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

© 2024 Stefano Magrini Alunno. All rights reserved

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

Version: December 2, 2024

Author's email: stefanomagrini99@gmail.com

Mater artium necessitas

Contents

List of Figures

Special contents

Chapter 1

Literature Review

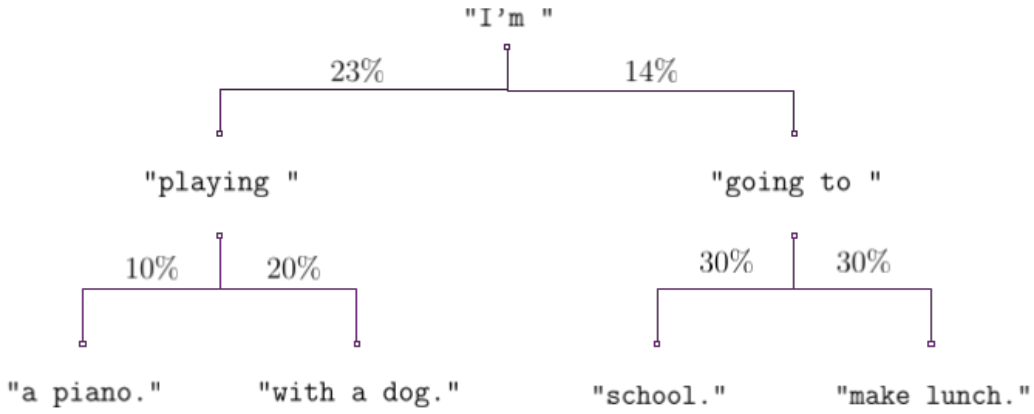
To Review

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.

1.1 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 However, as language modeling techniques have advanced, the n-gram model has largely been replaced by more sophisticated models, such as transformers[1].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 [5], 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 [4], 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)



1.1.1 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.
This probabilistic information can then be used to estimate the likelihood of subsequent characters appearing, with a certain margin of error. The model assumes that:

$$\mathbb{P}[w_{n+1}|w_1\dots w_n] = \mathbb{P}[w_{n+1}|w_2\dots w_n] \tag{1.1}$$

From this assumption we derive the following property:

Proposition 1.1.1. *Let $w_1,\dots,w_{n-1},w_n\dots,w_{n+k}$ be a sequence of characters. We have that:*

$$\mathbb{P}[w_n,\dots,w_{n+k}|w_1,\dots,w_{n-1}] = \prod_{i=0}^{k}\mathbb{P}[w_{n+i}|w_{i+1}\dots w_{n+i-1}] \tag{1.2}$$

Proof. This property is proved by using Bayes' theorem and the assumption of the model eq. (1.1):

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

□

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:

$$d(A,B) = \frac{1}{|D_A| + |D_B|} \sum_x \left(\frac{f_A(x) - f_B(x)}{f_A(x) + f_B(x)} \right)^2 \tag{1.3}$$

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 [4], 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:

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

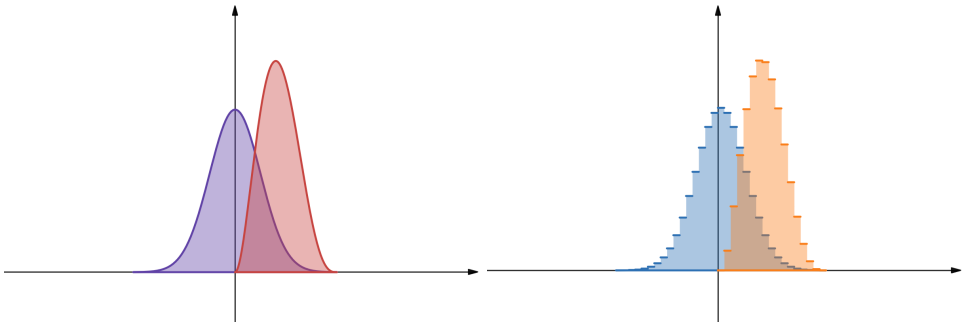
1.1.2 Implications

We conclude this section by discussing the implications of this theory. As seen in eq. (1.1), 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, 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 [4], 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 distance defined in eq. (1.3), particularly focusing on the problem of sparsity in n-grams.

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

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. (1.3) 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 [4], 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. 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.

1.2 Fuzzy clustering

Fuzzy clustering, also known as soft k-means, is a data analysis technique that allows data points to be assigned to more than one cluster. Unlike traditional "hard" clustering, where each data point is assigned exclusively to a single cluster, fuzzy clustering employs fuzzy logic to determine the degree of membership for each data point within multiple clusters. Instead of a binary membership statement (1 or 0), fuzzy clustering provides 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 k-means clustering algorithm in 1973. Dunn emphasized the increased accuracy and robustness of fuzzy clustering compared to hard clustering, particularly in handling anomalous data (outliers) [2].
The algorithm proposed by [2] 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.

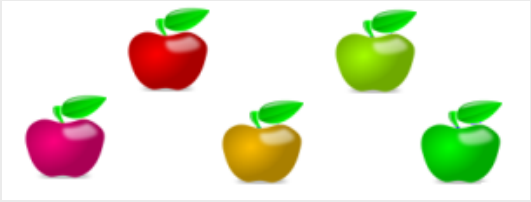
1.2.1 Insights

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.
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 weight matrix with
  $U = (u_{ij})$ dove $u_{ij} = \mu_{x_i}(C_j)$
- the quadratic Euclidean distance matrix with
  $D^2 = (d_{ij}^2)$ where $d_{ij} = \|x_i - C_j\|^2$

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 1.2.1 (membership measure)**.** Given a point x and a cluster C, the membership measure $\mu_x(C)$ is proportional to the inverse of the square of the Euclidean distance between the point x and the centroid of the cluster C, i.e. $\frac{1}{\|x-C\|^2}$. We assume that the sum of the membership measures of x to all clusters in the set $\mathcal{C}$ is equal to 1:

$$\sum_{C \in \mathcal{C}} \mu_x(C) = 1 \quad \forall x$$

Consequently, the membership measure u_{ij} of point x_i to cluster C_j is computed as:

$$u_{ij} := \mu_{x_i}(C_j) = \frac{1}{\sum_k \frac{d_{ij}^2}{d_{ik}^2}}$$

Remark. The KMeans algorithm, which is based on Boolean logic, seeks to minimise the sum of the squared distances between each data point and the centroid of the cluster to which it is assigned. In other words, the goal of the KMeans algorithm is to minimise the sum of intra-cluster variances, where the variance of a cluster 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 can be expressed as:

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

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:

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

where $\delta_x(C)$ represents the membership measure of point x in cluster C. This objective function reflects the weighting of the squared distances by the membership measure of each point in the clusters.

Definition 1.2.2 (Loss function and cluster weight)**.** We define the fuzzy clustering loss function from the KMeans algorithm as follows:

$$L_{\mathcal{S}}(\mathcal{C}) := \sum_{C \in \mathcal{C}} p_{\mathcal{S}}(C) := \sum_{C \in \mathcal{C}} \sum_{x \in \mathcal{S}} \mu_x(C)^2 \|x - C\|^2 \tag{1.4}$$

where $p_{\mathcal{S}}(C)$ is the 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. 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

Remark. The KMeans algorithm and fuzzy clustering via FCM share a similar approach in updating of cluster centroids. Both algorithms aim to find the optimal position of the centroids that minimizes the loss function based on the distances between the data points and the centroids themselves.
In the KMeans algorithm, the process of updating centroids occurs iteratively. Initially, centroids are randomly assigned to clusters. Data points are then assigned to the nearest clusters (based on Euclidean distance), after which the centroids are updated by averaging the points assigned to each cluster. This process is repeated until the centroids converge to stable positions.
Similarly, the FCM algorithm iteratively updates both the centroids of the clusters and the cluster membership measures for the points. In each iteration, the optimal centroids for a given configuration of membership measures are determined, and vice versa, until a stable configuration is reached that minimizes the loss function defined based on the membership measures and the distances between the data points and the centroids.

Theorem 1.1 (*Update formula 'M'*)

Fixed the matrix U, the optimisation of eq. (1.4) finds its minimum in the centroids according to the following update formula:

$$C_j^{\texttt{new}} = \frac{\sum_{i=1}^{N} u_{ij}^2 x_i}{\sum_{i=1}^{N} u_{ij}^2} \qquad \forall j \tag{1.5}$$

Proof. Fixed 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. 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 ideal centroids in fuzzy clustering and does not directly aim to minimise of eq. (1.4), but rather to iteratively updates the centroids and membership measures until a stable configuration is reached that best represents the input data.

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

procedure FUZZYCLUSTERING($\mathcal{S}, \mathcal{C}$)
  while not reached the exit criterion **do**
    $D^2 \leftarrow (d_{ij}^2)$ with $d_{ij} = \|x_i - C_j\|^2$
    $U^2 \leftarrow (u_{ij}^2)$ with $u_{ij} = \left(\sum_k D_{ij}^2 / D_{ik}^2\right)^{-1}$ ▷ see theorem 1.1
    for $j \leftarrow 1$ to M **do**
      $C_j^{\text{new}} \leftarrow \sum_{i=1}^N x_i U_{ij}^2 / \sum_{i=1}^N U_{ij}^2$
    end for
  end while
end procedure

1.2.2 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 minimize the sum of variances within these groups, resulting in well-defined clusters.

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, lets see how these methods behave in the presence of noisy data. Specifically, 6 random centroids were chosen in a two-dimensional space, from which 20, 20, 20, 30, 30 and 40 data points with different covariance matrices were generated,

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

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. 1.2. As can be seen, noise can create *outliers*, i.e. distant points that would be better ignored to achieve good clustering.

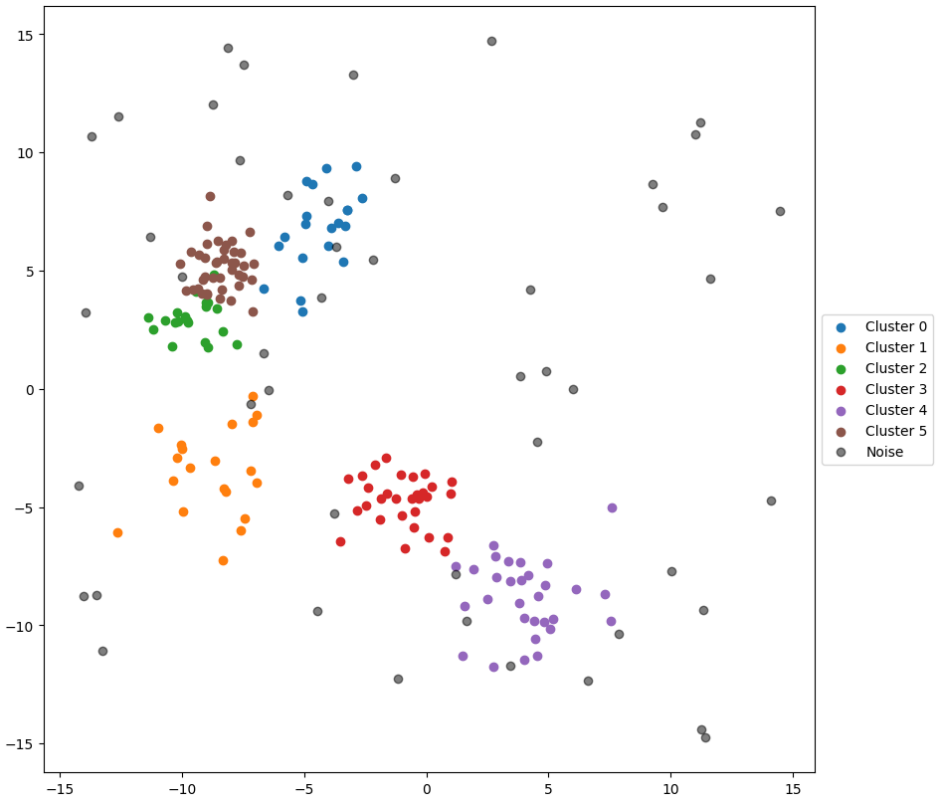


Figure 1.2. Data points are coloured according to their label, while noise is shown in grey.

Now let will try clustering with KMeans using the right 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. Although the result is quite satisfactory, it is not entirely clean.
Next, let is apply clustering with GMM again using 6 centroids. In fig. A.2 it can be seen that the weight of the cluster generated by the noise is very small. Furthermore, the covariance matrices allow the position of the centroids to be calculated more precisely, providing information on the nature of the clusters and creating a hierarchical structure. This is because, in reality, these are not real clusters, but normal distributions.
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. However, the algorithm GMM, although computationally onerous, also provides very valuable information about the nature of the data. In this thesis, we

will use the FCM algorithm to reduce the amount of information to be processed and the KMeans algorithm for efficient initialisation of centroids.

1.3 DFT

The DFT is a mathematical transformation used to analyse the internal periodicities of a time sequence of data. Specifically, it decomposes a periodic time series as a sum of sine waves at different frequencies, giving information on the amplitude and phase of each frequency present in the signal. The more data available, the more detailed DFT will be.
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].

1.3.1 Insights

The DFT comes about as a discrete version of the Continue Fourier Transform (CFT), which represents a continuous function as a potentially infinite sum of sine waves. More rigorously, given a function $f \in L^1(\mathbb{R})$ (i.e., a function integrable over the whole $\mathbb{R}$), its Fourier transform is a functional operator, denoted by $\mathcal{F}$, which associates the function f with its frequency representation:

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

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.

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

Definition 1.3.1. We define Fourier coefficients of the sequence $(x_n)_{n=0}^{N-1}$ as those complex numbers $(X_k)_{k=0}^{N-1}$ such that:

$$x_n = \frac{1}{N}\sum_{k=0}^{N-1} X_k e^{i2\pi\frac{k}{N}n} \tag{1.6}$$

The k-th coefficient gives information about the amplitude and phase of the frequency $\frac{2pi}{N}k$. We present some examples to better understand the meaning of the Fourier coefficients.

Exempli Gratia. Take as our first example 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 purely periodic, with amplitude $\frac{2A}{N}$ and frequency $\frac{2\pi}{N}$.

Let us now consider a series of different coefficients, where all values are zero except:

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

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. In addition, to obtain a real time series, symmetry on the coefficients is required.

At last, let us see how to derive the phase from a time series. Let us consider the following coefficients:

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

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. Again, 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 1.3.1. *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 scalar product between two rows of the matrix, one of which is hermitian

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

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

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

This prove 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 1.2 (*DFT*)

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

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi\frac{n}{N}k} \tag{1.7}$$

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 1.3.1, 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. □

1.3.2 Implications

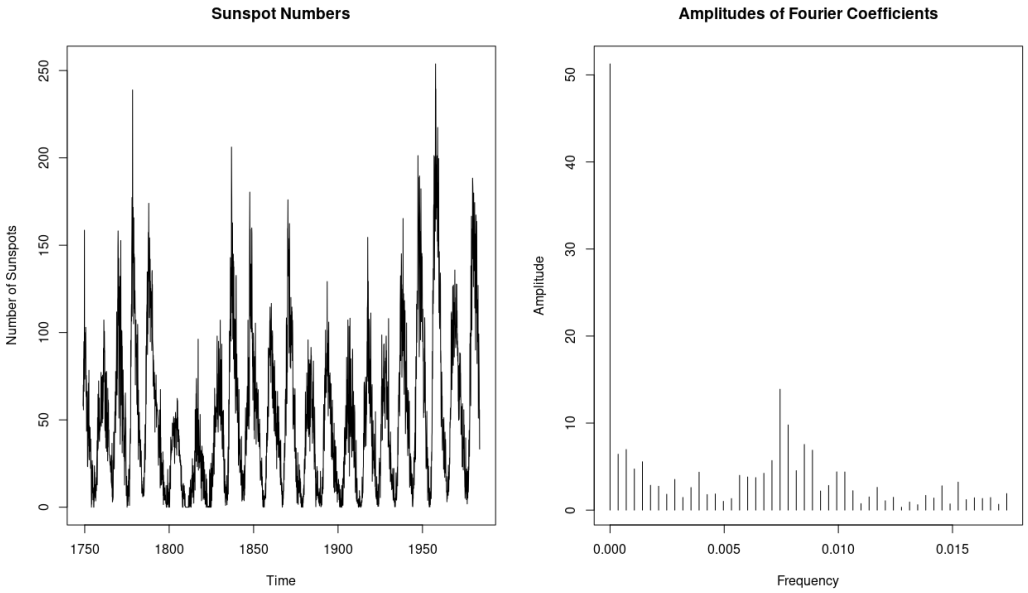
The possibility of calculating Fourier coefficients directly by means of a simple matrix-vector product not only makes the analysis very informative, but also easily achievable.

Algorithm 2 FFT Cooley-Tukey

```
procedure FFT(x)
    if N is 1 then return x
    end if
    even ← FFT([x_0, x_2, ··· , x_{N−2}])
    odd ← FFT([x_1, x_3, ··· , x_{N−1}])
    X is an array of size N
    for k ∈ 0, ..., N/2 − 1 do
        t ← exp(−2πik/N) · odd[k]
        X[k] ← even[k] + t
        X[k + N/2] ← even[k] − t
    end for
end procedure
```

Fast Fourier Transform (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 is a specific algorithm for Fourier series that reduces this cost to $O(N \log N)$, comparable to a sorting algorithm. This algorithm, named FFT, was designed by Cooley and Tukey.
It is possible to further optimise this algorithm using Graphics Processing Unit (GPU), which exploit the high parallelism of processors to further reduce the computational cost. Although algorithm 2 seems to be able to achieve a cost of $O(\log N)$, there are physical limitations of the GPU and in data structures that make its execution very fast, but asymptotically it remains of cost $O(N \log N)$.

Periodogram A periodogram graphically represents the amplitude of the different frequencies that make up a time series, highlighting the dominant components of the frequency spectrum. This tool is particularly useful for distinguishing between noise and the significant frequencies of a signal. For example, in the fields of climatology and astrophysics, it is often used to identify periodic cycles in data.
We see an application of DFT on the `sunspots` dataset from the R package, which contains the number of sunspots observed each month from 1700 to 1988. Sunspots are correlated with the Sun's magnetic activity and one of the goals of the analysis is to identify any periodic cycles in the data.



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.

1.4 Application

In this section, we will develop specific applications of the topics discussed in the chapter to see how they can be used in the context of the 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.

1.4.1 Comparing works, the idea of clustering

As we have seen, a problem of eq. (1.3) becomes apparent when the continuous space in which the n-tiles are defined turns out to be too fit. The proposed idea was to fix the fewest possible boxes and to work with the idea that 2 any tiles are as distinct as any other pair. However, this is not a credible assumption due to the loss of information, for this reason in this thesis we would like to approach the problem more dynamically by increasing the variety of tiles without running into sparsity problems.
The idea is to use the clustering of the merged data between the two samplings as a basis for fitting the continuous space. The expectation is that the result will

certainly be more accurate, however, it is necessary to consider the problem of the shape of the clusters (which will no longer be boxes).

Exempli Gratia (Distance between distributions reformulated). 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 samplings will be merged and clustered with KMeans using 32 centroids, resulting in a predictor $\mathcal{P}$. 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:

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

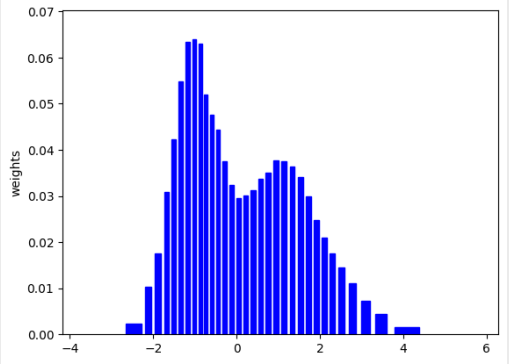
where $\mathcal{L}$ is the law of the two merged samples.
We can see in the first image that regions have been captured and that automatically the regions are little where the density is higher. Furthermore, the image shows the weights of each cluster c defined as $\mathbb{P}_{x\sim\mathcal{L}}[\mathcal{P}(x) = c]$
We now study the two 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:

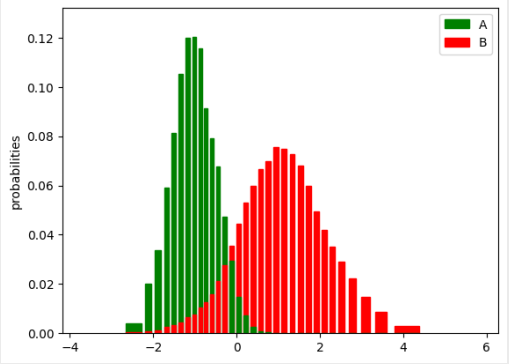
$$d_{\mathcal{A}}(c) = \frac{\mathbb{P}_{x\sim\mathcal{A}}[\mathcal{P}(x) = c]}{\mu(c)}$$

similarly for $\mathcal{B}$.
In the second figure we show the cluster membership probabilities of the two distributions $\mathcal{A}$ and $\mathcal{B}$.
Let us try to calculate the distance formulated eq. (1.3) considering the measure:

$$d(A,B) = (1 + J_{D_A,D_B})^{-1} \frac{1}{\mu(D_A \cup D_B)} \int d\mu(c) \left(\frac{r(c)-1}{r(c)+1}\right)^2$$

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, it is basically shown that the number of nodes can be reduced and can be adapted to the distributions of the tiles, and can also be very effective

at high dimensions by counteracting the curse of dimensionality. In fig. 1.3 we take an example very similar to the previous one but in 2 dimensions and with less data. Let us try clustering using more and more nodes, repeating the calculation carried

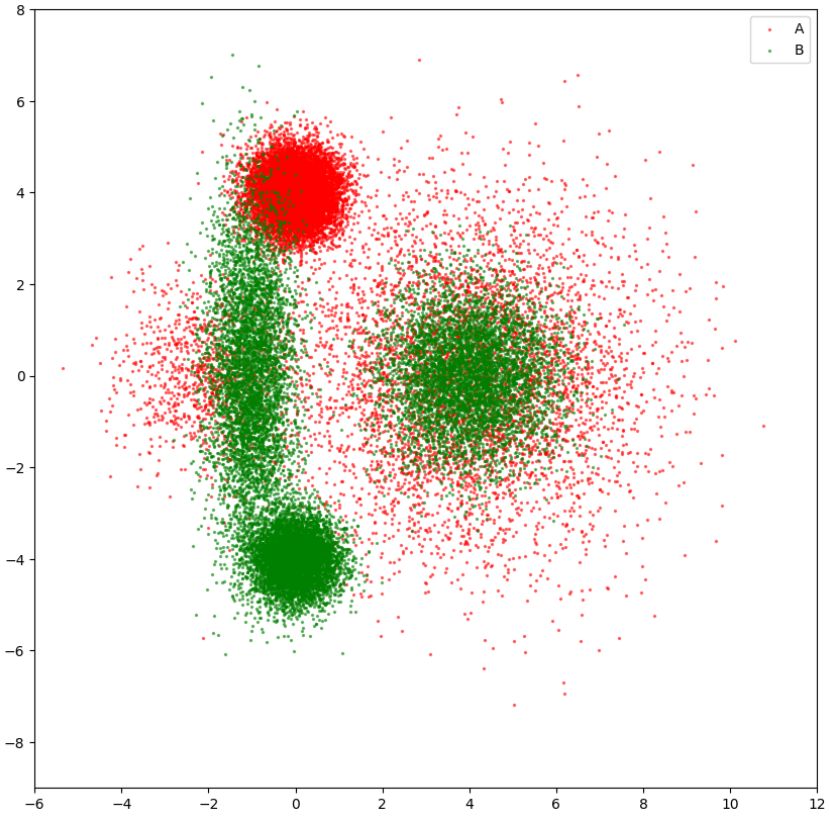


Figure 1.3. In the figure, the data from the distribution $\mathcal{A}$ are shown in red and the data from the distribution $\mathcal{B}$ in green. They were obtained by merging normal Gaussians with different representability.

out in the example, studying how the distance changes by increasing the number of nodes. In table 1.1 we can see how the clustering shows more stable results.

Number of nodes	1	4	16	64	256	1024
Distance with clustering	0.00	0.13	0.21	0.30	0.34	0.37
Distance without clustering	0.00	0.40	0.49	0.45	0.49	0.48

Table 1.1. Comparison between KMeans clustering and box tessellation with same number of nodes.

1.4.2 Fuzzy Clustering as a Noise Filtering Method

In this thesis, a variant of the algorithm FCM will be used to compare two datasets. The variant will be used both to consider data of different weights and to consider machine error in the computation of centroids.

data's weight We introduce a vector w indicating the weight of the data as a positive real value. As stated in theorem 1.1 the update follows the following law:

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

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 result:

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

machine error The use of FCM in this thesis involves millions of pieces of data, and it is possible that the classical algorithm will find itself making serious machine errors that must be kept under control. An example that 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 is infinity.
For this reason algorithm 3 proposes a more robust approach. We also remark that the computational cost in a sequential algorithm will be $O(NCk)$ however scalability allows us to reduce this cost to $O(\log(kC))$ by exploiting the independence of each cycle and scalable binary operations (details in chapter 2).

N is the number of data and C is the number of centroids, k is the size of the tiles

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

procedure MembershipUpdateStable($\mathcal{S}, \mathcal{C}$)
  $D^2 \leftarrow (d_{ij}^2)$ with $d_{ij} = \|x_i - C_j\|^2$
  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
      $u_{ij} \leftarrow \frac{l}{D_{ij}^2}\ \forall j$
    end if
    $S_i \leftarrow \sum_j u_{ij}$
    $u_{ij} \leftarrow u_{ij}/S_i\ \forall j$
  end for
end procedure

1.4.3 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:

$$x = \left(\cos\left(\omega_r n + \omega_c m\right)\right)_{n,m}$$

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:

$$x_{n,m} = \frac{1}{NM}\sum_{r,c} X_{r,c} e^{i2\pi\left(\frac{rn}{N}+\frac{cm}{M}\right)} \tag{1.8}$$

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

$$X_{r,c} = \sum_{n,m} x_{n,m} e^{-i2\pi\left(\frac{nr}{N}+\frac{mc}{M}\right)} \tag{1.9}$$

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:

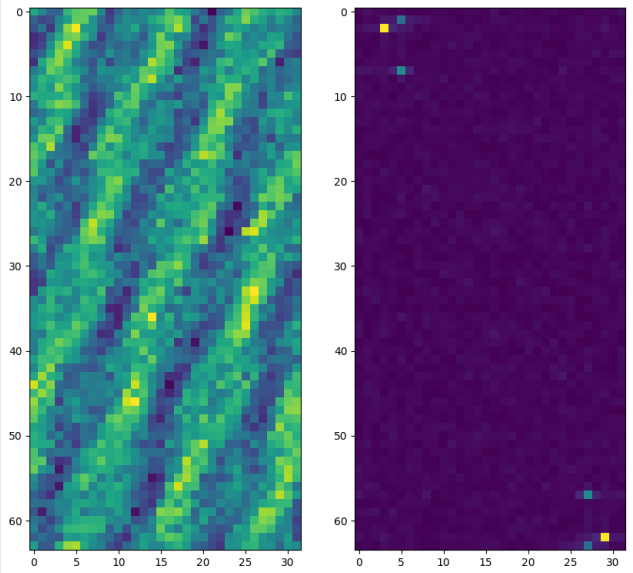
$$y_{r,m} = \sum_{n} x_{n,m} e^{-i2\pi\frac{nr}{N}} \qquad \text{over each column apply FFT}$$

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

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 Fourier coefficients are shown. They are computed from the matrix x using a `viridis` colour scale.



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 2

Methodology

To Review

This research aims to analyse the attribution of graphic works, taking its inspiration from the methodologies used for the attribution of literary works [see 4]. 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 4], graphics are considered as matrices of colours and an N-gram was defined as a square submatrix 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. 2.1). 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 2.1. 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 as the thesis of Magrini Alunno [4], 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.

2.1 Data Set

2.1.1 Description

In this study, the data set consists of a set of graphic works for which attribution is established and unambiguous and for which a uniform resolution expressed in dots per inch (DPI) is known. It was decided to use calligraphic works, as evidenced in [4]. The use of calligraphic works represents a promising first way for artistic attribution.
The dataset used in the reference article 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 often contain small squares that could be mistaken for the author'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 [4] 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:

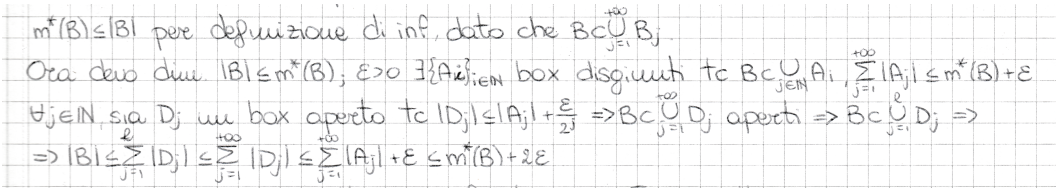
- The images are saved in the png format.
- Each image is composed of three colour channels Red Green Blue (RGB), each with a depth of 8 bits.
- The original sheets were scanned at a resolution of 400 DPI.
- The thickness of the pen used to write on the original sheets is 0.4 mm.

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 2.1. 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, it's compute as follow: $\sqrt{\frac{\text{Num of pixels}}{\text{Num of items}}}$

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.



2.2 Pre processing

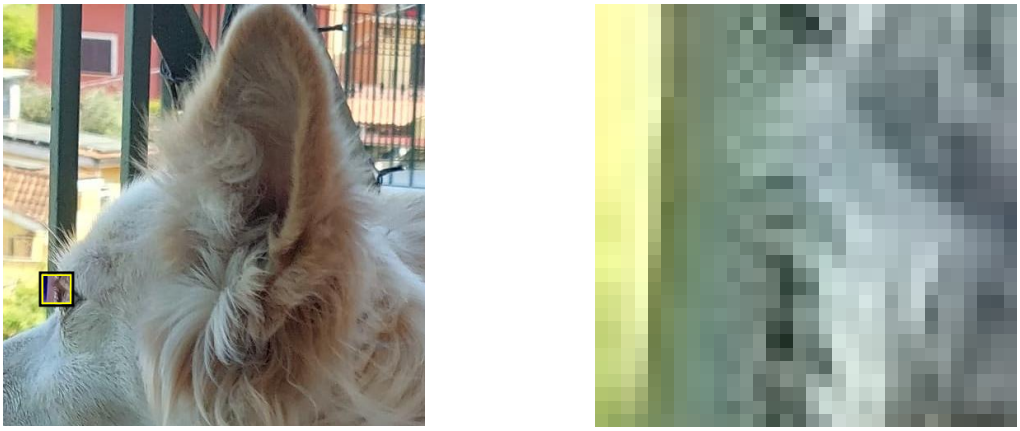
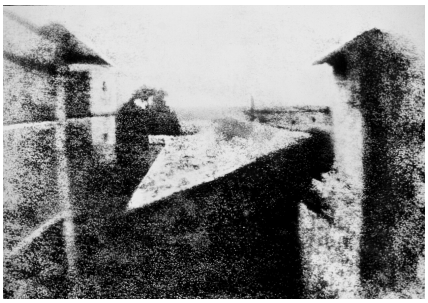


Figure 2.2. Image detail.

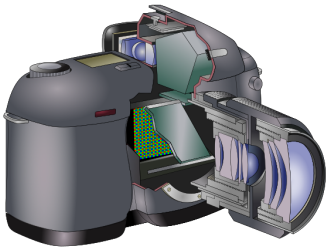
Digitising images is a fundamental step in preparing the data set for analysis. To fully understand how these transformations work, it is important to understand the process by which a work of art is captured by a camera and then digitised in an electronic device.

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]
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]
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 [3]). This model encodes each colour with three real numbers 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,

[1] for more about camera obscura, see
https://en.wikipedia.org/wiki/Camera_obscura
[2] for more about history of photography, see
https://en.wikipedia.org/wiki/History_of_photography



(a) 'View from the Window at Le Gras', Joseph Nicéphore Niépce, c.1826, Heliography, Harry Ransom Center, University of Texas at Austin, USA[6].



(b) Illustration of digital camera[7].

a digital image becomes a matrix of triples, representing the pixel values the three colour channels RGB.

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

2.2.2 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.

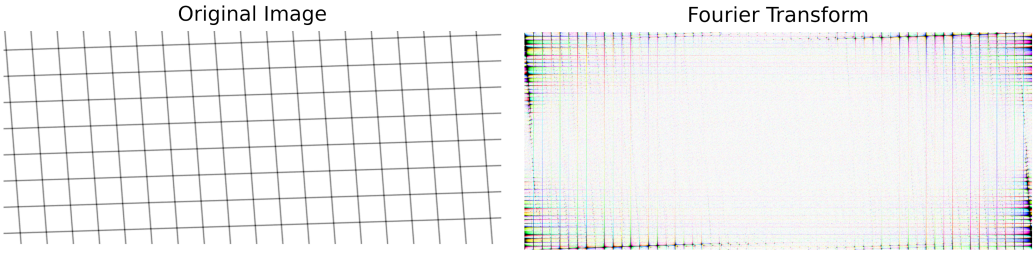
[3] For more about Himba people, see https://en.wikipedia.org/wiki/Himba_people

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:

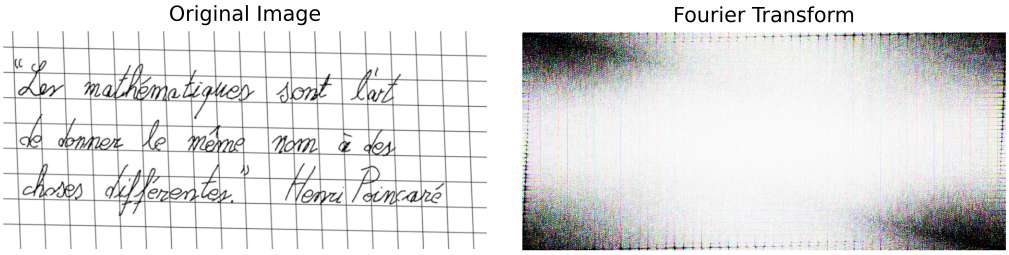
$$U\Sigma V^H = M$$

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 real 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.

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 drawings on it. In the case of the unmarked sheet, frequencies with larger amplitudes are mainly found aligned along the x and y axes (fig. 2.5a). By adding a pattern, the main frequencies remain almost the same (fig. 2.5b). To remove the squares, we remove these significant frequencies and reconstruct the image (fig. 2.5c).
Empirically, by analysing 10 randomly selected real images, 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.
However, this process introduces an issue of colour distortion. In the original image, the grey levels were distributed intuitively: most of the pixels were white, with only a small portion in darker shades of grey, representing the writing. After reconstructing the image using the FFT, a significant number of pixels took on intermediate shades of grey that weren't present before. This affects the clarity of the image, especially by blending the background and the lettering, making it harder to distinguish between the important features from the unwanted noise.
To reduce the effects of this distortion, we adopted a 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 certainly belong to the lettering, while those with a grey level above 0.8 correspond to the white background of the page. Consequently, the pixels of the lettering and white background in the original image are counted. In the cleaned image, the



(a) A slightly transformed grating, caused by an imperfect scan. The most significant frequencies are highlighted in black.



(b) A deformed grating with a superimposed inscription. The frequencies of the lattice remain visible.

(c) Lattice-less writing does not show significant Fourier coefficients, as recurring patterns are missing.

Figure 2.5. Fourier coefficient of the two-dimensional image. The squares show Fourier coefficients similar to those of the square waves: $\text{sqwave}(t) = \frac{4}{\pi} \sum_{k=1}^{\infty} \frac{\sin(2\pi(2k-1)t)}{2k-1}$. The most significant intensities are at regular intervals with decreasing intensities.

darkest pixels (with a grey value below 0.2 in the original image) are assigned to black, while the lightest pixels (with a grey value above 0.8 in the original image) are assigned to white.
To correct the distortion, we expanded the grey values of the reconstructed image so that pixels that should be black became darker and those that should be white became lighter. We then clamped these values within natural limits (0 to 1) to avoid out-of-scale colours. This process was called 'clamp normalisation'. A result of this procedure is shown in **??**.

Presente nel capitolo Results **Note**

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

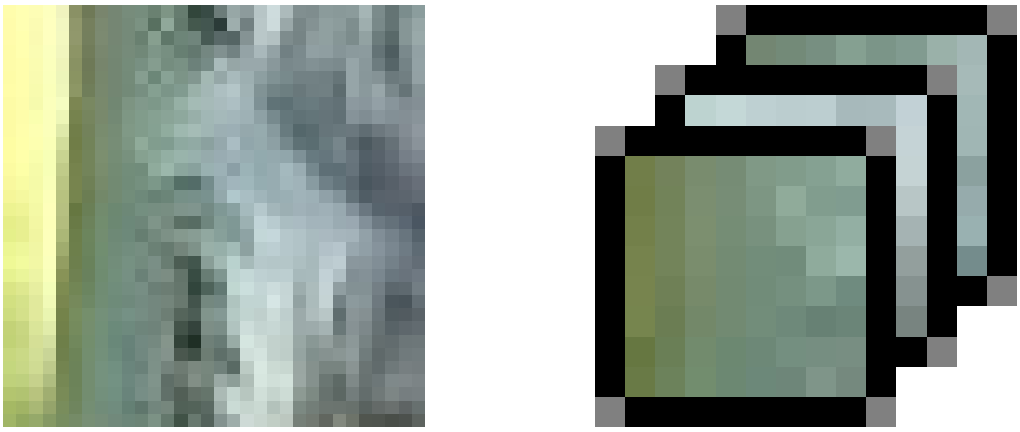
procedure CLEANFFT($\mathcal{I}, p, a$)
  $\hat{\mathcal{I}} \leftarrow \text{normalize}(\mathcal{I})$
  $F_{\hat{\mathcal{I}}} \leftarrow \text{fft2d}\left(\hat{\mathcal{I}}\right)$
  $f = (f_1, \ldots, f_{W \times H}) \leftarrow \text{argsort}\left(\text{flatten}\left(F_{\hat{\mathcal{I}}}\right), \text{rule=abs}\right)$
  $f_{\text{best}} \leftarrow f\left[\text{last } \lfloor p \cdot \text{length} \rfloor \text{ values}\right]$ ▷ 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
  $w_{\text{cnt}} \leftarrow$ **count** $\mathcal{I} \geq a(\max)$ ▷ normalization using dilation and clamp
  $b_{\text{cnt}} \leftarrow$ **count** $\mathcal{I} \leq a(\max)$
  $h_w \leftarrow \max_{w_{\text{cnt}}^{-\text{th}}} \hat{\mathcal{I}}$
  $h_w \leftarrow \min_{b_{\text{cnt}}^{-\text{th}}} \hat{\mathcal{I}}$
  $\mathcal{I}_{\text{new}} \leftarrow \frac{\hat{\mathcal{I}} - h_b}{h_w - h_b}$
  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

2.3 Synthesis

The tessellation phase extracts tiles $N \times N$ from each image. Starting from a matrix of grey pixels, represented as a matrix 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, which is why this phase is called 'synthesis': in fact, it is difficult to reconstruct the original image from the tiles. This process considerably complicates any attempt to falsify the work, as manually reconstructing an image with a tessellation distribution similar to the original is extremely complicated. Furthermore, the size of the tiles in the representation is small, but large enough to capture small automatisms of the author's hand, details that are difficult to reproduce voluntarily.

Algorithm 5 Algorithm for tile extraction

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



(a) Image detail 32×32.

(b) List of tiles 8×8.

Figure 2.6. The synthesis process convert an image in a list of its tiles.

2.4 Comparison

In order to compare two works, we decided to use the clustering FCM, which is less dependent on possible noise, especially since the comparison between two works will leverage smaller densities and therefore noise could pollute the results.
In this subsection, we will look in detail at the algorithm for comparing works and see an example application to better investigate its characteristics.

Definitions As already mentioned in chapter 1, the comparison between two distributions $\mathcal{A}$, $\mathcal{B}$ will pass through the use of a clustering algorithm.
In particular, we used the KMeans to implement a clusterisation that discretized the space. Finally, over this discretisation we applied this definition of distance:

$$d(\mathcal{A},\mathcal{B}) = (1 + J_{D_\mathcal{A},D_\mathcal{B}})^{-1} \frac{1}{\mu(D_\mathcal{A} \cup D_\mathcal{B})} \int d\mu(c) \left(\frac{r(c)-1}{r(c)+1}\right)^2 \tag{2.1}$$

Now, we will apply FCM to the fused distribution $\mathcal{L}$. For each cluster with centroid c, we will compute the size $\mu(c)$ and the density of the set of samples $d_A(c)$, $d_B(c)$. Finally, we will give a definition of a Jaccard index between dictionaries D_A and D_B using the fuzzy logic.
Since FCM is a generalisation of KMeans, let us try to define these quantities starting from KMeans. In particular, we define the weight of a cluster as a quantity proportional to the k-dimensional hypervolume of a hypersphere having radius the mean square distance of the data from its centroid:

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

In our case, there is no predictor since a measure of belonging of each datum to each cluster has been defined. However, it is recalled that FCM is derived from a generalisation of KMeans and that therefore the formula minimised by the algorithm EM (eq. (1.4)) can be taken.

Definition 2.4.1 (measurement of a cluster)**.** Applying FCM to the data set $\mathcal{S}$ with weights w, we obtain the centroids $\mathcal{C}$ and the membership matrix $u^2_{xc} = w_x \mu_x(c)^2$. Calling k the dimension of the space where the data are instantiated, we denote by $\mu(c)$ the weight of the cluster c. The weight will be proportional to the k-dimensional hypervolume of a hypersphere having radius the mean square distance of the data from the centroid c:

$$\mu(c) := \sqrt{\frac{\sum_{x\in\mathcal{S}} u^2_{xc}\|x-c\|^2}{\sum_{x\in\mathcal{S}} u^2_{xc}}}^k \tag{2.2}$$

Once the size of the clusters has been defined, it will then be necessary to talk about the density of belonging of the set A (or similarly B) to a cluster c. As mentioned above, FCM does not propose a predictor, so a more general definition will have to be given. In KMeans it is possible to reformulate the density as follows:

$$\frac{\mathbb{P}_{x\sim\mathcal{A}}[\mathcal{P}(x)=c]}{\mu(c)} = \frac{\int d\mu_\mathcal{A}(x)\delta_x(c)}{\mu(c)} = \frac{\int d\mu_\mathcal{A}(x)\mu_x(c)^2}{\mu(c)} \approx \frac{\frac{1}{|A|}\sum_{x\in A}\mu_x(c)^2}{\mu(c)}$$

The definition for FCM follows:

Definition 2.4.2 (density over a cluster)**.** We denote by $d_A(c)$ the density of the set A on the cluster c obtained with FCM.

$$d_A(c)\mu(c) := \frac{\sum_{x\in A} u_{xc}^2}{\sum_{x\in A} w_x} = \frac{\sum_{x\in A} w_x \mu_x(c)^2}{\sum_{x\in A} w_x} \tag{2.3}$$

Similarly for $d_B(c)\mu(c) = \frac{\sum_{x\in B} w_x \mu_x(c)^2}{\sum_{x\in B} w_x}$
We call the weight of samples over the cluster with centroid c:
$\omega_A(c) = d_A(c)\mu(c)$ and $\omega_B(c) = d_B(c)\mu(c)$

It is now necessary to define the Jaccard index, which is not an easy task since in fuzzy logic the cardinality of a set is not defined in the same way as in Boolean logic, since membership itself does not have a binary truth value.
Let us think of D_A and D_B as fuzzy sets of clusters. By $\mu_c(D_A)$ we denote how much the centroid c is in D_A, same for B, i.e. we mean the measure in which the centroid c relates to the set A.

Definition 2.4.3. (Jaccard index) Union and intersection operations in fuzzy logic are defined as follows:

$$\mu_c(D_A \cup D_B) = \max\{\mu_c(D_A), \mu_c(D_B)\}$$
$$\mu_c(D_A \cap D_B) = \min\{\mu_c(D_A), \mu_c(D_B)\}$$

and the cardinality of a set S is defined as: $|S| := \sum_c \mu_c(S)$
The membership of a cluster c in the dictionary D_A will be defined as follows:

$$\mu_c(D_A) = \max_{x\in A}(\mu_x(c)^2) \tag{2.4}$$

similarly for $\mu_c(D_B) = \max_{x\in B}(\mu_x(c)^2)$.
For this reason, the definition of the Jaccard index that follows will be:

$$J_{D_A,D_B} := \frac{|D_A \cap D_B|}{|D_A \cup D_B|} = \frac{\sum_c \min\left\{\max_{x\in A}\left(\mu_x(c)^2\right), \max_{x\in B}\left(\mu_x(c)^2\right)\right\}}{\sum_c \max\left\{\max_{x\in A}\left(\mu_x(c)^2\right), \max_{x\in B}\left(\mu_x(c)^2\right)\right\}} \tag{2.5}$$

definitions 2.4.1 to 2.4.3 are sufficient to formulate the distance between two proposed samplings following FCM.

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.

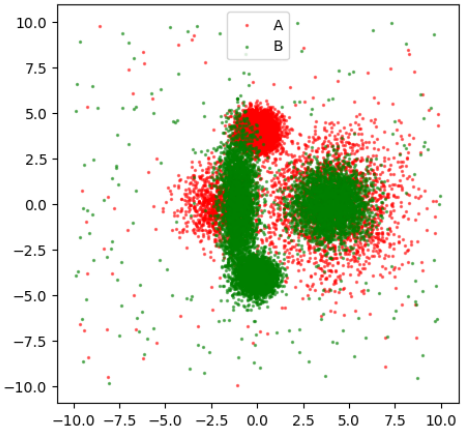
- $\mu(c) = \sqrt{\frac{\sum_{x\in\mathcal{S}} u_{xc}^2 \|x-c\|^2}{\sum_{x\in\mathcal{S}} u_{xc}^2}}^{\,k}$
- $\omega_A(c) = \frac{\sum_{x\in A} u_{xc}^2}{\sum_{x\in A} w_x}$ and similarly $\omega_B(c) = \frac{\sum_{x\in B} u_{xc}^2}{\sum_{x\in B} w_x}$

- J_{D_A,D_B} as in definition 2.4.3

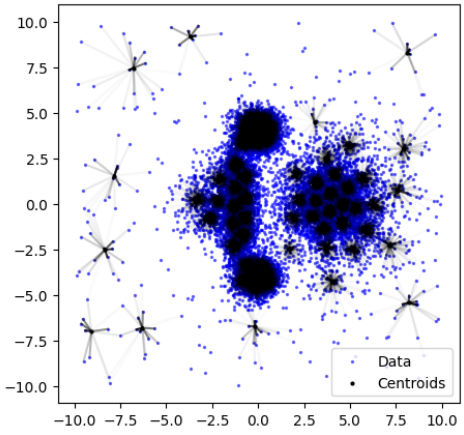
The distance between works will be defined as:

$$\begin{aligned} d_{FCM}(A,B) &= (1+J_{D_A,D_B})^{-1}\frac{1}{\mu(D_A\cup D_B)}\int_{\mathcal{C}} d\mu(c)\left(\frac{r(c)-1}{r(c)+1}\right)^2 \\ &= (1+J_{D_A,D_B})^{-1}\frac{\sum_c\left(\frac{r(c)-1}{r(c)+1}\right)^2\mu(c)}{\sum_c\mu(c)} \end{aligned} \tag{2.6}$$

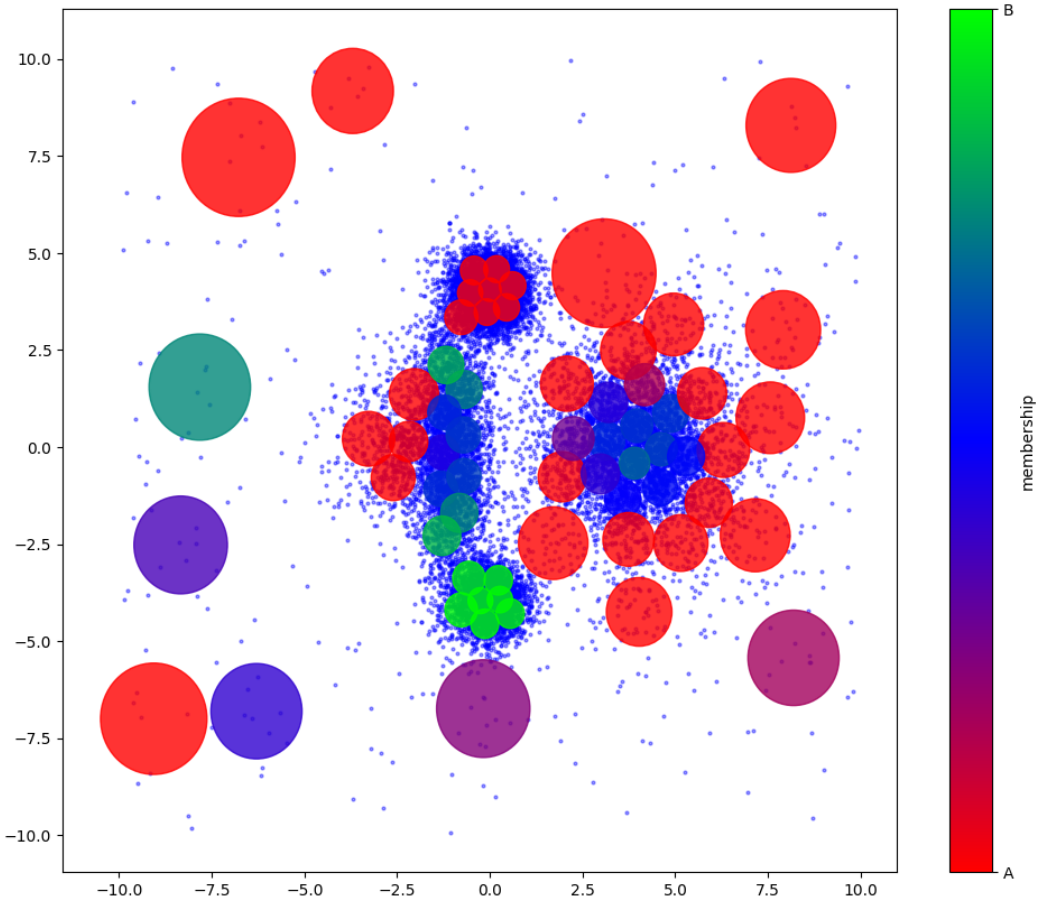
where $r(c)$ is the ratio of the weights i.e. between $\omega_A(c)$ and $\omega_B(c)$.
A simple application for a synthetic data set is shown below. As done in chapter 1, data is generated in a two-dimensional space and clustered with FCM into 64 clusters.
In fig. 2.7a the data used are shown, these are normal Gaussians with some representation and uniform noise. The two sets, called A and B, were then merged and clustered with FCM (in fig. 2.7b a representation). In fig. 2.7c the clusters are depicted with their respective representations between the set A and B by means of a colour map.



(a) The two sets of images are shown in the figure.



(b) A clustering FCM with 64 centroids is shown in the figure; significant relationships between data and clusters are represented by means of segments.



(c) Clusters are shown in the figure with an area covering the size of the cluster and a colour indicating its characterisation; in particular, the colour scale is mapped onto $\frac{\omega_A - \omega_B}{\omega_A + \omega_B}$.

We calculate the Jaccard index:

- $|D_A| = 53.17$ e $|D_B| = 52.26$
- $|D_A \cap D_B| = 43.91$ e $|D_A \cup D_B| = 61.53$
- $J_{D_A,D_B} = 0.71$

We estimate the integral and consequently the distance between the distributions:

- $\frac{\sum_c \left(\frac{r(c)-1}{r(c)+1}\right)^2 \mu(c)}{\sum_c \mu(c)} = 0.31$
- $(1 + J_{D_A,D_B})^{-1} = 0.58$
- $d(A, B) = 0.18$

We observe that the result in the presence of noise is not very dissimilar to the result obtained with KMeans in the absence of noise (see table 1.1). Another example is with 256 nodes where the distance will be 0.38 so again there are no noise-related problems.
We verify the potential of FCM by examining the distances computed by KMeans in the presence of noise:

- 64 nodes: the estimated distance is 0.12 versus the previous 0.30 from KMeans.
- 256 nodes: the estimated distance is 0.23 versus the previous 0.34 from KMeans.

The reduction in distance is an expected phenomenon, as the noise is identical for both samples, thus making them more similar. FCM can slightly clean the sets from noise and thus have a more accurate estimate.

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 equipped with 14 SM, each of which contains 1024 threads divided into 32 warps. This total of 14336 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 Compute Unified Device Architecture (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. 2.8 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 2.8. 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 3

Results

3.1 Architecture

To Do

Come accennato nei capitoli precedenti, il sistema di analisi delle immagini richiede di confrontare l'opera sconosciuta con quelle note.
L'immagine viene preprocessata rimuovendo elementi inquinanti che non riguardano la grafia, lasciando essenzialmente un'immagine in scala di grigi con la grafia dell'autore in tonalità scura e lo sfondo bianco. Inoltre, il tratto dell'autore potrebbe necessitare di uno spessore specifico rispetto ai caratteri impressi e la risoluzione (espressa in DPI) deve essere conforme al dataset.
L'immagine così preprocessata sarà quindi sintetizzata: un programma ne estrae tutte le tessere con una dimensione specifica conforme al dataset. In seguito, l'immagine sarà confrontata con ogni immagine del dataset (o con le più rappresentative per ogni autore).
In questo capitolo si effettueranno analisi di ogni passaggio illustrato in questa tesi al fine di vedere come ogni singola componente agisce sulle immagini esaminate. Non solo, si mostreranno anche le analisi effettuate per stabilire specifiche decisioni e parametri usati durante la fase di progettazione.

La costruzione del dataset avviene con un procedimento lievemente diverso. Ogni immagine di cui l'autore è noto viene confrontata completamente con tutte le altre già note. Essendo l'indice di dissimilarità simmetrico, sarà sufficiente esaminare la metà di tutti i confronti possibili. Se si è in possesso di n immagini, pensiamo di voler riempire una tabella quadrata $n \times n$ con valori reali. Assumendo dissimilarità 0 tra immagini con se stesse, si può concludere che in totale saranno richiesti $\frac{n^2-n}{2}$ confronti.
Per questa ragione, l'algoritmo per ogni riga esaminerà la metà dei possibili confronti come mostrato in fig. 3.1. In particolare, se n è dispari, per ogni riga si faranno $\frac{n-1}{2}$ confronti (e altrettanti saranno memorizzati per via della simmetria), mentre se n è pari, allora $\frac{n}{2}$ confronti saranno effettuati se la riga i è pari (la prima riga ha indice 0) e $\frac{n}{2} - 1$ se la riga i è dispari. Questa procedura permette una analisi più uniforme ed efficiente per numerosi dati, specie se accompagnata da shuffle per riga e per colonna.

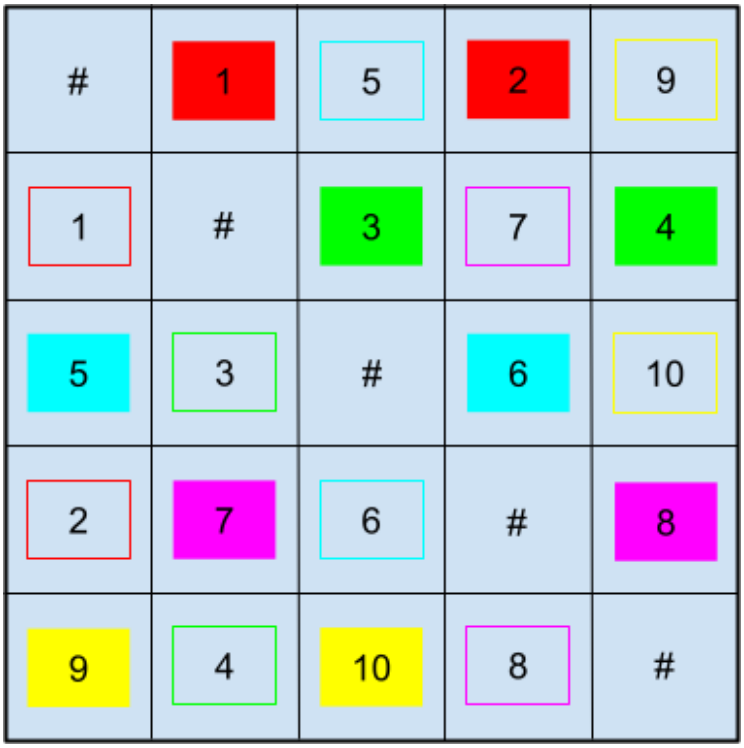


Figure 3.1. The algorithm for each row computes half of all components. In the image, the algorithm follows the order of the numbers and alternately takes the cells such that the lower diagonal is not directly computed and the upper diagonal is directly computed. For example, in the $3^{\text{-th}}$ row, the algorithm computes directly the first column (number 5 in cyan) and the next-to-last 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.

3.1.1 Challenges

L'implementazione del codice ha richiesto notevoli sforzi poiché gran parte dei framework richiesti non erano esistenti, inoltre le limitazioni della potenza di calcolo e il tempo a disposizione hanno rappresentato un forte vincolo. Questo ha richiesto la stesura di codice altamente prestante e sofisticato per ottimizzare l'efficienza computazionale.
In generale, il progetto ha richiesto largo uso di Python e CUDA. Durante la sua realizzazione si sono utilizzati software specifici per debugging, logging e testing, utili per monitorare il corretto funzionamento del software. Sono stati impiegati strumenti di documentazione automatica e gestione delle versioni del codice. Inoltre, è stato adottato un robusto sistema di gestione degli errori e delle sessioni di calcolo, con meccanismi di backup (detti "checkpoint") per scongiurare la perdita di dati preziosi. Il software utilizza anche file temporanei per aumentare la capacità di memorizzazione oltre i limiti della memoria principale.
Per queste ragioni, il codice (composto da un totale di 3480 righe) è difficilmente riportabile per intero in questo testo ed è consultabile nella repository GitHub: `https://github.com/StefanoMagriniAlunno/MMATH_thesis`.

La macchina che ha prodotto i risultati illustrati in questo capitolo ha le seguenti specifiche tecniche:

- **processor:** 11th Gen Intel(R) Core(TM) i7-11370H @ 3.30GHz
  - Cores: 4
  - Thread(s) per core: 2
  - CPU max MHz: 4800
  - Caches:
    * L1d 192KiB (4 instances)
    * L1i 128KiB (4 instances)
    * L2 5MiB (4 instances)
    * L3 12MiB (1 instances)
- **memory:** 16 GiB
  - SODIMM DDR4 Synchronous 3200 MHz (0,3 ns)
  - SODIMM DDR4 Synchronous 3200 MHz (0,3 ns)
- **memory:** 16 GiB
  - SODIMM DDR4 Synchronous 3200 MHz (0,3 ns)
  - SODIMM DDR4 Synchronous 3200 MHz (0,3 ns)
- **graphic processor unit:** NVIDIA GeForce GTX 1650
  - Cores: 896
  - stream multiprocessor: 14
  - VRAM: GDDR5 4096 MiB
  - CUDA capability: 7.5

Saranno visionate anche le prestazioni per sottolineare come l'uso di processori grafici possa essere estremamente utile per questo tipo di fini.

Il codice fa uso di wrapping, ossia di programmi in grado di far scorrere il flusso di uno script di Python in un codice precompilato di C++.In questo modo si è realizzato con codice Python uno script di avvio per le sessioni di calcolo che prepara file e report utili, recupera specifiche hardware e configurazioni del calcolo da eseguire. Laddove è richiesto un'intensivo calcolo lo script Python chiamerà una libreria precompilata scritta in C++. In questo modo il codice C++ potrà essere più essenziale e quindi leggibile.
Per fare un esempio di avvio di una comparazione tra opere, lo script principale di Python si occupa di preparare il sistema operativo, convalidare i parametri usati ed eventuali checkpoint. Quindi, fornisce tutti gli input richiesti al wrapper (scritto in C++) il quale convalida eventuali errori di sintassi e traduce i tipi di valori di Python in tipi comprensibili al C++. Di seguito, questi valori vengono passati ad uno starter che prepara un logger e legge eventuali file inizializzando di fatto la

memoria principale. Il programma scritto in C++ chiama il programma scritto in CUDA il quale si occupa di leggere i dettagli tecnici della GPU (usando apposite librerie) per configurare un piano di calcolo che massimizza le prestazioni e che sia robusto. Infine, verranno comunicate le istruzioni alla GPU come kernel (codice esplicito che compilerà la GPU stessa in runtime). Poiché la memoria della GPU è limitata, è importante far attenzione che i dati siano passati parzialmente e usare la memoria principale come appoggio.
I dati ottenuti dalla GPU vengono passati alla funzione chiamante in C++ la quale li memorizza in modo opportuno e restituisce il controllo allo script di Python chiamante. In caso di errori tutto viene documentato e riportato al chiamante per gestirli in modo opportuno.
Questo tipo di strategie per realizzare codice prestante sono tipiche nelle librerie di calcolo scientifico che spesso mostrano una funzione principale chiamante e funzioni compilate nascoste, un esempio è CuPy, una libreria Python la quale installazione prevede anche la compilazione.

3.2 Pre processing

Il pre processing è una fase importante del confronto tra opere ignote con il dataset. In particolare si preoccupa di far rispettare delle specifiche qualitative dell'immagine per lo più soggettive, non è quindi detto che si faccia uso dello stesso algoritmo per ogni immagine. Il fine ultimo è eliminare dati evidentemente inquinanti per far rispettare un certo standard e presentare l'immagine al software nel giusto modo. Essenzialmente si avvicina il dato reale al dato ideale ammissibile teoricamente dal software.
Come già illustrato in algorithm 4, il preprocessing adottato per i dati raccolti fa uso principalmente di una compressione con FFT. In questa sezione saranno illustrati i risultati passo passo seguendo l'algoritmo.

FFT Il primo step dell'algoritmo di preprocessing è applicare FFT all'immagine normalizzata, quindi con pixel grigi aventi valori di media 0 e varianza 1. Come osservato, attraverso immagini sintetiche, i quadretti nello sfondo delle pagine di appunti sono pattern ricorrenti che tipicamente non sono propri della scrittura di un testo. Questa ricorrenza è meglio visibile attraverso l'uso di trasformate di Fourier.
In fig. 3.2 è visibile il pattern dei quadretti come frequenze alte separatamente su x e su y di significiativa intensità.
Il programma rimuove il p-percentile delle frequenze con più alta intensità. Dopodiché ricostruisce l'immagine originale usando le frequenze rimaste, cui intensità non è stata annullata. In fig. 3.3 si vedono varie ricostruzioni usando differenti percentili e una normalizzazione tra $0, 1$ dei grigi ottenuti. Osserviamo come $p = 0.1\%$ è una scelta appropriata da cui cominciare le analisi.
Si può osservare che la normalizzazione effettuata per visualizzare queste immagini nasconde il fatto che i pixel sono in realtà tutti visibilmente uguali tra loro. L'idea quindi è individuare quanti pixel erano parte certamente dello sfondo bianco della pagina e quanti invece delle grafie. Analizzando le immagini si è osservato che i

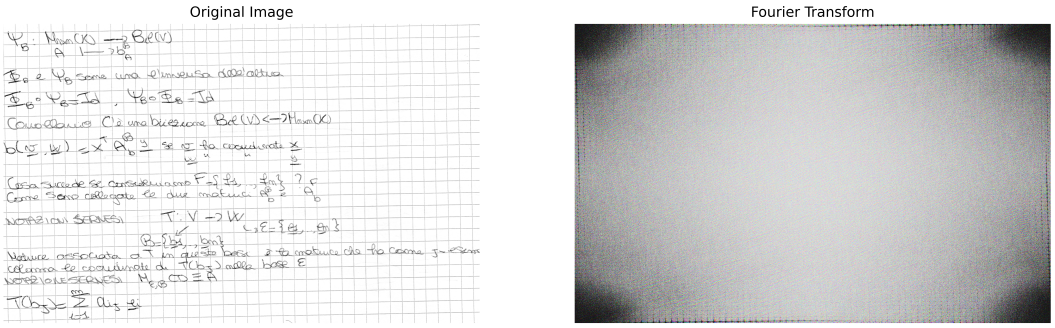


Figure 3.2. We can observe significant frequencies along the edge of the second image. The template of the sheet is captured from FFT, the frequencies along the edges are high for an ax and low for the other ax.

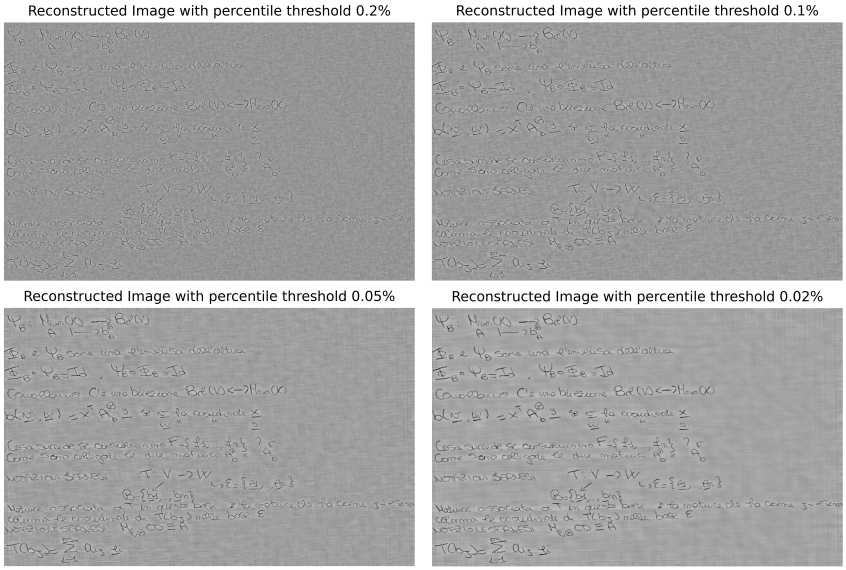


Figure 3.3. Si può osservare che un percentile tra 0.1% e 0.05% è un buon compromesso, i quadretti sembrano sparire e la scrittura rimane intaccata.

pixel con valore di grigio inferiore a 0.2 sono certamente parte delle grafie, invece se il valore è superiore a 0.8 sono certamente parte dello sfondo bianco della pagina. Quindi l'algoritmo recupera quanti pixel sono certamente delle grafie e quanti lo sono per la scrittura. In figura **??** un'illustrazione di questa analisi.

Mostrare i risultati del pre processing.

- serie di Fourier applicate a griglie sintetiche per dedurre quali sono le frequenze di nostro interesse: griglie dritte e ruotate, griglie chiare e scure, griglie con elementi inquinanti, confronto tra teoria e aspettativa
- serie di Fourier applicate a immagini vere, confronto con le immagini sintetiche
- mostrare alcuni risultati della pulizia dei quadretti attraverso le varie fasi dell'algoritmo di pulizia

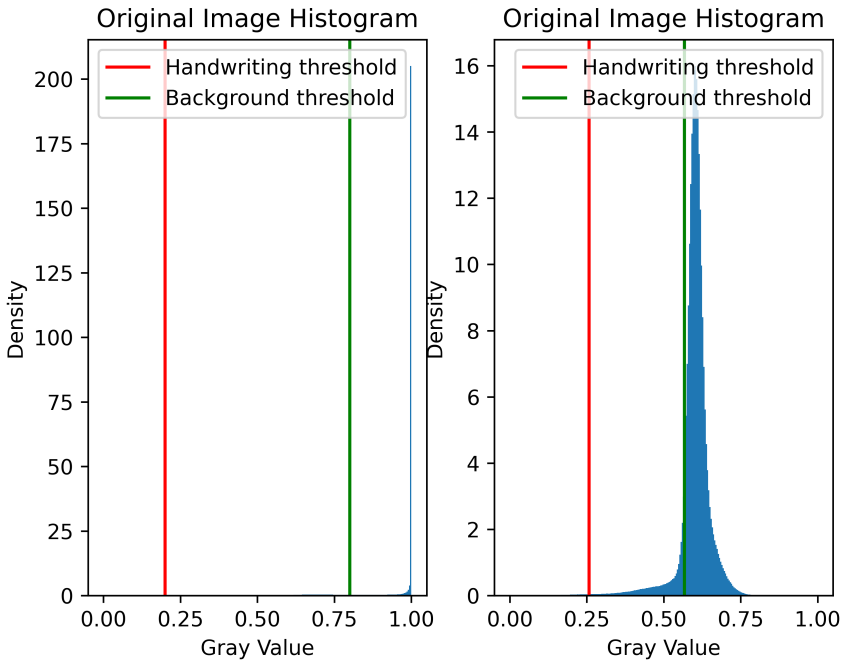


Figure 3.4. Si può osservare che la densità dei grigi nell'immagine originale presentava, per ovvie ragioni, molti pixel bianchi. Mentre nell'immagine ricostruita sono per lo più grigi. Per mantenere una distribuzione quanto meno simile a quella di partenza bisogna separare di più i colori tra loro.

3.3 Synthesis

Mostrare una piccola analisi delle sintesi ottenute e della velocità del programma, magari con dettagli implementativi

3.4 Comparison

Mostrare dettagli del clustering a livello esecutivo, come velocità del programma e confronto con l'esecuzione su CPU

3.4.1 Results

Mostrare dettagli in esecuzione dei vari valori ottenuti dall'algoritmo di comparazione. Quindi concludere con una panoramica dei risultati ottenuti.

Appendix A

KMeans, GMM, FCM compare figures

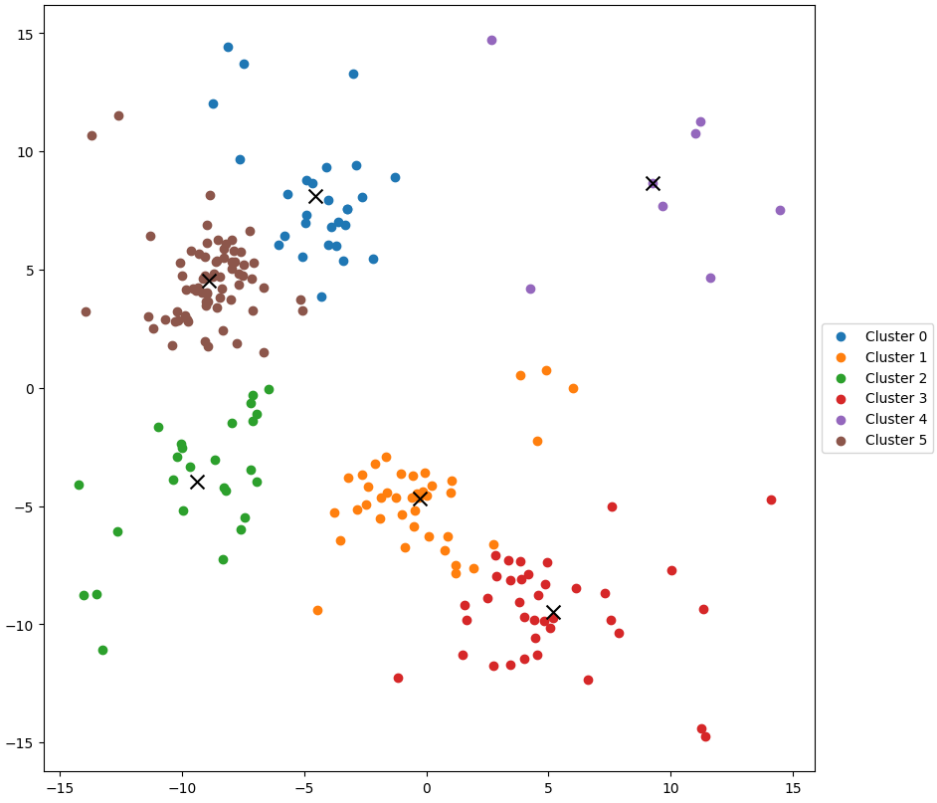


Figure A.1. The data points are coloured according to the calculated label and the estimated centroid is indicated with a `X`.

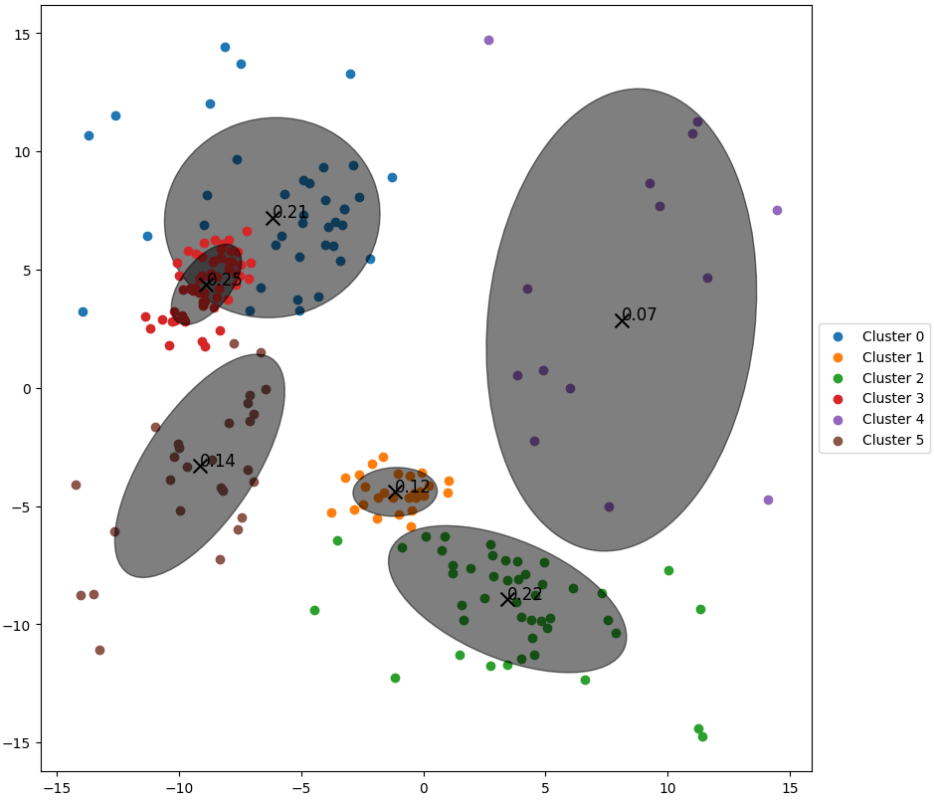


Figure A.2. The data points are coloured according to the Mahalanobis distance and the estimated centroid is indicated with a `X`. The ellipse of the normal distribution represents the covariance and is a confidence region of 95%.

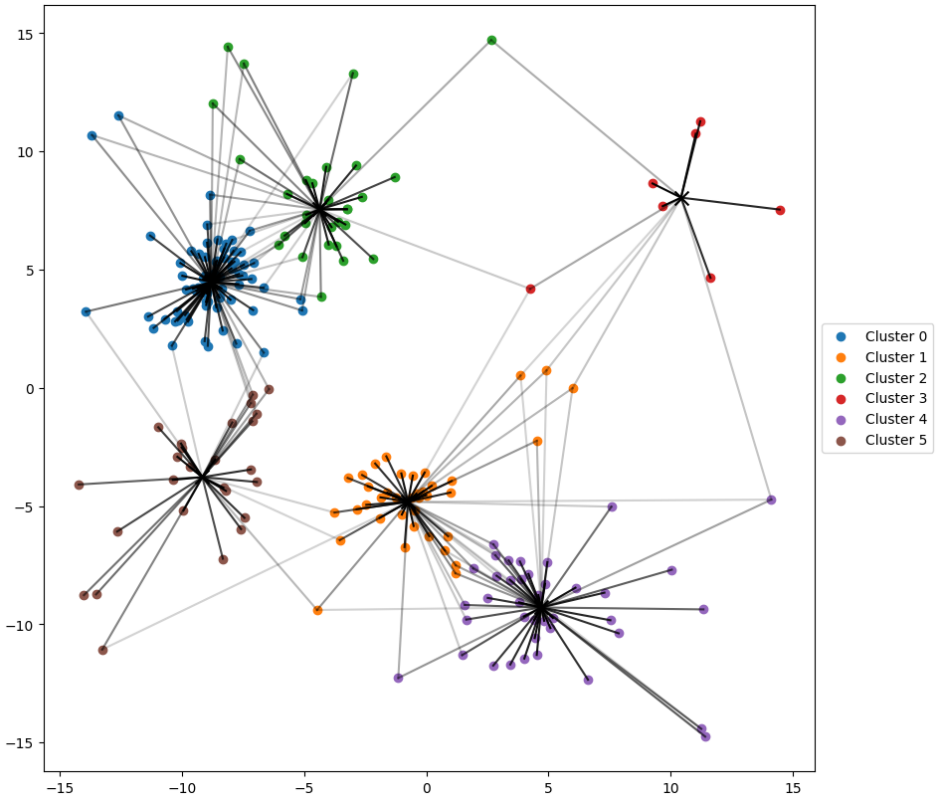


Figure A.3. The data points are coloured according to the most probable label and the estimated centroid is indicated with a `X`. The lines indicate the assignments with the greatest degree of affiliation of each point to the clusters; the darker the lines, the stronger the assignment.

Appendix B

FCM implementation with CUDA

CUDA è un'architettura hardware supportata dai processori grafici di *NVIDIA*, un'azienda statunitense. E' possibile realizzare il codice che definisce la comunicazione tra CPU e GPU in un dialetto del C++. Il codice che effettivamente esegue il processore grafico può essere già implementato con funzioni speciali delle librerie thrust e cuBLAS o realizzato personalmente attraverso dei kernel. In questa appendice si mostra il kernel usato per realizzare il calcolo della matrice U^2 in FCM. In particolare l'algoritmo è ottimizzato per compiere il calcolo su al più MAX_THREAD_PER_BLOCK centroidi, poiché nel medesimo

```
/**
 * @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 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,
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
  float value = 0;
  float min_value = 0;
  // compute the distance between the i-th data point and the j-th
  // centroid
```

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

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] 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.

[3] 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.

[4] 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/.

[5] 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.

[6] 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.

[7] 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.