not only related to the execution of the same operations on all threads, but also to the nature of these operations. Normally, an instruction takes much longer to be executed by a GPU than by a CPU. Arithmetic instructions are the most efficient, while the use of conditions tends to be avoided.

[4] for more details about GPU architecture, see
https://researchcomputing.princeton.edu/support/knowledge-base/gpu-computing
[5] for more details about CUDA language, see
https://docs.nvidia.com/cuda/

Algorithm 6 Parallel algorithm for sum reduction.
INPUT
v: array of values
N: number of components
This algorithm sum all values of an array and write in v[0] the result. The array is not preserved, in this way the algorithm does not allocate new memory. The computational cost is $O(\log(N))$. In fig. 3.9 an example over a vector with 7 components.

```
1: procedure KERNEL i, SUMREDUCTION(v,N)
2:     Let S a shared vector with 2^k ≥ N component
3:     S[i] ← v[i] if i < N else 0
4:     for L ← 2^k/2, 2^k/4, ..., 1 do
5:         if i < L then
6:             S[i] ← S[i] + S[i + L]
7:         end if
8:         require synchronisation between threads
9:     end for
10: end procedure
```

1	2	3	4	5	6	7
1+5=6	2+6=8	3+7=10	4	5	6	7
6+10=16	8+4=12	10	4	5	6	7
16+12=28	12	10	4	5	6	7

Figure 3.9. We want to calculate the sum of the values in the first row. The idea is to divide the vector into 2 regions, sum the components, and repeat over the new vector with half the size of the previous vector.

Chapter 4

Results

4.1 Architecture

As mentioned in the previous chapters, the image analysis system requires the comparison of an unknown work with known ones.
The image is pre-processed to remove non-handwriting artefacts, resulting in a greyscale image where the author's handwriting appears as dark tones on a white background. In addition, the thickness of the author's strokes may need to be adjusted relative to the printed characters, and the resolution (expressed in PPI) must match the dataset specifications.
The pre-processed image is then synthesised: a program extracts tiles of a certain size, according to the requirements of the dataset. The image is then compared with each image in the dataset (or with the most representative images for each author).
This chapter provides an analysis of each step outlined in this thesis, evaluating the impact of each component on the images studied. It also analyses and justifies the decisions and parameters chosen during the design phase.

Each image with a known author is fully compared to all other known images. Since the comparison value is symmetric, only half of all possible comparisons need to be evaluated. For n images, imagine filling a square $n \times n$ table with real values. Assuming a comparison value of 0 between identical images, the total number of comparisons required is $\frac{n^2-n}{2}$.
For this reason, the algorithm computes only half of the comparisons for each row, as shown in fig. 4.1. Specifically

- If n is odd, each row performs $\frac{n-1}{2}$ comparisons (with the remaining values derived by symmetry).
- If n is even, $\frac{n}{2}$ comparisons are performed for even-indexed rows (where the first row has index 0), while $\frac{n}{2} - 1$ comparisons are performed for odd-indexed rows.

This method provides a more uniform analysis for large datasets, especially when combined with row and column shuffling.

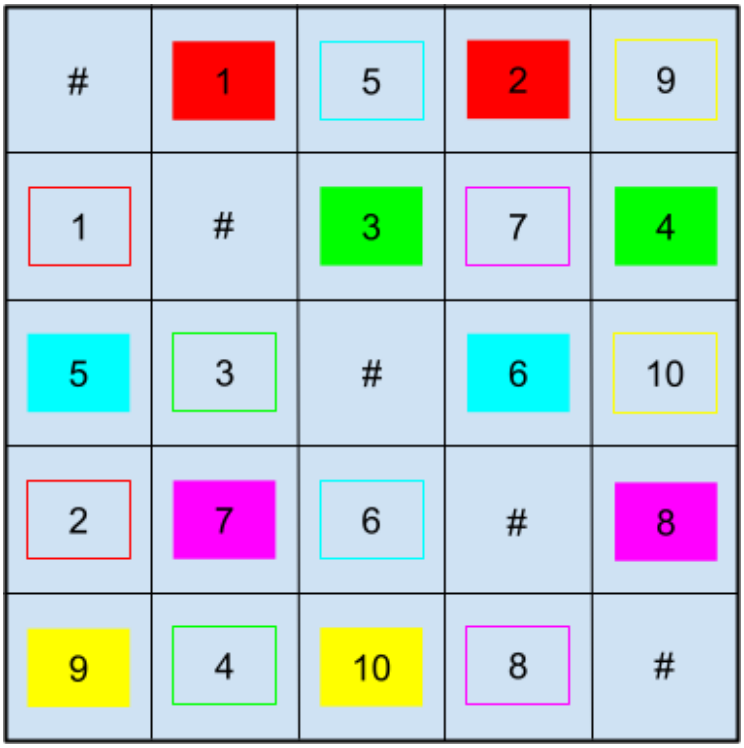


Figure 4.1. The algorithm calculates half of all components for each row. In the figure, the algorithm follows the order of the numbers, taking cells alternately so that the bottom diagonal is not directly computed and the top diagonal is directly computed. For example, in the row $3^{\text{-th}}$, the algorithm directly computes the first column (number 5 in cyan) and the penultimate column (number 6 in cyan), while the second column is already computed (number 3 in green square) and the last column is not yet computed.

4.1.1 Challenges

The implementation of the code presented significant challenges, as many of the required frameworks did not exist, and computational and time constraints posed additional hurdles. This necessitated the development of highly efficient and sophisticated code to optimise computational performance.
The project is heavily based on Python and CUDA. Specialised debugging, logging and testing software was used extensively to ensure that the software worked correctly. Automated documentation tools and version control systems were used to manage and track code development. In addition, a robust error handling and session management system was implemented, which included backup mechanisms (called "checkpoints") to prevent the loss of valuable data. Temporary files were also used to extend storage capacity beyond the limits of primary storage.
Due to its complexity and length (a total of almost 4000 lines), the code cannot be fully included in this document. However, it is available in the GitHub repository: `https://github.com/StefanoMagriniAlunno/MMATH_thesis`.

The code uses wrapping, which allows a Python script to interface with pre-compiled C++ code. This approach allows the Python script to handle high-level tasks such as preparing input files and reports, retrieving hardware specifications, and

configuring computing sessions. When intensive computations are required, the Python script calls precompiled C++ libraries, ensuring that the C++ code remains streamlined and more readable.
For example, when initiating a comparison between works, the main Python script prepares the operating system environment, validates the parameters used, and handles any existing checkpoints. It then passes all necessary input to the wrapper (written in C++), which validates syntax errors and translates Python types into C++-compatible types. The wrapper then calls a starter function which initialises logging mechanisms and reads input files, effectively setting up main memory. The C++ program then calls the CUDA program, which reads the GPU specifications (using dedicated libraries) and configures a computation plan to maximise performance and ensure robustness. Finally, the GPU receives kernel instructions that it compiles at runtime to perform the computations. Given the limited memory of GPUs, data is transferred in chunks, with main memory acting as a buffer.
The results computed by the GPU are returned to the calling C++ function, which stores them appropriately and returns control to the calling Python script. In case of errors, all relevant information is documented and passed back to the caller for appropriate handling.
These strategies for developing high-performance code are common in scientific computing libraries, which often have a main calling function with hidden compiled functions. An example is CuPy, a Python library that requires compilation as part of its installation process.

4.2 Pre processing

Preprocessing is a critical stage in the comparison of unknown works with the dataset. Its primary objective is to ensure that the quality specifications of the images meet certain standards. These specifications are often subjective, and it is not guaranteed that the same algorithm will be applied to each image. The ultimate goal is to remove clearly contaminating data, standardise the image quality and prepare it for analysis by the software. Essentially, pre-processing transforms the real data into an ideal format that is theoretically acceptable to the software.
As previously illustrated in algorithm 4, the preprocessing applied to the collected data primarily involves compression using the FFT. This section presents a step-by-step analysis of the results produced by the algorithm.

FFT The first step of the preprocessing algorithm is to apply FFT to the normalised image, where pixel values have a mean of 0 and a variance of 1. As noted in the summary images, the grid patterns in the background of the notebook pages are recurring features that are not typical of writing. These recurring patterns are more apparent when Fourier transforms are used.
In fig. 4.2 the grid pattern is visible as high-intensity frequencies almost aligned along the sides.

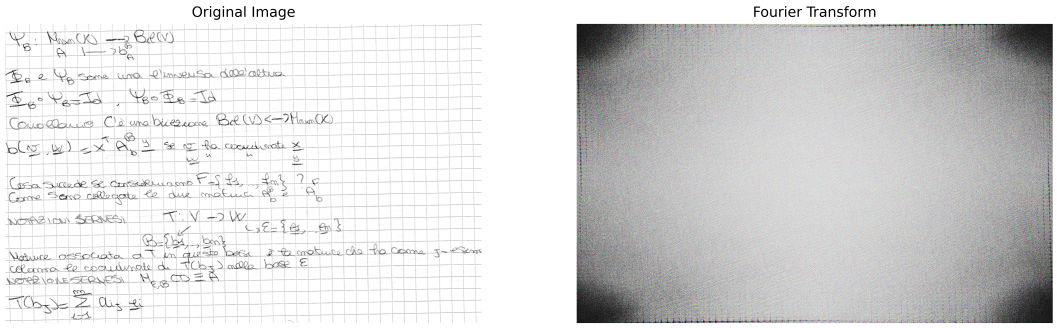


Figure 4.2. Significant frequencies along the edges of the second image reveal the grid pattern.

The algorithm removes the p percentile of the frequencies with the highest amplitude and reconstructs the original image using the remaining frequencies whose amplitudes are preserved. In fig. 4.3 different reconstructions using different percentiles are shown, with pixel intensities normalised between 0 and 1. The results suggest that $p = 0.1\%$ is a reasonable starting point for analysis.

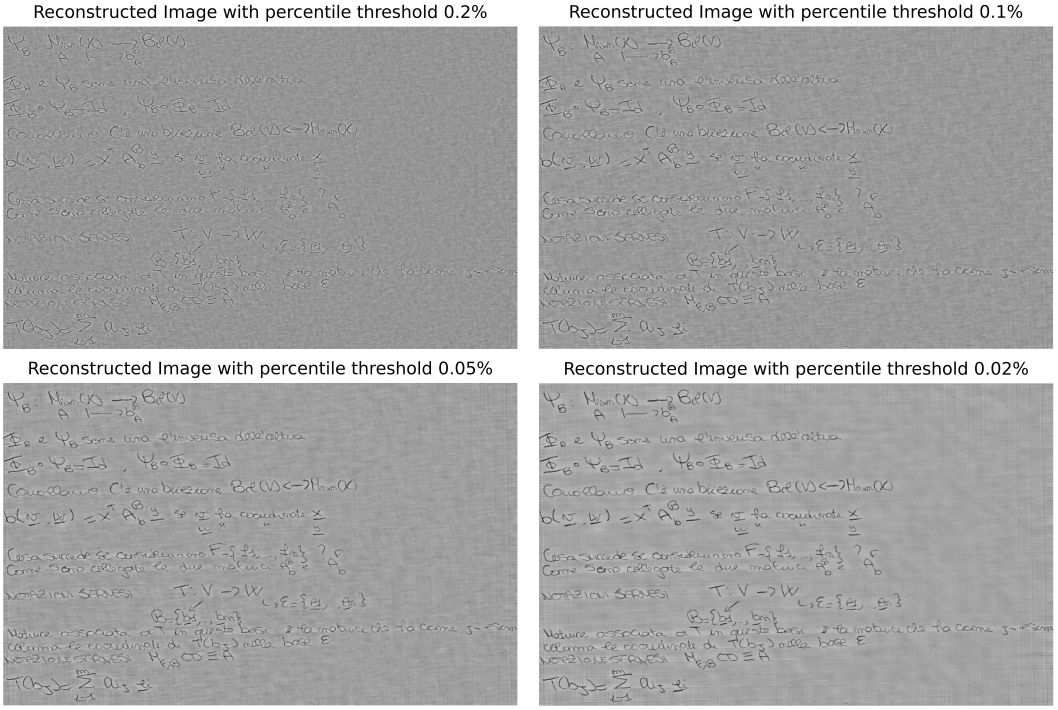


Figure 4.3. A percentile between 0.1% and 0.05% strikes a good balance by removing the grid patterns while leaving the handwriting intact.

While normalisation makes these images visually comparable, it masks the fact that the pixels in the reconstructed images tend to have very similar intensities. To overcome this, the algorithm identifies pixels that were certainly part of the white background or handwriting in the original image. Specifically

- Pixels with intensity below 0.2 are considered part of the handwriting.

- Pixels with intensity above 0.8 are considered part of the white background.

Using this analysis, the algorithm calculates how many pixels belong to the handwriting and how many belong to the background. fig. 4.4 illustrates this analysis.

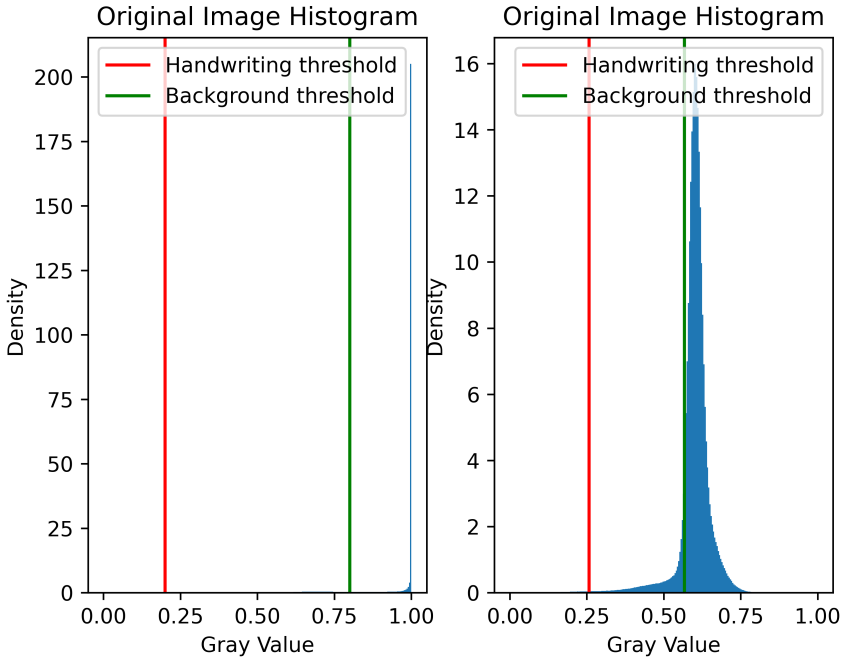


Figure 4.4. In the original image, most pixels are white, while the reconstructed image contains more mid-grey pixels. To preserve the original distribution, thresholds are computed based on the original image: pixels above 0.8 (green line) are assumed to be background, while pixels below 0.2 (red line) are assumed to be handwriting. These thresholds are used to define two new boundaries in the reconstructed image.

The algorithm identifies two thresholds in the reconstructed image:

- $\texttt{h}_{\texttt{up}}$: Above this intensity, the number of pixels matches (or slightly underestimates) the number of background pixels in the original image.
- $\texttt{h}_{\texttt{down}}$: Below this intensity, the number of pixels matches (or slightly underestimates) the number of handwriting pixels in the original image.

A linear transformation is applied to the pixel intensities in the reconstructed image to scale these thresholds to 1 and 0 respectively:

$$\texttt{gray}_{\texttt{new}} = \frac{\texttt{gray}_{\texttt{old}} - \texttt{h}_{\texttt{down}}}{\texttt{h}_{\texttt{up}} - \texttt{h}_{\texttt{down}}}$$

Pixel intensities outside the range $[0, 1]$ are clamped to 0 or 1 using the `clamp` function.

$$\texttt{clamp}\left(\texttt{gray}_{\texttt{new}}\right) = \begin{cases} 0 & \text{if} \quad \texttt{gray}_{\texttt{new}} < 0 \\ 1 & \text{if} \quad \texttt{gray}_{\texttt{new}} > 1 \\ \texttt{gray}_{\texttt{new}} & \text{otherwise} \end{cases}$$

The fig. 4.5 shows the results of this process for $p = 0.1\%$ and $p = 0.05\%$ on two samples. While the results appear visually similar, using $p = 0.05\%$ preserves the handwriting better, but at the cost of retaining some lattice artefacts. It is hoped that this residual noise will be captured by the FCM clustering.

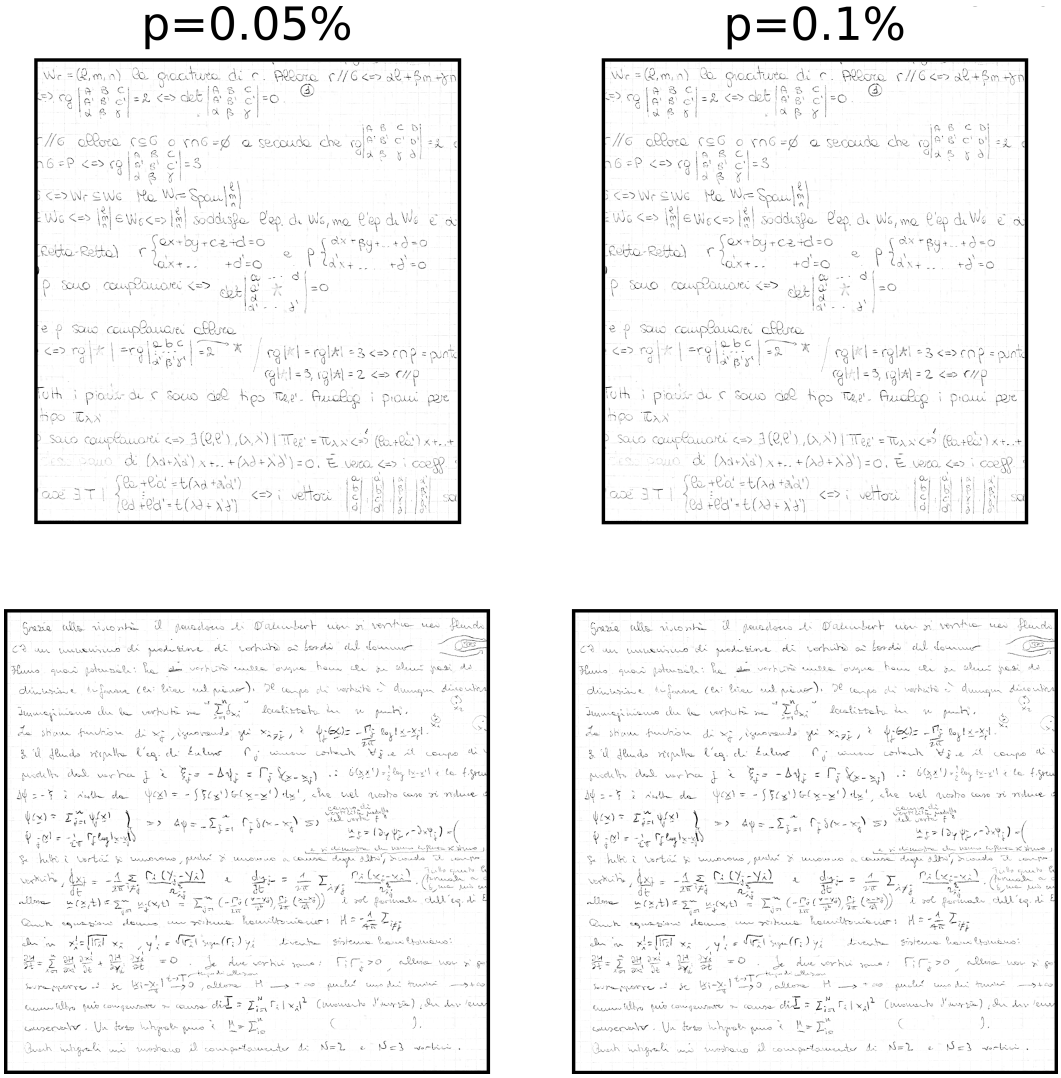


Figure 4.5. Two pre-processed images with different percentiles are shown. The first column corresponds to $p = 0.05\%$ and the second column corresponds to $p = 0.1\%$. Each row represents one work from the dataset.

4.3 Synthesis

Once the images have been pre-processed and adapted to the data set according to "human" parameters, the synthesis phase begins. This step ensures that the works are redefined in an objectively consistent manner. Specifically, the works are transformed into an (ordered or unordered) list of square tiles with uniform sizes. As mentioned above, the Python script manages the operating system and efficiently

retrieves input and hardware parameters for lower level pre-compiled programs. In this case, the `synthesis.py` script sets up the necessary directories to allow a C++ program to store the generated syntheses. This process is achieved using a simple code structure as shown below:

```
/** @brief In this code we use OpenMP API to realize an array with all tiles.
 *
 *  @param[in] byte_board : the array with all pixels
 *  @param[out] cells : the array with all tiles of the image
 *  @param n_threads : the number of used threads in the first loop
 *  @param height : the height of the image
 *  @param width : the width of the image
 *  @param n_tiles : the size of tiles
 *
 *  @details This code uses a bijective map from tiles and "short"image, i.e. an image with shape width_short, height_short. Each tiles will be assigned to its first pixel (up/sx).
 *  "omp parallel for" is a directive of OpenMP that executes a for cycle using more indipendent threads. In this case the schedule is "static" i.e. each thread knows in advance all its loops.
 */
unsigned width_short = width - n_tiles + 1,
         height_short = height - n_tiles + 1;
#pragma omp parallel for schedule(static) num_threads(n_threads)
for (unsigned row = 0; row < height_short; ++row)
for (unsigned col = 0; col < width_short; ++col)
  for (unsigned i = 0; i < n_tiles; ++i)
  for (unsigned j = 0; j < n_tiles; ++j) {
    long unsigned index_out = ((row * width_short + col) * n_tiles + i) * n_tiles + j; // [row-i, col-j, i,j]
    long unsigned index_in = (row + i) * width + col + j;
    cells[index_out] = byte_board[index_in];
    }
```

Using multiple threads to execute a loop significantly improves performance, achieving a multiplicative speedup. In addition, write access to `cells` for different `rows` does not raise concurrency issues, such as multiple threads attempting to write to the same memory location. In this case, the nominal execution time of the loop does not depend on the choice of row. For this reason, a static scheduling strategy greatly reduces the overhead associated with managing the set of generated threads. Multi-threading techniques are primarily used to optimise the execution time of small, parallelizable tasks. It is unusual to apply such techniques at a high level, such as analysing two images simultaneously. While starting multiple high-level processes can reduce overall startup times, it also increases memory requirements. The suitability of this method depends largely on the specific problem being solved. Since each image is transformed into a set of tiles, it is obvious that this can significantly increase memory usage. The number of elements in the `cells` array is given by:
$(w-n+1)(h-n+1)n^2$, where w and h are the dimensions of the original image and n is the size of the tiles. As a result, managing multiple high-level threads becomes challenging due to the significant memory requirements.

4.3.1 A Simple Analysis

Since syntheses are represented as sets of high-dimensional points, visualising them can be challenging. To address this, we propose a "coarse" visualisation technique

that reduces the dimensionality from $\mathbb{R}^{16}$ (for 4×4 tiles) to $\mathbb{R}^6$. This reduction is achieved using Principal Component Analysis (PCA), and only a subsample of 5000 tiles is visualised. It is important to note that this approach introduces a significant loss of both dimensionality and data quantity, so the proposed plots are purely illustrative.
As shown in fig. 4.6, there is a region where the tiles are densely clustered. The most reasonable hypothesis is that the background of the images produces many bright tiles that form a dense cluster. To confirm this hypothesis, these points are highlighted and linked to their original tiles.

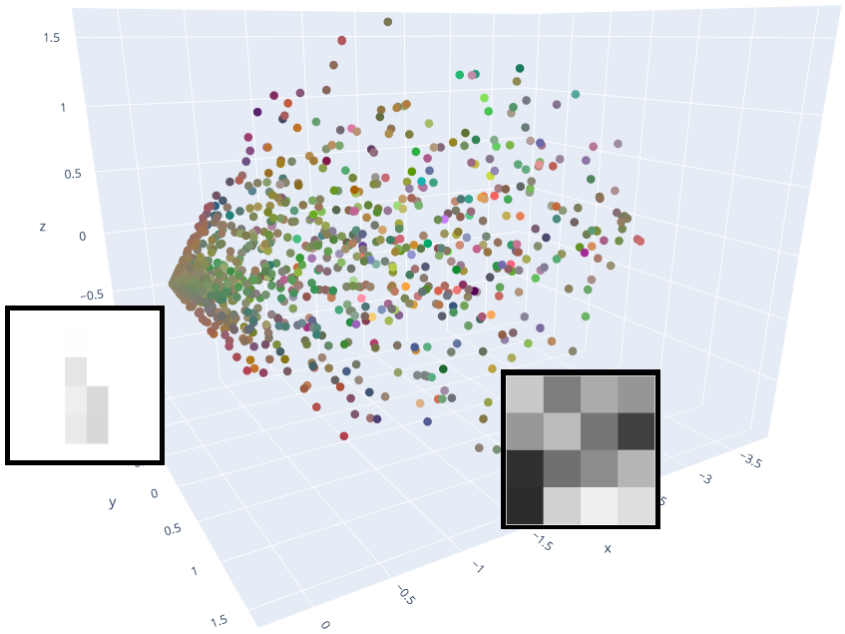


Figure 4.6. The figure shows a scatterplot of a sample of tiles extracted from an image. Six dimensions are represented using colours. As can be seen, the background tiles are mostly concentrated around 0, while the handwriting tiles are more scattered.

4.4 Comparison

The comparison between distributions defines the dissimilarity between works. Given the computational cost of this process, it is important to analyse the trade-off between the quality of the result and the time required, as well as the parameters chosen. Specifically, the following strategies are considered:

- Reducing the number of iterations used to calculate the centroid in FCM.
- Reduce the number of centroids in FCM.
- Reduce dimensionality using PCA before running FCM.
- Reducing the number of data points before running FCM.

4.4.1 Implementation

The implementation of the comparison algorithm is long and complex; only a general framework is presented here. Specifically, we focus on how the data is managed by the GPU, taking into account its physical limitations.
Examining the update formulas reveals the possibility of performing updates using batches. Due to the memory limitations of the GPU, it is not possible to compute a complete update in one step. Instead, partial computations must be accumulated in smaller batches of data.
The theoretical update formulae are as follows

$$C_j^{\text{new}} = \frac{\sum_{i=1}^{N} u_{ij}^2 w_i x_i}{\sum_{i=1}^{N} u_{ij}^2 w_i},$$
$$u_{ij} = \frac{1}{\sum_k \frac{d_{ij}^2}{d_{ik}^2}},$$
$$d_{ij} = \|x_i - C_j\|$$

A single iteration step can be further subdivided into steps across batches, indexed here by B:

$$C_j^{\text{new}} = \frac{\sum_B \sum_{i \in B} u_{ij}^2 w_i x_i}{\sum_B \sum_{i \in B} u_{ij}^2 w_i}$$

Consequently, the communication between the host (the programme flow executed by the CPU) and the device (the programme flow executed by the GPU) can be simplified as follows:

Algorithm 7 Host/Device communication
INPUT
• $\mathcal{X}$: set of data $x_1, \ldots, x_N \in \mathbb{R}^K$ and weights $w_1, \ldots, w_N$
• $\mathcal{C}$: centroids $C_1, \cdots, C_M$
• stop: stop criteria of the clustering (a positive value)

function FCMwrapper($\mathcal{C}$, $\mathcal{X}$, stop)
  $c^{\text{num}} \leftarrow$ array with M vectors in $\mathbb{R}^K$ into the Device
  $c^{\text{den}} \leftarrow$ array with M real values into the Device
  $\mathcal{C}'$ a copy of centroids $\mathcal{C}$ into the Device
  $s \leftarrow \infty$
  $L \leftarrow \infty$
  while $s >$ stop **do**
    Compute batch size
    $c^{\text{num}} \leftarrow \vec{0}$
    $c^{\text{den}} \leftarrow \vec{0}$
    $L_{\text{new}} \leftarrow 0$ ▷ Current value of loss function, see definition 2.2.2
    for each B batch **do**
      move B into the Device
      Call the data points of B as x
      Call the weights of B as w
      Compute U^2, D^2 between B and $\mathcal{C}'$ ▷ calling algorithm 3
      $L_{\text{batch}} \leftarrow \sum_{x \in B} \sum_{C \in \mathcal{C}'} u_{ij}^2 d_{ij}^2$
      $c_{\text{batch}}^{\text{num}} \leftarrow \sum_{i \in B} u_{ij}^2 w_i x_i$
      $c_{\text{batch}}^{\text{den}} \leftarrow \sum_{i \in B} u_{ij}^2 w_i$
      $c^{\text{num}} \leftarrow c^{\text{num}} + c_{\text{batch}}^{\text{num}}$
      $c^{\text{den}} \leftarrow c^{\text{den}} + c_{\text{batch}}^{\text{den}}$
      $L_{\text{new}} \leftarrow L_{\text{new}} + L_{\text{batch}}$
    end for
    $C'_j \leftarrow \frac{c_j^{\text{num}}}{c_j^{\text{den}}} \quad \forall j$
    $s \leftarrow L - L_{\text{new}}$
    $L \leftarrow L_{\text{new}}$
  end while
  move $\mathcal{C}'$ into the Host
  return $\mathcal{C}'$
end function

The computation of the U^2 matrix and L_{batch} is performed simultaneously, without the need to store D^2. This computation is performed by a CUDA kernel whose pseudocode is presented in algorithm 3, with implementation details in appendix B. The computational cost for each batch is $O(\log KM)$.

Hardware limitations may limit the size of the batch, both due to memory constraints and parallelism limits.
The use of GPU greatly improves performance. This has already been discussed in general terms in chapter 3. Here we examine it in detail, considering specific hardware limitations. While a GPU can perform many operations in parallel, it

cannot perform an infinite number of tasks simultaneously.

Examining algorithm 3, we observe that the computational cost for an unbounded hardware GPU is $O(\log KM)$. In particular:

1. The matrix D^2 can be computed with cost $O(\log K)$.
2. In the loop, each cycle is independent of the others and can be executed in parallel. The cost of a single loop will be:
   (a) $O(\log M)$ to calculate $l = \min_k D^2_{ik}$,
   (b) $O(1)$ to initialize U^2, due to parallelism,
   (c) $O(\log M)$ to initialize S_i,
   (d) $O(1)$ to complete the calculation of U^2 by parallelism.

However, in the presence of physical limits of GPU, the total cost changes. Denoting by threads the processes dedicated to synchronizable reductions and computations, and by core the parallel processes for independent computations, the total cost becomes $O\left(|B|MK\frac{\log \text{thread}}{\text{thread}\times\text{core}}\right)$ where $|B|$ denotes the batch size. In more detail:

1. The matrix D^2 can be computed with cost $O(|B|MK\frac{\log \text{thread}}{\text{thread}\times\text{core}})$.
2. In the loop, each loop is independent and can be parallelized on core processes. The cost of a single loop executed by a single core will be:
   (a) $O\left(M\frac{\log \text{thread}}{\text{thread}}\right)$ to compute $l = \min_k D^2_{ik}$,
   (b) $O\left(\frac{M}{\text{thread}}\right)$ to initialize U^2, treating the threads as independent processes,
   (c) $O\left(M\frac{\log \text{thread}}{\text{thread}}\right)$ to initialize S_i,
   (d) $O\left(\frac{M}{\text{thread}}\right)$ to complete the calculation of U^2.

   Consequently, the total computational cost of the loop is $O\left(|B|M\frac{\log \text{thread}}{\text{thread}\times\text{core}}\right)$.

Examining algorithm 7, we deduce that the innermost loop has all its computational cost determined by the call to algorithm 3. Therefore, the cost of the entire algorithm is:

$$O\left(INMK\frac{\log \text{thread}}{\text{thread}\times\text{core}}\right)$$

where:

- I is the number of iterations,
- N is the number of data point,
- M is the number of clusters, and
- K is the number of dimensions.

It is worth noting that the parallelism of the GPU does not asymptotically reduce the computational cost in terms of algorithmic parameters, since scalability limits are irrelevant from an asymptotic perspective. However, the practical efficiency gain is significant. On a CPU, a single iteration can take hours, whereas on a GPU it takes only a few seconds.
In addition, memory size does not significantly affect the asymptotic cost, but it can affect computation times due to internal optimisations by the CUDA compiler. These optimisations allocate the handling of large datasets to deeply nested loops, improving efficiency.
This asymptotic analysis makes it possible to evaluate the speed of the algorithm, taking into account hardware constraints. The table table 4.1 compares the expected algorithm performance on different processors. However, these estimates are not entirely indicative, as many technical factors such as memory size, driver optimisation and compiler performance significantly influence real-world results.

GPU	Num of cores	threads per core	Frequency	Efficiency
Intel Core i7	4	2	3300MHz	8
GTX 1650	14	64	1485MHz	1
RTX 3090	82	128	1395MHz	0.11
RTX 4090	128	128	2235MHz	0.04

Table 4.1. The values in the last column are the proportions of the operation $\frac{\log_2 \text{thread}}{\text{thread} \times \text{core}}$/Frequency for each GPU respecting the current GPU.

As highlighted in fig. 3.8c, the aim is to use FCM clustering to improve on the box-based clustering approach. If the tiles are in $\mathbb{R}^K$, it is possible to start with 2^K centroids, which is the number of boxes used in [5]. However, for $K = 36$ (i.e. tiles of shape 6×6), managing such a large number of clusters can be computationally demanding, even for preliminary testing. The adopted implementation supports up to 1024 centroids, constrained by hardware and the need for optimal performance. Consequently, the dimensionality K should not exceed 10.

4.4.2 Stability

Having discussed the implementation details of the algorithm, we now examine its stability. How does the calculated comparison value between works vary with different initialisations? A fast and noisy way to initialise the centroids is to randomly select M data points as centroids and then slightly perturb them.
To evaluate the stability, we compute the distance starting from 10 different initialisations, each perturbed with Gaussian noise of variance 0.01. The tile dimensions are 3×3, obtained by setting the resolution to 200 PPI instead of the original 400 PPI. This adjustment ensures that the coverage of 3×3 tiles is equivalent to that of 6×6 tiles at 400 PPI. The number of centroids is 512.
The table 4.2 compares the results between works using the stopping criterion 10^{-6} and 10^{-8}.

stop criteria			
10^{-6}	10^{-6}	10^{-8}	10^{-8}
0.00233	0.00188	0.00279	0.00074
0.00231	0.00194	0.00289	0.00076
0.00232	0.00201	0.00302	0.00076
0.00206	0.00197	0.00308	0.00075
0.00216	0.00198	0.00293	0.00076
0.00211	0.00212	0.00306	0.00073
0.00213	0.00211	0.00310	0.00081
0.00183	0.00203	0.00289	0.00072
0.00234	0.00208	0.00309	0.00072
0.00182	0.00208	0.00303	0.00076

Table 4.2. This table shows the distances following 10 repeated calculations with different stops. It can be seen that a higher stop criterion implies less accurate results and that in general for `stop` $= 10^{-8}$ the values remain fairly stable except for a few outliers.

4.4.3 Tuning

Having established that the stability of the algorithm depends on the stopping criterion, we now examine how the computed distance varies with the number of centroids and the dimensionality of the data. In particular, we set a stopping criterion of 10^{-9} in order to obtain more stable results than those previously observed. This section examines how to determine an appropriate number of centroids, and evaluates the effects of applying PCA.

Number of centroids The number of centroids can be analysed using the Jaccard index calculated for the comparison values. A study of tens of thousands of comparisons showed that the cardinality $|D_{\mathcal{A}} \cup D_{\mathcal{B}}|$ rarely exceeded 64 and never exceeded 128. This suggests that the clustering used as the basis for the distance calculation does not fully utilise all centroids. Instead, it is likely to be based on fewer than 32 centroids, or only partially using the centroids that are available.
In table 4.3 we show how the calculated result between works changes when the number of centroids is varied from 64 to 512. The results show that it is not necessary to use 512 centroids; 64 or 128 centroids suffice and produce results that are not significantly different from those obtained with 512 centroids. In particular, 128 centroids seem to provide an acceptable balance between accuracy and computational efficiency.

number of centroids	64	128	256	512
$\lvert D_A \cup D_B \rvert$	9.2	10.9	12.2	19.0
comparison value	0.00228	0.00241	0.00237	0.00222
$\lvert D_A \cup D_B \rvert$	12.5	14.0	15.4	17.0
comparison value	0.01175	0.01208	0.01219	0.01230
$\lvert D_A \cup D_B \rvert$	6.5	6.9	10.2	12.1
comparison value	0.01531	0.01539	0.01436	0.01482
$\lvert D_A \cup D_B \rvert$	15.3	17.7	22.2	30.0
comparison value	0.01351	0.01347	0.01351	0.01356
number of centroids	64	96	128	512
$\lvert D_A \cup D_B \rvert$	15.8	17.0	19.0	27.5
comparison value	0.01003	0.00986	0.01022	0.01019

Table 4.3. Comparison values for different numbers of centroids between 2 works, using hyperparameters: `stop` $= 10^{-9}$, tiles with shape 3×3 from images with 200 PPI. In particular, there are 5 tests and generally the results are stable.

Dimensions Another important aspect to consider is the dimensionality of the data and whether it can be reduced using PCA. This section analyses how the calculated comparison value changes as the dimensionality is reduced. Instead of extracting 3×3 tiles from images at 200 PPI, we reduce the dimensionality of 6×6 tiles from images at 400 PPI to 9 dimensions using PCA, down from the original 36. While this preserves the higher resolution of the tiles, it also filters out noise. However, excessive dimensionality reduction risks losing rare data points that are crucial for the algorithm to determine the comparison value between works.
Another key consideration is the **plateau effect**. Essentially, reducing the dimensionality without adjusting the number of centroids or data points can lead to an increased number of iterations before the stopping criterion is met. In other words, the clustering algorithm struggles to determine the optimal position of the centroids. Paradoxically, in the computational cost $O(INMK)$, even though K decreases, I increases, sometimes worsening the overall computational cost.
In table 4.4 we compare the results obtained using 3×3 tiles extracted from images at 200 PPI with those obtained using 6×6 tiles extracted from images at 400 PPI, where the dimensionality of the tiles has been reduced from $\mathbb{R}^{36}$ to $\mathbb{R}^9$ using PCA.

size of tiles			
3×3	6×6	3×3	6×6
0.00178	0.00261	0.01554	0.00340
0.01074	0.00000	0.01836	0.02821
0.01016	0.00435	0.01056	0.01203
0.00618	0.01518	0.00285	0.00639
0.01829	0.00559	0.00368	0.00453

Table 4.4. Comparison values for different dimensionalities were compared between 10 works using two approaches: 3×3 tiles extracted from images at 200 PPI, and 6×6 tiles extracted from images at 400 PPI with dimensionality reduced from 36 to 9 using PCA. A significant difference in the results was observed between the two methods. The reduction in dimensionality introduced noticeable changes in the comparison values, particularly when using 6×6 tiles, suggesting that the dimensionality reduction method and resolution choice both influence the final outcome.

Conclusion These experiments demonstrate that using 128 centroids yields results comparable to those obtained with 512 centroids, while reducing computational costs by a factor of 4. In addition, it was observed that using 6×6 tiles extracted from images at 400 PPI and reducing their dimensionality from 36 to 9 using PCA leads to different results compared to the original dimensionality. This indicates that dimensionality reduction causes considerable changes in the results, highlighting the importance of carefully choosing the resolution and reduction parameters. It was also observed that using PCA allows the dimensionality of 6×6 tiles to be reduced to 10, enabling the use of 1024 initial clusters for FCM. Repeating the stability and optimal centroid detection analyses for dimensionality reduction from 36 to 10, it was found that 128 clusters are sufficient.

4.4.4 pre-Test

Based on the previous analysis, comparisons between works can be performed using 128 clusters and PCA to reduce the dimensionality from 36 to 10. However, before performing large-scale comparisons across all possible pairs of works, it is prudent to first evaluate these hyperparameters in terms of stability and attribution quality on smaller samples. This preliminary examination will also ensure that the stopping criterion remains consistent.

Test 1 The first test involves recalculating the stopping criterion, as the reduction in dimensionality and the number of centroids may allow for fewer iterations to achieve convergence.

stop criteria			
10^{-8}	10^{-8}	10^{-9}	10^{-9}
0.00756	0.03902	0.00984	0.02824
0.00752	0.03913	0.00993	0.02810
0.00753	0.03906	0.00984	0.02805
0.00740	0.03883	0.00993	0.02805
0.00750	0.03895	0.00995	0.02809
0.00752	0.03900	0.00999	0.02806
0.00750	0.03908	0.00994	0.02824
0.00751	0.03879	0.00987	0.02775
0.00744	0.03906	0.00993	0.02782
0.00746	0.03894	0.00994	0.02792

Table 4.5. This table shows the comparison values after repeated 10 calculations with different stops. As can be seen, 10^{-9} is a stop criterion that gives more than stable results.

Test 2 This test compares the attribution quality using 3×3 tiles extracted from images at 200 PPI and 6×6 tiles extracted from images at 400 PPI, subsequently embedded into a $\mathbb{R}^{10}$ space via PCA. A small portion of comparison values between works were computed. For each work, the attributed author was the one associated with the "closest" work, i.e., the work yielding the smallest comparison value. While this method is not fully reliable and would require more detailed analyses, it serves as a good indicator of the model's adaptability to real-world scenarios.

Author	1	2	3	4
1	36	2	2	1
2	0	5	0	0
3	3	0	2	0
4	8	0	0	4

Table 4.6. This table shows the confusion matrix of attributions using 3×3 tiles. Each row represents the known author of the work, while each column indicates the attributed author. In this test, only 13 169 comparisons were performed and only the works with at least 40 computed comparisons are shown. It can be observed that the dominant author (first author, with 270 works) has a relatively low number of false positives (23%) and false negatives (12%).

Author	1	2	3	4
1	51	1	1	1
2	4	5	0	2
3	5	0	4	1
4	2	1	0	5

Table 4.7. This table shows the confusion matrix of attributions using 6×6 tiles from an image of 400PPI with PCA from $\mathbb{R}^{36}$ to $\mathbb{R}^{10}$. Each row represents the known author of the work, while each column indicates the attributed author. In this test, only 5982 comparisons were performed and only the works with at least 40 computed comparisons are shown. It can be observed that the dominant author (first author, with 270 works) has a relatively low number of false positives (18%) and false negatives (6%).

We observe that using PCA to reduce the dimensionality of the tiles is a qualitatively better approach. However, it is important to note that the computational costs are approximately four times higher. This increase is mainly due to the fact that the number of tiles extracted from an image is almost quadrupled (and at least tripled asymptotically). In particular, $(2w - 2n + 1)(2h - 2n + 1)$ tiles were generated, where w and h represent the image size (or pixel grid) with a resolution of 200 PPI compared to $(w - n + 1)(h - n + 1)$ tiles extracted directly from the image with 200 PPI.
In terms of computation time, 5982 comparisons took about four days of computation, while 13169 comparisons were completed in about two days, highlighting the different efficiency in relation to the volume of data processed.
This highlights how the choice of method must balance the quality of the results with the computational costs, depending on the specific objectives of the analysis.

4.4.5 Monte Carlo Test

This test compares the works using images with the following parameters:

- resolution of 400 PPI,
- tiles of size 6×6,
- a PCA that reduces the dimensionality of the tiles from $\mathbb{R}^{36}$ to $\mathbb{R}^{10}$,
- 128 centroids for the FCM algorithm,
- a stopping criterion of 10^{-9} for the energy update in the FCM algorithm.

In addition, to reduce the computational time, a sampling method was applied to each set of tiles for each comparison, accepting a potential increase in imprecision. Specifically, half of the available tiles were randomly selected for each comparison. The aim of this method is to see if the analysis can produce qualitatively similar results while reducing the computational time by half (or even more, as it may also reduce the number of iterations required).

Appendix A

KMeans, GMM, FCM compare figures

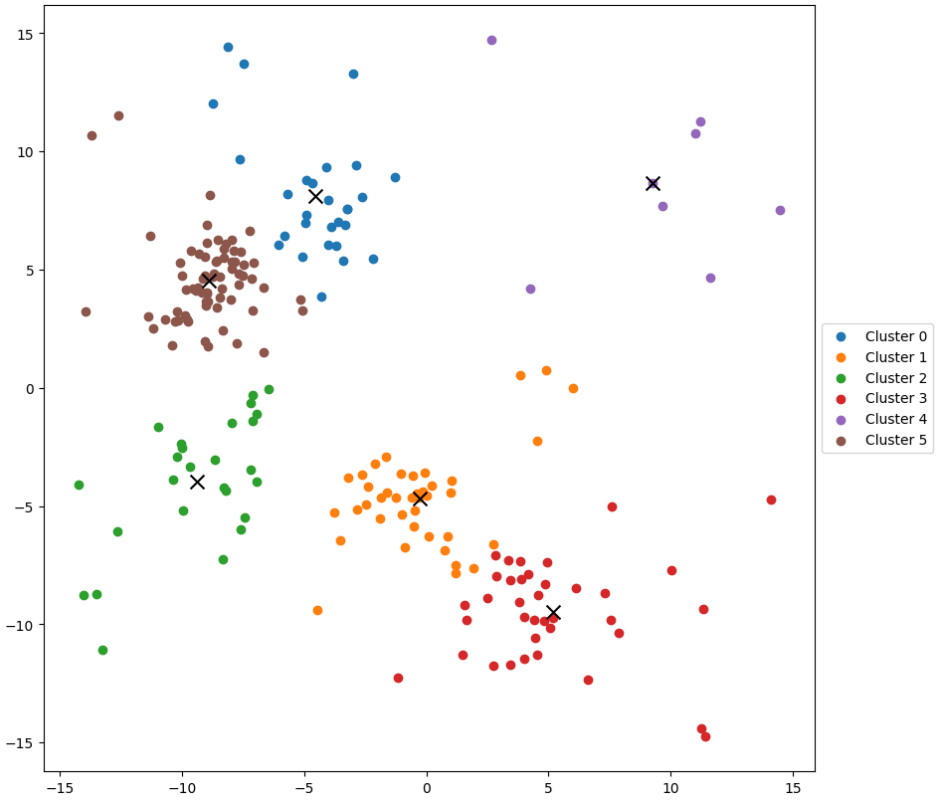


Figure A.1. The data points are coloured according to the calculated label and the estimated centroid is indicated with an ×.

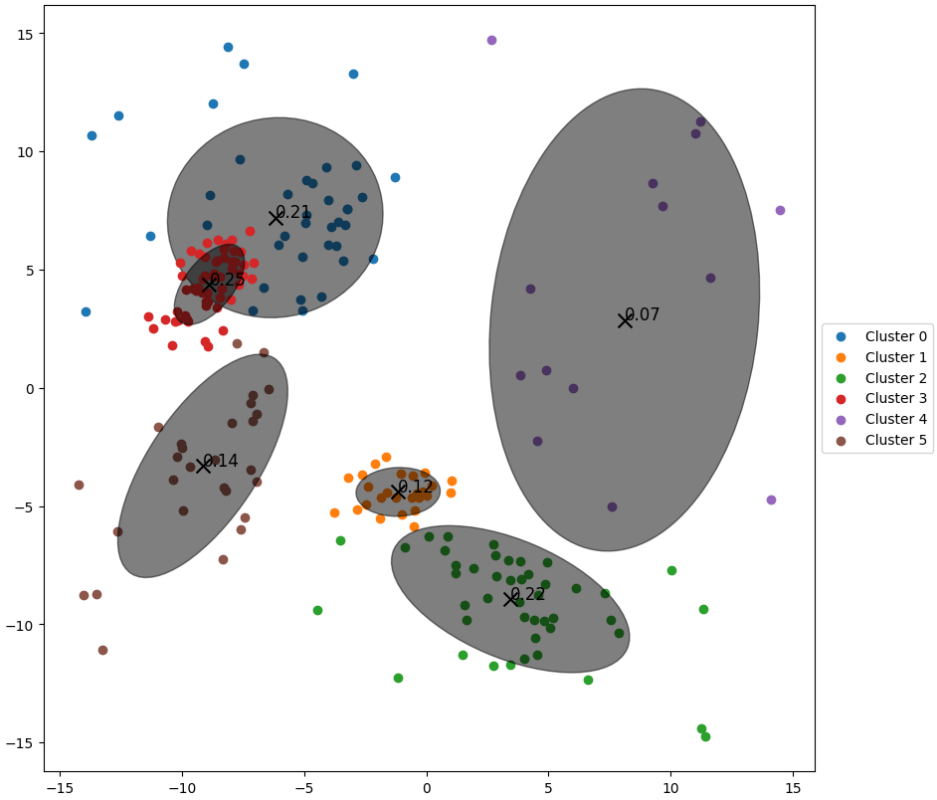


Figure A.2. The data points are coloured according to the Mahalanobis distance and the estimated centroid is indicated with an ×. The ellipse of the normal distribution represents the covariance and is a confidence region of 95%.

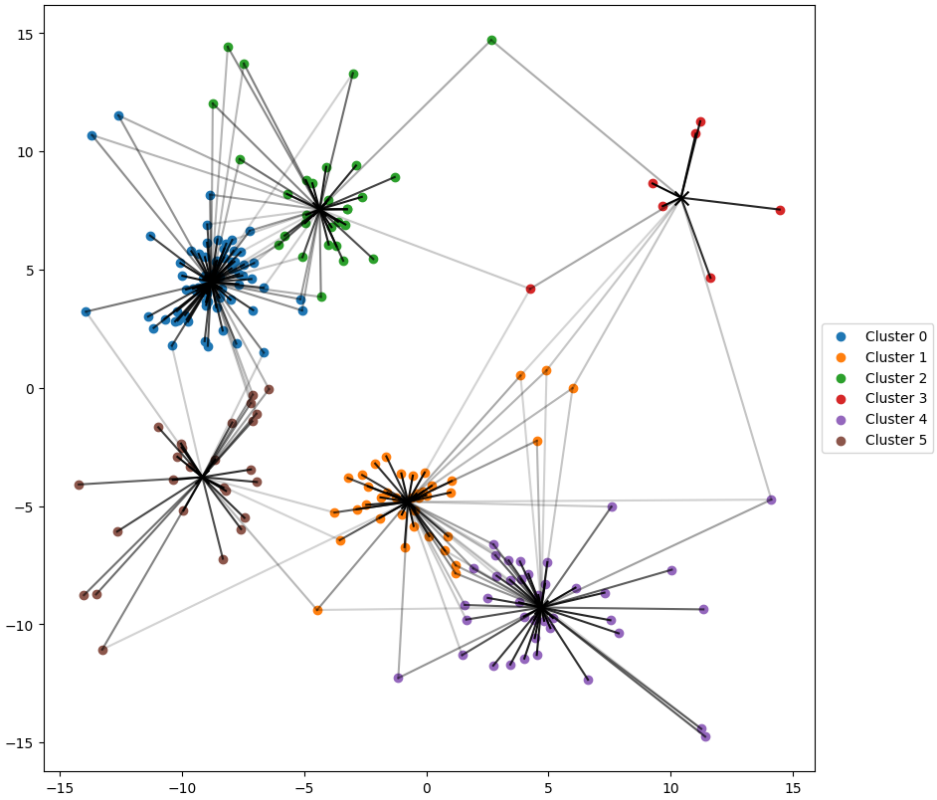


Figure A.3. The data points are coloured according to the most probable label and the estimated centroid is indicated with an ×. The lines indicate the assignments with the greatest degree of affiliation of each point to the clusters; the darker the lines, the stronger the assignment.

Appendix B

FCM implementation with CUDA

CUDA is a hardware architecture supported by the graphics processors of *NVIDIA*, a US company. It is possible to implement the code that defines the communication between CPU and GPU in a dialect of C++. The code that actually executes the graphics processor can already be implemented with special functions of the thrust and cuBLAS libraries or realised by means of kernels. In this appendix we show the kernel used to realise the calculation of the u^2 matrix in FCM. In particular, the algorithm is optimised to perform the calculation on at most `MAX_THREAD_PER_BLOCK` centroids, which provides the highest performance since the hardware properties of GPU can be used to the fullest.

```
/**
* @brief This kernel computes the matrix U2 of membership between
* data points and centroids
*
* @param[in] d_data : the i-th is d_data[i * n_dimensions + k]
* for k = 0, ..., n_dimensions - 1
* @param[in] d_weights : the weight of the i-th data point is
* d_weights[i]
* @param[in] d_centroids : the j-th is
* d_centroids[j * n_dimensions + k] for k = 0, ..., n_dimensions - 1
* @param[out] d_matrix : the weighted membership between the i-th data point
* and the j-th centroid is stored in d_matrix[i * n_centroids + j]
* @param[out] d_energies : the energy of the i-th data point is stored in
* d_energies[i]
* @param n_data : number of data points
* @param n_dimensions : dimensions of data points
* @param n_centroids : number of centroids
*
* @details This kernel requires a grid of blocks with n_data blocks
* and MAX_THREADS_PER_BLOCK threads for each block.
*
* @note This kernel synchronize threads at the end of the computation
*/
__global__ void
kernel_compute_U2 (const float *const d_data, const float *const d_weights,
const float *const d_centroids, float *const d_matrix,
float *const d_energies, size_t n_data, size_t n_dimensions,
size_t n_centroids)
{
  __shared__ float sdata[MAX_THREADS_PER_BLOCK];
  size_t i = blockIdx.x;  // i-th data
  size_t j = threadIdx.x; // j-th centroid
```

```
33  float value = 0;
34  float reduction_solution = 0;
35  float d2 = 0;
36
37  // compute the distance between the i-th data point and the j-th
38  // centroid
39  if (i < n_data && j < n_centroids)
40  {
41    for (size_t k = 0; k < n_dimensions; k++)
42    {
43      float diff = d_data[i * n_dimensions + k]
44      - d_centroids[j * n_dimensions + k];
45      value += diff * diff;
46    }
47  }
48  d2 = value;
49  // syncronyze threads of this block
50  __syncthreads ();
51
52  // compute the min value of the block
53  if (j < n_centroids)
54  sdata[j] = value;
55  else
56  sdata[j] = FLT_MAX;
57  __syncthreads ();
58  for (size_t s = MAX_THREADS_PER_BLOCK / 2; s > 0; s >>= 1)
59  {
60    if (j < s && sdata[j] > sdata[j + s])
61    sdata[j] = sdata[j + s];
62    __syncthreads ();
63  }
64  reduction_solution = sdata[0];
65  // syncronyze threads of this block
66  __syncthreads ();
67
68  // prepare the row to a stable normalization
69  if (reduction_solution == 0.0)
70  {
71    // let to 1 the components that are 0 and to 0 the others
72    if (i < n_data && j < n_centroids)
73    value = value == 0.0 ? 1.0 : 0.0;
74  }
75  else
76  {
77    // for each component of the row, assign min/value
78    if (i < n_data && j < n_centroids)
79    value = reduction_solution / value;
80  }
81  // syncronyze threads of this block
82  __syncthreads ();
83
84  // compute the sum of the row
85  if (j < n_centroids)
86    sdata[j] = value;
87  else
88    sdata[j] = 0.0;
89  __syncthreads ();
90  for (size_t s = MAX_THREADS_PER_BLOCK / 2; s > 0; s >>= 1)
91  {
92    if (j < s)
93      sdata[j] += sdata[j + s];
94    __syncthreads ();
95  }
96  min_value = sdata[0];
97  // syncronyze threads of this block
98  __syncthreads ();
```

```
99
100   // assign the value to the matrix
101   if (i < n_data && j < n_centroids)
102   {
103     value /= min_value;
104     d_matrix[i * n_centroids + j] = value * value * d_weights[i];
105   }
106   // syncronyze threads of this block
107   __syncthreads ();
108
109   // compute energy
110   if (i < n_data && j < n_centroids)
111     value = d_matrix[i * n_centroids + j] * d2;  // compute partial energy
112   // syncronyze threads of this block
113   __syncthreads ();
114
115   // compute the sum of the partial energies
116   if (j < n_centroids)
117     sdata[j] = value;
118   else
119     sdata[j] = 0.0;
120   __syncthreads ();
121   for (size_t s = MAX_THREADS_PER_BLOCK / 2; s > 0; s >>= 1)
122   {
123     if (j < s)
124     sdata[j] += sdata[j + s];
125     __syncthreads ();
126   }
127   value = sdata[0];  // energy of the data point
128   // syncronyze threads of this block
129   __syncthreads ();
130
131   // assign the energy to the matrix
132   if (i < n_data && j == 0)
133     d_energies[i] = value;
134   // syncronyze threads of this block
135   __syncthreads ();
136 }
```

Glossary

C++ A general-purpose computer language widely used to build high-performance code for embedded systems. 58, 59, 63, 77

CUDA Dialect of the C++ language for handling the video card. 58

GitHub GitHub is a platform for version control and collaboration, allowing developers to host, share, and manage code using Git. 58

KMeans K-means is an unsupervised clustering algorithm that divides a data set into k distinct groups based on distance. Each group is defined by its centroid, and the goal of the algorithm is to minimise the variance within the groups. v, 2, 5, 11, 14–16, 20–22, 30–32, 48, 49, 52–54

Python A high-level, general-purpose programming language. 5, 55, 58, 59, 62

R A computer language used mostly for statistical analysis. Includes not only many features but also datasets. 29

boost In computer science, "boost" generally refers to enhancing the performance of a system, application, or algorithm through code optimization, leveraging more efficient hardware resources, or implementing advanced techniques. 54

core A core is a phisical group of threads, in this context it is inside the GPU. 55, 67, 68

cuBLAS CUDA-Toolkit library specializing in linear computation and data management on graphics processors. 77

png The PNG (Portable Network Graphics) format is a raster image file format used for lossless compression. This means that images saved in this format do not lose quality or detail during compression. 39

thread A thread is a fundamental unit of execution in computing, allowing a program to perform multiple tasks concurrently, akin to different lines of thought in a mathematical problem-solving process. 54, 55, 63, 67, 68

thrust library of C++ that can also handle data through the graphics processor. 77

warp A warp is a group of threads executed together in a synchronized manner within a GPU, allowing for efficient parallel processing of data. 54, 55

Bibliography

[1] Chiara Basile, Dario Benedetto, Emanuele Caglioti, and Mirko Degli Esposti. An example of mathematical authorship attribution. *Journal of Mathematical Physics*, 49, 12 2008. doi: 10.1063/1.2996507.

[2] Tukey J. W. Cooley J. W. An algorithm for the machine calculation of complex fourier series. pages 297–301, 1965. URL https://www.jstor.org/stable/2003354.

[3] J. C. Dunn. A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters. *Journal of Cybernetics*, 3(3):32–57, 1973. doi: 10.1080/01969727308546046. URL https://doi.org/10.1080/01969727308546046.

[4] I. B. HOPLEY. Clerk maxwell's experiments on colour vision. *Science Progress (1933-)*, 48(189):46–66, 1960. ISSN 00368504, 20477163. URL http://www.jstor.org/stable/43424831.

[5] Stefano Magrini Alunno. Analisi automatica di disegni per attribuzione d'autore. Bachelor's thesis, La Sapienza, March 2021. Available at https://www.linkedin.com/feed/update/urn:li:activity:7182499324092694529/.

[6] Daniel Jurafsky & James H. Martin. An introduction to natural language processing, computational linguistics, and speech recognition. 2024. URL https://web.stanford.edu/~jurafsky/slp3/ed3bookfeb3_2024.pdf.

[7] Joseph Nicéphore Niépce. View from the window at le gras. https://upload.wikimedia.org/wikipedia/commons/5/5c/View_from_the_Window_at_Le_Gras%2C_Joseph_Nic%C3%A9phore_Ni%C3%A9pce.jpg, c.1826. public domain.

[8] Lawrence R. Rabiner, Ronald W. Schafer, and Charles M. Rader. The chirp z-transform algorithm and its application. *The Bell System Technical Journal*, 48(5):1249–1292, 1969. doi: 10.1002/j.1538-7305.1969.tb04268.x.

[9] Jean François WITZ. Illustrazione di una fotocamera digitale di tipo reflex. https://upload.wikimedia.org/wikipedia/commons/3/31/Reflex_camera_numeric.svg, 2008. CC BY-SA 3.0 Deed.