Bachelor Degree Project

# Multi-Method NDI Detection Using a Layered Architecture Framework for Optimized Simplicity and Flexibility

9/4/2024

*Author 1:* Logan Fouts
*Author 2:* Henry Pap
*Supervisor:* Rafael Martins
*Semester:* VT/HT 2024
*Subject:* Computer Science

**Abstract**

It comes as no surprise; complex problems require complex solutions, but what if they don't always have to be this way? What if an approach was taken where current solutions are combined to not only reap benefits from each but to see if the whole is greater than the sum of its parts? Occam's Razor, a philosophical principle suggesting that the simplest explanation is often the correct one, is quite applicable to our situation. In today's digital world, we see a large increase in media. This media contains portions of exactly and nearly exactly duplicated image content that hogs precious storage space. Techniques to handle these near-duplicated images have grown into highly complex systems utilizing many methods in a combined manner. While effective, these methods are, by nature, hard to build. By building a framework that uses a layered architecture, we combine near-duplicate image detection algorithms in a new way to explore possible benefits. This approach allows for easy setup of image processing systems, and exhaustive testing can be done to quickly create a combination that suits the context.

To address the difficult task of near-duplicate image detection across highly varied use cases, we used a multi-method approach with a layered architecture at the core that combines algorithms easily. Each algorithm was selected for strengths in certain aspects of image comparison as well as for its behavior when combined with others. This allows for a robust, adaptable, yet simple detection system. Our methodology follows the design science approach and involves testing and tuning many different algorithm combinations and configurations to find the best performances in different image datasets.

The results indicate that our approach can retain the impressive accuracy, precision, and general efficacy seen in other multi-method approaches, while its optimized simplicity lends itself to easy adaptability to different types of images and duplication definitions. Furthermore, the filter-like idea seems to have benefits in terms of detection system execution time. This suggests that a layered multi-method approach may be able to effectively enhance the capabilities of NDI detection systems, making them approximately as effective but more adaptable in handling various situations.

**Keywords:** Near Duplicate Images, Image Processing, Layered Architecture, Algorithm Integration

# Contents

# 1 Introduction

The world as we know it is becoming more and more digital. Thus, the amount of image content that grows with this is increasing. However, with all this growth, not all image content is unique. A portion of it can be seen as either exact or near duplicates of other existing images. These images take up valuable storage space and computation power on both personal and cloud computers. With this situation, this thesis from Linnaeus University's Bachelor Programme in Software Technology hopes to contribute information to the intricate challenge of near-duplicate (ND) image detection across various domains and use cases. Traditional methods alone cannot be expected to work properly across all use cases while managing the fluid definitions of what might be considered a near-duplicate image. As Vonikakis et al. stated, "ND detection in personal photo libraries is a very challenging task, mainly due to the semantic gap, which may result in different interpretations between observers" [1]. Current solutions that aim to address this problem are often highly complicated tending to be less flexible.

As the sheer number of images grows, the need for more generalizable methods of detection on differing datasets and definitions of ND becomes clear. This work is an endeavor to engineer a new layered architecture-based framework—one that makes it easier to combine image processing algorithms for more flexible and effective solutions.

In this respect, the thesis proposes a possible solution to the missing link between theoretical and algorithmic complexity and engineering solutions with a flexible architectural approach. The following chapters look into the design of the architecture in more detail, describe the practical implementation, and outline the evaluation—all with a single aim: contributing an idea as to how modern solutions for multi-method image processing systems could be built.

Before we continue, a key definition should be made.

1. NDI means near duplicate image. Hence, ND means near duplicate.

## 1.1 Background

In modern times, society is finding that the amount of images existing is ever-increasing. In a study by the International Data Corporation (IDC) showed the digital fingerprint of the world is around `1.8ZB` in 2012 and by 2020 it should exceed `40ZB` where latest claims by 2025 we are going to reach `175ZB` [2]. These images are not all insignificant selfies; some may provide information for health care, important law enforcement concerns, surveillance logs, or any other important information. Thus, the necessity to efficiently manage and process these images in each unique domain is growing.

### Application Areas

Image processing, in general, is a very useful tool in many different areas for a plethora of reasons. Below are a few informal examples of areas in which customized algorithmic systems could provide benefits:

**Surveillance and Security:** Processing of images could more effectively remove old surveillance logs that are considered normal while avoiding any that have potential abnor-

malities. Furthermore, images from videos can be processed for automated monitoring without the need for human eyes.

**Digital Libraries:** In places where users store various types of media, such as pictures and videos, duplicate detection could prove a handy tool. The use of detection could greatly reduce storage resource utilization, which could help reduce overall computational resource utilization.

**Social Media:** Refined duplicate detection could be used heavily in this area to mitigate many issues. These include spam, copyright infringement, posting of illegal images, etc. This would have a direct effect on improving user experience within the platform.

### Research Area

The idea of duplicate image detection done in a new simple yet effective manner falls nicely in the field of image processing within computer science. More specifically, this project aims to contribute information and a new useful tool to the duplicate image detection research area. The framework built for the technique may be considered as part of the software engineering field.

### Current Knowledge and Why Change

Research for duplication image detection solutions includes work that ranges from basic hashing techniques to more intense ones, such as Convolutional Neural Networks (CNNs). All these have their strengths and weaknesses, usually with trade-offs between execution speed, accuracy, and, in some cases, the ability to apply on other datasets. In order to try to have the benefits while minimizing negatives, solutions nowadays largely use a combination of these individual methods. The main way in which this combination is accomplished seems to be a horizontal approach, but this proves to be highly complex and more difficult to get working. This approach is characterized by the combination of algorithms on the same layer. This means things such as hashes or features are combined between the algorithms into a larger knowledge; then this is used to make all decisions about whether two images are NDIs. Two examples of implementations using this approach are GeoMinHash[3] and the fingerprinting algorithm [4]. In our approach, a layered architecture as the core of the framework is proposed, where known methods are applied at each layer to progressively filter out images in hopes that suitable solutions for each problem domain can be made. Our approach aims to allow specifically tailored algorithms to be made in a more simple manner for whatever the user's input domain and NDI definitions are. This idea is analogous to a real-world filter, and we will refer to it as the vertical approach.

### Related work

In a paper written by Min et al. [5], a new approach for ND video clip detection is explored that is leveraged in order to increase accuracy. This technique is called bimodal fusion, and it involves combining low-level features with high-level semantic information. Inspired by this idea of combining different algorithms to extract more information, our proposed approach is a layered architecture that includes different algorithms at each level for NDI detection. Since their new approach provided promising results, we intend

to reform and expand upon it in a different direction.

Research from Chum et al. [3] builds upon the classic min-hash algorithm with its own Geometric Min-Hash or GmH, traditionally used for efficient similarity detection by visual word hashing. They identify unique central features that give spatial and scale information, which is used to perform secondary feature extraction and construct a richer set of geometric features. This method results in a higher discriminative representation of visual words that leverages the inherent strengths of the min-hash technique. The GmH approach demonstrates substantial improvements in recall and reduces the false positive rate, making it a powerful algorithm for image processing, particularly in ND detection. Similar to GmH, our methodology targets the use of many different techniques together, which can each individually extract different aspects of an image. However, our approach differs in that we aim to integrate various algorithms together vertically while GmH focuses on enhancing the min-hash algorithm. This aims to show improved results for near-duplicate image detection. GMH may, in the end, be one layer of our system that could be used in combination with others.

The work done by Dongre et al. [4] shows promising results by combining three distinct algorithms, Dhash, SIFT, and LSH models, in a horizontal manner. The initial two algorithms, Dhash and SIFT, are used for feature extraction, resulting in several hashes; these are combined into one condensed hash representation of a fingerprint image. LSH receives the hash representation from the previous step, where it employs a final feature extraction, which puts respective features into specific buckets, reducing the search space. Finally, hamming distance is applied to find matching fingerprints; their approach shows a significant promise of 99.99 percent accuracy. Their method is similar to ours in that they utilize the advantages of multiple algorithms. However, ours diverges as we adopt a more modular approach to the integration of algorithms by using a layered architecture. We believe this offers improved flexibility when choosing different algorithms, which can lead to better performance and similar efficacy throughout differing application areas.

### 1.2 Motivation

Microsoft conducted a study in 2012 that showed between 50% to 85% of their data is considered as near duplicated data [6]. The need for improved data storage devices grows with the rapid growth of data as a whole, "almost three-quarters of digital information is duplicated" [7]. This solution could have significant societal, ethical, and economic benefits. Further improvements in the detection of duplicate images would be an improvement in the current quality of information on the Internet, making access to the correct form more reliable and even lessening the amount of misinformation. Ethically, it complies with copyright enforcement, and hence the reduction of illegal content on the web [5]. Implementation for each specific domain could minimize economic waste of computing resources in a more ecologically disposed online world both for companies and individuals [8]. All these considered together, because of the complexities of current horizontal approaches, this thesis aims to allow for tailored NDI detection systems to be more accessible.

### 1.3 Problem formulation

The central challenge addressed by this thesis revolves around the complexities and inflexibilities inherent in current methods of near-duplicate image detection. As the digital

world expands, the need for effective and more easily adapted detection technologies grows as well. Current technologies often struggle to accommodate the variability in data sets and the lack of a uniform definition for what constitutes an NDI.

### Knowledge Gap

Despite all the progress in image processing techniques and the current horizontal approach, a gap remains in the development of a method that can dynamically combine algorithms into a pipeline that is adapted to various data sets and definitions of near-duplicate images while remaining relatively simple to understand. The existing solutions often require complex configurations that do not lend themselves to easy customization for diverse application domains. This gap highlights the necessity for a more flexible architectural framework that can simplify the customization process of combining many algorithms for specific user requirements.

The gap we intend to fill is the integration of NDI detection algorithms in a layered architecture for optimized simplicity while retaining benefits associated with the use of multiple methods. The resulting artifact will be a simple framework for building NDI detection systems in this manner. A visual representation of this approach is shown in Figure 1.1.
The concept of combining multiple algorithms into a single system is not a new idea; it has been explored in various use cases. However, its application in near-duplicate image detection seems unexplored.

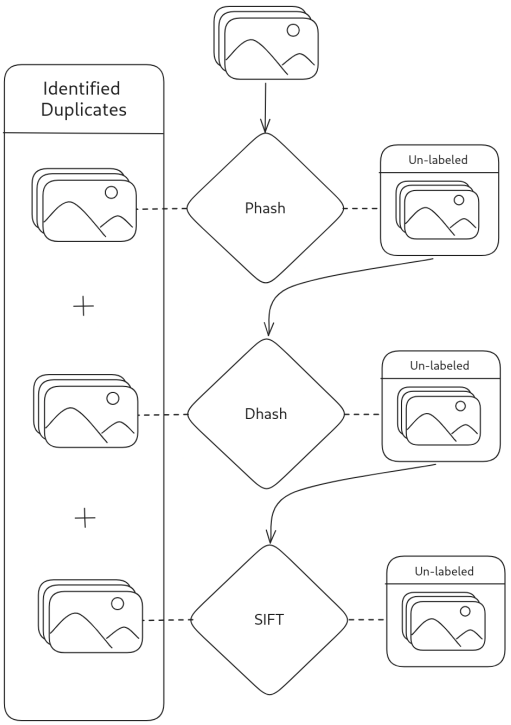


Figure 1.1: Vertical approach simplified flow.

To attempt to use inherent benefits from a layered architecture, the framework follows a few key concepts. The layers act together to form a sort of filter where, at the top, the fastest methods are placed, and they can get progressively slower toward the bottom. This way of building makes it easier to classify images that can be removed at an earlier time so the heavier, more accurate methods at the bottom do not have to do as much work. Furthermore, the methods placed before the most accurate method(s) should be tuned in such a way that they make the least amount of false positives possible. This is due to the fact that for the images labeled as duplicates, only one will go on to the next layer while the potentially many others will stay out, not giving the strongest method a chance to correctly classify them as not duplicates. Finally, the main driving idea behind the whole framework is that each method or layer should be a black box. Simply, a list of image paths is passed in, and classified image paths are returned. Thus, each layer is independent and can be moved and modified independently of the other algorithms that it may be

working with.

A clear potential downside to the filtering idea is that images that are labeled as duplicates do not move on to the next layer. This means that if mislabeled as duplicates, further layers cannot correct this mistake. In addition, the whole idea of filtering relies upon the idea that all duplicate images can be identified at the same layer. This assumption was too significant of a risk to accept, so to mitigate this issue, after each layer is run, a random image from each duplicate group is allowed to continue to run onto the next layer.

### Research Objectives

The primary objective of this research was to design and evaluate a versatile framework that simplifies the process of developing and deploying image detection algorithms. By integrating various algorithmic layers, this architecture aims to bring forward a solution that can easily be adapted to meet individual and domain-specific needs. This approach hopes to bridge the gap between complex theoretical image processing algorithms and practical engineering solutions, providing a more accessible approach that supports extensive customization.

### Problem Statements

The problem can be summarized into two main questions that guide our research and development:

1. Can a layered architecture be used to provide a flexible and effective way to combine image processing algorithms while retaining the benefits of multi-method approaches?

2. In what ways can the proposed framework improve the efficacy and flexibility of near-duplicate detection compared to existing methods?

## 1.4 Results

We evaluate our solution through design science, including controlled experiments, which are guided by methodologies seen in related work. Our results are compared with standard metrics such as accuracy, precision, recall, and efficiency that were detailed in articles we targeted as use cases. Finally, with good understanding, we assess the ease of use and understanding for our method compared to others.

### Evaluation Method

Direct comparisons are made under controlled conditions using selected metrics; evaluation is conducted with different image datasets to demonstrate the effectiveness and adaptability of this new solution in relation to results gathered from each specific article.

### Performance Metrics

The metrics used for this evaluation include F1 Score, Computational Efficiency, and Performance across differing datasets. These metrics are utilized to assess how this solution compares with other current solutions for each use case, as recall precision and F1 score are the most common metrics used for near-duplicate image benchmarking [1].

#### Expected Outcomes

The results of our experiment show two main ideas. The vertical approach can benefit from the multi-method approach, and it can provide competitive results even when placed against the benchmarks while remaining more simple.

### 1.5 Scope/Limitation

#### Development of the Layered Architecture

This involves the development of a modular framework based on a layered architecture that will allow for easy composition of various NDI detection algorithms. The approach should be flexible to allow for adaptation to specific needs within different datasets and use cases.

#### Evaluation through Controlled Experiments

In this evaluation, the effectiveness and performance of our proposed idea are compared with traditional methods of image detection, and examples of horizontal combinations of algorithms have been found. The goal is to see if a layered approach can be highly flexible as well and remain less complex than previous approaches.

## Limitations

#### Algorithm Selection

Existing image detection algorithms form the basis of our study. Our research covers tuning and combinations of these algorithms under the new architecture, but there is no development of completely unique detection algorithms within this thesis scope.

#### Data Constraints

Given the time and resources are very limited, the experiments are conducted with a small and limited subset of publicly available datasets. Even though datasets are selected to be representative of the different kinds of images and settings, these do not span all conceivable scenarios.

#### Generalizability of the Results

The level of generalizability that could be accorded to the results will mainly be based on the specific algorithms and datasets used in our research. For example, although valid for indicating the potential of the framework, these results should not be generalized to all cases of image data or detection scenarios.

#### Production Deployment

The thesis does not discuss the deployment of the project into any production environment. Most issues in scalability, continuous integration, and real-time processing are acknowledged but not the focus. With these boundaries set, the aim of the thesis is to take a focused look at the proposed architecture, understanding that there are practical limits met during its research.

## 1.6 Target group

The target group for our thesis is a broad set of people and organizations that stand to benefit from advanced, customizable NDI detection technologies. Some interested parties could include, academic researchers, software developers, digital content managers, security analysts, and social media organizations who require adaptable tools for managing large datasets of images across various domains. By addressing some of the needs of these groups, the thesis hopes to provide an idea that has many practical applications.

## 1.7 Outline

This paper is structured as follows:

- **Chapter 1, Introduction:** This chapter acts as the introductory part to the research problem; it outlines the significance and provides objectives.
- **Chapter 2, Method:** Involves a description of the methodological framework, research methods, and ethical considerations that were involved in the study.
- **Chapter 3, Theoretical Background:** We present a comprehensive account of the theoretical foundations of our study, which take care of existing knowledge gaps.
- **Chapter 4, Research Project Implementation:** This chapter elaborates on the aims of developing an evaluation method used to guide the design and implementation that follows.
- **Chapter 5, Results:** In this chapter, we present the first results of applying the developed technique.
- **Chapter 6, Analysis:** Herein, the analysis of the results obtained from the application of the technique across three case studies is provided.
- **Chapter 7, Discussion:** This chapter will discuss the design of the framework with theory and data.
- **Chapter 8, Conclusions and Future Work:** The final chapter will state the main findings of this research, discuss them, and suggest possible future research.
- **Chapter 9, References:** Lists all the cited sources used in the study to give the readers reference to the resources for further reading.

# 2 Theoretical Background

This section lays the foundational theories involving the integration of multiple near duplicate image processing algorithms within a layered architecture. It explores the growth of image detection technologies, and examines current known methodologies to highlight the exact contributions of this study.

## 2.1 Historical and Current Technologies

The field of image processing has started off with simple image manipulation techniques, such as cropping, resizing, and color adjustments. In recent times, it has grown into complex detection systems that utilize advanced techniques, including heavy artificial intelligence networks [9]. NDI detection, an area of critical importance within this field, has relied upon largely hash-based, feature extraction, and comparison techniques, each with its inherent advantages and disadvantages. Hash-based methods tend to be faster and simpler but are not able to understand higher-level aspects of images, while feature-based methods work well to understand and classify images but tend to be more complicated and more computationally expensive. In general, one method on its own cannot accommodate the complexities inherent in many NDI domains [10].

## 2.2 Combining Approaches

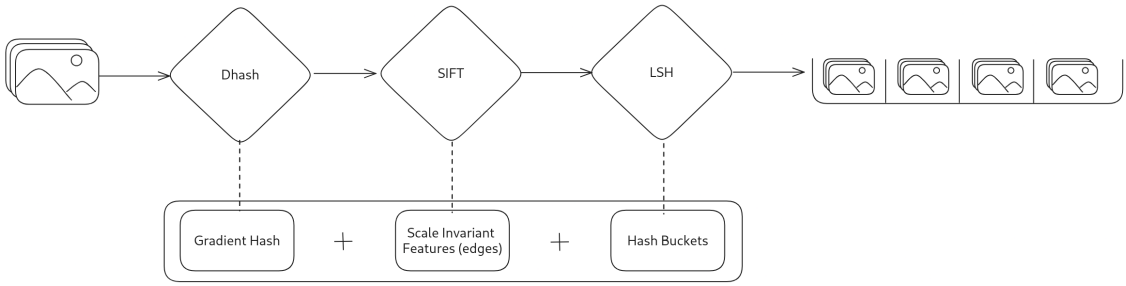


Figure 2.2: Horizontal approach simplified flow.

Each method has its pros and cons. It seems that if there were some way to benefit from many methods together in one algorithm, a very strong approach could be made. This leads us to the idea of fusing two or more different techniques together. As Szeliski discusses, integrating multiple image processing techniques can enhance performance beyond what is achievable with isolated applications, suggesting a compounded effect from diverse methodologies [11]. As of now, research in this field of NDI detection has used a horizontal approach where algorithms are used on the same layer directly together with combining hashes, features, etc; see the figure above. For example, Dongre et al. [4] combined three different algorithms together, Dhash, SIFT, and LSH, in order to accomplish astonishing results when compared to previous methods. Their approach can be seen simplified in Figure 2.2. However, this horizontal approach, despite being highly accurate, is also highly complex, difficult to implement, and hard to understand. This means that we do have a way of combining algorithms to bolster the performance as a whole, but the inherent complexities make it so significant understanding and expertise are needed to build in this manner for your own use cases.

## 2.3 Near Duplicate Image Philosophy

As discussed by Li et al., near duplicate detection is subjective and dependent on the context and intended application [12]. The need for all these different algorithms that target the same goal but try to accomplish it in different ways is indicative of this fact. A photo of a person smiling versus not smiling with the same background may not want to be identified as a duplicate, but then, for fingerprinting, high levels of accuracy are needed to distinguish what fingerprints are from the same person. An algorithm built for one domain may perform poorly on another. Data sets like California ND provide a non-binary ground truth in annotations. In simple words, they include multiple different people's opinions on what images are and are not duplicates. Figure 2.3 is a graph showing the distribution of agreement for image pairs from the ten people. This is a great example of how varied definitions of near duplicate images may even be within the same data set [13].

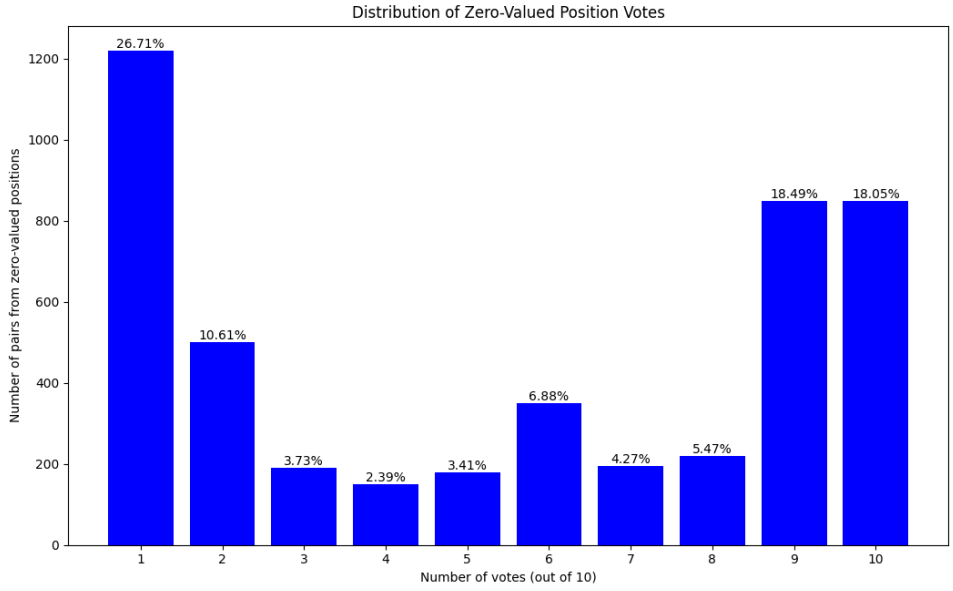


Figure 2.3: Reproduced results from California-ND article.

## 2.4 Similarity Measures

An essential part of near-duplicate image detection algorithms is the method used to measure the similarity between images. The choice of similarity measure can have a great impact on the performance of the systems, influencing both their accuracy and computational cost. Below are paraphrased overviews of a few common techniques as outlined by Murty et al. [14].

- **Euclidean Distance**: This method is typically used in feature-based methods where the features of two images are represented as points in a vector space. The similarity is calculated as the inverse of the distance between the points.
- **Cosine Similarity**: The cosine of the angle between two vectors. This method is useful when the scale of the vector is not important compared to the direction. This makes it suitable when working with high-dimensional data.

- **Jaccard Index**: Used with hash-based techniques, it measures similarity as the size of the hash set intersection in a high dimensional space divided by the total size of the union of different sets.
- **Hamming Distance**: Seen as a more simple approach, it is used with hash values to measure the number of different bits between two binary strings.

## 2.5 Review of Image Detection Techniques

An examination of the available image detection techniques reveals many different methodologies, from simple deterministic algorithms to complex neural networks. By understanding current methods, this research can identify areas where improvements can be made and how a layered approach can utilize them.

To give a quick understanding a few commonly used methods and their theories are explained briefly in the following section.

### 2.5.1 Dhash

Difference Hash (Dhash) is a well-known image hashing algorithm used for very quick near-duplicate image identification. It works by converting images into grayscale, resizing them, and then comparing adjacent pixel values to generate a binary hash, which is often then converted into a hexadecimal format. Then, typically, the hamming distance is used as the similarity measure [15].

### 2.5.2 MinHash

MinHash (Minimum Hashing) is an algorithm used to give a quick estimate of the Jaccard similarity between two sets. This makes it suitable for tasks such as the detection of duplicate or near-duplicate images. It is founded on the concept of locality-sensitive hashing, which is used to reduce the dimensionality of high-dimensional data while preserving similarity by keeping dimensions with the highest variability. MinHash builds a summary or "sketch" of a set (e.g., the characteristics of an image). Standard MinHash then takes a set intersection using the sketches as its similarity measure [16].

### 2.5.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a type of deep neural network whose architectures are excellent in the analysis of images. Their performance is one of the benchmarks across scopes of image processing. CNNs use an operation called convolution, which deals with data having a grid-like structure. Images are prime examples of this topology, as they are a matrix of pixels. They work by processing images through layers that apply differing filters to detect patterns (convolution layers). Some of these layers classify the image based on the extracted features (fully connected layers). The performance of these models is heavily dependent on the depth of the system. As Krizhevsky et al. observed in their experiments with CNN, "It is notable that our network's performance degrades if a single convolutional layer is removed. For example, removing any of the middle layers results in a loss of about 2 percent for the top-1 performance of the network. So the depth really is important for achieving our results" [17].

### 2.5.4 SIFT

Scale-Invariant Feature Transform (SIFT) is a computer vision algorithm made to find and describe important parts of an image. It performs very well when handling changes in images, such as scaling, rotation, or lighting changes. SIFT works by grabbing specific spots in an image that are considered unique and then describing them in a way that can be identified in separate images. We can say these key points are "invariant to scale changes" [4]. Examples of these key points and their matching counterparts can be seen in Figure 2.4.

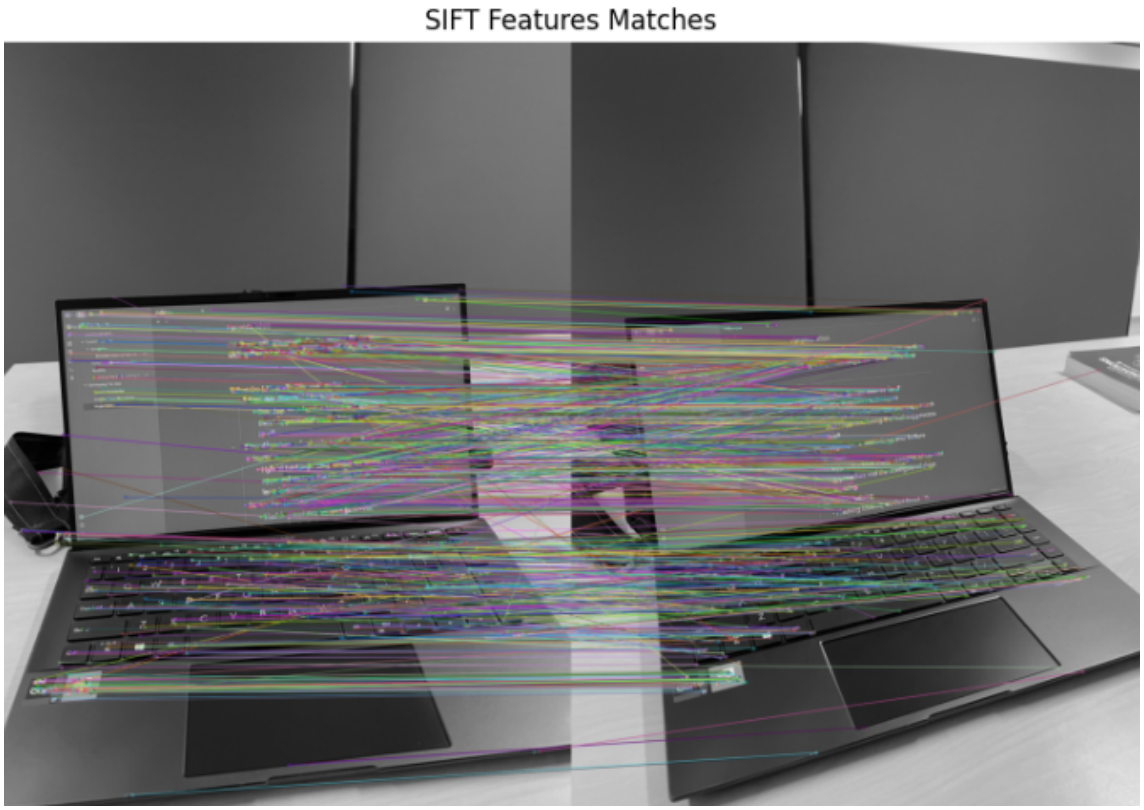


Figure 2.4: Sift Feature matching example.

## 2.6 Conclusion

This theoretical background helps to set the stage for the development of the layered architectural framework aimed at simplifying while retaining multi-method performance for NDI detection. By building on existing knowledge and introducing our own ideas, this research contributes to theoretical and practical advancements in the NDI field.

# 3 Method

## 3.1 Overview of Design Science

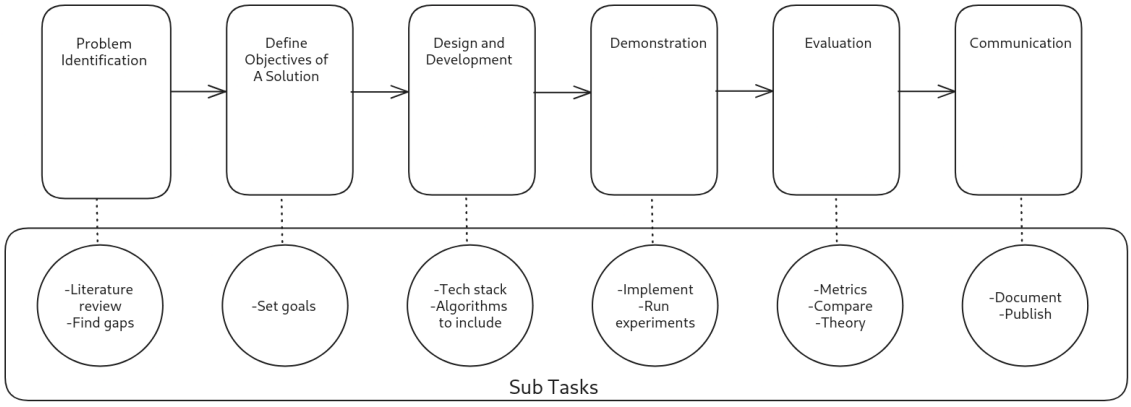


Figure 3.5: Our DSRM flow chart adapted from the DSMR article [18].

For our thesis, we use the design science research methodology (DSRM). It is aimed at the creation and evaluation of IT artifacts intended to solve identified problems, offering a rigorous process to design, evaluate, and communicate the artifacts and their attributes [18]. This description closely fits our needs as we design a framework that would be our artifact. Unlike purely empirical methods, such as observational studies, which hope to understand through observation, design science aims to address issues through the design and refinement of practical solutions. The DSRM process has six main steps: problem identification, definition of solution objectives, design and development, demonstration, evaluation, and communication. Our specific steps can be seen in Figure 3.5. This structured approach ensures comprehensive and proper treatment of the research and development of concrete solutions.

### Problem Identification

The challenge we address is within the definition and identification of NDI. Depending on the scenario and who you are asking, what exactly defines two images as near-duplicates can change vastly. After reviewing related NDI literature, this phenomenon is discussed often; a great example is in the classification of images in the California-ND paper [13] paper. Hence, there is a need for a method that could allow easy adaptation and configuration for this seemingly infinite range of situations.

The current methods lack flexibility, rely upon a horizontal approach of combining methods, and/or are simply too computationally expensive to be feasibly run on integrated or even consumer hardware. The horizontal approach is the closest to our proposed idea, and it is very effective when built correctly to address the issues of complex NDI definitions and scenarios. This can be seen in the GeoMinHash paper [3], and furthermore in the fingerprinting paper as well [4]. Despite the impressive performance, this approach is highly complex and requires deep knowledge in the area; therefore, building your own tailored system by combining and tuning methods in this way proves to be quite difficult.

### Objectives of a Solution

Since the combination of many methods shows promise, the goal for our thesis is a modular architecture that allows for the integration of various algorithms for NDI detection. This architecture aims to enhance flexibility by optimizing simplicity, all while maintaining the efficacy of multi-method approaches.

### Design and Development

We propose a layered architecture approach where different algorithms are placed as individual layers and are tuned individually first, then refined while combined. The use of many methods hopes to allow for better performance, much like the horizontal approach, while also providing improvements in terms of easy adaptability/flexibility.

Figure 3.6 is an example of how permutation tests are run. First, a list of potential algorithms and a number of algorithms to combine are provided to create the permutation sets. Then, for every set in the list of permutation sets we created, we create the layers and run the experiment.

```
def run_experiments(dataset, size):
    permutation_sets = permutations(list_of_algorithms, size)
    for set in permutation_sets:
        layered = Layers(set)
        results = layered.experiment(dataset)
```

Figure 3.6: Python code to run experiments using permutations of algorithm sets.

### Demonstration

The prototype is shown running two vastly different datasets: the Sokoto Coventry Fingerprint Dataset (SOCOFing) and the California-ND. Example images from these can be seen in Figures 3.7 and 3.8. Initially, controlled experiments are run on "training" subsets of the data for each dataset. This involves tuning individual layers and running automated testing of permutations (combinations) to determine the direction of the optimal configuration. After a configuration is decided upon, each layer is tuned by hand to improve performance further. This could mean in terms of execution time or accuracy, but in our situation, a balance of the two is targeted in both datasets.

Figure 3.7: example of fingerprints from SOCOFing dataset.

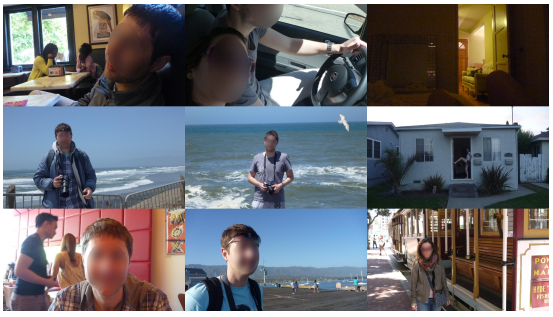

Figure 3.8: example of images from California-ND dataset.

**Details About the Datasets** SOCOFing [19] was created from 600 participants capturing each of their ten fingers, building a subset of 55,273 images in total. The SOCOFing is split into real, easy, medium, and hard subsets with a resolution of 96x103 pixels. Due to resource constraints, we will use a subset of the easy section for our demonstration and evaluation. This dataset is great for capturing high-detail differences in images that are all very similar. The California-ND data set [13] contains 701 images from a real user's holiday photo collection with a resolution of 1024×768. The target of the data was to represent a real personal photo collection. Ten participants have individually created annotations, making California a good set for detecting real-life near-image duplication.

### Evaluation

Our evaluation entails the use of both empirical and theoretical arguments. The prototype is compared against traditional and horizontal ND detection methods using metrics like precision, recall, runtime, and resource usage. Comparative analysis aims to highlight the potential advantages and disadvantages of our approach. Furthermore, a detailed assessment of our vertical approach compared to the horizontal is conducted. This provides insight into the less easily proven data aspects of each. The main issues are ease of use/implementation and complexity.

### Communication

Findings are documented and shared through scholarly publications in the form of the thesis paper. Additionally, all resources, code, and documentation developed throughout are available on GitHub (https://github.com/Logan-Fouts/Thesis).

## 3.2 Reliability and Validity

Reliability in research refers to the consistency of the results obtained from the study. It explores whether other researchers can produce similar results under the same conditions using the same methods. In our study, we aim to document in detail all relevant testing and experimentation information in order to allow others to run them for themselves.

Validity relates to the accuracy and legitimacy of our findings. To aid in the validity of our study, we engage in exhaustive testing to find a base algorithm and then tune the layers to fit the domain, maximizing our algorithm's ability to identify near duplicate images while limiting wrong classifications.

### 3.2.1 Addressing Reliability and Validity Issues

**Threats to Reliability**:

- The main potential threat to reliability comes from non-uniform test environments or inconsistent data sets. To mitigate this, we use subsets of defined image sets for each algorithm and make sure to have consistent testing conditions across all experiments.

**Threats to Validity**:

- One threat to validity in our study could come from the subjective definition of what constitutes a "near duplicate" image. To deal with this, we use clearly defined and

specific criteria for what constitutes near duplicity for each algorithm targeting a certain domain. When possible, we use a specific example of the ground truth from the articles.

- Most likely, the largest threat to validity would be the use of subsets of the original experiment data due to a lack of computing and time resources. This could lead to skewed or invalid results depending on how the subset is taken. To help minimize this issue and allow for more generalizability, subsets are chosen at random. However, the fact that only a subset is being taken will regardless limit the validity of our results.

- Quality of algorithm implementation could be a risk for our validity. Assuming one algorithm is implemented far better than another, this would mean a chance that our results would be skewed against the un-optimized algorithm versus optimized ones. This threat is mitigated since the majority of the algorithms we are using are implemented heavily using well-known libraries. Furthermore, since we generally know how an algorithm should behave on a dataset, we look for any irregularities in results.

- Algorithm Configuration Consistency could be a threat to our validity. Different configurations of the algorithms with different parameters could affect results. To avoid this, we ensure that all algorithms are configured with standard parameters across all test cases and document any changes to account for their impact on our results.

### 3.3 Ethical Considerations

For our study, we have made the following ethical considerations:

- **Confidentiality**: If the prototype was put into production, many security issues would need to be ironed out, such as securely storing images both in the databases as well as during processing. However, we do not intend for this to be put into production; simply, we are exploring if the idea of layers of image processing is a viable option to allow for greater flexibility as well as reaping the benefits of the filter-like flow.

- **Sampling and Bias**: The choice of images and datasets needs to be assessed to avoid biases that could skew the results. For instance, diverse datasets will be chosen to represent real-world scenarios and avoid overfitting, as well as to highlight the flexibility of the layered approach.

- **Risk of Harm**: Our research involves the processing of images and does not entail any human participation, so harm is not a concern.

- **Participation and Consent**: While our study does not involve people directly, the use of image datasets will be done following their terms of use. If user-generated content or more personal images are used, it will be done with the consent of the owners.

# 4 Research project – Implementation

This chapter aims to unfold the implementation of our new framework as well as the steps taken to evaluate and compare the effectiveness of our multi-method vertical approach for NDI detection. The objective of our implementation was to create a robust and simple architecture by which the integration of algorithms could be performed effectively to detect ND from datasets.

## 4.1 Permutation Experiments

The setup of permutations uses the `itertools` built-in Python library for its permutations function that takes two arguments, `list` and `radius`. This outputs a list of lists and looped to build a layer and pass the images to run experiments to get results, which are then saved to a desired folder within the outputs folder. The `experiment` function handles this part (inside the permutation file); it takes a couple of parameters:

1. **dataset**: a JSON structure that contains `paths` - array of image URLs.
2. **layer_size**: the radius for the `itertools` permutation function.
3. **dataset_size**: the number of images processed by the layer.
4. **presets**: configuration of algorithms containing name and parameters, normally an object within the `presets.json` file.
5. **config**: also the configuration of algorithms that mostly can be ignored but can be useful to experiment with parameters. If set to *"all"*, it uses the `presets` array instead.
6. **accuracy_calculator**: function to compute scores, defaults to the built-in method if none is passed.
7. **foldername**: destination subfolder within the outputs folder where results should be saved.

## 4.2 Manual Parameter Tuning

Users are responsible for manually adjusting algorithm parameters to optimize performance. This step can be tricky, but a general flow we have seen is to set parameters to have low false positives, especially for the faster algorithms. It is recommended to store these parameters; in our experiments, we store them inside the "presets.json" file for each experiment. Figure 4.9 is an example of how we store these parameters.

```
"california": {
    "dhash": [0.9],
    "phash": [11],
    "sift": [16, 1.6, 10, 3, 0.04, 0.1],
    "vgg": [0.7]
}
```

Figure 4.9: Preset we use for the California-ND dataset.

### 4.3 Implementation of Existing NDI Detection Algorithms

Algorithms that are readily available in libraries and that are also effective in their techniques were selected as the starting tools for the framework. The libraries used are imagehash [20], cv2 [21], and tensor flow [22]. The following are the algorithms we are using.

- **Dhash (Difference Hash)**: Difference in brightness of neighboring pixels.
- **Phash (Perceptual Hash)**: Phash allows for visual examination of image contents with the intention of finding relations between images in a more human vision-like approach.
- **Ahash (Average Hash)**: Finds duplicates through comparison of averages in pixel intensity.
- **SIFT (Scale-Invariant Feature Transform)**: Identification and description of local features in images.
- **VGG (Visual Geometry Group)**: Deep learning for recognizing patterns in images. The classification layer is left out.
- **SSIM (Structural Similarity Index Measure)**: Structural similarity measure of two images.

### 4.4 Framework Design

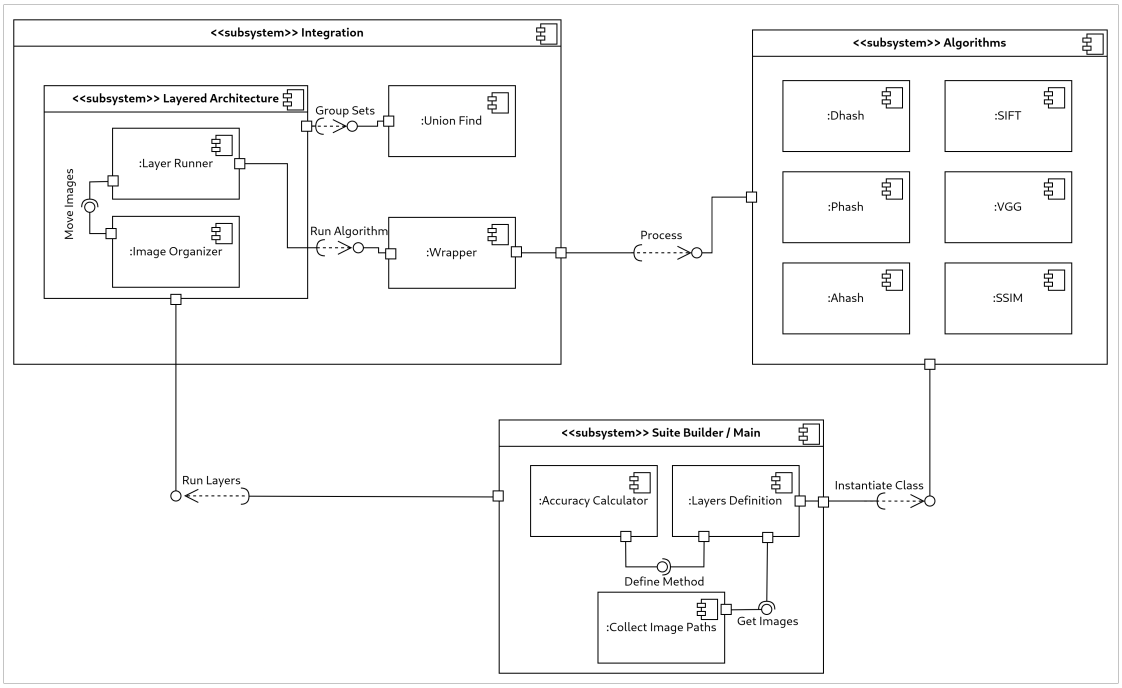


Figure 4.10: Component diagram of the framework architecture [23].

The framework is written in Python due to its widespread use in machine learning. It is structured around classes that allow for flexible integration of various image detection algorithms. A detailed component diagram in Figure 4.10 gives a visual as to what the system looks like. The main aspects to consider include:

- **Algorithms inside python classes**: Each detection algorithm is encapsulated in a Python class that provides a `process` function. This function accepts a list of image paths and updates two lists: one for images labeled as duplicates and another for those not labeled yet. This allows each algorithm to act as a black box, focusing solely on input and output. Parallel processing may be used within these layers.

- **Initialization parameters**: Each algorithm class constructor (`__init__`) takes named arguments as parameters, facilitating easy hyperparameter tuning. Default values are set for cases where no parameters are provided by the user.

- **Compatibility layer**: This layer wraps each algorithmic layer, ensuring methods are callable in the same way across all algorithms. Methods like `run`, `get_results`, `print_results`, etc., are included in the wrapper to abstract away complexities from the end user.

- **Layered architecture**: Algorithms are passed as a list of layers, each wrapped in the compatibility layer. Processes are completed in a filter-like manner, layer by layer, transferring selected images from one layer to the next and progressively refining the labeled sets of image paths. This includes using a union-find data structure to minimize duplicate processing. One image from each group of related duplicates is selected to continue on, minimizing inherent issues in the layered methodology.

- **Accuracy calculator**: Integrated within the architecture, this component computes accuracy at the end of processing. This method is tailored to specific domain requirements for metric calculations, with results used to calculate precision, recall, f1 score, and overall accuracy. Users, if needed, can define their own function and pass it as a parameter to the architecture class.

- **Organize images**: Automatically organizes processed images into specific group folders based on their classification, utilizing the union-find data structure to group-related sets.

- **Error handling**: Proper handling is implemented throughout the program to gracefully resolve potential errors.

- **Easy expansion**: The architecture allows for easy integration of other algorithms, whether directly or through external APIs. This scalability is crucial, as it demonstrates the framework's potential beyond our initial limited implementation.

No graphical interface has been implemented to allow the building of custom suites, but all a user needs to do to build with this approach is import the required modules, instantiate and put them into a list, and provide the desired path to the images folder. An example of this can be seen in Figure 4.11.

```
# Import modules
from Layers.layers import Layers
from Phash.phash import Phash
from SURF.surf import SURF
from SIFT.sift import SIFT

image_paths = get_image_paths(path)

layers = [Phash(), SURF(), SIFT()]
layered_architecture = Layers(raw_layers=layers, accuracy_calculator=
    None)

layered_architecture.run(test_paths)
```

Figure 4.11: Example setup of an algorithmic suite.

### 4.5 Dataset Utilization for Demonstration

Two different datasets were used to show the new framework's abilities. These are the SOCOFing and California-ND datasets.

- **Subset Selection**: For illustration purposes and due to constraints, random subsets of these images have been selected to illustrate the capabilities of our framework when tested under the same conditions as the articles we plan on comparing against.
- **Exhaustive testing of permutations**: Using a training subset of the data, all combinations of algorithms are tested. The suite with the best-recorded metrics is selected for further optimization by hand.
- **Manual Tuning**: The selected suite of algorithms is tuned by hand on the training subset of data to achieve a balance between accuracy and time taken.
- **Final suites tested**: Once tuned, each suite is run on increasingly larger testing sets of the data.

### 4.6 Data Collection

All experiments were conducted with the following hardware and software setup: 11th Gen Intel i7-1165G7 CPU, 16 GB RAM, and Linux kernel 6.8.9-arch1-1. A collection of outputs of the automated testing is maintained in text files. Each includes used algorithms with presets and the processing outcomes with execution times. These files are used for the comparative analysis and creation of relevant graphs using matplotlib [24] in order to provide insightful visuals. This includes both the permutation and final tests.

# 5 Results/Demonstration

In this section we present permutation results, individual final results, and combined suite final results without analysis. Along with combined tuned parameters and accompanying graphs for both the SOCOFing and California-ND datasets.

## 5.1 SOCOFing Data Set

### Permutations (Random Subset of size 90)

| Algorithms Combination | Time Elapsed (s) | Accuracy (%) | Groups of Duplicate Images |
|---|---|---|---|
| Phash[4], SIFT[17, 1.8, 10000, 3, 0.01], VGG[0.6] | 4.67 | 90.41 | 30 |
| Phash[4], SIFT[17, 1.8, 10000, 3, 0.01], Dhash[3] | 3.19 | 100.00 | 28 |
| Phash[4], Dhash[3], VGG[0.6] | 8.53 | 48.89 | 17 |
| Phash[4], Dhash[3], SIFT[17, 1.8, 10000, 3, 0.01] | 3.18 | 100.00 | 28 |
| Dhash[3], VGG[0.6], SIFT[17, 1.8, 10000, 3, 0.01] | 9.00 | 35.71 | 13 |
| Dhash[3], VGG[0.6], Phash[4] | 9.11 | 35.71 | 13 |
| Dhash[3], SIFT[17, 1.8, 10000, 3, 0.01], VGG[0.6] | 4.90 | 87.84 | 29 |
| Dhash[3], SIFT[17, 1.8, 10000, 3, 0.01], Phash[4] | 3.25 | 100.00 | 28 |
| Dhash[3], Phash[4], VGG[0.6] | 8.21 | 48.53 | 17 |
| Dhash[3], Phash[4], SIFT[17, 1.8, 10000, 3, 0.01] | 3.22 | 100.00 | 28 |
| Phash[4], VGG[0.6], Dhash[3] | 8.90 | 39.36 | 13 |
| Phash[4], VGG[0.6], SIFT[17, 1.8, 10000, 3, 0.01] | 8.98 | 39.36 | 13 |
| SIFT[17, 1.8, 10000, 3, 0.01], Dhash[3], Phash[4] | 3.26 | 100.00 | 28 |
| SIFT[17, 1.8, 10000, 3, 0.01], Dhash[3], VGG[0.6] | 4.35 | 96.34 | 30 |
| SIFT[17, 1.8, 10000, 3, 0.01], Phash[4], Dhash[3] | 3.26 | 100.00 | 28 |
| SIFT[17, 1.8, 10000, 3, 0.01], Phash[4], VGG[0.6] | 4.18 | 98.72 | 31 |
| SIFT[17, 1.8, 10000, 3, 0.01], VGG[0.6], Dhash[3] | 4.34 | 96.34 | 30 |
| SIFT[17, 1.8, 10000, 3, 0.01], VGG[0.6], Phash[4] | 4.30 | 96.34 | 30 |
| VGG[0.6], Dhash[3], Phash[4] | 8.62 | 23.99 | 6 |
| VGG[0.6], Dhash[3], SIFT[17, 1.8, 10000, 3, 0.01] | 8.88 | 23.99 | 6 |
| VGG[0.6], Phash[4], Dhash[3] | 8.79 | 23.99 | 6 |
| VGG[0.6], Phash[4], SIFT[17, 1.8, 10000, 3, 0.01] | 8.64 | 23.99 | 6 |
| VGG[0.6], SIFT[17, 1.8, 10000, 3, 0.01], Dhash[3] | 8.78 | 23.99 | 6 |
| VGG[0.6], SIFT[17, 1.8, 10000, 3, 0.01], Phash[4] | 8.68 | 23.99 | 6 |

Table 5.1: SOCOFing permutation results table.

Table 5.1 shows the result of different permutations of algorithms by running them on 90 random images from the SOCOFing dataset. The algorithm combinations show each of the layers and their respective parameter tunings. More information on the parameters can be seen in 5.13. Each row shows a unique permutation being used for NDI detection, providing run-time, accuracy, and the total count of groups of duplicate images it detected. Permutations vary in ways, such as computational efficiency and image detection capabilities.

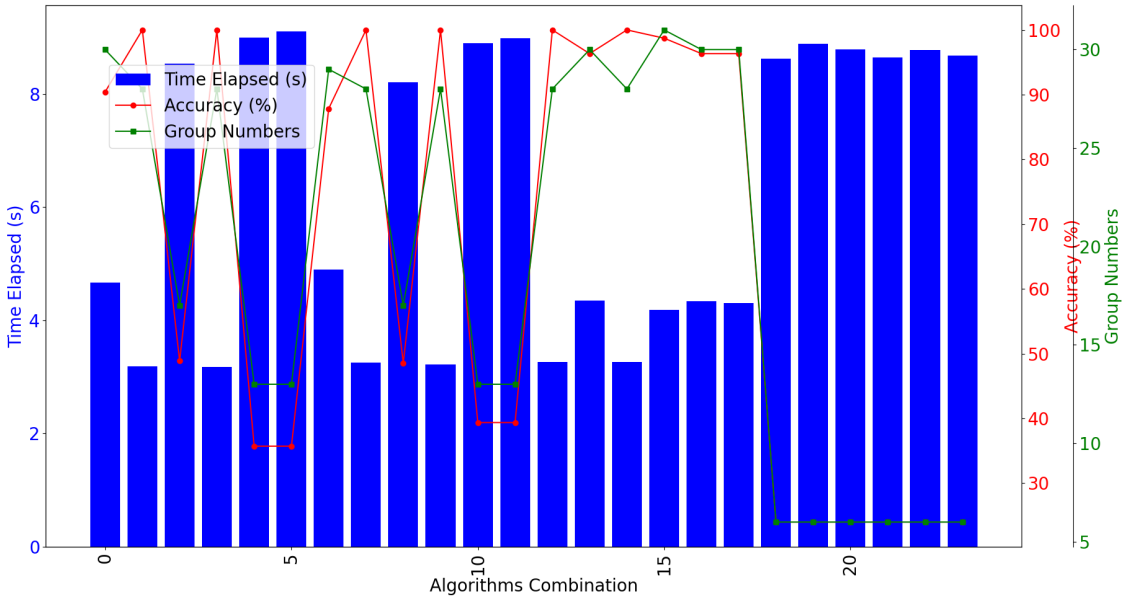


Figure 5.12: SOCOFing permutation results graph.

The graph 5.12 displays the results of various algorithm combinations tested on a random subset of 90 images from the SOCOFing dataset. The x-axis represents different combinations of algorithms used for NDI detection. The left y-axis (blue bars) shows the time elapsed for processing each combination in seconds. The right y-axis (red line) shows the accuracy percentage of each algorithm combination, and the green line indicates the number of groups of NDI's identified. The graph highlights the performance differences among the combinations, showing trade-offs between computational efficiency and detection accuracy.

### Final Experiment

```
"layers": [
  {"type": "phash", "parameters": {"threshold": 4}},
  {"type": "dhash", "parameters": {"sim": true, "threshold": 0.95
  {
    "type": "sift",
    "parameters": {
      "threshold": 13,
      "sigma": 1.2,
      "edge_threshold": 1000**10,
      "noctavelayers": 8,
      "contrast_threshold": 0.01
    }
  }
]
```

Figure 5.13: Tuned suite for SOCOFing dataset.

Figure 5.13 is the final configuration that defines a suite of algorithms for NDI detection on images taken from the SOCOFing dataset. It has three layers: the first perceptual

hash algorithm (phash) with a threshold of 4, the third is the difference hash algorithm (dhash) with similarity comparison and a threshold of 0.95, and finally the scale-invariant feature transform (SIFT) algorithm with a threshold of 13, sigma of 1.2, $1000^{10}$ as the edge threshold, 8-octave layers, and a contrast threshold of 0.01. This figure reflects the fusion of these delicately balanced permutations of algorithms from a suite developed specifically for the SOCOFing dataset.

**Individual and Combined Results**

| Data Size | TP | FP | FN | Precision | Recall | F1-Score | Accuracy | Elapsed Time (s) |
|---|---|---|---|---|---|---|---|---|
| 300 | 297 | 3 | 3 | 0.9900 | 0.9900 | 0.9900 | 0.9900 | 6.2310 |
| 600 | 593 | 7 | 7 | 0.9883 | 0.9883 | 0.9883 | 0.9883 | 18.3165 |
| 900 | 887 | 13 | 13 | 0.9855 | 0.9855 | 0.9855 | 0.9855 | 40.6721 |
| 1200 | 1186 | 14 | 14 | 0.9882 | 0.9882 | 0.9882 | 0.9882 | 72.0654 |
| 1500 | 1484 | 16 | 16 | 0.9893 | 0.9893 | 0.9893 | 0.9893 | 112.9932 |

Table 5.2: SOCOFing Dhash testing results.

| Data Size | TP | FP | FN | Precision | Recall | F1-Score | Accuracy | Elapsed Time (s) |
|---|---|---|---|---|---|---|---|---|
| 300 | 136 | 0 | 164 | 1.0000 | 0.4533 | 0.6239 | 0.4533 | 0.5607 |
| 600 | 264 | 4 | 336 | 0.9851 | 0.4430 | 0.6111 | 0.4400 | 1.6792 |
| 900 | 398 | 6 | 502 | 0.9851 | 0.4430 | 0.6111 | 0.4400 | 3.5723 |
| 1200 | 529 | 9 | 671 | 0.9832 | 0.4408 | 0.6094 | 0.4408 | 6.4721 |
| 1500 | 665 | 15 | 835 | 0.9779 | 0.4433 | 0.6109 | 0.4433 | 10.3482 |

Table 5.3: SOCOFing Phash testing results.

| Data Size | TP | FP | FN | Precision | Recall | F1-Score | Accuracy | Elapsed Time (s) |
|---|---|---|---|---|---|---|---|---|
| 300 | 300 | 0 | 0 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 8.9167 |
| 600 | 600 | 0 | 0 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 35.1776 |
| 900 | 900 | 0 | 0 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 79.6654 |
| 1200 | 1200 | 0 | 0 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 142.9932 |
| 1500 | 1500 | 0 | 0 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 221.4671 |

Table 5.4: SOCOFing Sift testing results.

| Data Size | TP | FP | FN | Precision | Recall | F1-Score | Accuracy | Elapsed Time (s) |
|---|---|---|---|---|---|---|---|---|
| 300 | 300 | 0 | 0 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 2.5530 |
| 600 | 598 | 0 | 2 | 1.0000 | 0.9967 | 0.9983 | 0.9967 | 9.1599 |
| 900 | 883 | 7 | 10 | 0.9921 | 0.9888 | 0.9905 | 0.9811 | 19.7361 |
| 1200 | 1162 | 23 | 15 | 0.9806 | 0.9873 | 0.9839 | 0.9683 | 33.8846 |
| 1500 | 1433 | 47 | 20 | 0.9682 | 0.9862 | 0.9772 | 0.9553 | 51.8370 |
| 3000 | 2796 | 153 | 51 | 0.948 | 0.982 | 0.964 | 0.932 | 187.554 |

Table 5.5: SOCOFing combined suite testing results.

The results presented in Tables 5.2-5.4 provide precision, recall, F1-score, and run time for each of the algorithms alone and the increasing data size. Results in Table 5.5 are where a suite of algorithms—Phash, Dhash, and SIFT were all combined as described in Figure 5.13. The results from Table 5.5 will be heavily used when comparing against existing benchmarks for this dataset.

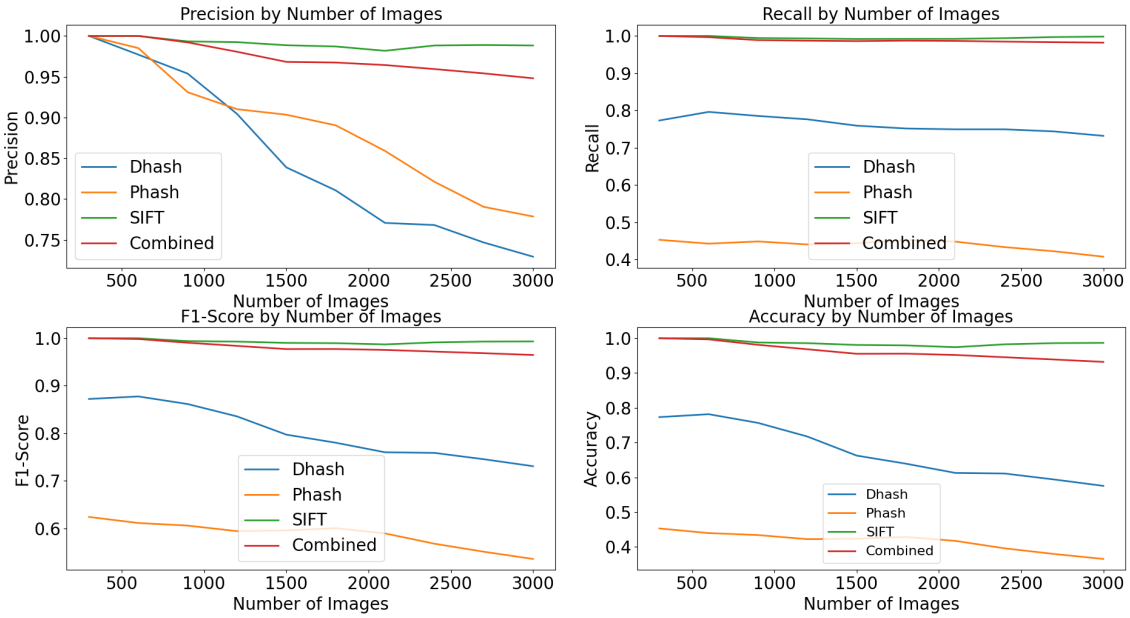


Figure 5.14: SOCOFing resulting metrics graph.

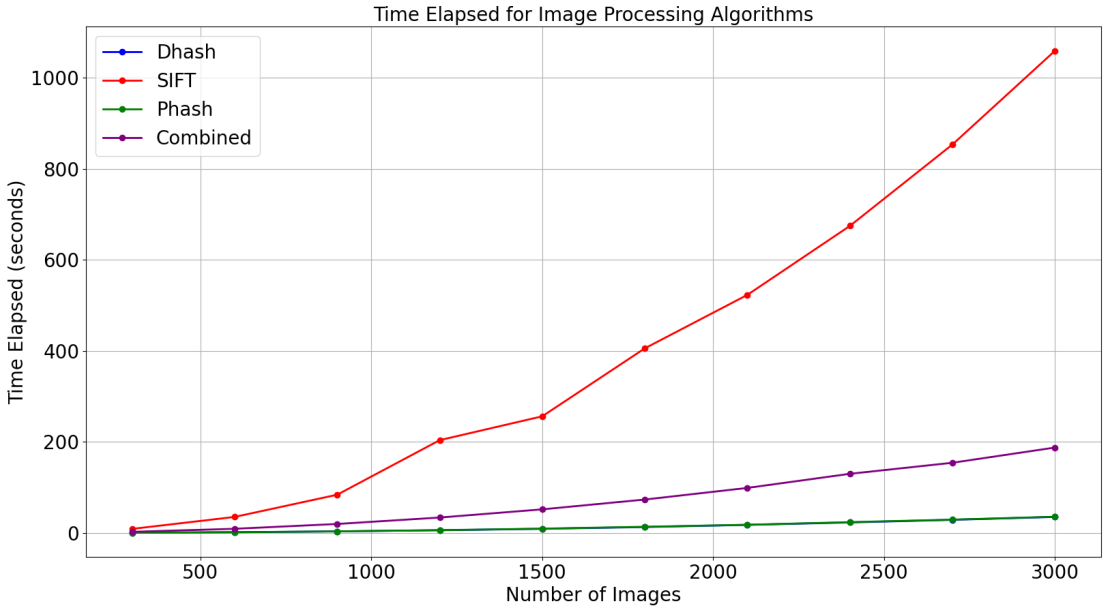


Figure 5.15: SOCOFing resulting elapsed time graph.

The graphs in Figures 5.14 and 5.15 plot the performance metrics and time elapsed for the NDI detection suites applied to the SOCOFing dataset as the number of images increases. Figure 5.14 displays the precision, recall, F1-score, and accuracy for the Dhash, Phash, and SIFT algorithms, as well as a combined suite of these algorithms. Figure 5.15 shows the time elapsed for processing using each Dhash, SIFT, Phash, and the combined suite.

## 5.2 California-ND Data Set

### Permutations

| # | Filename | Data size | TP | FP | FN | Precision | Recall | F1-Score | Accuracy | Elapsed Time (s) |
|---|---|---|---|---|---|---|---|---|---|---|
| **1** | **sift-dhash-vgg** | **613** | **512** | **10** | **91** | **0.9808** | **0.8491** | **0.9102** | **0.8352** | **170.5191** |
| 2 | dhash-sift-vgg | 613 | 512 | 10 | 91 | 0.9808 | 0.8491 | 0.9102 | 0.8352 | 173.2788 |
| 3 | sift-vgg-phash | 613 | 512 | 10 | 91 | 0.9808 | 0.8491 | 0.9102 | 0.8352 | 555.8749 |
| 4 | sift-vgg-dhash | 613 | 512 | 10 | 91 | 0.9808 | 0.8491 | 0.9102 | 0.8352 | 804.7373 |
| 5 | phash-sift-vgg | 613 | 511 | 9 | 93 | 0.9827 | 0.8460 | 0.9093 | 0.8336 | 151.2948 |
| 6 | vgg-dhash-sift | 613 | 511 | 8 | 94 | 0.9846 | 0.8446 | 0.9093 | 0.8336 | 320.8972 |
| 7 | dhash-vgg-sift | 613 | 510 | 8 | 95 | 0.9846 | 0.8430 | 0.9083 | 0.8320 | 206.3728 |
| 8 | vgg-sift-phash | 613 | 510 | 8 | 95 | 0.9846 | 0.8430 | 0.9083 | 0.8320 | 269.8956 |
| 9 | sift-phash-vgg | 613 | 510 | 11 | 92 | 0.9789 | 0.8472 | 0.9083 | 0.8320 | 558.8285 |
| 10 | vgg-phash-sift | 613 | 509 | 11 | 93 | 0.9788 | 0.8455 | 0.9073 | 0.8303 | 278.4222 |
| 11 | phash-vgg-sift | 613 | 507 | 11 | 95 | 0.9788 | 0.8422 | 0.9054 | 0.8271 | 201.7340 |
| 12 | vgg-sift-dhash | 613 | 503 | 17 | 93 | 0.9673 | 0.8440 | 0.9014 | 0.8206 | 271.1108 |
| 13 | **dhash-phash-sift** | **613** | **477** | 0 | **136** | **1.0000** | **0.7781** | **0.8752** | **0.7781** | **38.0510** |
| 14 | sift-dhash-phash | 613 | 475 | 2 | 136 | 0.9958 | 0.7774 | 0.8732 | 0.7749 | 57.9484 |
| 15 | dhash-sift-phash | 613 | 474 | 1 | 138 | 0.9979 | 0.7745 | 0.8721 | 0.7732 | 41.9424 |
| 16 | phash-dhash-sift | 613 | 473 | 3 | 137 | 0.9937 | 0.7754 | 0.8711 | 0.7716 | 32.2344 |
| 17 | sift-phash-dhash | 613 | 473 | 1 | 139 | 0.9979 | 0.7729 | 0.8711 | 0.7716 | 56.5184 |
| 18 | phash-sift-dhash | 613 | 471 | 3 | 139 | 0.9937 | 0.7721 | 0.8690 | 0.7684 | 32.4826 |
| 19 | dhash-vgg-phash | 613 | 421 | 8 | 184 | 0.9814 | 0.6959 | 0.8143 | 0.6868 | 231.5593 |
| 20 | vgg-phash-dhash | 613 | 421 | 8 | 184 | 0.9814 | 0.6959 | 0.8143 | 0.6868 | 264.1087 |
| 21 | vgg-dhash-phash | 613 | 421 | 8 | 184 | 0.9814 | 0.6959 | 0.8143 | 0.6868 | 297.9276 |
| 22 | dhash-phash-vgg | 613 | 420 | 8 | 185 | 0.9813 | 0.6942 | 0.8132 | 0.6852 | 181.9143 |
| 23 | phash-vgg-dhash | 613 | 420 | 8 | 185 | 0.9813 | 0.6942 | 0.8132 | 0.6852 | 228.7846 |
| 24 | phash-dhash-vgg | 613 | 418 | 8 | 187 | 0.9812 | 0.6909 | 0.8109 | 0.6819 | 172.0623 |

Table 5.6: California-ND permutations.

Table 5.6 provides performance results of algorithm permutations used on the California-ND dataset. It lists different combinations of algorithms, the size of the dataset, true positives (TP), false positives (FP), false negatives (FN), precision, recall, F1-score, accuracy, and the time taken to process each permutation. Two of our most exciting results are bolded. These are the most accurate, number 1, and our fastest, while still reasonably accurate, number 13.

### Tuning Experiments

```
"layers": [
  {"type": "dhash", "parameters": {"sim": true, "threshold": 0.9}},
  {"type": "phash", "parameters": {"threshold": 11}},
  {"type": "vgg", "parameters": {"threshold": 0.7}},
  {
    "type": "sift",
    "parameters": {
      "threshold": 16,
      "sigma": 1.6,
      "edge_threshold": 10,
      "noctavelayers": 3,
      "contrast_threshold": 0.04,
      "image_ratio": 0.1
    }
  }
]
```

Figure 5.16: Tuned suite for California-ND.

Figure 5.16 is the final configuration that defines a suite of algorithms for NDI detection on images from the California-ND dataset. It has four layers: the first is the difference hash algorithm (dhash) with similarity comparison and a threshold of 0.9, the second is the perceptual hash algorithm (phash) with a threshold of 11, the third is the Visual Geometry Group (VGG) algorithm with a threshold of 0.7, and finally the scale-invariant feature transform (SIFT) algorithm with a threshold of 16, sigma of 1.6, an edge threshold of 10, 3-octave layers, a contrast threshold of 0.04, and an image ratio of 0.1.

### Individual and Combined Results

| Data size | TP | FP | FN | Precision | Recall | F1-Score | Accuracy | Elapsed Time (s) |
|---|---|---|---|---|---|---|---|---|
| 100 | 50 | 0 | 50 | 1.0000 | 0.5000 | 0.6667 | 0.5000 | 1.2313 |
| 200 | 147 | 33 | 20 | 0.8167 | 0.8802 | 0.8473 | 0.7350 | 2.3574 |
| 400 | 197 | 146 | 57 | 0.5743 | 0.7756 | 0.6600 | 0.4925 | 4.6754 |
| 701 | 279 | 0 | 334 | 1.0000 | 0.4551 | 0.6256 | 0.4551 | 7.7574 |

Table 5.7: California-ND Dhash results.

| Data size | TP | FP | FN | Precision | Recall | F1-Score | Accuracy | Elapsed Time (s) |
|---|---|---|---|---|---|---|---|---|
| 100 | 69 | 0 | 41 | 1.0000 | 0.5900 | 0.7421 | 0.5900 | 1.8090 |
| 300 | 202 | 6 | 92 | 0.9712 | 0.6871 | 0.8048 | 0.6733 | 3.4085 |
| 701 | 319 | 0 | 294 | 1.0000 | 0.5204 | 0.6845 | 0.5204 | 7.8921 |

Table 5.8: California Phash results.

| Data size | TP | FP | FN | Precision | Recall | F1-Score | Accuracy | Elapsed Time (s) |
|---|---|---|---|---|---|---|---|---|
| 10 | 8 | 0 | 2 | 1.0000 | 0.8000 | 0.8889 | 0.8000 | 4.6451 |
| 100 | 78 | 0 | 22 | 1.0000 | 0.7800 | 0.8764 | 0.7800 | 49.6771 |
| 701 | 415 | 8 | 190 | 0.9811 | 0.6860 | 0.8074 | 0.6770 | 292.8172 |

Table 5.9: California-ND VGG results.

| Data size | TP | FP | FN | Precision | Recall | F1-Score | Accuracy | Elapsed Time (s) |
|---|---|---|---|---|---|---|---|---|
| 10 | 10 | 0 | 0 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 0.7154 |
| 80 | 76 | 0 | 4 | 1.0000 | 0.9500 | 0.9744 | 0.9500 | 1.0857 |
| 200 | 77 | 121 | 2 | 0.3889 | 0.9747 | 0.5560 | 0.3850 | 231.2505 |
| 400 | 338 | 0 | 62 | 1.0000 | 0.8450 | 0.9160 | 0.8450 | 22.7410 |
| 701 | 434 | 4 | 175 | 0.9909 | 0.7126 | 0.8290 | 0.7080 | 44.5817 |

Table 5.10: California-ND SIFT results.

The results presented in Tables 5.7-5.10 provide precision, recall, F1-score, and elapsed time for each of the algorithms—Dhash, Phash, VGG, and SIFT—applied to the California-ND dataset across increasing data sizes. These tables show how each algorithm performs individually with increasing data size while finding proper parameters.

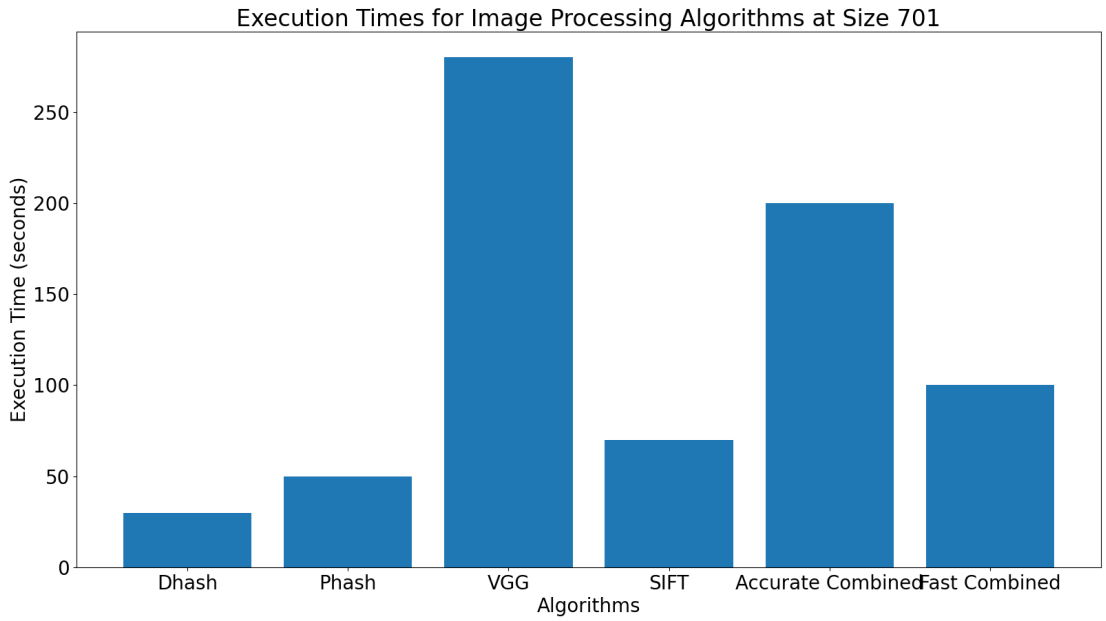


Figure 5.17: California-ND resulting execution times graph.

Figure 5.17 visually represents the run time for each algorithm alone as well as the combined suite. These metrics are taken from runs using the entirety of the dataset. The blue bars represent the time elapsed for each.

# 6 Analysis

## 6.1 Comparative Analysis

This section compares the results obtained from our layered architecture framework with existing studies on similar datasets. We focus on comparing the performance metrics such as precision, recall, F1-score, and accuracy.

### California-ND Dataset

**Existing Benchmark** Eshkol et al. [25] utilized the MPEG-7 Color Structure Descriptor for NDI detection on the California-ND dataset, yielding the following performance metrics:

- **Number of images**: 701
- **Precision**: 54.4%
- **Recall**: 78.2%
- **F1-Score**: 64.16%

**Our Framework Results** In contrast, our layered approach using a combination of Phash, Dhash, SIFT, and VGG algorithms produced notably higher performance metrics. These can be seen in Table 5.6:

- **Number of images**: 701
- **Precision**: Ranges from 98% to 100% based on permutation configurations
- **Recall**: Consistently above 84%
- **F1-Score**: Typically exceeds 90%

The improvement in precision and F1-score illustrates the effectiveness of our multi-method, layered architecture in handling the complex variations within the California-ND dataset. The layered approach not only improves accuracy but also maintains high recall rates, demonstrating robustness across varied image conditions.

### SOCOFing Dataset

**Existing Benchmark** The research on the SOCOFing dataset that we are comparing against uses the horizontal approach, where multiple algorithms operate at the same level, combining their outputs to enhance the detection process. These studies have achieved high performance, often reaching perfect metrics under controlled conditions. The exact results we are comparing against are as follows:

- **Number of images**: 17,000
- **Precision**: 1
- **Recall**: 1
- **F1-Score**: 1
- **Accuracy**: 1
- **Elapsed Time**: 50 minutes

**Our Framework Results** Our experiments on the SOCOFing dataset, employing a vertical layered architecture framework, yielded the following results for a size of 3000 images. This can be seen in Table 5.5:

- **Number of images**: 3,000
- **Precision**: 0.9481
- **Recall**: 0.9821
- **F1-Score**: 0.9648
- **Accuracy**: 0.9320
- **Elapsed Time**: 3 minutes

The results of our method against the benchmark show that ours performs slightly worse across all the precision, recall, F1-Score, and accuracy. While our results are not quite as good as what we are comparing against, it is notable that despite the vertical approach we were able to provide competitive results to the complex horizontal method used in the benchmark results. Our results should be taken with a grain of salt as they were taken from only a random subset of the full SOCOFing dataset.

### 6.2 Empirical Comparative Discussion

Our empirical observations seen in the final combined suite results of both experiments, made using our vertical layered architecture, argue that this approach can give competitive results when compared to horizontal approaches, specifically in the context of the SOCOFing dataset. Moreover, it can be seen that the vertical approach retains the benefits of the multi-method approach because it significantly outperforms the more standard approach used on the California-ND dataset. The empirical comparison, therefore, shows the advantages and disadvantages of each approach.

To discuss further, data seen in our results supports the conclusion that horizontal approaches are capable of delivering the maximal performance but at the cost of complex and therefore less flexible configurations. On the other hand, the vertical approach, though without the peak performance seen in the horizontal method, gives comparable performance, easy integration, and management of various algorithms with the ability to be highly adaptable to specific operational needs.

This flexibility is particularly important when rapid deployment, system modifiability, and operational simplicity are needed. The evidence suggests that vertical architecture with the framework as a tool to build it, can offer a ambitious performance appropriate for ever changing real-world applications, especially where systems require updating often to adapt to new data types or detection tasks or where simpler detection system building is desired.

### 6.3 Further Discussion

After researching the vertical method in comparison to the horizontal approach, we would like to present our understanding of both systems.

### 6.3.1 Horizontal Method

**Definition and Architecture:**

The horizontal method defines the relation of different algorithms at the same level. Their output is then aggregated. The usual methods to handle this are feature concatenation, ensemble averaging, or voting systems.

**Strengths:**

1. **Performance Optimization:** By using multiple algorithms simultaneously, horizontal methods can optimize for the best possible performance, achieving high accuracy and efficiency. The combination of outputs directly seems to allow for a greater understanding of an image.

2. **Image Understanding and Reliability:** This approach does not have the issue where some layers may never touch certain images if they are processed incorrectly before. Each method creates and combines an output for every image, thereby increasing the system's reliability.

**Weaknesses:**

1. **Complexity:** Managing multiple algorithms and how they are combined can significantly increase the system's complexity. This requires a deep understanding of each selected method's internal workings.

2. **Scalability Issues:** When new algorithms are needed and added, this would require significant reconfiguration of the system, making it more difficult to modify and adapt.

### 6.3.2 Vertical Method

**Definition and Architecture:**

Our vertical method structures the system as a set of layers, where each layer applies an algorithm. The output of one layer feeds directly into the next as a list of unclassified images. This allows for refinement of results as images are filtered out.

**Strengths:**

1. **Modularity and Flexibility:** Vertical architectures are inherently modular, making it easier to add or modify layers without needing to reconfigure the entire system.

2. **Ease of Implementation and Maintenance:** Since the layers are black boxes, each can be implemented and optimized independently, simplifying development and maintenance.

3. **Adaptability:** Vertical methods, since more modular, can adapt easier to different types of data or changing requirements, as changes can be made to specific layers instead of the whole system.

**Weaknesses:**

1. **Error Propagation:** Errors in early layers can propagate through to the rest, in worst cases potentially compounding and destroying the overall system accuracy. This is mitigated to an extent by passing images from each duplicate group to the next layer for further processing.

2. **Understanding of Images:** Since layers are separate from each other, they can not directly benefit from the understanding of the previous layers. This results in the layered approach performing slightly worse than the horizontal approach.

### 6.3.3 Comparative Implications

There are good reasons for choosing either horizontal or vertical methods, depending on the needs and constraints of the application. Ideal applications for horizontal methods are those where very high performance is critical and implementation time and maintenance are not a significant concern. In contrast, vertical methods are more flexible and easier to manage, being more appropriate for applications requiring frequent alteration or adaptation to new data types or processing techniques. Although potentially less performative, the vertical approach and the framework provide a structured way to easily integrate diverse algorithms and adapt to changes.

# 7 Discussion

Our research aimed to address the challenges of NDI detection by proposing a layered architecture-based framework that combines multiple detection algorithms. This approach was hypothesized to increase flexibility and simplicity while benefiting from these many algorithms in terms of accuracy and performance.

## 7.1 Integration of Multi-Method Approach

Our findings highlight the practicality of the vertical architecture, which effectively utilizes the power of diverse algorithms. This approach, characterized by a 'pipe and filter' paradigm, has demonstrated its adaptability to various datasets, which is often difficult for more conventional methods.

For instance, the application on the SOCOFing dataset showed that while the peak performance of horizontal methods might be better in a controlled environment, the vertical approach offers competitive accuracy with considerably reduced complexity and increased adaptability. This adaptability is what we are targeting and is crucial for real-world applications where media is ever-increasing.

## 7.2 Simplifying Complex Solutions

One of the original propositions of this project was inspired by Occam's Razor—simpler solutions are preferable under equivalent outcomes. Our framework architecture allows for a simple approach to combine or stitch algorithms together. By treating them as individual modules or black boxes, we simply give them the input they expect and collect what they would give without having to delve within. However, some inherent complexity still exists, such as tuning parameters and understanding which algorithms will be used at what layer will affect the parameters for the specific algorithm. Despite the lack of a proper user interface for building systems, the framework provides simple interfaces to interact with from inside the code, which greatly decreases the difficulty of building using our system.

## 7.3 Comparison with Existing Approaches

The results closely align with the current horizontal approach while outperforming the more traditional single-method approaches when it comes to image detection results. Ours, however, offers great flexibility, as demonstrated by its performance on the two very different datasets. This flexibility has only been seen in rich CNNs with millions or billions of parameters, which comes at high costs—something our approach avoids.

# 8 Conclusions and Future Work

This thesis presents the effectiveness of the layered architecture-based framework in NDI detection using the SOCOFing and California-ND datasets. It achieves results that are similar to the alternatives or, in the case of California-ND, exceed the benchmarks set by the existing study we are comparing. This argues the promising ability of this framework and approach in terms of its high accuracy, precision, recall, and F1 scores. The approach has been successful in terms of both matching benchmarks and improving simplicity thanks to the multi-method approach and its modular design. Furthermore, easy integration of different algorithms allows for quick adaptation and robustness to the variations in different datasets. These findings of our research contribute a useful tool as well as information to the image processing community.

## 8.1 Future Work

The promising results of this research open up several avenues for future studies and development:

- **Algorithm Integration:** To further increase both the potential adaptability and performance of our framework, more algorithms should be implemented inside of the Python classes.
- **Further domain testing:** Testing the framework on more diverse datasets of images would be useful in further assessing the adaptability and robustness of the framework under various situations.
- **Usability and Deployment:** End user studies targeting the ease of use of the framework could provide some insights as to how simple the system really is or isn't.
- **Use of better hardware:** Running some experiments on the framework using significantly better hardware would be very interesting to see. As of now, we are limited to our laptops, but better hardware could further argue whether the approach really is effective or not.
- **Image domain presets:** For increase ease of use, more presets for differnt domains of image could be made. This would allow a user to simply pick their domain and go from their.
- **Ethical and Privacy Considerations:** The framework currently will require some re-work with respect to privacy. Data would need to remain safe both during storage and during processing.

By highlighting the potential here, we hope to encourage further development and research into this approach. The open-source aspect of this project should allow any interested parties to make pull requests with changes or fork their own versions.

## References


[1] V. Vonikakis, A. Jinda-Apiraksa, and S. Winkler, "Photocluster a multi-clustering technique for near-duplicate detection in personal photo collections," in *2014 International Conference on Computer Vision Theory and Applications (VISAPP)*, vol. 2, 2014, pp. 153–161.

[2] J. Gantz and D. Reinsel, "The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east," *IDC iView: IDC Analyze the future*, vol. 2007, no. 2012, pp. 1–16, 2012.

[3] O. Chum, M. Perd'och, and J. Matas, "Geometric min-hashing: Finding a (thick) needle in a haystack," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 17–24.

[4] S. Dongre, S. Mehta, P. Raut, and V. Kulkarni, "Elevating fingerprint matching with a unified dhash, sift, and lsh model," in *2023 IEEE International Conference on Computer Vision and Machine Intelligence (CVMI)*. IEEE, 2023, pp. 979–8–3503–0514–2/23/31.00.

[5] H. seok Min, J. Y. Choi, W. De Neve, and Y. M. Ro, "Bimodal fusion of low-level visual features and high-level semantic features for near-duplicate video clip detection," *Signal Processing: Image Communication*, vol. 26, no. 10, pp. 612–627, 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0923596511000397

[6] D. T. Meyer and W. J. Bolosky, "A study of practical deduplication," *ACM Trans. Storage*, vol. 7, no. 4, feb 2012. [Online]. Available: https://doi.org/10.1145/2078861.2078864

[7] S. Ahmed and L. George, "Lightweight hash-based de-duplication system using the self detection of most repeated patterns as chunks divisors," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, 04 2021.

[8] S. F. N. Shandiz, "A fast and low-cost method to detect near-duplicate images in large dataset based on fingerprint extraction and deep learning."

[9] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed. Pearson, 2018.

[10] P. Ghosh, E. D. Gelasca, K. R. Ramakrishnan, and B. S. Manjunath, "Duplicate image detection in large scale databases," *World Scientific Review*, vol. 9.75, pp. 1–17, 2007.

[11] R. Szeliski, *Computer Vision: Algorithms and Applications*. Springer Science & Business Media, 2010. [Online]. Available: https://www.springer.com/gp/book/9781848829343

[12] S. Li and A. Jain, *Handbook of Face Recognition*, 01 2004.

[13] A. Jinda-Apiraksa, V. Vonikakis, and S. Winkler, "California-nd: An annotated dataset for near-duplicate detection in personal photo collections," in *2013 Fifth International Workshop on Quality of Multimedia Experience (QoMEX)*, 2013, pp. 142–147.

[14] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," *ACM Comput. Surv.*, vol. 31, no. 3, p. 264–323, sep 1999. [Online]. Available: https://doi.org/10.1145/331499.331504

[15] N. Krawetz, “Kind of like that,” The Hacker Factor Blog, January 2013. [Online]. Available: https://www.hackerfactor.com/blog/index.php?/archives/529-Kind-of-Like-That.html

[16] O. Chum, J. Philbin, and A. Zisserman, “Near duplicate image detection: min-hash and tf-idf weighting,” 01 2008.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Commun. ACM*, vol. 60, no. 6, p. 84–90, may 2017. [Online]. Available: https://doi.org/10.1145/3065386

[18] K. Peffers, T. Tuunanen, M. Rothenberger, and S. Chatterjee, “A design science research methodology for information systems research,” *Journal of Management Information Systems*, vol. 24, pp. 45–77, 01 2007.

[19] Y. I. Shehu, A. Ruiz-Garcia, V. Palade, and A. James, “Sokoto coventry fingerprint dataset,” 2018.

[20] J. Buchner, “Imagehash,” 2023. [Online]. Available: https://github.com/JohannesBuchner/imagehash

[21] G. Bradski and A. Kaehler, “Opencv,” https://opencv.org/, 2000.

[22] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[23] “Component diagrams,” https://www.uml-diagrams.org/component-diagrams.html, accessed: 2023-05-07.

[24] J. D. Hunter, “Matplotlib: A 2d graphics environment,” pp. 90–95, 2007. [Online]. Available: https://matplotlib.org

[25] A. Eshkol, M. Grega, M. Leszczuk, and O. Weintraub, “Practical application of near duplicate detection for image database,” in *Multimedia Communications, Services and Security*, A. Dziech and A. Czyżewski, Eds. Cham: Springer International Publishing, 2014, pp. 73–82.