FINGERPRINT CLASSIFICATION USING KMCG ALGORITHM UNDER VARYING WINDOW AND CODEBOOK SIZES

WINNIE GIFT ODONGO

MASTER OF SCIENCE IN

SOFTWARE ENGINEERING

JOMO KENYATTA UNIVERSITY

OF

AGRICULTURE AND TECHNOLOGY

2026

Fingerprint Classification Using Kmcg Algorithm under Varying Window and Codebook Sizes

Winnie Gift Odongo

A Thesis Submitted in Partial Fulfilment of the Requirements for the Degree of Master of Science in Software Engineering of the Jomo Kenyatta University of Agriculture and Technology

2026

DECLARATION

This thesis is my original work and has not been presented for a degree in any other University.

Signature..... Date: .....

Winnie Gift Odongo

This thesis has been submitted for examination with our approval as the University Supervisors:

Signature.....Date.....

Prof. Waweru Mwangi, PhD

JKUAT, Kenya

Signature.....Date: .....

Dr. Richard Rimiru, PhD

JKUAT, Kenya

DEDICATION

To my beloved Husband Isaiah Odhiambo Sei, Sons Ryan and Rylan Sei, my daughters Rehema and Radiant Sei, Brothers, Sisters and Friends, I love you all. Your support and prayers have enabled me to come this far. May almighty God bless you all abundantly and may you live long to enjoy the fruits of your hard work.

ACKNOWLEDGEMENT

First and foremost, I humbly express my greatest gratitude to the Almighty God for grace, strength, wisdom and favour throughout my study. Also, I would like to thank my supervisors, Prof. Waweru and Dr. Rimiru for supporting, guiding, inspiring and restraining me during my work presented in this thesis. I thank all my friends who tirelessly supported me and encouraged me throughout this study. Especially I would like to appreciate my friends Antonne and Everlyn for their support and kindness throughout my study. I thank all the SCIT staff at JKUAT for creating a professional educational climate that facilitated my studies. A special thanks to all my classmates for their support. Especially, I would like to thank Mr. Ominde for providing the opportunity for many rewarding discussions on Software engineering topics without withholding any information. I am grateful to my husband Isaiah Sei and all my family members for their support, love and for putting up with me from the beginning to the end of this thesis. To God be the Glory.

TABLE OF CONTENTS

DECLARATION ..... **ii**

DEDICATION ..... **iii**

ACKNOWLEDGEMENT ..... **iv**

LIST OF TABLES ..... **x**

LIST OF FIGURES ..... **xi**

LIST OF APPENDISES ..... **xii**

ABREVIATIONS AND AGRONYMES ..... **xiii**

ABSTRACT ..... **xiv**

CHAPTER ONE **1**

INTRODUCTION **1**

1.1 Background of the Study 1

1.2 Problem Statement 2

1.3 Rationale 3

1.4 General and Specific Objectives 4

1.4.1 General Objective 4

1.4.2 Specific Objectives 4

1.5 Research Questions 5

1.6 Structure of the Thesis 5

CHAPTER TWO 7

LITERATURE REVIEW 7

2.1 Introduction 7

2.2 Biometric Systems and Fingerprint Recognition 7

2.3 Fingerprint Pattern Classes 8

2.4 Early Fingerprint Classification and Techniques 10

2.5 Feature Extraction in Fingerprint Classification 11

2.6 Kekre's Median Codebook Generation (KMCG) 12

2.6.1 Concept and Theoretical Basis of KMCG 12

2.6.2 KMCG and Traditional Vector Quantization Techniques 13

2.6.3 Relevance of KMCG to Fingerprint Classification 14

2.6.4 Strengths and Limitations of KMCG 15

2.6.5 Effect of Window and Codebook Size Variation in KMCG on Fingerprint Classification 15

2.6.6 Implications for the Current Study 19

2.7 Traditional Machine Learning Models for Fingerprint Classification 19

2.7.1 Support Vector Machine 20

2.7.2 k-Nearest Neighbours 20

2.7.3 Random Forest 21

2.7.4 XGBoost (Extreme Gradient Boosting) 22

2.7.5 Comparative Review 23

2.8 Deep Learning in Fingerprint Recognition 24

2.8.1 MobileNetV2 25

2.8.2 InceptionV3 26

2.8.3 DenseNet201 27

2.8.4 Strengths and Limitations of Deep Learning Models 28

2.8.5 Transfer Learning in Fingerprint Recognition 29

2.8.6 Comparative Review 29

2.9 Research Gaps and Motivation for Current Study 31

2.10 Summary 32

CHAPTER THREE 34

METHODOLOGY 34

3.1 Introduction 34

3.2 Research Design 34

3.3 Dataset Description 36

3.4 Data Preprocessing 37

3.5 Feature Extraction Using KMCG 38

3.6 Feature Extraction Using PCA 40

3.7 Model Selection and Training 41

3.7.1 Traditional Machine Learning Model Selection and Training 41

3.7.2 Deep Learning Model Selection and Training 43

3.8 Evaluation Metrics 45

3.9 Tools and Techniques 45

3.10 Ethical Considerations 45

CHAPTER FOUR **47**

RESULTS AND DISCUSSION **47**

4.1 Introduction 47

4.2 Results 47

4.2.1 Effect of varying KMCG window and codebook sizes on fingerprint classification performance 47

4.2.2 Performance of KMCG-Based Feature Extraction 50

4.2.3 Performance of PCA-Based Feature Extraction 53

4.2.4 Performance of Deep Learning Models 55

4.2.5 Comparative Analysis Across Methods 57

4.3 Discussion 59

4.3.1 Effect of KMCG Window and Codebook Size Variation 59

4.3.2 KMCG-Based Machine Learning Compared with Previous Studies 60

4.3.3 PCA-Based Models Compared with KMCG and Literature 61

4.3.4 Deep Learning Performance Compared with Previous Studies 62

4.3.5 Dataset, Preprocessing, and Evaluation Differences 63

4.3.6 Computational Cost and Practical Trade-Offs 63

4.3.7 Generalizability, Limitations, and Practical Implications 64

CHAPTER FIVE **66**

CONCLUSION **66**

5.1 Summary of the Study 66

5.1.1 Objective 1: Effect of Varying KMCG Window and Codebook Sizes 66

5.1.2 Objective 2: Performance of Traditional Machine Learning Using KMCG Features 66

5.1.3 Objective 3: Performance of PCA-Based Feature Extraction 67

5.1.4 Objective 4: Comparative Performance Across KMCG, PCA, and Deep Learning Models 67

5.2 Implications and Recommendations 68

5.2.1 Implications of the Study 68

5.2.2 Recommendations 69

5.3 Limitations of the Study 70

5.4 Conclusion 71

REFERENCES 72

APPENDICES 80

LIST OF TABLES

Table 2.1: Summary of Related Studies 33

Table 3.1: Distribution of Fingerprint Images by Class 36

Table 4.1: KMCG Performance under Varying Window and Codebook Sizes 48

Table 4.2: Performance of KMCG-Based Feature Extraction with Different Classifiers 51

Table 4.3: Performance of PCA-Based Feature Extraction with Different Classifiers 53

Table 4.4: Performance of Deep Learning Models 55

Table 4.5: Comparative Analysis Across Methods 58

LIST OF FIGURES

Figure 3.1 Research Design 35

Figure 3.2 KMCG Implementation 39

Figure 3.3 PCA Implementation 41

Figure 3.4 Deep Learning Models Implementation 44

Figure 4.1: KMCG Performance under Varying Window and Codebook Sizes 50

Figure 4.2: Confusion Matrix of SVM Classifier Using KMCG Features 52

Figure 4.3: Confusion Matrix of KNN Classifier Using KMCG Features 52

Figure 4.4: Confusion Matrix of Random Forest Classifier Using KMCG Features 52

Figure 4.5: Confusion Matrix of XGBoost Classifier Using KMCG Features..... 52

Figure 4.6: Confusion Matrix for PCA + SVM 54

Figure 4.7: Confusion Matrix for PCA + KNN 54

Figure 4.8: Confusion Matrix for PCA + Random Forest 54

Figure 4.9: Confusion Matrix for PCA + XGBoost..... 54

Figure 4.10: MobileNetV2 Confusion Matrix 57

Figure 4.11: InceptionV3 Confusion Matrix..... 57

Figure 4.12: DenseNet201 Confusion Matrix..... 57

Figure 4. 13: Comparison of Model Performance 59

LIST OF APPENDISES

Appendix I: Publications 80

Appendix II: Loading Dataset 81

Appendix III: KMCG Implementation 83

Appendix IV: SVM + KMCG 85

Appendix V: KNN + KMCG 87

Appendix VI: Random Forest + KMCG 89

Appendix VII: XGBoost + KMCG 91

Appendix VIII: MobileNetV2 93

Appendix X: InceptionV3 96

Appendix XI: DenseNet 201 99

ABREVIATIONS AND AGRONYMES

AI	Artificial Intelligence
AFIS	Automated Fingerprint Identification System
CNN	Convolutional Neural Network
DL	Deep Learning
KMCG	Kekre's Median Codebook Generation
KNN	k-Nearest Neighbors
LBP	Local Binary Patterns
ML	Machine Learning
PCA	Principal Component Analysis
RF	Random Forest
SVM	Support Vector Machine
XGBoost	Extreme Gradient Boosting
USD	United States Dollar
CAGR	Compound Annual Growth Rate

ABSTRACT

Fingerprint classification is a key task in biometric recognition because it supports faster identification, verification, and retrieval in automated fingerprint systems. However, accurate classification remains difficult when fingerprint images contain similar ridge patterns, noise, partial impressions, or variations caused by acquisition conditions. Traditional machine learning methods are efficient and interpretable, but their performance depends on the quality of handcrafted features. Deep learning models often achieve higher accuracy, but they require greater computational resources and provide limited transparency. This study evaluated fingerprint classification using Kekre's Median Codebook Generation (KMCG) under varying window and codebook sizes and compared its performance with Principal Component Analysis (PCA) and deep learning approaches. The main objective was to determine the effectiveness of KMCG-based feature extraction and compare its performance with PCA-based machine learning and deep learning models. Fingerprint images from the NIST Special Database 302 were used. KMCG extracted texture descriptors under varying window and codebook sizes, while PCA served as a dimensionality-reduction baseline. Support Vector Machine, K-Nearest Neighbours, Random Forest, and XGBoost classifiers were trained using the extracted features. Three convolutional neural network architectures were also implemented to represent different deep-learning capabilities: MobileNetV2 was selected as a lightweight and computationally efficient model, InceptionV3 was used to evaluate multiscale feature extraction, and DenseNet201 was included to assess whether dense feature reuse and greater network depth improved fingerprint classification. Performance was evaluated using accuracy, precision, recall, F1-score, average score, confusion matrices, and execution time. MobileNetV2 achieved the best overall performance, with an accuracy of 91.7% and an average score of 94.9% across the evaluation metrics. DenseNet201 achieved 90.0% accuracy, while InceptionV3 recorded 83.4%. Among the traditional machine learning approaches, PCA-XGBoost obtained the highest accuracy of 73.8%, followed by KMCG-SVM at 70.2%. These findings demonstrate that deep learning models provide greater classification accuracy, although KMCG-based approaches remain useful where computational efficiency, compact feature representation, and interpretability are important. The study therefore provides a comparative framework for selecting fingerprint-classification approaches according to performance requirements and available computing resources.

CHAPTER ONE

INTRODUCTION

1.1 Background of the Study

Fingerprint recognition is a mature biometric authentication system. This is due to the unique, permanent nature of fingerprints, which are easy to obtain but hard to duplicate. Its low cost and reliable identity verification make it popular in security, border control, law enforcement, mobile authentication, and access control (Sarkar and Singh, 2020). The rise in demand for fast biometric systems is projected to drive the global biometrics market, which is expected to exceed USD 27.1 billion by 2020. Fingerprint classification is a useful technique that narrows down the search space before matching.

The performance of fingerprint classification depends heavily on the features extracted. Features such as ridge endings, bifurcations, minutiae points, ridge flow, and texture patterns are used in traditional methods. They can be difficult to identify in messy, dirty, twisted, or partial fingerprints. They can also be tedious and prone to errors when dealing with large datasets (Adadi, 2021). In digital fingerprints, automated feature extraction can yield more uniform images, reduce human involvement, and capture more complex ridge structures (Li et al., 2022). It is hard to obtain discriminative features for various acquisition scenarios.

Kekres Median Codebook Generation (KMCG) is a feature-extraction technique based on vector quantization of image features, using compact feature descriptors derived from a codebook of image regions. KMCG divides an image into small windows and uses a few values from a codebook to compress the image pixel patterns in each window for fingerprint classification. It is relevant because it could preserve a region's texture while maintaining small feature sizes, which would be beneficial in limited biometric systems. The window size is significant because a small window captures the details of the ridges, and a large window captures the texture of the context. The size of the codebook is important as it dictates feature discrimination and compression. They have a direct impact on classification accuracy, computational costs, and robustness.

Although less computationally intensive and more interpretable, conventional machine learning classifiers such as SVM, KNN, Random Forests, and XGBoost can be used to classify fingerprints based on KMCG-extracted features and are considered useful classifiers (Linardatos, Papastefanopoulos, and Kotsiantis, 2020). Convolutional neural networks (CNNs) can, however, extract hierarchical features from raw fingerprint images and have yielded accuracies above 90% (MobileNetV2 and DenseNet201), outperforming traditional designs (Meiramkhanov and Tleubayeva, 2024). Deep learning techniques require more labelled data and more computing power. Thus, a thorough analysis of the performance of machine learning and deep learning techniques, including feature representation via k-means clustering and the impact of window sizes and codebooks, is necessary. This comparison will enable us to decide which is best in terms of accuracy, efficiency, and suitability for practical real-life fingerprint classification applications.

1.2 Problem Statement

Despite advances in biometric recognition, previous studies show that fingerprint classification remains difficult when systems process noisy, partial, rotated, smudged, or otherwise poor-quality images. Variations in image quality, sensor conditions, pressure, and incomplete ridge impressions can weaken feature extraction and reduce classification accuracy (Sumalatha et al., 2024). Traditional machine learning classifiers such as SVM, KNN, Random Forest, and XGBoost depend heavily on the quality of extracted ridge, texture, and minutiae descriptors. When these descriptors capture only dominant ridge patterns and fail to represent subtle local and global differences, structurally similar classes, particularly arches and tented arches or left and right loops, may be incorrectly classified (Dey et al., 2019). Although KMCG provides a compact approach for representing fingerprint texture, previous research has not adequately examined how variations in window and codebook sizes affect feature discrimination, classification accuracy, and computational efficiency. Consequently, there is limited empirical evidence regarding the KMCG configuration that offers the most effective balance between accuracy and computational cost. This study addresses that problem by systematically evaluating KMCG under varying

window and codebook sizes and comparing its performance with PCA-based machine learning and deep learning models.

Deep learning models, particularly CNN-based architectures, achieve high classification accuracy when learning hierarchical features directly from image data, with the ability to classify fingerprints with greater than 90% accuracy (Alzubaidi et al., 2021). While they generally demand large amounts of annotated data, substantial memory, GPU acceleration for training, longer training times, and fine-tuning, they are difficult to use in small, mobile, or resource-limited biometric applications. They are also used as black-box models that restrict interpretation, particularly for high sensitivity applications, e.g., forensics, border control and security verification. They are also used as black-box models that limit interpretation, particularly in high-sensitivity applications, e.g., forensics, border control, and security verification. It is therefore not only a matter of high accuracy, but also of the approach that is most suitable in terms of accuracy, computational effort, interpretability, and practical application. The main contribution of this work is to investigate fingerprint classification using KMCG with different window sizes and codebook sizes, and to compare its performance with PCA-based machine learning and deep learning algorithms. This is done by comparing the most effective configuration of the KMCG, the traditional classifiers with the KMCG features, the deep learning models and the best approach for practical purposes of fingerprint classification, which directly supports the study objectives.

1.3 Rationale

The rationale for this study is that fingerprint classification methods should achieve high accuracy while remaining computationally efficient and sufficiently interpretable. Although fingerprint-based biometric systems are widely used, classification performance can decline when images are noisy, partial, or contain closely related ridge patterns. Deep learning models have demonstrated strong performance in image-based recognition because they can automatically learn hierarchical features from raw data (Alzubaidi et al., 2021). However, these models often require large, labelled datasets, substantial processing power, longer training periods, and specialised hardware. Their complex decision-making processes may also be difficult to interpret,

which can limit their suitability for forensic investigations, mobile devices, and other environments with restricted computational resources. It was therefore necessary to investigate whether lighter feature-extraction techniques combined with traditional machine learning models could provide an effective balance between classification accuracy, computational efficiency, and interpretability.

The KMCG method was chosen because it is a vector quantization-based approach that produces compact image descriptors, thereby avoiding noise caused by pixel-level variation (Kekre et al., 2022). Its use was particularly relevant because the window size and the codebook size may affect feature granularity, discrimination, and computational cost. PCA was chosen as a comparison dimensionality reduction method because it aims to reduce data dimensionality while preserving significant variance (Jia et al., 2022). To capture state-of-the-art automated feature learning, deep learning models were added. Comparing KMCG, PCA, and deep learning then provided a clear basis for identifying the most practical fingerprint classification approach for different deployment conditions.

1.4 General and Specific Objectives

1.4.1 General Objective

The general objective of this study is to examine how varying KMCG window and codebook sizes affect fingerprint classification and to compare KMCG-based traditional machine learning models with PCA-based and deep learning models in terms of accuracy, precision, recall, F1-score, confusion matrices, computational efficiency, interpretability, and practical suitability deployment contexts.

1.4.2 Specific Objectives

To achieve the general objective, the study is guided by the following specific objectives:

1. To determine the effect of varying KMCG window and codebook sizes on fingerprint classification performance.
2. To evaluate the performance of traditional machine learning algorithms using KMCG-generated fingerprint descriptors.

3. To examine the performance of PCA-based feature extraction using traditional machine learning classifiers.
4. To compare KMCG-based traditional machine learning models, PCA-based models, and deep learning models using accuracy, precision, recall, F1-score, confusion matrices, and computational efficiency.

1.5 Research Questions

This study seeks to answer the following research questions:

1. How do varying KMCG window and codebook sizes affect fingerprint classification performance?
2. How effective are traditional machine learning algorithms in classifying fingerprints using KMCG-generated fingerprint descriptors?
3. How does PCA-based feature extraction perform in fingerprint classification when used with traditional machine learning classifiers?
4. How do KMCG-based traditional machine learning models, PCA-based models, and deep learning models compare in terms of accuracy, precision, recall, F1-score, confusion matrices, and computational efficiency?

1.6 Structure of the Thesis

This dissertation is organized into five chapters, each serving a distinct purpose in achieving the research objectives:

Chapter 1: Introduction

This chapter introduces the study by presenting the background, defining the research problem, outlining the aim and objectives, and setting the scope. It also highlights the significance of the research and provides an overview of the entire dissertation structure.

Chapter 2: Literature Review

This chapter reviews existing research on fingerprint recognition systems. It discusses traditional machine learning approaches, deep learning architectures, and feature

extraction techniques, with particular emphasis on KMCG. The chapter identifies research gaps that this study aims to address.

Chapter 3: Methodology

This chapter outlines the research design, dataset selection, preprocessing methods, model implementation, and evaluation metrics. It describes how both classic ML and deep learning, were trained, validated and tested, using KMCG and transfer learning techniques.

Chapter 4: Results and Discussion

The chapter provides empirical findings of all models and compares the models regarding accuracy, precision, recall, and F1-score. It has visualizations and verbatim documentation of results to the taking of quality and the inadequacy of any of these methods.

Chapter 5: Conclusion and Future Research

The last chapter presents an overview of the chief conclusions, includes reflection on the study's contributions, and addresses the study's limitations. It also indicates possible courses for future study into fingerprint classification and biometric system optimization.

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

This chapter reviews the literature on fingerprint classification using KMCG, PCA, traditional machine learning, and deep learning models. It starts with biometric systems and fingerprint recognition, then focuses on fingerprint pattern classes and early classification techniques. It then investigates the feature extraction, specifically KMCG, window size, codebook size and PCA as a competitor dimensionality reduction technique. In addition, the chapter discusses the following models: SVM, KNN, Random Forest, XGBoost, MobileNetV2, InceptionV3, and DenseNet201. Finally, it discusses comparative studies, identifies gaps in the research, and explains the significance of these gaps in justifying the current study.

2.2 Biometric Systems and Fingerprint Recognition

Biometric systems recognize or confirm a person's identity based on their physiological or behavioural attributes, such as fingerprints, faces, irises, voices, or gaits. Of these, fingerprint recognition is one of the most popular techniques since fingerprints are unique, permanent, relatively easy to capture, and can be used for both forensic and civilian purposes (Abdulrahman and Alhayani, 2023; Sumalatha et al., 2024). The basic steps involved in fingerprint recognition are image acquisition, image preprocessing, feature extraction, classification, and matching. Such stages are highly dependent on image quality, particularly when fingerprints are noisy, partial, rotated, smudged or distorted (Sumalatha et al., 2024). Thus, feature extraction and classification are key to improving fingerprint recognition accuracy.

Fingerprint classification is used to help fingerprint recognition by clustering prints into known pattern classes to facilitate matching. This decreases the search space and retrieval efficiency in the Automatic Fingerprint Identification System. Traditional classifiers rely on features extracted from fingerprint images, whereas deep learning models learn features directly from them without human intervention (Adnan et al., 2024; Khan et al., 2020). This means that for comparing handcrafted feature extraction

methods like KMCG, the dimensionality reduction methods like PCA and deep learning models, the field of fingerprint recognition is applicable.

2.3 Fingerprint Pattern Classes

Pattern classes are based on fingerprint structures. The most popular are Arch, Tented Arch, Left Loop, Right Loop, and Whorl. These classes are characterized by ridge flow, core position, delta points, and/or the overall fingerprint pattern shape (Dey et al., 2019). Arch patterns are distinguished from other arches by ridges which curve from side to side, but not from back to front, and also by the more pronounced peak in the centre of the tented arches. The ridges in loop patterns enter and exit from the same side, and there is one core and one delta. The pattern of the deltas is more spiral or circle-shaped, and two or more deltas may be present.
These classes are crucial because they share a structural similarity that affects classification performance. For example, arch and tented arch patterns may not be distinguished, and if orientation is unknown, left- and right-hand loops may be confused. Such a class overlap suggests the need to use features extracted from local textural properties of ridges as well as from larger pattern structure. Figure 2.1 shows the various fingerprint classes considered for this study:

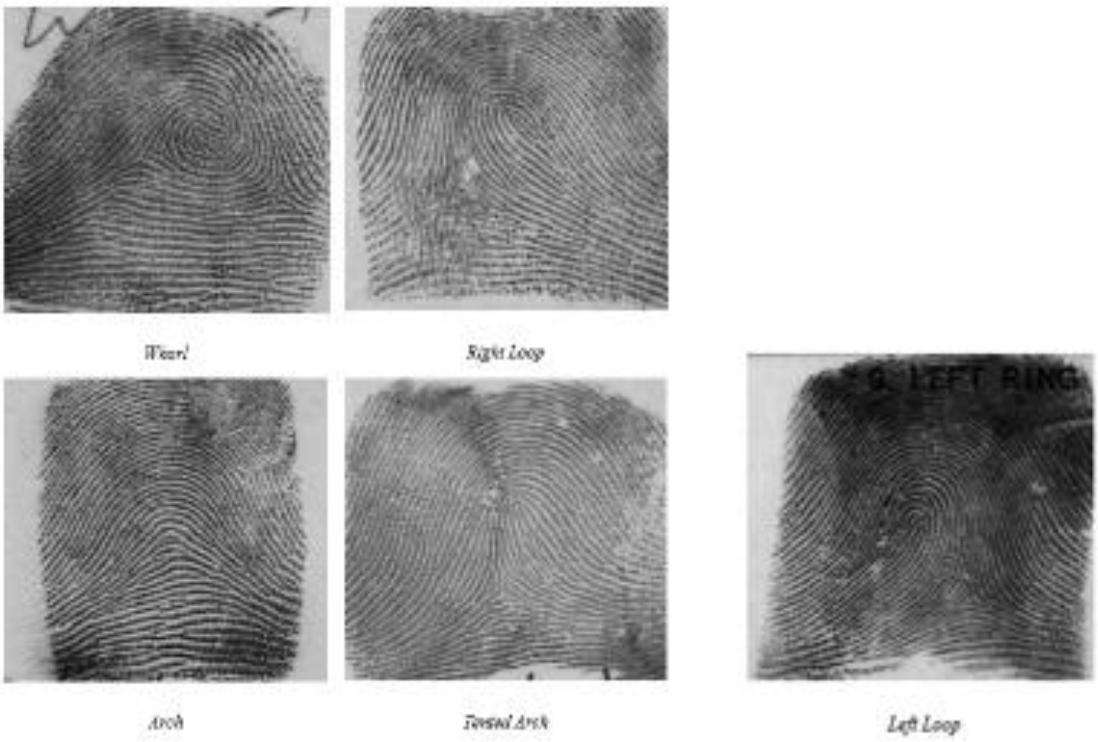

Figure 2.1: Fingerprint Classes

Source: Monpara et al. (2022)

Fingerprint patterns can be understood better by looking at the small details found along the ridges. These include the core, delta, ridge ending, bifurcation, island, pore, and crossover (See Figure 2.2). The core is the central part of the fingerprint pattern, while the delta is the point where ridges spread in different directions. A ridge ending shows where a ridge stops, and a bifurcation shows where one ridge separates into two. Islands and dots are very short ridges, while pores and small spaces add more detail to the fingerprint. These features help to tell the difference between arches, loops, and whorls. They are also useful in feature extraction and support accurate fingerprint classification in automated systems.

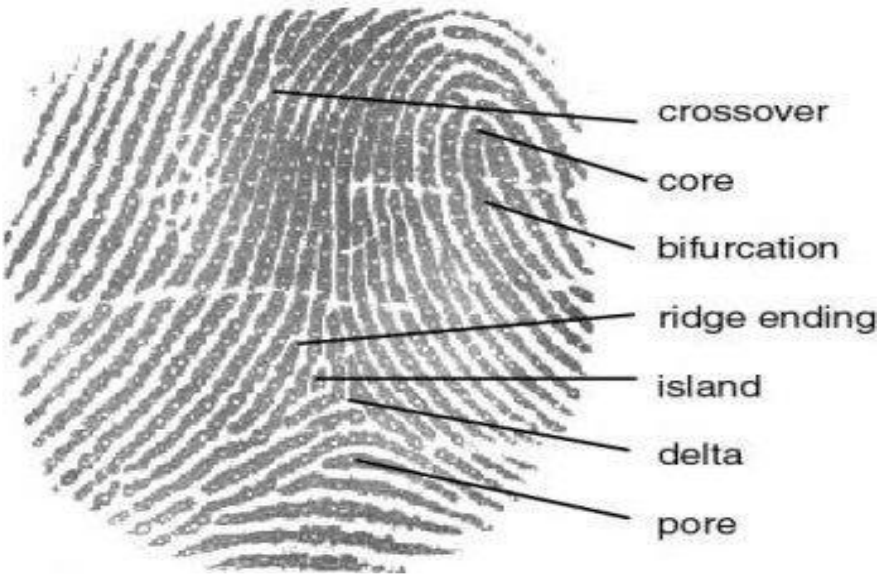

Figure 2.2: Fingerprint patterns

(Source: Verma et al., 2012)

2.4 Early Fingerprint Classification and Techniques

Initial classification of fingerprints was done manually and based mainly on their visible ridge patterns. The Henry Classification System was one of the most significant early and influential, categorizing fingerprints into general types such as loops, whorls, and arches (Ghosh and Pahari, 2021; Hawthorne, Plotkin, and Douglas, 2021). This provided the basis for organising fingerprint records and reduced the number of records to review in large collections. Manual classification was closely related to the expert judgment and less efficient for large automated systems. The processes of rolling prints to get a fingerprint also changed from ink to digital scanning. Digital acquisition had some benefits, such as storage, retrieval, and automated image processing, but also encountered difficulties due to image quality, sensor variability, partial impressions, and latent prints. In particular, latent fingerprints are incomplete and often distorted, making classification more difficult (Sankaran, Vatsa, and Singh, 2014). This limitation is the reason for switching to computation-based classifications such as feature extraction, machine learning and deep learning. This is the direction of the current study, which aims to assess whether the descriptors developed under

KMCG can be used to reliably classify the data without being too computationally expensive.

2.5 Feature Extraction in Fingerprint Classification

A key step in fingerprint classification is the extraction of features to be passed to the classifier, and the information they contain. Ridge endings and bifurcations are common features employed in traditional systems. These features are effective for high-quality fingerprint images but may not work when fingerprints are noisy, smudged, partial, or poorly captured (Socheat and Wang, 2020). Ridge-oriented methods, ridge-frequency methods, texture-descriptor methods, and hybrid methods are other approaches that represent fingerprint structure in a more general way (Rangnekar, 2022; Safavipour, Doostari, and Sadjedi, 2022). Thus, accurate feature extraction is essential to boost classification performance over other fingerprint classes.

Local Binary Patterns are texture-based methods that are effective at representing local intensity variations and capturing ridge patterns; however, they are not always discriminatory when used in isolation (Ranjan et al., 2023). The other approach is a codebook-based method that encodes local fingerprint image windows with compact descriptors, such as KMCG. Thus, KMCG is beneficial in scenarios where interpretability, reduced feature size, and computational efficiency are significant (Shrivastava and Gajjar, 2023). The importance of KMCG in this study is that the window size and codebook can affect the ridge texture captured and the separation of fingerprint classes.

Principal Component Analysis (PCA) is used in fingerprint classification as a feature-extraction and dimensionality-reduction technique. It transforms high-dimensional fingerprint image data into a smaller set of principal components that preserve most of the important variation in the original features (Gewers et al., 2021). By reducing redundancy and feature dimensionality, PCA can lower computational demands and improve the efficiency of traditional machine learning classifiers (Jia et al., 2022). However, PCA is basically a linear technique and fails to account for the non-linear ridge structures and subtle differences in the classes of fingerprint images.

In this study, PCA is used as a baseline for comparison with KMCG-based feature extraction methods and deep learning models. It is not used to supersede KMCG but rather to serve as a generic dimensionality reduction technique to evaluate if KMCG yields more discriminative fingerprint descriptors. Deep learning differs from PCA and KMCG because it extracts hierarchical features directly from raw fingerprint images rather than designing feature extraction methods (Khan et al., 2020). Hence, the basis for comparing PCA, KMCG, and Deep learning approaches in terms of accuracy, efficiency, and suitability for fingerprint classification is laid down by Feature extraction. Therefore, feature extraction is the foundation for comparing PCA, KMCG, and deep learning approaches in terms of accuracy, efficiency, and suitability for fingerprint classification.

2.6 Kekre's Median Codebook Generation (KMCG)

2.6.1 Concept and Theoretical Basis of KMCG

Kekre's Median Codebook Generation (KMCG) is a feature-extraction method for high-dimensional image data that is compactly represented via vector quantization. The basic idea behind VQ is to cluster together similar image vectors and code each cluster by storing a codeword in a codebook. It is found to be useful in image processing applications such as compression, segmentation, pattern recognition and biometric feature extraction (Kekre et al., 2022). In fingerprint classification, KMCG is relevant because it addresses the local ridge-and-valley structure of fingerprint images, which can be represented using compact local descriptors rather than the entire fingerprint image.

KMCG works in two phases: first, it divides an image into fixed-size blocks or windows; then, it converts these blocks into feature vectors and creates representative codebook entries using a median-based method. KMCG differs from feature extraction techniques that rely exclusively on manually identified minutiae points by capturing the local texture distribution from the image's intensity patterns. This is helpful in cases where it is difficult to detect ridge endings, bifurcations, or singular points due to image noise, partial impressions, or sensor-related distortions (Socheat and Wang, 2020; Mohamed Abdul Cader, Banks and Chandran, 2023). Thus, KMCG offers a

practical approach to numerically represent the fingerprint images for use in machine learning classifiers.

2.6.2 KMCG and Traditional Vector Quantization Techniques

Most conventional VQ-based techniques, such as the Linde-Buzo-Gray (LBG) algorithm, generate codebooks by iteratively dividing vectors into groups and calculating representative centroids until convergence (Ravikiran et al., 2023). The codebooks created using these methods can be helpful but require multiple passes and are subject to the starting point. This increases the computation costs, especially for large sets of images. These iterative VQ methods differ from KMCG, which uses the median method to generate the codebook to prevent repeated centroid recalculation (Kekre et al., 2022). This can be especially attractive in applications that need to output features quickly and efficiently.

The use of median values also provides a pragmatic benefit to KMCG when representing noisy images. The median is less affected by extreme pixel values than the mean, and it is helpful when there is isolated noise or a local intensity change. Some of the possible reasons for variations in fingerprint images include dryness, pressure changes, smudging, partial contact, and limitations of the sensor (Sumalatha et al., 2024; Mohamed Abdul Cader, Banks, and Chandran, 2023). Therefore, to minimize the effect of abnormal local pixel values and maintain the main texture structure of fingerprint blocks, the KMCG is used to extract texture features. But this does not mean that KMCG can solve all noise issues. This still requires correct pre-processing, a window size, a code-book size, and a classifier following the pre-processing step.

KMCG is less flexible than traditional vector quantization in modelling extremely complex image variations, but simpler and more direct. In some cases, iterative and adaptive codebook methods can further enhance the codebook's discriminative ability by continually updating the clusters (Zheng and Vedaldi, 2023). This is good; however, this is usually more complex to compute. KMCG, therefore strikes a balance between the simplicity of computation and the discrimination of features. The compromise is important in the context of fingerprint classification applications,

including high-resource systems such as a central forensic database and low-resource systems such as embedded or mobile authentication devices.

2.6.3 Relevance of KMCG to Fingerprint Classification

Local ridge texture and more global pattern structure are required in feature descriptors for fingerprint classification. Classification can be difficult because some classes may overlap, especially when weak or absent descriptors are present (Dey et al., 2019). The relevance of KMCG is that it stores the fingerprint image locally on the Windows system, and the classifier has to learn the pattern across various parts of the image. This can help preserve spatial texture information that may be necessary to distinguish between similar fingerprint classes.

One more benefit of using KMCG is that it doesn't rely solely on minutiae extraction. Minutia-based systems, however, are useful for high-quality fingerprint images but not for low-quality images with noise, smudges, or partial images (Socheat and Wang, 2020). KMCG can generate descriptors even when ridge endings or bifurcations are not readily apparent in the image, using local image blocks. This makes it suitable as an alternative/complementary feature-extraction technique for fingerprint classification. Similar codebook-based and handcrafted descriptor approaches have been adopted in image recognition, enabling the compact description of local visual structures, which is well-suited to efficient traditional machine learning classifiers (Siddiqui et al., 2018).

Using classifiers such as SVM, KNN, RF, or XGBoost, it is easy to compare the performance of handcrafted, PCA-based, and deep learning texture descriptors. This is important because there are systems where traditional machine learning models remain viable because they should be interpretable, easily trained, and cheap to compute (Linardatos, Papastefanopoulos, and Kotsiantis, 2020). The quality of the KMCG descriptors, however, is strongly dependent on the parameters used (especially codebook and window sizes). Thus, these two parameters should be carefully investigated.

2.6.4 Strengths and Limitations of KMCG

The biggest advantage of KMCG is that it generates compact feature representations out of image data. This can reduce the dimensionality of these fingerprint images prior to classification, thereby lowering memory usage and training time. In addition, it has a feature-extraction procedure that is more comprehensible than deep learning, since the descriptors are extracted from image blocks defined in the codebook. For applications in forensic and security, the basis of classification may have to be explained (Linardatos, Papastefanopoulos, and Kotsiantis, 2020).

One benefit is that, because KMCG is represented in a median sense, it will be less sensitive to individual noisy pixels. It can apply to fingerprint images, as pressure, skin condition, sensor type, rotation, or partial fingerprint impressions can all impact image quality (Sumalatha et al., 2024; Mohamed Abdul Cader, Banks, and Chandran, 2023). KMCG has its disadvantages, however, important information about local ridges is smoothed or lost when the window size is large. A small window may result in excessive local noise. Similarly, a small codebook might not be able to distinguish between classes, while a large one may overfit the data and be computationally costly. The limitations motivate studying the KMCG with different window sizes and codebook sizes.

2.6.5 Effect of Window and Codebook Size Variation in KMCG on Fingerprint Classification

Window Size and Local Feature Granularity:

The window in fingerprint classification refers to the entire size of the fingerprint image. In this research we used fingerprints of 512x512 pixels with 32 rows of white space at the bottom of the print as shown in Figure 2.3.

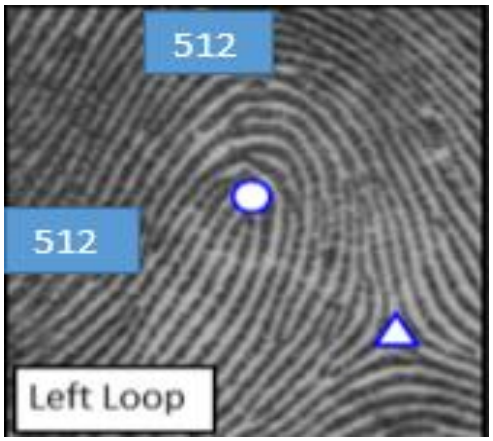

Figure 2.3: Window

The window size is the spatial block size before fingerprint image division for feature extraction. It is the portion of the image cropped from entire size of the fingerprint image (window). Figure 2.4 shows an example of an 8*8 window size cropped from the left Loop fingerprint.

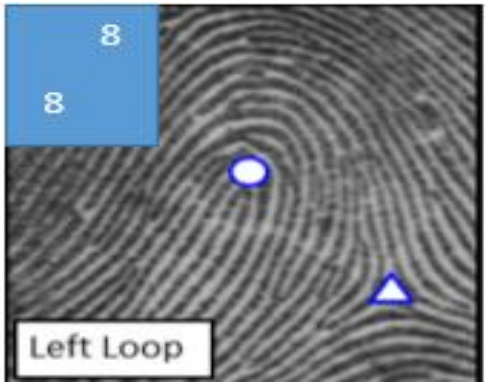

Figure 2.4: Window Size

The window size defines the amount of information that comes from the local image in each descriptor. This is significant in fingerprint classification, as ridge orientation, ridge spacing, texture, cores, deltas, and local curvature all help separate classes. Fine-grained ridge details and local variations can be captured in a small window, which may be beneficial in discriminating between similar classes. However, an extremely small window size can lead to sensitivity to noise, broken ridges, and to image acquisition conditions (Sumalatha et al., 2024; Mukoya, 2025).

The bigger the window, the more of the image texture and context is seen. This could assist in recognizing global ridge flow, which is helpful for classifying loops and whorls. However, large windows can smooth or summarize larger areas. That is, the difference between Arch and Tented Arch, or between Left Loop and Right Loop, may not be as apparent to the classifier. It sets a good compromise between the close-up and the bigger picture. Thus, the size of windows must be an empirical matter, not an assumption.

The number and type of feature vectors generated are directly related to the window size in KMCG-based fingerprint classification. Smaller windows tend to produce more local descriptors and/or increase computation. Larger windows increase the number of descriptors but may yield a less detailed set of features. This should be the optimum window size as a function of image resolution, fingerprint quality, the similarity of the class and the used classifier after feature extraction. The objective of this study is to explore different window sizes and determine the optimal window size that balances feature richness and computation time.

Codebook Size and Descriptor Discrimination:

The size of the codebook is the number of representative codewords used to encode the image block patterns. The higher the number of codewords in the codebook, the more variations can be captured in the feature space; the lower the number, the more compact the feature space representation. The size of a codebook influences the ability to distinguish local ridge and texture patterns in fingerprint classification. Though it may be efficient (computationally) to have a small codebook, it may be possible to use different ridges for the same representation, which would result in reduced discrimination. A larger codebook can capture more fine-grained differences in the pattern, but will take up more memory, require more time to execute, and overfit more (Yang et al., 2026).

The design of codebooks influences recognition results, as evidenced by work on visual codebooks (Tang et al., 2024), which shows that changes in codebook design can alter clustering and the representation of local image features, leading to changes in recognition results. Borrowing from the field of efficient design of codebooks, Della

Libera et al. (2026) also call for compact codebooks that would increase efficiency while retaining enough discriminative information. The same can be applied to fingerprint classification as well, because the codebook should be able to assign fingerprint ridge patterns to different classes, separating those with similar patterns. If the codebook is not large enough, it may be possible to miss out important variations of the fingerprint. If it's too large, then the model could be very expensive to run, but it may not be important.

Therefore, the size of the codebook should be carefully determined for KMCG. Stating the number of codewords is no longer the only objective; a level of representation needs to be found. The ideal codebook size is one that preserves meaningful fingerprint texture differences while keeping computation time low. It is especially crucial for systems with limited resources, such as memory, speed, and interpretation, where an accurate biometric system is a real challenge.

Joint Effect of Window Size and Codebook Size:

The window and codebook sizes should not be treated independently, as they interact during feature extraction. The size of the window determines the spatial scale of the information that is captured; the size of the codebook determines how the information is represented. A small window with detailed descriptors, but it might also lead to high dimensionality and high sensitivity to noise. A large window with a small codebook could be efficient, but it could oversimplify the fingerprint image, thus decreasing classification accuracy (Zhu et al., 2025).

This interaction forms a two-dimensional parameter space which needs to be optimized. The optimal configuration of fingerprint classification may depend on both image characteristics and the downstream classifier. For instance, SVM and XGBoost might benefit from a compact yet discriminative set of descriptors, whereas KNN may be affected by noise or high-dimensional feature spaces, as it relies on distance calculations (Halder et al., 2024). Random Forest can cope with some degree of feature variations but can be impacted when the extracted descriptors do not make the classes well separated. Hence, it is important to systematically tune the window and codebook sizes to find the optimal sizes for both.

The main purpose of this study is achieved through joint optimization of window and codebook sizes. The subject and goals of the title do not represent minor technical specifications; there are published works that provide the technical details. These factors directly affect classification performance, computational cost, and the model's suitability. This makes their empirical evaluation crucial for determining whether KMCG can serve as a practical alternative to the PCA-based approach and deep learning methods.

2.6.6 Implications for the Current Study

The literature reviewed shows that KMCG is a lightweight, interpretable feature extraction method, but whether it works well compared to other feature extraction methods has not yet been assessed. It is observed from the traditional VQ literature that the design of the codebook is of great importance, and from the fingerprint recognition literature, it is noted that extracting stable features from noisy, variable fingerprint images is difficult (Sumalatha et al., 2024; Mohamed Abdul Cader, Banks, and Chandran, 2023). Therefore, the purpose of this work is to analyse fingerprint classification performance across various window and codebook sizes and to examine their effects on classification results using the KMCG. This focus provides a firmer foundation for the study, as KMCG can be directly related to the research problem, objectives, and methodology. It also justifies a comparison with the PCA and deep learning. PCA is a statistically based descriptor, KMCG is a codebook-based descriptor, and deep learning is a feature-learning benchmark. These approaches can be evaluated and contrasted with the question of whether KMCG can offer a fair balance between accuracy, efficiency, and interpretability in fingerprint classification.

2.7 Traditional Machine Learning Models for Fingerprint Classification

Although deep learning is widely used, the traditional machine learning models remain important in fingerprint classification, and provide interpretable solutions that are efficient, hard to replicate with deep learning. Conventional classifiers do not learn features directly from raw fingerprint images, as do convolutional neural networks. They perform differently depending on the quality of the extracted descriptors, including ridge orientation, minutiae, texture features, PCA features, or features from

a codebook generated using KMCG (Militello et al., 2021). This allows them to be used to assess the feasibility of obtaining discriminative representations of fingerprints via handcrafted feature extraction, such as KMCG. In this study, SVM, KNN, Random Forest, and XGBoost are chosen because they use different learning principles: margin-based classification, distance-based classification, bagging-based ensemble learning, and boosting-based ensemble learning. Including them is to have a balanced comparison of the performance of KMCG and PCA features as classifiers.

2.7.1 Support Vector Machine

Support Vector Machine is a supervised learning algorithm that classifies the classes by finding the optimal hyperplane (that maximizes the margin between data points). SVM is employed in fingerprint classification because the descriptors used can lead to a very high-dimensional feature space, particularly when features are extracted from local ridge patterns, texture descriptors, or codebook-based methods like KMCG (Dushku, 2024). SVM can also be used to model the non-linear relationship between fingerprint classes using kernel functions, such as radial basis functions. This is important because the separate classes of fingerprint patterns (Arch, Tented Arch, Left Loop, Right Loop, Whorl) might not be linearly separable.

The SVM is used as the base-line classifier for hand-crafted feature descriptor, because of its high performance. It is beneficial as it generalizes well in a small and discriminative feature space. This facilitates the evaluation of the KMCG window and codebook configurations for generating useful fingerprint descriptors. But, SVM performance is highly sensitive to parameter tuning, kernel choice, and feature scaling (Ali et al., 2023). It can also be very expensive computationally when the data is large or the feature vectors are high-dimensional. So SVM serves as an evaluation tool for accuracy and generalization; however, it must be used alongside considerations of computational cost.

2.7.2 k-Nearest Neighbours

K-Nearest Neighbours is an instance-based classifier that assigns an unknown sample to the majority class of its k nearest neighbours in the feature space. For fingerprint classification, the KNN algorithm uses the set of features extracted from a fingerprint

image and the classes and assigns an unlabelled fingerprint to the class whose features best match its feature vector (Chen et al., 2024). That's why KNN is easy to implement: it doesn't require a complex training process. It is also helpful to examine whether the extracted features create distinct class clusters as a baseline to assess whether there is any difference.

KNN is chosen in this study because it facilitates testing the separability of KMCG and PCA-generated features. Because the features are quite discriminative, fingerprints of the same class should be near in the feature space. But KNN has certain limitations. It is sensitive to the value of k, choice of distance functions, feature scaling, noise, and irrelevant/redundant features. It can also be slow to make predictions because it compares each test sample with stored training examples. When classes are similar and overlapping, or when descriptors generated by the feature extraction are noisy, these weaknesses are significant in fingerprint classification. Hence, KNN can be used to indicate whether KMCG and PCA features are naturally separable; however, it may not perform well when the feature boundaries are complex.

2.7.3 Random Forest

Random Forest is an ensemble learning method that creates multiple decision trees and makes a final classification decision based on their predictions. Training for each tree is performed on a random subset of data and features to avoid overfitting and improve generalization. Random Forest is suitable for fingerprint classification to model non-linear relationships and handle the diverse feature patterns generated by handcrafted descriptors (Alhijaj and Khudeyer, 2023). It also has a fairly high tolerance for partial feature sets or noise in fingerprint images, which is useful when the images contain partial ridge structures, image distortions caused by capture conditions, or smudges.

The selection of the random forest model in this study is based on its ability to serve as a comparison to both SVM and KNN. Random Forest is based on decision rules from multiple trees, unlike SVM, which is based on margin separation, and KNN, which is based on distances. This allows it to be used to determine whether there are patterns in the KMCG descriptors relevant to decisions about separating classes. It is robust, easy to use, can capture non-linear features, and does not suffer from overfitting

as a single decision tree does. Another issue is that Random Forests can be less interpretable than simpler classifiers, since each new tree adds little information. However, it may not be as discriminative when the extracted features are not sufficiently different between similar fingerprint classes Accordingly, Random Forest was included to determine whether ensemble-based learning could improve classification accuracy when applied to KMCG- and PCA-derived features.

2.7.4 XGBoost (Extreme Gradient Boosting)

XGBoost is a gradient boosting algorithm that sequentially builds decision trees, with each new tree correcting the mistakes of previous trees. It is thus very useful for structured feature-based classification applications. In biometric applications, XGBoost is useful because it can handle complex feature interactions, introduce regularisation to reduce overfitting, and enhance classification performance through iterative error correction (Odya, Górski and Czyżewski, 2023). Features such as KMCG or PCA descriptors can be extracted from these characteristics for fingerprint classification.

The XGBoost model is used in this study because it is a high-performing traditional machine learning model. It provides a good benchmark for assessing the ability of handcrafted KMCG descriptors to match those of more computationally expensive deep learning methods. Its most notable feature is the ability to learn complex decision boundaries, and consequently, its predictive power improves over simpler models. It also comprises regularization, tree pruning, and effective management of missing and weak feature patterns. But XGBoost must be properly fine-tuned using its hyperparameters, such as the learning rate, tree depth, number of estimators, and regularization parameters. Unfortunately, it can overfit or produce erratic results if not properly tuned. In this work, it is shown that KMCG features can be used to improve the classification performance of an XGBoost ensemble and to examine whether the benefits justify the computational resources required.

The four classifiers here are traditional classifiers used to test features from the KMCG and PCA methods under various classification logics. SVM considers separability in terms of margins, KNN in terms of distance, Random Forest in terms of bagging-based

decision learning, and XGBoost in terms of boosting-based prediction refinement. This allows for a better comparison than using a single classifier alone.

2.7.5 Comparative Review

Traditional machine learning algorithms differ in how they classify fingerprint features, which affects their applicability to KMCG- and PCA-based classification. Among the various methods, SVM is widely used in fingerprint studies because it can separate high-dimensional classes and handle complex classes using kernel functions. For example, SVM-based techniques using minutiae and texture features improve fingerprint recognition on the FVC 2000 benchmark data set when hand-crafted descriptors are available (Zhang et al., 2020). However, SVM must be parameterized and tuned to the problem, and it performs poorly when the extracted features are not well separated or noisy (Ali et al., 2023). Therefore, SVM is suitable for this research to evaluate KMCG's ability to produce compact, separable fingerprint descriptors. KNN is easier than SVM because it uses the distance to stored fingerprints in the database to classify new fingerprints. For the Ridge Frequency and Orientation features, KNN performs well on small datasets but doesn't scale well with dataset size on the FVC 2002 dataset, according to Lee et al. (2018). The advantage is that it provides a clear indication of the class clusters formed by the extracted features. It is sensitive to noisy data, overlapping classes, feature scaling, and the choice of k. This makes KNN an appropriate baseline for this study, but it may not be applicable if the fingerprint classes, as in this study, Left Loop and Right Loop, have overlapping ridge patterns.

Random Forest overcomes the disadvantages of using one decision tree classifier and avoids overfitting by fusing multiple decision trees. Random Forest can be applied to fingerprint classification due to its ability to handle non-linear relationships between features and to handle noise in the extracted descriptors (Alhijaj and Khudeyer, 2023). It is generally more stable than KNN across varying feature sets. It’s less sensitive to kernel choice than SVM. Random Forest, however, may require many trees to perform well, and its decision process might not be as straightforward as simpler models. It is thus appropriate to use it to test for decision-relevant patterns across different fingerprint classes in KMCG features.

XGBoost can also be more powerful than traditional tree-based classifiers, as it can be constructed sequentially and correct misclassifications from previously constructed trees. For structured feature vectors, such as those used in biometric or fingerprint classification, XGBoost has been employed for its ability to handle structured vectors and for its regularization, which helps prevent overfitting (Adnan et al., 2024; Odya, Górski and Czyżewski, 2023). While features are informative, XGBoost can outperform Random Forest; it also needs more tuning. It is appropriate for this study because both KMCG and PCA generate structured numerical features, and it is possible to determine whether boosted decision learning can enhance classification accuracy.

From a global perspective, the reviewed studies revealed that there is no one best traditional machine learning algorithm. SVM is appropriate for high-dimensional, well-separated descriptors; KNN can be used to test whether the natural feature space is separable; Random Forest can be used for noisy, non-linear features; XGBoost is effective for feature vectors with strong discriminatory information. But all four models are highly reliant on feature quality. This helps address a research gap identified in the present study: whether the KMCG, with different window and codebook sizes, can produce better fingerprint descriptors than PCA and whether these handcrafted methods can compete with deep learning models.

2.8 Deep Learning in Fingerprint Recognition

Deep learning is widely used in modern fingerprint recognition because it can automatically learn features from image data. CNNs are particularly suitable for fingerprint images, as they can identify local edges, ridge flow, fingerprint curves, texture transitions, and hierarchical spatial patterns at multiple levels (Alazzawi, 2022; Khan et al., 2020). Such a reduction means there is less need for manually designed descriptors, such as minutiae, ridge orientation, PCA components, or KMCG codebook features. Deep learning is thus applicable to fingerprint classification problems, where the patterns may be very complex and hard to describe manually, particularly when noisy, distorted, partial, or structurally similar fingerprint images occur within the same class.

However, there are practical limitations to deep learning as well. CNN-based models are complex and require large, labelled datasets, high memory, GPU support during training, longer processing time, and tuning. Also, they are not as interpretable as traditional machine learning models, which may be a challenge for applications in which biometric information needs to be interpreted in forensic, legal, or high-security contexts (Linardatos et al., 2020; Minaee et al., 2023). This research will then use deep learning to evaluate accuracy and compare it with interpretability and computational cost. The chosen models are all CNN-based, representing different philosophies: lightweight and efficient; multi-scale feature extraction; dense feature reuse; MobileNetV2; InceptionV3; and DenseNet201.

2.8.1 MobileNetV2

MobileNetV2 is an efficient CNN structure that is lightweight for image classification on mobile and embedded devices. It uses separable convolutions that process each input channel independently, together with inverted residual blocks, to reduce computational demands while preserving feature-extraction capability (Gulzar, 2023). Therefore, MobileNetV2 is appropriate for fingerprint recognition systems that require faster performance, less memory, and real-time operation. Moreover, it's lightweight, which could be advantageous in fingerprint classification because it doesn't require as much knowledge as deeper networks.

In this study, the selected network is MobileNetV2, a strong benchmark for resource-constrained biometric applications in deep learning. Its main virtue is that it's accurate and efficient. It can learn hierarchical image features with fewer computations than those required by deeper architectures. This is particularly relevant when comparing deep learning to KMCG-based machine learning, as they both focus on deployment. But MobileNetV2 might have drawbacks when the fingerprint image is highly distorted, or the ridge details are very fine, requiring deeper feature extraction. It may be less representative than heavier networks, due to its lightness.

The MobileNetV2-based fingerprint classification pipeline is shown in Figure 2.5. The figure demonstrates the resized fingerprint images passing through the preprocessing layers and the convolutional feature extractor, resulting in a class probability from the

fully connected classifier. The architecture illustrates why the MobileNetV2 is appropriate for the present study: It enables automatic feature learning without a heavy computational burden.

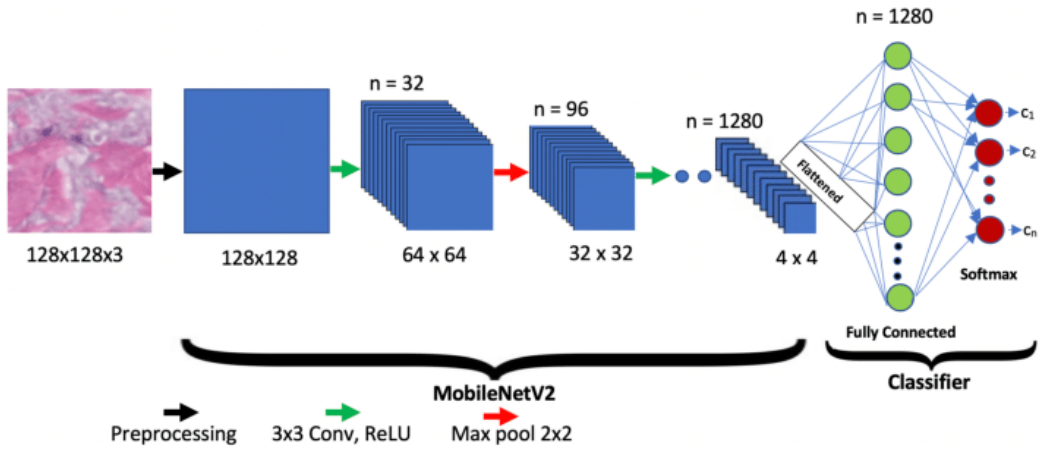

Figure 2.5 MobileNetV2-Based Fingerprint Classification Pipeline
Source: Hussain et al. (2022)

2.8.2 InceptionV3

InceptionV3 is a CNN architecture designed to enhance image recognition by extracting features at multiple scales. It features inception modules that process the image with a set of convolutional filters of varying sizes simultaneously, enabling the network to capture both fine- and wide-level patterns. (Tyagi, Vats, and Vashisht, 2024) This is significant for fingerprint classification because fingerprint images have features at both local and higher levels; loop, arch, and whorl are some of the higher-level features. Thus, a model that can represent features at multiple scales is expected to improve discrimination between structurally similar fingerprint classes. In this study, a deeper, more complex CNN than MobileNetV2, InceptionV3, is selected. The main benefit is that it can learn multiscale spatial features for fingerprint classification, even when ridge flow and textural features vary. InceptionV3 has a more complex input dimension and network blocks, making it more computationally intensive. This can result in longer training time and higher memory usage. It may also be less practical to use for resource-constrained biometric systems. Thus, InceptionV3

can serve as an important benchmark to assess whether the additional classification benefit can be achieved with the added architectural complexity. The architecture for fingerprint feature learning is shown in Figure 2.6, which is called "InceptionV3". The figure shows several inception modules and several grid reduction steps, allowing the model to learn features at different spatial scales. The model is able to distinguish between the small differences in the fingerprint pattern that might require more computational resources in this architectural design.

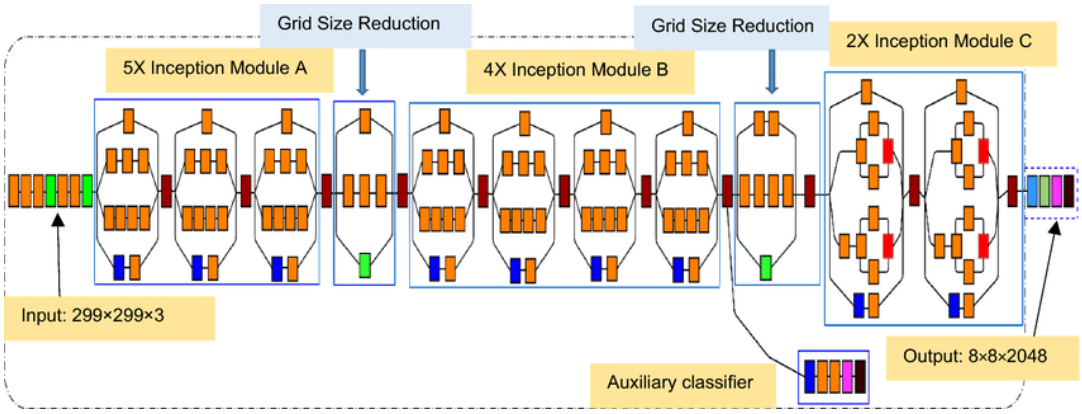

Figure 2.6 InceptionV3 Architecture for Fingerprint Feature Learning
Source: Minarno et al. (2023)

2.8.3 DenseNet201

DenseNet201 is a deep CNN architecture that establishes dense connections between layers. DenseNet models incorporate features from layers above and below to enhance feature reuse and mitigate the vanishing gradient problem (Sanghvi et al., 2023). This structure is useful for fingerprint classification, as low-level and high-level features may be required for accurate classification. Earlier layers might include edges and ridge texture, and deeper layers might include broader class-level structure.

In this study, DenseNet201 is chosen as it is one of the high-capacity deep learning models for complex feature learning. The dense connections enable the propagation of strong features, and it is well-suited for fingerprints with similar ridge patterns or quality. Hence, DenseNet201 can be very effective when simpler models fail to capture sufficient detail. But this power comes at a high price in terms of computing resources.

Since DenseNet201 will need to consume more memory, train for longer and need more hardware support, it will generally be more resource-intensive than MobileNetV2. It could therefore be applied to high-accuracy biometric systems, rather than lightweight or embedded systems.

The DenseNet201 architecture is shown in Figure 2.7. The diagram shows dense connectivity, with layers exchanging feature information at multiple stages before the final classifier. The rich representations of fingerprints by DenseNet201 are attributed to feature reuse. The depth and connectivity of the model are responsible for the increased computational cost at the same time.

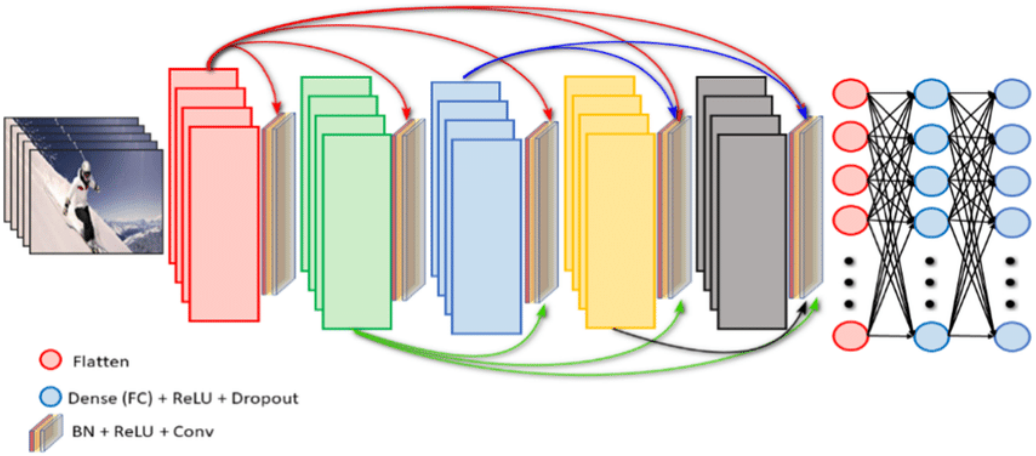

Figure 2.7: DenseNet201 Architecture
Source: Charisma and Adhinata (2023)

2.8.4 Strengths and Limitations of Deep Learning Models

The key advantage of deep learning models is their capability to learn the discriminative features directly from raw fingerprint images. This eliminates the need for manual feature design and enables CNNs to automatically learn ridge edges, curves, and local and global structures (Jain et al., 2024). In complex fingerprint classification problems where handcrafted features fail to capture subtle class differences, deep learning models prove highly beneficial. By improving their practicality, transfer learning enables adaptation of a pre-trained model to fingerprint datasets with only a small number of training images.

While powerful, deep learning models have drawbacks. They require a higher computational cost than PCA- and KMCG-based machine learning models. Moreover, they might need data augmentation, regularization, and strict cross-validation to avoid overfitting, especially when dealing with a small or imbalanced fingerprint data set (Ametefe et al., 2023). They are also black boxes in nature and, as such, are not easy to interpret, potentially lowering user trust in sensitive biometric applications. Thus, deep learning is likely to achieve high accuracy but is subject to hardware availability, dataset size, deployment environment, and requirements for explainability.

2.8.5 Transfer Learning in Fingerprint Recognition

In the context of fingerprint classification, transfer learning is crucial as it enables models trained on extensive image datasets to be fine-tuned for fingerprint images. In this case, rather than training all the CNN layers from scratch, low-level visual features, such as edges, lines, textures, and curves, are supplied using pretrained models, which can be useful for fingerprint ridge analysis. The final classification layers can then be adjusted for each fingerprint class to improve performance. This reduces training time and can even affect system performance when the database contains few fingerprints.

This study uses transfer learning to enable the use of the following deep learning models as comparative models: MobileNetV2, InceptionV3, and DenseNet201. In resource-constrained environments, MobileNetV2 is an efficient transfer learning model. InceptionV3 is a multi-scale feature learning approach. DenseNet201 is an even deeper feature reuse and more powerful representation. These models hence serve as a suitable benchmark for comparison with the traditional machine learning methods, such as KMCG-based and PCA-based.

2.8.6 Comparative Review

Deep learning models differ from conventional machine learning techniques in that they learn fingerprint features directly from images rather than relying on hand-coded descriptors. Comparative studies show that CNN-based models are superior to traditional methods for fingerprint and biometric classification, as they can capture large-scale spatial structures, edges, curves, and local texture variations in fingerprint

ridge patterns (Khan et al., 2020; Militello et al., 2021). In general, Rundo et al. (2021) compared the performance of pre-trained CNN architectures like AlexNet, ResNet and GoogLeNet to two fingerprint datasets: PolyU and NIST datasets and found that deeper CNN models were good at capturing the complex features of fingerprints. This provides a compelling explanation for the benchmarking provided by deep learning in this study.

For efficient computation, generally, you will need to use MobileNetV2. It employs depthwise separable convolutions, which enable the model to be simpler while maintaining high classification accuracy (Gulzar, 2023). Kaur et al. (2023) reported an accuracy of 93.50%, precision of 93.51%, and an F1-score of 92.69% for MobileNetV2. This enables MobileNetV2 to provide a solid balance of accuracy/performance. The significant advantage it has over heavier CNNs is that it is well-suited for mobile, embedded, and resource-constrained biometric systems. However, it may be a disadvantage when images are highly distorted or when there is little difference between the ridges of classes.

InceptionV3's strength is that it uses several types of convolutional filters of different sizes and dimensions, making it more effective at capturing features at various scales. This is suitable for fingerprint images in which both fine ridges and larger patterns are significant. Besides, Ratnakar (2024) demonstrated that the InceptionV3 model can be trained to produce useful representations for fingerprints, achieving 89.47% accuracy in fingerprint classification. However, InceptionV3 requires more dimensions in the input and computation than does MobileNetV2. Images can also be obscured by noise; global texture patterns can be smudged, may be partially printed, or may be picked up incorrectly. If multi-scale feature learning is generally worthwhile, as it improves classification accuracy, then it may be worth testing with InceptionV3 using a large number of nodes.

The DenseNet201 model is suitable for cases that require more depth in feature reuse. The dense connections allow the following layers to share features from previous layers, thus facilitating gradient flow and reinforcing feature representation (Sanghvi et al., 2023). Even DenseNet201 achieves 91.7% classification accuracy, as reported by Diarra et al. (2021), using augmented fingerprint images. DenseNet201 is

particularly well-suited to fingerprints with overlapping ridge patterns, distorted inputs, and slight class differences. But it is more memory-intensive and has higher latency than MobileNetV2, making it not feasible for resource-constrained systems.

In these studies, deep learning models consistently achieve higher accuracy than classical classifiers; however, this is not always the case in real-world deployment scenarios. MobileNetV2 is better suited to scenarios where speed and efficiency are key. When multi-scale feature extraction is required, InceptionV3 comes in handy. DenseNet201 can be used in high-accuracy applications with computational resources. But these three models all require labelled data, careful data preprocessing, regularization, and computational resources. They are also less interpretable than machine learning approaches based on KMCG or PCA (Linardatos, Papastefanopoulos, and Kotsiantis, 2020; Minaee et al., 2023). This forms the main gap in the literature, which the present study aims to bridge: the assessment of the best trade-off between accuracy, computational efficiency and interpretability between the deep-learning models and traditional classifiers (KMCG and PCA) under the same experimental conditions, on the same dataset, and with the same metrics.

2.9 Research Gaps and Motivation for Current Study

Despite extensive research on fingerprint classification, the relevant literature still has significant gaps in methodology. Several works are devoted to traditional machine learning using hand-crafted descriptors, and others to deep learning models that learn features directly from fingerprint images (Militello et al., 2021; Minaee et al., 2023). Traditional methods are useful because they are interpretable and computationally efficient, but their performance is heavily influenced by feature quality. Deep learning techniques tend to perform better, but need more data, more powerful hardware, and more training time. Thus, further investigation, comparing these methods under similar experimental conditions, is still required.

Another gap concerns KMCG. The use of vector quantization or codebook approach has been investigated for image recognition, but in fingerprint classification, few studies have examined the performance of KMCG across different window and codebook sizes. Previous work in the literature shows that feature discrimination,

computational cost, and generalization depend on the size of the codebook (Jurie and Triggs, 2005; Li, Mei, and Kweon, 2008). But little is known about how the KMCG window size and codebook size impact fingerprint classification accuracy, particularly when compared to PCA and deep learning. PCA is widely used as a reference standard for dimensionality reduction, although it may fail to capture some nonlinear ridge characteristics (Jia et al., 2022).

The present study is essential because it directly addresses these gaps, explores the performance of the KMCG across different window and codebook sizes, and compares the performance of the traditional machine learning model with that of the PCA and deep learning models. It is directly related to the study title and objectives, specifically by comparing the impact of KMCG parameters, traditional classifiers with KMCG descriptors, PCA as a baseline method, and all methods on accuracy, precision, recall, F1-score, confusion matrices, and computational efficiency. The study provides a more concrete basis for selecting a fingerprint classification method based on accuracy, interpretability, and available resources.

2.10 Summary

The chapter has thoroughly reviewed the literature on fingerprint classification, including the basics of biometric systems, feature extraction methods, and traditional and deep learning models. It reviewed the lightweight, noise-resistant feature extractor (KMCG) and how models such as SVM, KNN, RF, and XGBoost have been successfully applied with engineered features. It also highlighted the potential of deep learning architectures, especially MobileNetV2, InceptionV3, and DenseNet201, for cross learning their features and achieving good performance. The review, however, highlighted a number of areas for further improvement, such as the lack of integration of KMCG with deep learning, the absence of clear benchmarking practices, and the lack of comparative studies to date comparing both types of models under standard test conditions. These gaps provide the foundation and impetus for the following study.

Table 2.1 summarizes the previous research projects on fingerprint classification

Table 2.1 Summary of Related Studies

Paper	Method	Dataset	Main Contribution	Results
Rundo et al. (2021)	CNN (AlexNet, ResNet, GoogLeNet)	PolyU, NIST	Comparison of pre-trained CNN architectures for fingerprint classification, establishing performance benchmarks for high-precision and efficient models.	Achieved highest accuracy with PolyU dataset; ResNet outperformed others due to deep architecture's ability to capture complex patterns.
Zhang et al. (2020)	SVM and Minutiae-Based Features	FVC 2000	Developed a hybrid system combining minutiae and texture features for improved fingerprint classification accuracy.	Improved classification rate over traditional minutiae-based systems, especially in noisy environments.
Lee et al. (2018)	K-Nearest Neighbors with Ridge Orientation Features	FVC 2002	Proposed a KNN-based approach using ridge orientation and frequency as the primary feature vectors	KNN showed acceptable performance for small datasets but struggled with scalability for larger fingerprint databases
Shafie et al. (2023)	XGBoost with Minutiae Points and Local Binary Patterns (LBP)	FVC 2004	Integrated minutiae points and texture-based descriptors using LBP with XGBoost for high-precision fingerprint classification	XGBoost delivered improved accuracy and faster processing times compared to Random Forest and SVM models.

CHAPTER THREE

METHODOLOGY

3.1 Introduction

The chapter describes the methodology used to achieve the study's goal of evaluating fingerprint classification using the KMCG, PCA, traditional machine learning, and deep learning approaches. It introduces the research design, data description, data preprocessing, feature extraction, model implementation, evaluation, and ethical issues. The chapter also describes the experimental procedures designed to compare the performance of the chosen models in terms of accuracy, precision, recall, F1 score, confusion matrices, and computational efficiency. This methodological structure allows for a systematic, reproducible and research goal-oriented study.

3.2 Research Design

The study is a hybrid between the experimental and comparative research design in an effort to investigate the potential of classification given prints by using three methods: classical ML models using hand-crafted features, dimensionality reduction using PCA models and DL models with transfer learning. This research is primarily concerned with the systematic endeavour to examine, under a single paradigm the strength, limitations, and applicability of both of the methods.

The basis of research works on a tri-road strategy comprising of three parallel pipelines.

- In the former, conventional ML classifiers (SVM, KNN, Random Forest and XGBoost) are fitted on feature vectors obtained through KMCG.
- In the second path, PCA is applied as a dimensionality reduction technique on the fingerprint images. PCA reduces the feature space into orthogonal principal components that capture maximum variance while minimizing redundancy. The resulting low-dimensional representations are then used to train the same traditional ML classifiers. This provides an additional comparative baseline by assessing how PCA-based representations perform relative to KMCG features and raw deep learning approaches.

- In the third path, pretrained DL models (MobileNetV2, InceptionV3, and DenseNet201) are fine-tuned using resized fingerprint images without handcrafted features.

This design enables a fair and comprehensive comparison of KMCG-based feature engineering, PCA-based dimensionality reduction, and deep learning models, all tested on the same dataset with consistent evaluation metrics. The methodology follows a structured flow: dataset acquisition → preprocessing → feature extraction (KMCG or PCA for ML) → model training → performance evaluation. This process ensures that the distinct impacts of handcrafted feature engineering, linear dimensionality reduction, and end-to-end learning are clearly observed, thereby providing meaningful insights into which approach offers better accuracy, scalability, and efficiency in real-world fingerprint recognition systems.

Figure 3.1visualizes the research design

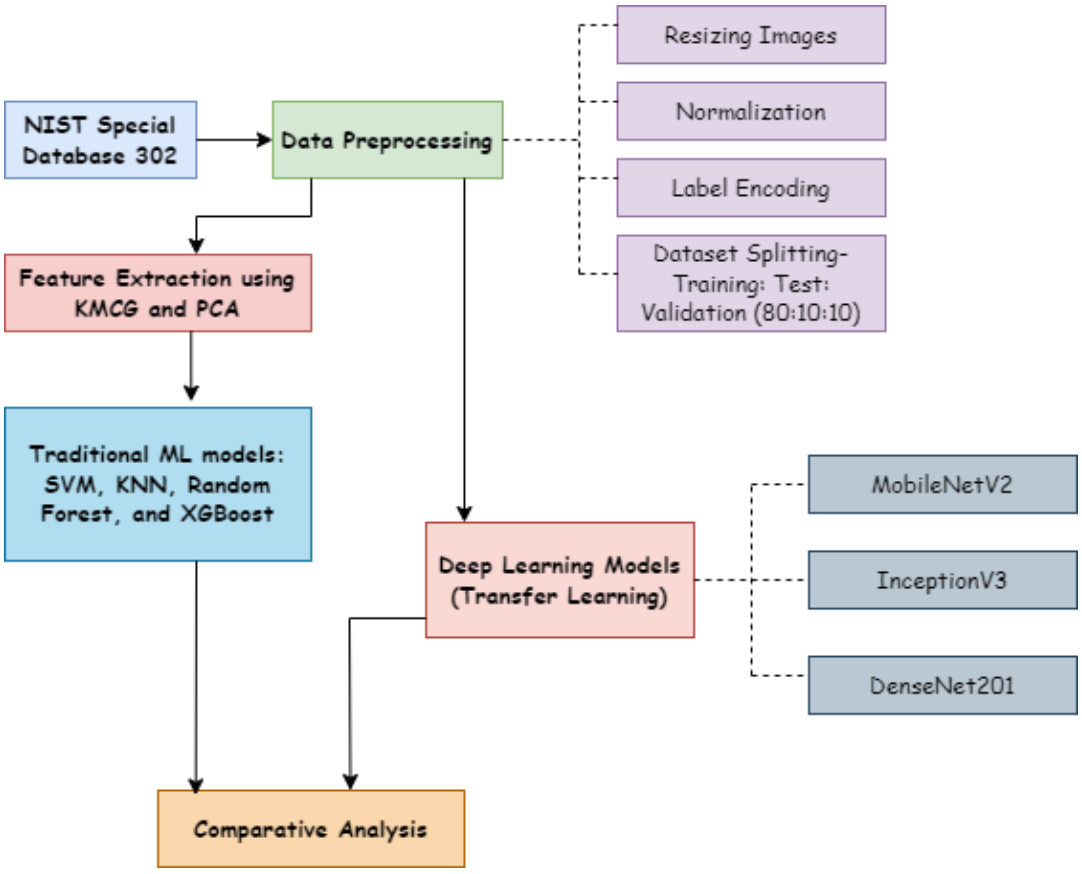

Figure 3.1: Research Design

3.3 Dataset Description

This study uses the fingerprint benchmark dataset, NIST Special Database 302, which is publicly available for biometric research. The images collected in the database were taken under realistic conditions and can be used for evaluation of fingerprint classification algorithms. In this study 4000 grayscale fingerprint images were used. The images were saved as PNGs and the metadata, such as the labels for the classes required for supervised learning, were saved as a TXT file.

The set of fingerprints included five main fingerprint classes: Arch, Tented Arch, Left Loop, Right Loop, and Whorl. The classes have been chosen as they are the most common structural types used in fingerprint classification. The distribution of records used in this study is shown in Table 3.1.

Table 3.1: Distribution of Fingerprint Images by Class

Fingerprint class	Number of images	Percentage
Arch	800	20%
Tented Arch	800	20%
Left Loop	800	20%
Right Loop	800	20%
Whorl	800	20%
Total	**4,000**	**100%**

The images used were high-resolution, grey-scale fingerprint samples. To ensure model compatibility, the fingerprint image dimensions were standardized during preprocessing to account for differences in acquisition sources and conditions. In InceptionV3, it is recommended to resize images to 299 × 299, whereas for MobileNetV2 and DenseNet201, it may be 224 × 224. First, for KMCG-based feature extraction, the grayscale images were divided into non-overlapping windows of varying sizes, and then PCA was applied to the flattened image vectors without dimensionality reduction.

There was also substantial variation in fingerprint samples, including ridge flow, ridge density, pattern structure, and image quality, which proved helpful for classification.

The study aimed to test the capacity of the KMCG model, the PCA model, and the deep learning model to classify fingerprints under realistic biometric conditions, and to examine the degree of variation. The dataset was therefore appropriate for this study due to the following reasons: (i) fingerprints were labelled into classes, (ii) there were enough samples in the dataset, (iii) the dataset had balanced classes, and (iv) the dataset contained sufficient diversity of images for evaluating the different traditional machine learning and deep learning approaches fairly.

3.4 Data Preprocessing

Data preprocessing was a crucial step to ensure that fingerprint images and associated metadata were in a consistent and model-ready format. Preprocessing was performed differently for the two model pipelines (traditional machine learning and deep learning) based on their respective input requirements.

For traditional ML models, fingerprint images were first converted to grayscale (if not already), and no resizing was applied to preserve local ridge details. The images were divided into non-overlapping windows as part of the KMCG feature extraction process, eliminating the need for manual dimension scaling.

In contrast, deep learning models required colour image inputs of fixed dimensions. Each image was resized to match the expected input size for each network: 224×224 pixels for MobileNetV2 and DenseNet201, and 299×299 pixels for InceptionV3. All pixel values were normalized to the range [0, 1] to improve convergence during training.

The class labels were parsed from the associated metadata files and encoded as numerical categories. The dataset was then split into training, testing, and validation sets using an 80:10:10 ratio.

The train, validation, and test sets were prepared prior to model training to avoid dataset leakage. The split was stratified by fingerprint class, meaning the images in Arch, Tented Arch, Left Loop, Right Loop, and Whorl were distributed evenly according to their proportions in the total sample. Images whose identifiers were in the same set were grouped together to prevent the same person from appearing in both the training and validation sets. Even duplicate file names and metadata were checked

prior to splitting. For PCA and KMCG, the feature-extraction parameters were obtained from the training set, and the resulting parameters were used for validation and test sets. This resulted in independence between the subsets and minimized the risk of inflated model performance.

3.5 Feature Extraction Using KMCG

KMCG is a data-driven, lightweight approach used herein to extract usable features from fingerprint images for classification using traditional machine learning. It compresses image local patches into codebook-based representative patterns through clustering, thereby offering compact feature encoding that is resistant to noise.

This algorithm begins by dividing each of the fingerprint images into non-overlapping window portions of a fixed size (flattened as feature vectors of pixel values). These feature vectors, in turn, are clustered together using K-Means (with a typical value of k = 2) to produce a codebook of image structural details. KMCG utilizes median clustering in a way that minimizes noise and outlier impact, resulting in robust local descriptors.

Experiments were carried out to determine the optimal configuration to extract features. Their window size varied from 2×2 up to 64×64, and codebook size of 2, 4, 8, respectively. For a silhouette score measure and classification accuracy for training time, the best performance has been observed for a codebook of 2, and a window size of 16×16 pixels.

The primary feature vectors for these machine learning models were final codebook. Final codebook was used as the primary feature vectors for machine learning models such as SVM, KNN, Random Forest and XGBoost. The low computation cost of KMCG and the fact that it is very adaptable in high dimensional biometric data, enabling to have an effective classification with low deep model complexity, were some reasons that made KMCG to be considered. The arrows in figure 3.2 represent the implementation approach of the KMCG:

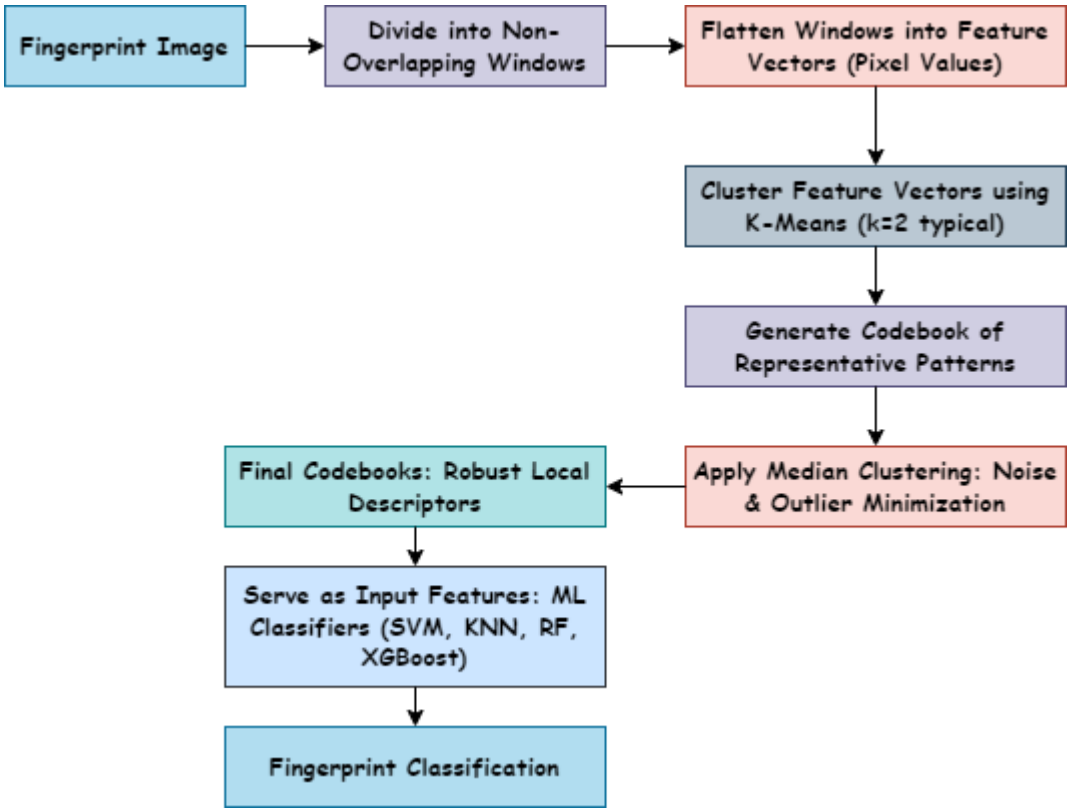

Figure 3.2: KMCG Implementation

The mathematical formulation of how KMCG works is described as follows:

i. Starting with a grayscale fingerprint image: $I \in R^{H \times W}$

ii. Dividing the image into fixed-size, non-overlapping windows: $s = h \times w$

iii. Flattening each window into a vector: $x_i \in R^{h \times w}$

iv. Collecting all flattened windows from the image: $X = \{x_1, x_2, x_3, \dots, x_n\}$

v. Choosing the codebook size K, which represents the number of clusters.

vi. Applying clustering to group similar window vectors together.

vii. For each cluster, compute a representative centre: $C = \{c_1, c_2, \dots, c_k\}$

viii. The clustering objective is to minimize the distance between each window vector and its closest cluster centre:

ix. $$C^* = \arg\min_{C} \sum_{i=1}^{N} \min_{1 \le k \le K} \| x_i - c_k \|^2$$

x. The final KMCG feature vector is obtained by flattening all cluster centres:

xi. $F_{KMCG}(I; K, s) = vec(C^*)$

xii. Combining the clustering objective and feature-vector formation gives the final formula:

$$F_{\mathrm{KMCG}}(I; K, s) = \mathrm{vec}\left(\arg\min_{C} \sum_{i=1}^{N} \min_{1 \le k \le K} \| x_i - c_k \|^2 \right)$$

In the implementation:

i. K (Codebook Size) = 2 and Window size $s = 16 \times 16$
ii. Therefore, the final feature vector length is $2 \times 16 \times 16 = 512$

Overall, KMCG works by converting each fingerprint image into a compact feature vector that summarizes the most representative local ridge-pattern windows.

3.6 Feature Extraction Using PCA

The PCA method was used in this study as an alternative feature extraction method to the KMCG-based method. It has been found to be useful in biometric applications, removing the redundancy, reducing the noise and emphasising the dominant structural patterns to enable efficient classification (Jia et al., 2022).

In this research, PCA has been used for pre-processing of the fingerprint images prior to feeding them to the traditional machine learning classifiers. The grayscale fingerprint images were raw, and they were initially unfolded as high-dimensional vectors. PCA was used to extract a subgroup of orthogonal principal components that have a chance to contain a lot of the data variance. The initial data was then mapped onto these main components to obtain a low-dimensional feature space, which retained much of the important ridge flow and texture information and eliminated repetitive and noisy features.

Retention of principal components was determined by looking at the cumulative explained variance, with 90 and 95 percent built-in thresholds. This ensured that most of the fingerprint data of interest remained intact, and dimensionality of the input data was greatly reduced. These reduced dimensional feature vectors were then given to the learning of traditional machine learning models like SVM, KNN, RF, XGBoost, under the same conditions of experimentation as in KMCG-based features.

Figure 3.3 visualizes the PCA implementation process.

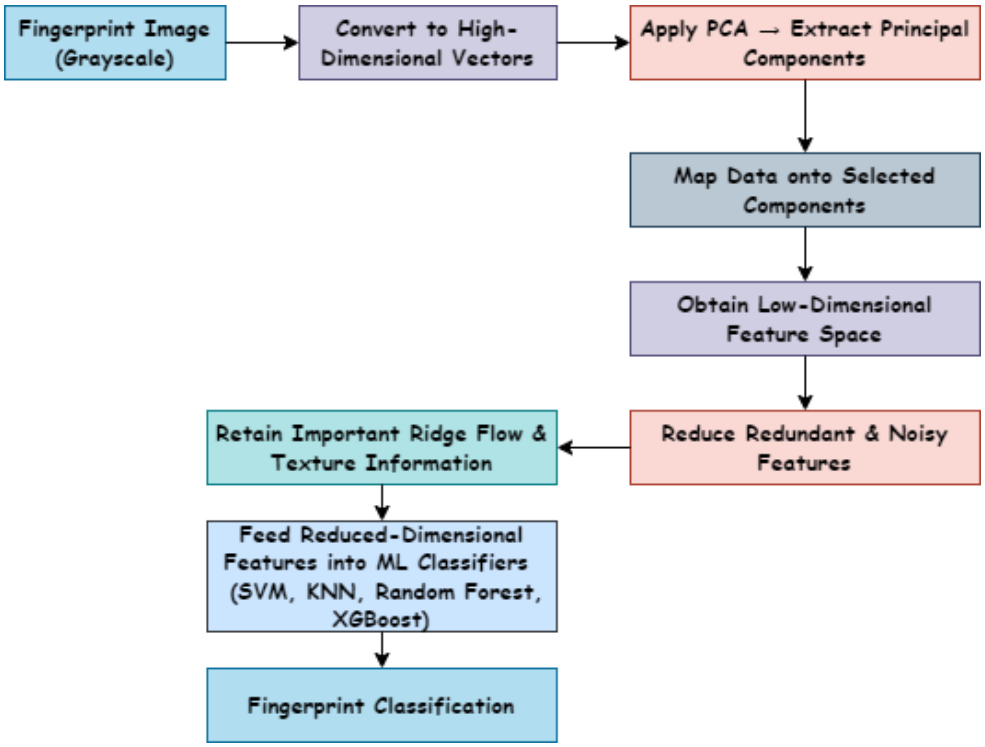

Figure 3.3: PCA Implementation

3.7 Model Selection and Training

This section describes the methods for selecting and training the traditional machine learning and deep learning models employed in this study. The selection of these models enabled a fair comparison between handcrafted feature-based classification, dimensionality-reduction-based classification, and automated deep feature learning. The chosen methods were also useful for the study goal to compare the accuracy of classification, computational efficiency, and practical feasibility.

3.7.1 Traditional Machine Learning Model Selection and Training

Traditional machine learning models were chosen for their ability to provide interpretable, computationally efficient classification methods for feature-based fingerprint recognition. The classifiers used are SVM, KNN, Random Forest and XGBoost. The models chosen were representative of various classification principles. SVM was selected because it is a margin-based classifier suitable for high-dimensional

feature spaces. KNN was chosen as a distance-based classifier for testing the separability of natural extracted fingerprint descriptors. Random Forest was selected as an ensemble model based on the bagging method and can handle non-linear relationships among features. Due to its ability to yield high performance in the presence of structured, numerical features, the boosting-based ensemble model XGBoost was chosen in this study (Ali et al., 2023; Odya, Górski and Czyżewski, 2023).

In traditional machine learning, the purpose of preprocessing was to preserve fingerprint ridge and texture information. Images were converted to grayscale because KMCG and PCA are based on pixel intensity patterns, not colour. The choice of KMCG was made because it produces compact codebook-based descriptors from windows of local fingerprint data, allowing us to study the effects of changing window and codebook sizes. This is because PCA was chosen as a baseline for dimensionality reduction, as it can compress high-dimensional fingerprint image data while retaining the dominant variance (Jia et al., 2022). This enabled comparison of the KMCG descriptors with a common statistical feature-reduction method.

The traditional classifiers were trained using features extracted with the KMCG and PCA approaches. To determine the optimal parameter combination for fingerprint classification, several window and codebook sizes were tested in KMCG experiments. The grayscale image vectors were flattened to create PCA features, which were then reduced to the principal components using thresholds based on the explained variance. The same classifiers were then trained on both KMCG and PCA feature sets for a fair comparison.

Analysis of the results indicated that hyperparameter tuning was performed to improve model performance and reduce bias caused by default values. The SVM kernel type, regularization parameter, and gamma were considered. KNN was tuned with the number of neighbours and the distance metric. Random Forest: number of trees, maximum depth, and minimum samples per split were taken into consideration. The number of estimators, learning rate, maximum tree depth and regularization parameters were fine-tuned for XGBoost. Training was performed on the training set,

tuning was based on the validation set's performance, and final evaluation was conducted on the test set.

3.7.2 Deep Learning Model Selection and Training

Deep learning was chosen for its ability to extract hierarchical fingerprint features directly from image data, which is well-suited to convolutional neural networks. The CNN architectures employed are: MobileNetV2, InceptionV3, and DenseNet201. The models are selected because they differ in architectural complexity and computational cost. MobileNetV2 was chosen for its lightweight nature and its ability to be implemented in a mobile or resource-constrained biometric system. Since it uses multi-scale convolution, it was deemed appropriate for extracting fine ridge details and broader fingerprint patterns, making it the choice for InceptionV3. The DenseNet201 is selected as the network due to its dense connectivity, which allows for feature sharing and deep representation learning (Gulzar, 2023; Sanghvi et al., 2023).

For deep learning, a preprocessing scheme was developed to enable the selected CNN architectures to be fed with the data. The fingerprint image dimensions for MobileNetV2 and DenseNet201 were 224 × 224, while those for InceptionV3 were 299 × 299. To improve convergence during training, the Pixel values were normalised to [0, 1]. Training images were augmented to improve generalization and prevent overfitting. These included small rotations, translations, and zooming, with no alteration to the class structure of the fingerprint. The deep learning models were developed in Python with the TensorFlow and Keras libraries. The models were trained using pre-trained ImageNet models via transfer learning. The base convolutional layers were frozen to preserve their identity as low-level visual features, including edges, curves, and texture patterns. The head was then fitted with a specific classification head designed for each model. For multi-class fingerprint classification, the head consisted of a GlobalAveragePooling2D layer, a Dense (128 units, ReLU activation) layer, and a Dense (SoftMax activation) output layer.

A sparse categorical cross-entropy loss was used along with the Adam optimizer to train the models. The number of training epochs and the batch size were set to 16 and 20, respectively. To avoid overfitting, early stopping was used, and the best model

(based on the validation loss) was saved using model checkpointing. Training histories were kept for subsequent analysis of the degree to which the models converged and whether their performance remained stable. The accuracy, precision, recall, F1-score, confusion matrix and execution time were used to assess the performance of the final model. The deep learning implementation pipeline is shown in Figure 3.4 and involves preprocessing, model selection, transfer learning, designing a classification head, setting training parameters, and evaluating results.

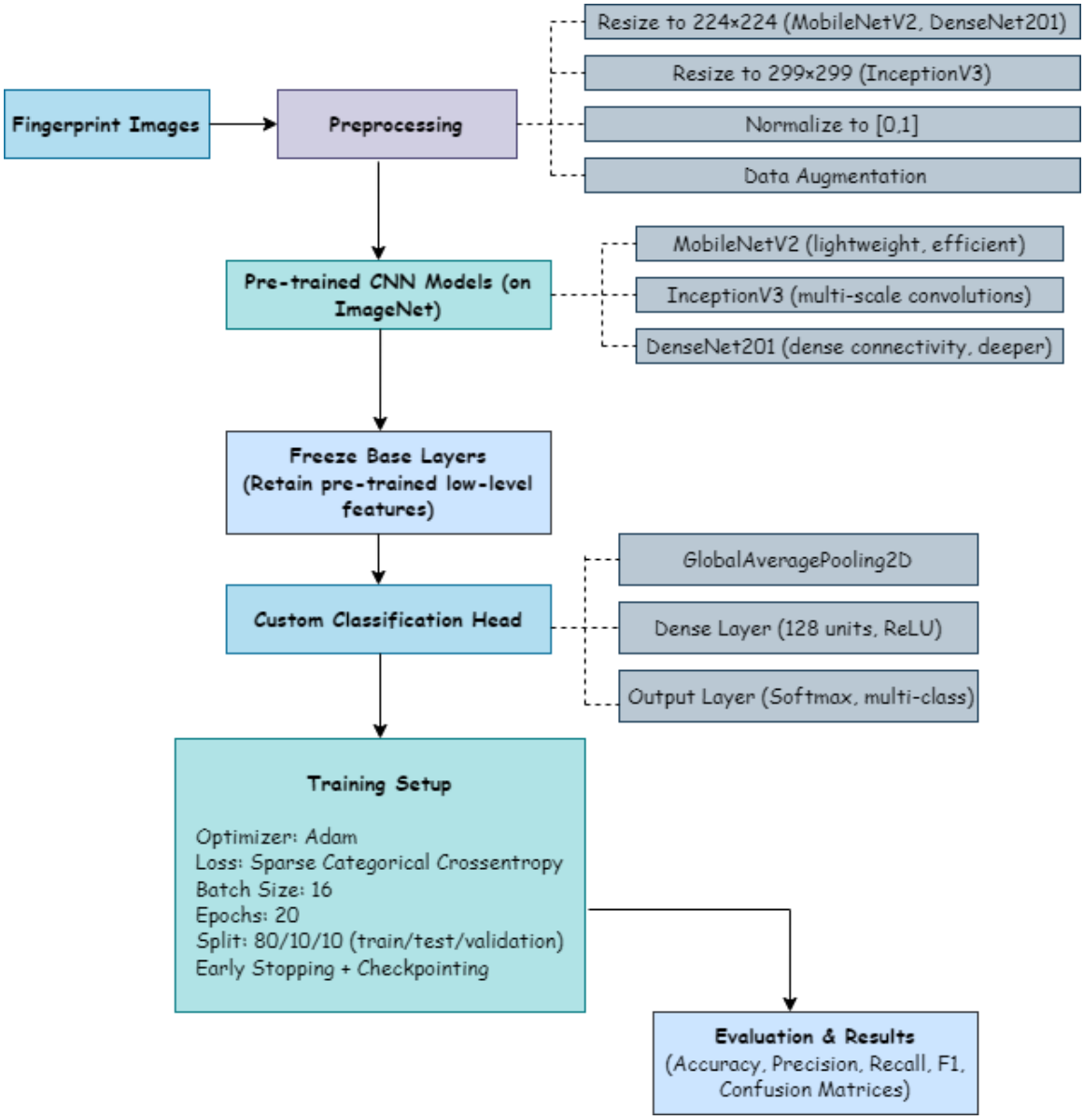

Figure 3.4 Deep Learning Models Implementation

3.8 Evaluation Metrics

To evaluate the performance of both traditional machine learning and deep learning models, several key metrics were employed, including accuracy, precision, recall, F1-score, confusion matrix, and execution time. Accuracy served as a general measure of correct predictions, while precision and recall served as measures of the power of models in correctly detecting and retrieving provided fingerprint classes. The F1-s, core, being the harmonic mean of precision and recall, helped balance the measure in cases of skewed class distributions. Classification performance was represented in confusion matrices across all five fingerprint classes, highlighting the most frequent confusion regions. Computational execution time was recorded at both training and inference times to compare computationally lightweight ML models with heavier deep learning models. All these measures were computed using standard libraries such as scikit-learn as well as TensorFlow, with multiclass evaluations reported using weighted means in order to cater for class imbalances in the set.

3.9 Tools and Techniques

The study used Python as the primary programming language. Pre-processing images, KMCG feature extraction, PCA transformation, training, and evaluating models were performed using OpenCV, NumPy, Pandas, Scikit-learn, TensorFlow, and Keras. The performance plots and confusion matrices were created using matplotlib and scikit-learn's visualization utility. Traditional machine learning models were implemented in Scikit-learn, while transfer learning with TensorFlow/Keras was used for MobileNetV2, InceptionV3, and DenseNet201. The experiments were conducted on a computer with an Intel Core i7 processor, 16GB RAM, a NVIDIA GPU (4GB dedicated), and Windows 11. They were recorded because execution time was one of the metrics evaluated, and the models' performance depended on the processor speed, available memory, and the availability of the graphics processing unit (GPU) during training.

3.10 Ethical Considerations

This research was conducted with due attention to the ethical standards for handling biometric data. The fingerprint images acquired were publicly available NIST Special

Database 302, intended for research and academic purposes, and did not include personally identifiable information (PII). All processing and handling of this data were restricted to this dataset, and no sensitive or private user data was accessed or disclosed. Furthermore, the models developed in this work were used solely for academic testing and not for deployment in any biometric identification system in use. The principles of data minimization and fairness were followed in the research, including remaining unbiased when comparing models and avoiding any form of unfairness or discrimination in the research findings. The study aimed to ensure methodological transparency, promoting the reproducibility and responsible use of AI.

CHAPTER FOUR

RESULTS AND DISCUSSION

4.1 Introduction

This chapter reports findings of the experiments that were conducted in this section in determining and comparing the effectiveness of different feature extraction and classification algorithms in the fingerprint recognition. The study can involve three important approaches, namely PCA, KMCG, and Deep Learning architectures. The chapter has a couple of tables, figures, and confusion matrices to display the results. The results are then compared and contrasted to shed light on the merits, constraints and feasibility of methods that have been compared.

4.2 Results

4.2.1 Effect of varying KMCG window and codebook sizes on fingerprint classification performance

The first objective of the study was to determine the impact of different KMCG windows and codebook sizes on fingerprint classification accuracy. To achieve this objective, KMCG was evaluated with three codebook sizes (2, 4, and 8) across six window sizes (2×2, 4×4, 8×8, 16×16, 32×32, and 64×64). The performance scores achieved from these combinations are listed in Table 4.1, and the experimental output from these combinations is displayed in Figure 4.1.

Table 4.1: KMCG Performance under Varying Window and Codebook Sizes

Codebook Size	Window Size	Performance Score
2	2×2	0.7519
2	4×4	0.7571
2	8×8	0.7712
2	16×16	0.7720
2	32×32	0.7573
2	64×64	0.6713
4	2×2	0.4382
4	4×4	0.4654
4	8×8	0.4864
4	16×16	0.4823
4	32×32	0.4678
4	64×64	0.4532
8	2×2	0.2599
8	4×4	0.2475
8	8×8	0.2798
8	16×16	0.2950
8	32×32	0.3062
8	64×64	0.2556

The best configuration of KMCG was found to be codebook size = 2 and window size = 16×16, with a performance score of 0.7720 (as shown in Table 4.1). This result indicates that a window size of 255 was a good compromise between retaining the fingerprint image's texture and capturing ridge details in a local region. In very small windows (2×2, 4×4), they yielded scores of 0.7519 and 0.7571, respectively, suggesting that very small windows can capture fine details but may be more sensitive to noise at the pixel level. The 8×8 window size achieved 0.7712, and the 16×16 window size achieved the best performance.

The window size was reduced to 16×16 for best performance. Under codebook size 2, the score dropped to 0.7573 at 32×32 and further declined to 0.6713 at 64×64. This implies that, for very large windows, the fingerprint features may be compressed within the window, making the differences in ridges and valleys at the window level too small for easy classification. Thus, large windows can make discrimination between similar fingerprint classes more difficult.

The results further indicate that a larger codebook did not increase the performance. The best score with codebook size 4 was 0.4864, obtained at a window size of 8×8. The optimal codebook size is 8, and the best score is 0.3062 with a window size of 32x32. These scores were well below the scores recorded for codebook size 2. This suggests that the larger codebooks have either introduced more complexity or a more divided feature space, leading to decreased generalization. Hence, this dataset was better represented using a compact codebook representation.

Figure 4.1 verifies the trend of Table 4.1, which shows that code book size 2 and window size 16×16 with 0.772 is the best combination. In summary, objective 1 was met, as it was clearly shown that window size and codebook size affected the performance of fingerprint classification based on KMCG. The optimum configuration was thus chosen as the starting point for the following experiments with the KMCG machine learning classifiers.

Figure 4.1: KMCG Performance under Varying Window and Codebook Sizes

4.2.2 Performance of KMCG-Based Feature Extraction

The results of fingerprint classification using KMCG features are summarized in *Table 4.2*, with detailed confusion matrices presented in Figures 4.2–4.5. The performance of KMCG is better than PCA-based FE in most of the classifiers with the highest accuracy of 70.2% for the SVM with an average score of 64.7.

Random Forest and XGBoost classifiers also showed competitive performances with accuracy of 64.8% and 68.9%, respectively, and had a balanced precision recall trade-off. The KNN, however, performed very poorly at 56.3% indicating its sensitivity to overlapping feature spaces. Execution times for KMCG classifiers were higher than PCA but remained significantly lower than deep learning models, with Random Forest requiring 50.3s and XGBoost 43.3s.

Table 4.2: Performance of KMCG-Based Feature Extraction with Different Classifiers

Method	Classifier	Accuracy	Precision	Recall	F1-Score	Average Score	Execution Time
KMCG	SVM	70.2	60.4	70.2	64.8	64.7	44.6
	KNN	56.3	57.4	56.3	55.9	56.5	32.6
	Random Forest	64.8	63.9	64.8	64.1	64.4	50.3
	XGBoost	68.9	68.5	68.9	68.5	68.7	43.3

The confusion matrices in Figures 4.2–4.5 offer more detailed class-level explanations of the classifiers based on the KMCG. As shown in Figure 4.2, the SVM classifier achieved the best recognition rate for the Whorl class (149 correct recognitions), and the other classes were confused, particularly Arch and Left Loop, which were misclassified as Whorl. It indicates that SVM found greater separability between Whorl patterns and was unable to achieve good separability between structurally similar classes. The predictions of the KNN classifier in Figure 4.3 were almost evenly distributed, with no class being dominant, suggesting that the KNN classifier had lower capacity to separate overlapping KMCG feature spaces. Although Random Forest in Figure 4.4 achieved the best recognition of Whorl, it also confused Right Loop and Whorl, meaning that some similarities in ridge flow were not overcome. The recognition rates of XGBoost in Figure 4.5 were relatively high in Tented Arch (66) and Whorl (79), but it still misclassified some loop patterns. In general, the confusion matrices indicate that the classification performance in KMCG was moderate, but there were still plenty of examples where similar ridge structures between the Arch, Tented Arch, Left Loop and Right Loop could not be separated.

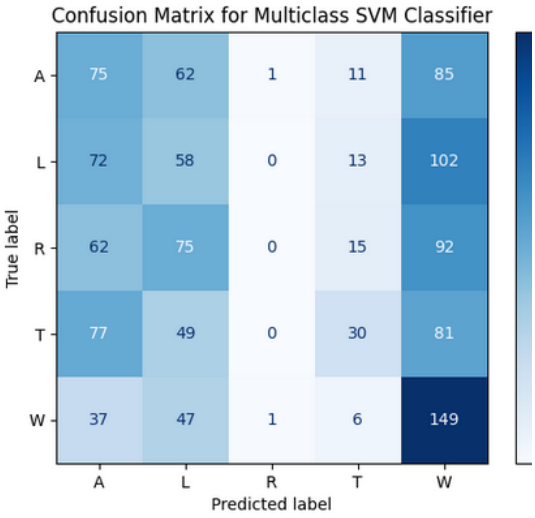

Figure 4.2: Confusion Matrix of SVM Classifier Using KMCG Features

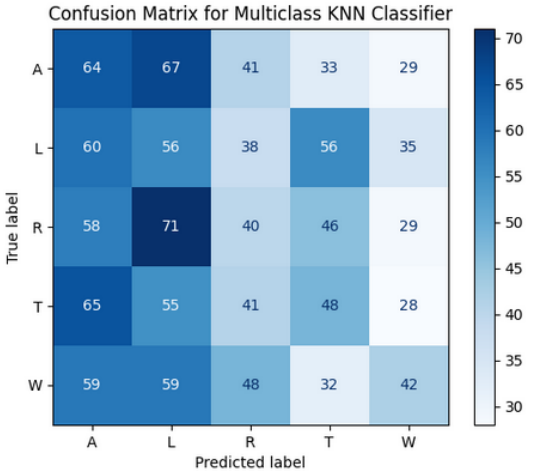

Figure 4.3: Confusion Matrix of KNN Classifier Using KMCG Features

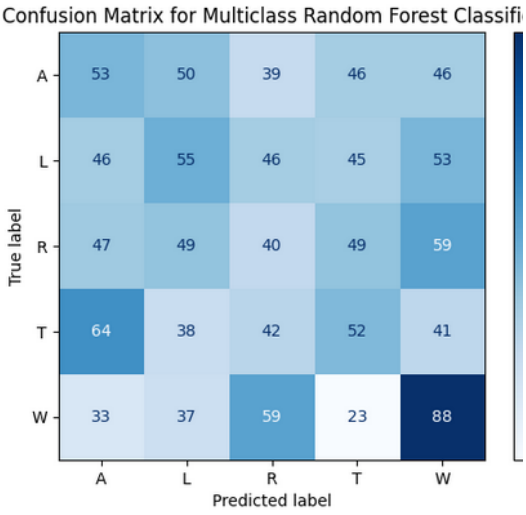

Figure 4.4: Confusion Matrix of Random Forest Classifier Using KMCG Features

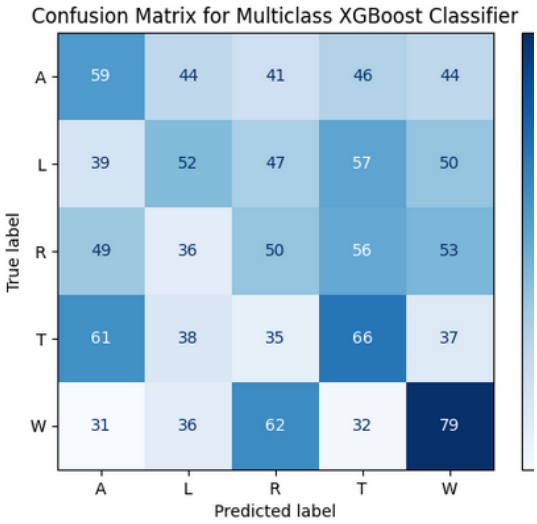

Figure 4.5: Confusion Matrix of XGBoost Classifier Using KMCG Features

Overall, these results demonstrate that KMCG is more effective than PCA at capturing discriminative fingerprint features, particularly local texture patterns. Although KMCG cannot achieve the highest accuracy of deep learning models, it has relatively high interpretability, efficiency and classification performance and is also a good traditional feature extraction method.

4.2.3 Performance of PCA-Based Feature Extraction

PCA was used to project the features of the fingerprint image into the lower dimensional subspaces and then applied different machine learning classifiers. The performance of SVM, KNN, Random Forest, and XGBoost classifiers was evaluated in terms of accuracy, precision, recall, F1-score, and execution time (*Table 4.3*). XGBoost achieved the highest accuracy (73.8%), demonstrating balanced precision and recall, though it also required a comparatively long execution time. SVM followed closely with an accuracy of 65.03%, while Random Forest achieved moderate performance at 61.9%. KNN underperformed, with only 56.7% accuracy and significant misclassifications.

Table 4.3: Performance of PCA-Based Feature Extraction with Different Classifiers

Method	Classifier	Accuracy	Precision	Recall	F1-Score	Average Score	Execution Time
PCA	SVM	65.03	70.6	65	61.2	65.4	28.96
	KNN	56.7	48.7	56.7	38.1	50.1	18.58
	Random Forest	61.9	59.2	61.9	52.3	58.8	23.78
	XGBoost	73.8	76.9	73.8	69	73.4	244.34

Figures 4.6–4.9 depict the confusion matrices of the four classifiers, obtained by using the PCA-based feature extraction for the four classifiers. The number of correct classifications for Left Loop was higher than that of Right Loop and Tented Arch in the PCA + SVM classification in Figure 4.6: 101 correct classifications, 79 correct classifications and 78 correct classifications, respectively. Unfortunately, Whorl was not very well recognized, with only 12 correct predictions, and a number of samples were classified as Tented Arch. Figure 4.7 shows that there was excessive bias towards the Arch and Tented Arch class; many fingerprints from the, left loop, Right loop, and Whorl classes were misclassified as Tented Arch. This is why the KNN F1 score is low in Table 4.3. Likewise, the PCA + Random Forest model focused the majority of its predictions on the Tented Arch class, with 141 correct examples, but few accurate predictions of Whorl, with only 5 correct predictions. Figure 4.9 indicates that PCA +

XGBoost achieved the most well-balanced PCA-based classification, and that the number of correct predictions improved for Arch, Left Loop, Right Loop, and Tented Arch. But Whorl was still a hard one to classify: it still had 17 correct predictions. Concluding overall, the confusion matrices confirmed that PCA reduced dimensionality was effective, but not sufficient to retain the ridge-pattern discriminative information for the consistent separation of classes, particularly for the Whorl and structurally overlapping classes.

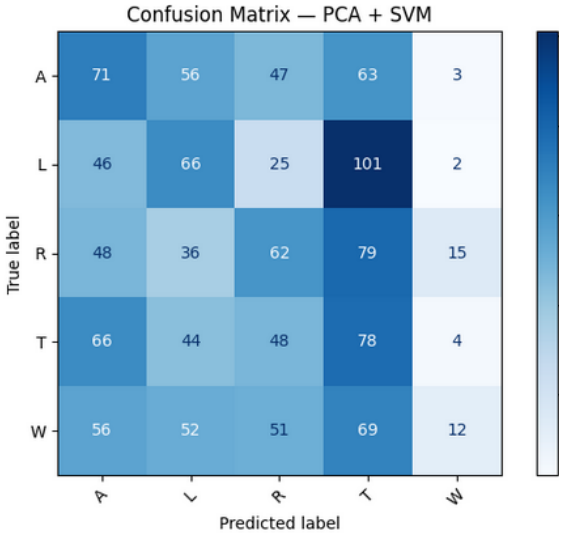

Figure 4.6: Confusion Matrix for PCA + SVM

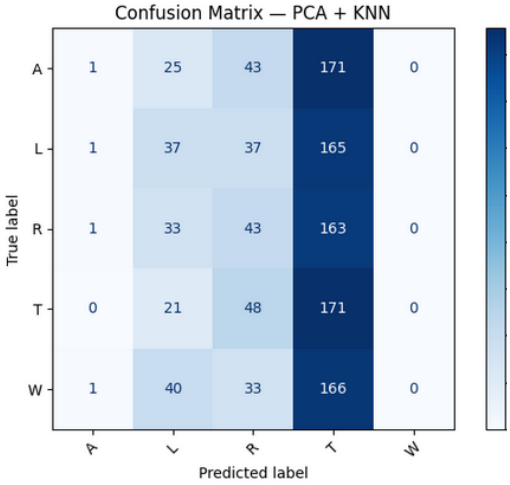

Figure 4.7: Confusion Matrix for PCA + KNN

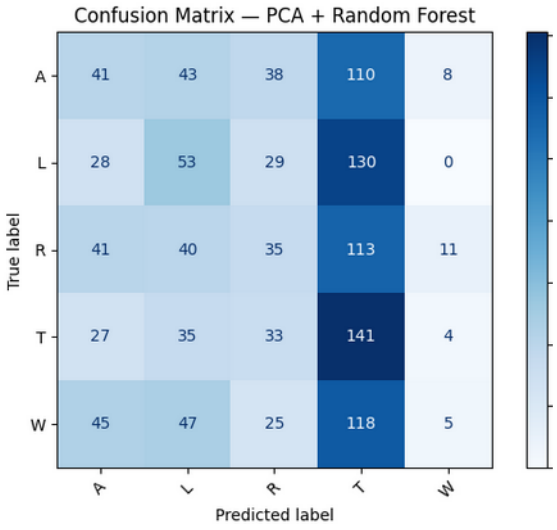

Figure 4.8: Confusion Matrix for PCA + Random Forest

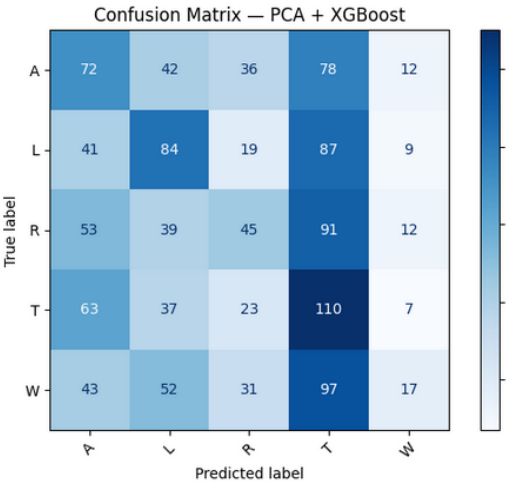

Figure 4.9: Confusion Matrix for PCA + XGBoost

The above results indicate that PCA, while effective in reducing feature dimensionality, does not always preserve the discriminative power required for high-accuracy fingerprint classification. Nonetheless, it provides a useful computationally efficient baseline against which the more complex KMCG and deep learning approaches can be compared.

4.2.4 Performance of Deep Learning Models

Fingerprint classification with deep learning models was also assessed in this work, including MobileNetV2, InceptionV3, and DenseNet201. Their performance on accuracy, precision, recall, F1-score, average score, and execution time is shown in Table 4.4. The confusion matrices are presented in Figures 4.10–4.12. In general, the deep learning models achieved better results than PCA- or KMCG-based ones. With an accuracy of 90.0%, DenseNet201 delivered the most balanced results, followed by precision (92.0%), recall (91.8%), and F1-score (91.7%), with a relatively high execution time (4041s). In terms of execution speed, MobileNetV2 outperformed the other models, achieving 91.7% accuracy with a much lower execution time of 1159s, making it suitable for resource-limited environments. InceptionV3, however, achieved only 83.4% accuracy and the longest execution time (6102s), making it impractical to use, despite competitive precision and recall.

Table 4.4: Performance of Deep Learning Models

Method	Classifier	Accuracy	Precision	Recall	F1-Score	Average Score	Execution Time
Deep Learning	MobileNetV2	**91.7**	**95.9**	**95.9**	**95.9**	**94.9**	**1159**
	InceptionV3	83.4	87.2	86.8	86.5	85.9	6102
	DenseNet201	90.0	92.0	91.8	91.7	91.4	4041

The confusion matrices shown in Figures 4.10–4.12 offer more details on the performance of the deep learning models at the class level. As seen in Figure 4.10, the MobileNetV2 model correctly predicted the majority of Whorl and Arch samples, with 147 and 127 predictions, respectively. Some samples on the Right Loop were misclassified as the Left Loop, and some on the Tented Arch were misclassified as Arch, indicating that some samples were still confused among the different loops and

Arches. InceptionV3 also performed well on the other two datasets, Whorl and Arch, with 155 and 136 predictions, respectively. It displayed, however, more confusion between Tented Arch and Arch, with 40 Tented Arch samples misclassified as Arch, indicating it is less successful at differentiating between upward-ridge structures similar to the eye. As shown in Figure 4.12, DenseNet201 made the most correct Whorl predictions (163) and also performed well on Arch, with 126 correct predictions. However, DenseNet201 incorrectly labelled some Right Loop samples as Left Loop and some Tented Arch samples as arch, and vice versa. The overall results of the confusion matrices showed that the deep learning models achieved better class-level recognition than the PCA and KMCG models, particularly for Whorl and Arch, but there were similarities between the loop and tented arch that caused misclassification.

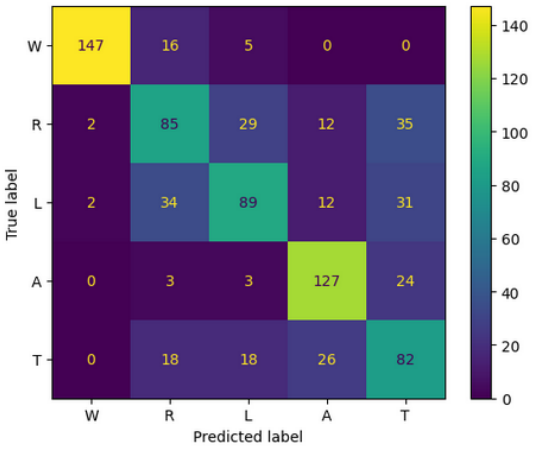

Figure 4.10: MobileNetV2 Confusion Matrix

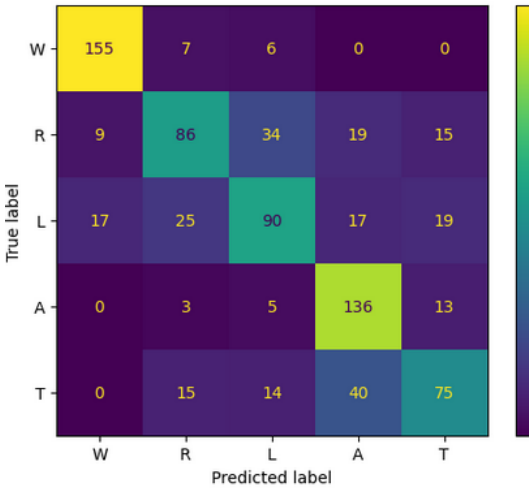

Figure 4.11: InceptionV3 Confusion Matrix

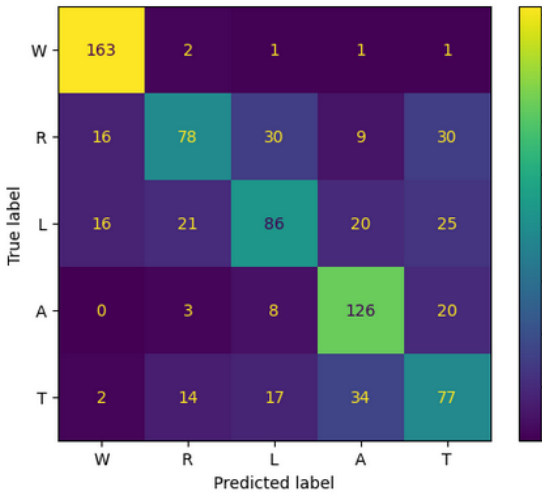

Figure 4.12: DenseNet201 Confusion Matrix

4.2.5 Comparative Analysis Across Methods

Table 4.5 presents an overall comparison of all the models examined in the study, including traditional machine learning, deep learning, and PCA-based traditional machine learning models of KMCG. The outcomes reveal that the deep learning models perform best across accuracy, precision, recall, F1-score, and average score. MobileNetV2 achieved the best average score of 94.9, DenseNet201 with 91.4 and InceptionV3 with 85.9. This means that deep learning models learned discriminative

fingerprint features directly from the image data, better than traditional feature extraction methods.

The traditional machine learning techniques yielded satisfactory results, with PCA and XGBoost achieving accuracies of 73.8 and an average score of 73.4, respectively. Among the KMCG-based models, the XGBoost model performed best, with an average score of 68.7, followed by the SVM and RF models, with average scores of 64.7 and 64.4, respectively. The KMCG features yielded the worst KNN performance (Score=56.5 on average), indicating that distance-based classification was less effective with these features. The results indicate that the ensemble-based classifiers (in particular, XGBoost) outperformed simpler classifiers when structured fingerprint descriptors were used.

Table 4.5: Comparative Analysis Across Methods

Method	Classifier	Accuracy	Precision	Recall	F1-Score	Average Score	Execution Time
PCA	SVM	65.03	70.6	65	61.2	65.4	28.96
	KNN	56.7	48.7	56.7	38.1	50.1	18.58
	Random Forest	61.9	59.2	61.9	52.3	58.8	23.78
	XGBoost	73.8	76.9	73.8	69	73.4	244.34
KMCG	SVM	70.2	60.4	70.2	64.8	64.7	44.6
	KNN	56.3	57.4	56.3	55.9	56.5	32.6
	Random Forest	64.8	63.9	64.8	64.1	64.4	50.3
	XGBoost	68.9	68.5	68.9	68.5	68.7	43.3
Deep Learning	MobileNetV2	91.7	95.9	95.9	95.9	94.9	1159
	InceptionV3	83.4	87.2	86.8	86.5	85.9	6102
	DenseNet201	90.0	92.0	91.8	91.7	91.4	4041

Figure 4.13 visually supports the comparison in Table 4.5 by showing the ranking of models by average performance. The figure clearly shows that MobileNetV2, DenseNet201, and InceptionV3 rank among the top three, confirming the superior performance of deep learning models. However, Table 4.5 also shows that this improved performance came at the cost of higher execution times. For example, MobileNetV2 required 1159 seconds, DenseNet201 required 4041 seconds, and InceptionV3 required 6102 seconds. In contrast, most PCA- and KMCG-based models required much shorter execution times.

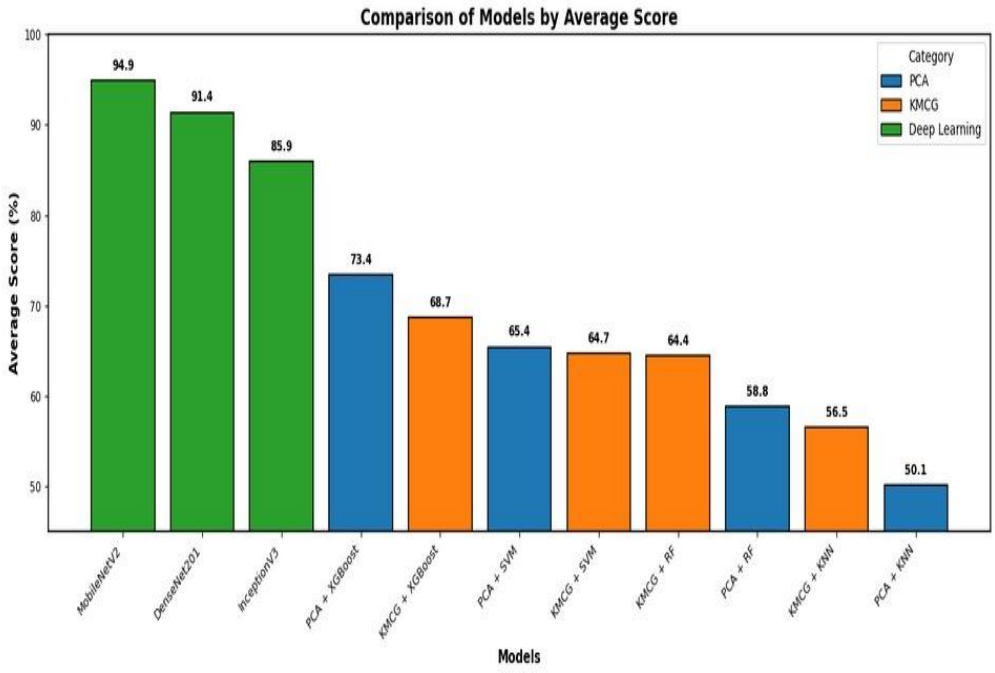

Figure 4. 13: Comparison of Model Performance

The comparative results confirm that there is a trade-off between classification performance and computational efficiency. The deep learning models performed best, and among them, the best-performing model was MobileNetV2, which had the highest accuracy and a reasonably lower execution time. The PCA-based and KMCG-based models were less accurate, but more computationally simple and interpretable. Therefore, for high-accuracy fingerprint classification systems, deep learning is more appropriate, whereas for biometric systems that lack resources or require interpretability, KMCG and PCA are useful.

4.3 Discussion

4.3.1 Effect of KMCG Window and Codebook Size Variation

The first goal of the study was to determine the influence of changes in KMCG window size and codebook size on fingerprint classification performance. The results indicated that KMCG's performance was highly sensitive to the parameters. The codebook size was set to 2, and the window size to 16×16. This result indicates that the medium-sized window included sufficient local ridge texture while maintaining the larger-scale fingerprint structure. The smaller window sizes (2×2, 4×4) tended to be less accurate,

possibly due to more noise and pixel-level variations they picked up, and also because they could see smaller local variations. This explains why larger windows (32×32 and 64×64) also resulted in a negative performance. With larger windows, too much ridge-level information was lost because of excessive spatial compression, which is needed to separate classes.

This is consistent with the results obtained in the study of visual codebooks. It indicates that different codebook designs and feature granularity influence recognition performance (Siddiqui et al., 2018). However, the results show that there is no direct correlation between codebook size and effectiveness. The scores were lower for codebook sizes 4 and 8 than for codebook size 2. This indicates that selecting larger codebooks could have generated a non-uniform feature space, leading to a decline in generalisation capability. There was no gain in discrimination. Therefore, to get the best from KMCG, its parameters must be balanced for detail, compactness, and classifier usability.

4.3.2 KMCG-Based Machine Learning Compared with Previous Studies

Moderate classification performance was achieved with the KMCG-based models. SVM achieved the highest accuracy, i.e., 70.2%, followed by XGBoost (68.9%), Random Forest (64.8%), and KNN (56.3%). The results indicate that the KMCG descriptors could be useful but not sufficiently discriminative for deep learning models. This better performance of SVM and XGBoost suggests that KMCG performs better with classifiers other than distance-based methods, such as margin-based methods and boosting. This is consistent with earlier research, which showed that the performance of traditional ML models relies heavily on features and classifier types (Militello et al., 2021; Ali et al., 2023).

The KMCG results are compared with previous traditional machine learning studies and found to be better than those that used highly specialised fingerprint descriptors such as ‘minutiae’, ‘ridge orientation’, or ‘texture features’ in combination. For instance, Shokrzade et al. (2022) achieved better classification results by selecting SVM and minutiae and texture features, and Shokrzade et al. (2021) demonstrated that KNN with ridge orientation and frequency features can be used acceptably with

smaller data sets. The disparity could be due to the method. The previous research was based on descriptors developed specifically for fingerprint ridge structure, whereas KMCG uses image summaries based on compact codebooks. KMCG is therefore simple and interpretable; however, it may not contain sufficient ridge information to distinguish structurally similar classes.

The confusion matrices also revealed difficulties in distinguishing between the respective classes of fingerprint images for the KMCG, particularly for Arch, Tented Arch, Left Loop, and Right Loop images. This is significant because there are similar ridge-flow structures in both of these classes. Degrading conditions for KMCG include noisy fingerprint images, weak local ridge patterns, partial prints, and window/codebook combinations that over compress the image. So, KMCG can be applied in resource-constrained systems, but in some cases, it may require additional preprocessing or a combination of features to achieve better class separation.

4.3.3 PCA-Based Models Compared with KMCG and Literature

The PCA-based models also achieved moderate performance. The highest accuracy of the traditional machine learning models was obtained by PCA combined with XGBoost (73.8%), PCA-SVM (65.03%), PCA-Random Forest (61.9%) and PCA-KNN (56.7%). The PCA-XGBoost yielded better results than the other KMCG-based classifiers, implying that PCA kept sufficient dominant variance to enable boosted tree learning to detect relevant patterns. This aligns with Jia et al., 2022, who describe dimensionality reduction as a technique that can enhance the computational efficiency and eliminate redundancy in high-dimensional feature spaces.

The PCA confusion matrices, however, revealed that the dimensionality reduction did not capture all the details of the ridge patterns that were important to the discrimination. For several of the PCA-models there was a prediction bias towards certain classes, particularly the Tented Arch, and also in some cases Whorl was poorly predicted. This constraint is what is known to be the weak point of PCA as the linear versions: they preserve dominant variance but lose non-linear texture structures and subtle differences in ridge-flow (Gewers et al., 2021). So, PCA is useful as a baseline

method that is computationally efficient but might not be enough for complex fingerprint class discrimination by itself.

4.3.4 Deep Learning Performance Compared with Previous Studies

The best performance was obtained by deep learning models. MobileNetV2 achieved the highest overall accuracy of 91.7% and an average score of 94.9%, making it the top performer. The DenseNet201 network achieved 90.0% accuracy and 91.4% average score, while the InceptionV3 network achieved 83.4% accuracy and 85.9% average score. The results align with the other literature on the subject that CNN-based models have been found to be more successful than handcrafted ones because of their ability to capture hierarchical image features such as edges, ridge flow, texture transitions, and class-specific spatial patterns (Khan et al., 2020; Minaee et al., 2023).

The accuracy of the MobileNetV2 result is similar to that reported in Kaur et al. (2023), which achieved 93.50% accuracy in biometric image classification using MobileNetV2. Currently, the accuracy of MobileNetV2 is 91.7%, a bit lower perhaps due to the presence of similar ridge features within fingerprint classes and to acquisition variability in the dataset. This DenseNet201 achieved 90.0%, similar to Diarra et al. (2021), who reported 91.7% accuracy in fingerprint classification with DenseNet. InceptionV3 did not perform as well as MobileNetV2 and DenseNet201 in terms of accuracy. The accuracy of 89.47% reported by Ratnakar (2024) for InceptionV3 in the task of fingerprint alteration detection is lower than that obtained in this study, which might be attributed to differences in data sets, data preprocessing, class distribution, and the task.

The results also apply to studies based on NIST fingerprint databases. For instance, Wang et al. (2014) achieved 93.1% classification accuracy using a deep neural network on NIST-DB4, and Hou et al. (2021) achieved 97.3% classification accuracy using unsupervised deep learning on NIST-DB4. These studies outperformed current deep learning models, but this comparison should be interpreted with caution, as differences in the NIST datasets, the number of classes used, and the pre-processing methodology may have affected the results. Grosz and Jain (2023) reported the results of latent-to-rolled fingerprint matching using NIST SD302 in a different task setting. Their work

shows that SD302-like fingerprint data is difficult to handle, as it is affected by image quality, acquisition conditions, and latent or operational variability during recognition. The currently obtained classification results, therefore, are to be understood as classification results of this selected subset, the preprocessing pipeline and the evaluation metrics.

4.3.5 Dataset, Preprocessing, and Evaluation Differences

Some discrepancies between the present results and those reported in earlier research studies are likely due to data dependence. The number of samples, class balance, image quality, sensor type, resolution, and whether the image is rolled, plain, latent, or contactless all affect fingerprint classification results. The performance associated with NIST SD302 may differ from that reported in studies using the FVC, PolyU, NIST-DB4, or SOCOFing datasets, as it is a public dataset containing fingerprint images from multiple sources. Thus, the answer is, in general, similar but not necessarily the same numerically.

Processing performance is also affected by preprocessing. The images were resized to fit the deep learning input, and the KMCG images used window-based grayscale descriptors, while the PCA images used flattened grayscale vectors. These preprocessing differences affect the information each model can receive. The spatial image structure was directly fed to deep learning models for hierarchical feature learning. The image data were compressed using KMCG or PCA prior to classification, thereby speeding up classification but losing some details. In addition, there were differences between the studies in how these evaluations were done. Some earlier studies report only accuracy; in this work, we report accuracy, precision, recall, F1-score, average score, confusion matrices, and execution time. This is a more comprehensive evaluation that offers a more balanced perspective on the models' performance.

4.3.6 Computational Cost and Practical Trade-Offs

The results show that there is indeed a clear trade-off between accuracy and computation efficiency. Models based on deep learning achieved the highest accuracy but ran significantly longer. The computation time for MobileNetV2 was 1159

seconds, for DenseNet201 it was 4041 seconds, and for InceptionV3 it was 6102 seconds. Most classifiers using PCA and KMCG ran much faster. The argument is that deep learning is better suited to high-accuracy contexts where hardware resources are available, whereas in low-resource or interpretability-focused contexts, traditional feature-based methods are more relevant.

MobileNetV2 was the best-performing deep learning model across overall performance and execution time, whereas DenseNet201 and InceptionV3 were the least performing models in terms of execution time. This renders it more suitable for use in mobile or embedded biometric systems. DenseNet201 proved useful for regions requiring high feature learning capacity, but due to its computational burden, it is not suitable for resource-limited regions. The least efficient deep learning model in this study was InceptionV3, which had the lowest accuracy and the highest execution time.

4.3.7 Generalizability, Limitations, and Practical Implications

The results can be transferred to a fingerprint classification application based on similar image datasets and a supervised learning process. But cautions should be taken when generalizing to real-world biometric systems. In a real deployment, the fingerprint images might not be of the same quality; there might not be a camera; the dataset could be skewed; there may be more or fewer prints; the prints might be worn; or the images might come from a population not in the training set. As a result of these conditions, performance can be compromised, particularly for KMCG and PCA, because handcrafted/compressed features may not capture complex variations. The study also demonstrates the strengths and limitations of KMCG. It has the advantages of easy explanation, small representation, and a lower computational burden than deep learning. It has drawbacks in terms of window size, codebook size, noise, and class overlap. The performance of KMCG can suffer due to the following reasons: (1) If the window size is too small, then the sign's shape might be underrepresented within a particular codeword. (2) If the window size is too large, the feature vector from that window size may include a number of irrelevant features. (3) If the codebook size is too large, it could cause the codebook to fragment the feature space. Therefore, KMCG is not suited to high-accuracy applications such as deep learning but is useful if simplicity and transparency of computation are desired.

The overall results align with the study objectives. Objective 1 was addressed by selecting the optimal codebook size of 2 and the window size of 16×16 for the KMCG configuration. Objective 2 was addressed by demonstrating that SVM and XGBoost were the most effective traditional classifiers based on KMCGs. In objective 3, it is demonstrated that among PCA-based models, PCA-XGBoost achieved the best result. Objective 4 was addressed by the fact that deep learning models outperformed KMCG and PCA, with the MobileNetV2 achieving the best performance. In practice, the choice of the model should be determined according to the deployment requirements: If the accuracy of the deep learning model is required to be as high as possible and the efficiency is required to be relatively high, the MobileNetV2 model is more suitable; If interpretability, speed, and low computation are needed, the KMCG model and the PCA model are both appropriate.

CHAPTER FIVE

CONCLUSION

5.1 Summary of the Study

This work compares the performance of the fingerprint classification based on KMCG, PCA, traditional machine learning and deep learning models. The primary goal was to investigate the impact of the window and codebook sizes on the classification accuracy of KMCG classifiers and to compare the accuracy of KMCG classifiers with that of PCA based and DNN classifiers. The results are presented in a summary for each of the four research objectives.

5.1.1 Objective 1: Effect of Varying KMCG Window and Codebook Sizes

The first goal was to find out how different sizes of KMCG window and codebook affects fingerprint classification performance. This objective was achieved by testing codebook sizes of 2, 4, and 8 across different window sizes, including 2×2, 4×4, 8×8, 16×16, 32×32, and 64×64. Both parameters had a significant influence on KMCG performance, as seen in the results. The optimal codebook size and window size were found to be 2 and 16×16 respectively, with a score of 0.7720. This demonstrated that a medium sized window and a compact codebook gave the best compromise between retaining local ridge information and the absence of unnecessary complexities in the code.

5.1.2 Objective 2: Performance of Traditional Machine Learning Using KMCG Features

The second goal was to assess the performance of the traditional machine learning algorithms based on the fingerprint descriptors created by KMCG. This objective has been accomplished by training SVM, KNN, Random Forest and XGBoost using the best KMCG feature set. The results indicated that SVM had the best accuracy (70.2%) based on KMCG followed by XGBoost (68.9%), Random Forest (64.8%) and KNN (56.3%). The results suggest that the average classification performance of KMCG was in the medium range. SVM and XGBoost are superior to this due to their ability

to perform better on structured feature spaces, while the KNN was more impacted by features like the overlapping fingerprint class features.

5.1.3 Objective 3: Performance of PCA-Based Feature Extraction

The third goal was to analyse the performance of feature extraction by PCA method with traditional machine learning classifiers. To achieve this, the principal component analysis (PCA) was used as a dimensionality reduction technique and then used to classify the reduced features using classifiers such as SVM, KNN, random forest (RF), and XGBoost. The best accuracy was obtained with PCA-XGBoost model with accuracy of 73.8%, followed by SVM with 65.03%, Random Forest 61.9% and KNN with 56.7%. The findings suggested that PCA can serve as a simple baseline model, particularly in conjunction with XGBoost. It was found that the confusion matrices did not completely retain all the details of the ridge pattern discrimination, particularly for classes with similar structures.

5.1.4 Objective 4: Comparative Performance Across KMCG, PCA, and Deep Learning Models

The fourth goal was to evaluate the accuracy, precision, recall, F1-score, confusion matrices, and computational efficiency of traditional machine learning models, PCA-based models, and KMCG-based deep learning models. All models were compared side-by-side to achieve this objective. The best overall results were obtained with deep learning models. Overall, MobileNetV2 achieved the best accuracy of 91.7% and the highest average score of 94.9, followed by DenseNet201 with 90.0% accuracy and 91.4% average score. The accuracy of InceptionV3 is 83.4%, and the average score is 85.9. Demonstrations of deep learning models, however, took longer than those of KMCG and PCA-based models.

Overall, the study revealed that the deep learning models were the most effective for fingerprint classification tasks, whereas KMCG and PCA models were also effective in scenarios where interpretability and computational efficiency are crucial. MobileNetV2 had the best trade-off between accuracy and execution time among deep learning models. KMCG was useful for extracting features that were compact and easily interpretable; however, the results were found to be very sensitive to window

and codebook size. While this PCA was rather computationally economical, it did not retain complex ridge structures in fingerprints as well. From this, the study revealed that the most suitable fingerprint classification technique is dependent on the deployment environment, computational resources, and the level of classification accuracy needed.

5.2 Implications and Recommendations

5.2.1 Implications of the Study

This research has significance with regard to the theoretical and practical aspect of fingerprint classification. The study showed that, theoretically at least, deep learning-based feature learning is more effective for fingerprint classification than the handcrafted approach and the statistical reduction approach. MobileNetV2 got the highest accuracy (91.7%) and the highest average score (94.9), while DenseNet201 got the next best accuracy (90.0%) and average score (91.4). This supports the notion that a CNN is able to learn complex ridge patterns, texture differences and class-level spatial features better than traditional feature extraction methods (Khan et al., 2020).

The study also shows that KMCG and PCA are still valid methods for fingerprint classification although they do not perform the best as compared to deep learning. The experimental results indicate that the parameter selection for KMCG directly affects classification performance, with a codebook size of 2 and a window size of 16×16 as the best settings. The best traditional machine learning performance was achieved by PCA combined with XGBoost, with an accuracy of 73.8%, followed by KMCG combined with SVM, with an accuracy of 70.2%. The results suggest that traditional feature-based methods can be applied when interpretability, reduced computational cost, and ease of deployment are desired.

From a practical standpoint, the results indicate that the model's deployment choice should be based on the deployment environment. Deep learning models work better for high-accuracy biometric systems that have high computational power. For when accuracy and feature richness are the primary concerns, DenseNet201 might be helpful, but when both accuracy and running efficiency are required, more people might find MobileNetV2 more viable. KMCG- and PCA-based approaches can also

be applied, as they require less computational time and offer more transparent feature processing than deep learning models, making them suitable for resource-constrained systems.

5.2.2 Recommendations

The results show that MobileNetV2 is suitable for fingerprint classification systems that must execute efficiently and be accurate. It showed the best overall performance of all the models and had the lowest execution time compared to DenseNet201 and InceptionV3. Thus, MobileNetV2 can be applied in practical biometric applications such as mobile authentication, access control, and embedded recognition systems.

DenseNet201 is recommended for biometric systems with high security requirements or for server applications when computational resources are available. It had a much longer execution time but achieved good classification performance and good recognition of complex fingerprint patterns. Its use should be validated with sufficient hardware, optimized training, and care to prevent unnecessary computational burden.

The KMCG is suggested for use in fingerprint classification methods with limited resources and/or interpretable classification. The study found that the maximum performance is achieved with a codebook size of 2 and a window size of 16×16. So, the window and codebook parameters of the future KMCG-based system should be tuned before classification. In addition, a stronger classifier such as SVM or XGBoost would be recommended, as both models proved more efficient than KNN and Random Forest when KMCG descriptors were used.

PCA is recommended as a base preprocessing technique rather than a good feature extraction technique itself. Overall, traditional Machine Learning methods, specifically PCA-XGBoost, had the highest accuracy, showing that PCA effectively reduced dimensionality without loss of classification information. It is to be noted, however, that PCA can miss some nuances of the ridges, which may be non-linear, so it is best used with strong classifiers or as part of a hybrid system.

Hybrid models such as KMCG, PCA, and deep learning should be investigated in the future. For example, the handcrafted KMCG descriptors can be combined with CNN generated features to jointly leverage handcrafted interpretability and deep feature

discrimination. Alternatively, PCA can be used to reduce the dimensionality of the features before classification and/or combined with lightweight deep learning models. These hybrid architectures can enhance the accuracy, efficiency, and interpretability.

A real-time application is also suggested for the optimization of the model. The deep learning models need to be improved using pruning, quantization, fine-tuning, attention mechanisms and data augmentation. The following techniques are often used to reduce the size of the model, speed up the inference speed and deploy the model to mobile or edge devices. EfficientNet, Vision Transformers, and attention-based lightweight CNNs are emerging architectures that also need to be tested and co-modelled with KMCG or PCA features to determine whether newer architectures can provide improved fingerprint classification without a significant increase in computational complexity.

Last but not least, explainability needs to be incorporated into future fingerprint classification systems. Saliency maps, Grad-CAM, or feature importance can be used to provide insights into the fingerprint region that contributes most to classification. This is particularly critical in forensic, legal, and security environments, where decisions made by a model are critical and need to be clear and believable.

5.3 Limitations of the Study

This study had some limitations that may affect the generalizability of the results. Although the dataset is representative, it might not fully reflect the diversity of real-world fingerprint images, including challenges such as partial images, noise, and environmental factors. A limited number of classifiers and models were used. With PCA, only four classifiers (SVM, KNN, Random Forest, XGBoost) were investigated, and with deep learning, only three pre-trained architectures (MobileNetV2, InceptionV3, DenseNet201) were fine-tuned. More classifiers and networks, including ResNet, EfficientNet and Vision Transformers, may offer other insights.

Additionally, execution times have been measured within a particular computational context, and might have influenced the results because of hardware, software, and implementation details. A small amount of tuning of the accuracy of the deep learning models was done and additional experiments can be conducted on hyper parameter

tuning or in data augmentation or altering the architecture of the model to get additional improvements. The study also ignored explainability and robustness to adversarial attacks; two important issues for the deployment of deep learning in biometric systems. These suggest the results are valuable but require further research to expand their scope and validity.

5.4 Conclusion

In general, PCA and KMCG remain effective, with limited use and specific applications, but no alternative to deep learning models for fingerprint classification exists today. Finally, it is important to note that MobileNetV2 offers a reasonable trade-off between efficiency and accuracy, making it suitable for practical applications. For other cases, however, in which accuracy is not as important as cost, DenseNet201, even though it is a high-cost, high-accuracy network, is the state of the industry. Overall, this study has demonstrated that there is no universal solution for the approach; what is appropriate will always depend on the constraints, needs, and objectives of a particular biometric system. The comparative study contributes to the existing biometric debate at the academic level, confirming the viability of deep learning and recognizing that conventional methods are not useless. The study also presents potential future research opportunities that may lead to an efficient, explainable, secure, and effective fingerprint classification system.

REFERENCES

Abdul Hamid, L.B., Mohd Khairuddin, A.S., Khairuddin, U., Rosli, N.R. and Mokhtar, N., 2022. Texture image classification using improved image enhancement and adaptive SVM. *Signal, Image and Video Processing*, *16*(6), pp.1587-1594.

Abdulrahman, S.A. and Alhayani, B., 2023. A comprehensive survey on the biometric systems based on physiological and behavioural characteristics. *Materials Today: Proceedings*, *80*, pp.2642-2646.

Abed, A.J. and Abdulah, D.A., 2025. Advancements and Challenges in Low-Quality Fingerprint Identification: A Comprehensive Survey. *Bilad Alrafidain Journal for Engineering Science and Technology*, *4*(1), pp.127-136.

Adadi, A., 2021. A survey on data-efficient algorithms in big data era. *Journal of Big Data*, *8*(1), p.24.

Adnan, M.M., Priyanka, K., Lande, A. and Swamy, I.M.K., 2024, July. Fingerprint Recognition System Based on XGBoost Matching Algorithm for Forensic Analysis. In *2024 International Conference on Data Science and Network Security (ICDSNS)* (pp. 1-4). IEEE.

Alazzawi, M.S.J., 2022. Facial detection using image processing techniques and deep neural network.

Alhijaj, J.A. and Khudeyer, R.S., 2023. Integration of efficientnetb0 and machine learning for fingerprint classification. *Informatica*, *47*(5).

Ali, Y.A., Awwad, E.M., Al-Razgan, M. and Maarouf, A., 2023. Hyperparameter search for machine learning algorithms for optimizing the computational complexity. *Processes*, *11*(2), p.349.

Alkinani, F.S. and Rahma, A.M.S., 2019. A comparative study of KMCG segmentation based on YCbCr, RGB, and HSV color spaces. *Journal of Al-Qadisiyah for computer science and mathematics*, *11*(1), pp.Page-53.

Alkinani, F.S. and Rahma, A.M.S., 2019. Enhanced cartooning system based on dynamic augmented kmcg and lip. *Iraqi Journal of Science*, pp.653-661.

Alzubaidi, L., Zhang, J., Humaidi, A.J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaría, J., Fadhel, M.A., Al-Amidie, M. and Farhan, L., 2021. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. *Journal of big Data*, *8*(1), p.53.

Ametefe, D.S., Sarnin, S.S., Ali, D.M. and Muhammad, Z.Z., 2023. Fingerprint pattern classification using deep transfer learning and data augmentation. The Visual Computer, 39(4), pp.1703-1716.

Biometric Technology Market Size & Share Report, 2030 (no date). https://www.grandviewresearch.com /industry-analysis/biometrics-industry.

Charisma, R.A. and Adhinata, F.D., 2023, February. Transfer learning with densenet201 architecture model for potato leaf disease classification. In *2023 International Conference on Computer Science, Information Technology and Engineering (ICCoSITE)* (pp. 738-743). IEEE.

Chen, B., Ma, J., Zhang, L., Xiong, Z., Fan, J. and Lan, H., 2024. An improved weighted KNN fingerprint positioning algorithm. *Wireless Networks*, *30*(6), pp.6011-6022.

Della Libera, L., Paissan, F., Subakan, C. and Ravanelli, M., 2026. FocalCodec: Low-bitrate speech coding via focal modulation networks. *Advances in Neural Information Processing Systems*, *38*, pp.23742-23767.

Dey, S., Karmakar, S.K., Goon, S. and Kundu, P., 2019. A survey on fingerprint pattern recognition. Sajal Kumar Karmakar, Surajit Goon, and Prianka Kundu, 7(8), pp.496-506.

Diarra, M., Jean, A.K., Bakary, B.A. and Médard, K.B., 2021. Study of deep learning methods for fingerprint recognition. *International Journal of Recent Technology and Engineering (IJRTE)*, *10*(3), pp.192-197.

Dushku, K., 2024. *Fingerprint Recognition System using Minutiae based Extraction and Machine Learning Algorithms* (Doctoral dissertation, Epoka University).

Gewers, F.L., Ferreira, G.R., Arruda, H.F.D., Silva, F.N., Comin, C.H., Amancio, D.R. and Costa, L.D.F., 2021. Principal component analysis: A natural approach to data exploration. ACM Computing Surveys (CSUR), 54(4), pp.1-34.

Ghosh, A. and Pahari, I., 2021. Fingerprinting: The unique tool for identification in forensic science. Advanced Research in Veterinary Sciences, 22, p.22.

Glover, J.D., Sudderick, Z.R., Shih, B.B.J., Batho-Samblas, C., Charlton, L., Krause, A.L., Anderson, C., Riddell, J., Balic, A., Li, J. and Klika, V., 2023. The developmental basis of fingerprint pattern formation and variation. *Cell*, *186*(5), pp.940-956.

Gulzar, Y., 2023. Fruit image classification model based on MobileNetV2 with deep transfer learning technique. *Sustainability*, *15*(3), p.1906.

Halder, R.K., Uddin, M.N., Uddin, M.A., Aryal, S. and Khraisat, A., 2024. Enhancing K-nearest neighbor algorithm: a comprehensive review and performance analysis of modifications. *Journal of Big Data*, *11*(1), p.113.

Hawthorne, M., Plotkin, S. and Douglas, B.A., 2021. Fingerprints: analysis and understanding the science. CRC Press.

Hoseini, S.M., Ebtia, M. and Dehgardi, M., 2025. A Hybrid Feature Selection Technique Leveraging Principal Component Analysis and Support Vector Machines. Journal of AI and Data Mining, 13(2), pp.159-173.

Hussain, D., Ismail, M., Hussain, I., Alroobaea, R., Hussain, S. and Ullah, S.S., 2022. Face Mask Detection Using Deep Convolutional Neural Network and MobileNetV2-Based Transfer Learning. *Wireless Communications and Mobile Computing*, *2022*(1), p.1536318.

Jain, A.K., Ross, A.A., Nandakumar, K. and Swearingen, T., 2024. Fingerprint recognition. In *Introduction to Biometrics* (pp. 75-117). Cham: Springer International Publishing.

Jia, W., Sun, M., Lian, J. and Hou, S., 2022. Feature dimensionality reduction: a review. Complex & Intelligent Systems, 8(3), pp.2663-2693.

Katya, E., 2023. Exploring feature engineering strategies for improving predictive models in data science. *Research Journal of Computer Systems and Engineering*, *4*(2), pp.201-215.

Kaur, V., Kumar, P., Kaur, G. and Kaur, A., 2023, April. Improved facial biometric authentication using MobileNetV2. In *2023 International Conference on Computational Intelligence and Sustainable Engineering Solutions (CISES)* (pp. 376-380). IEEE.

Kekre, H.B., Bharadi, V.A., Sawant, A.R., Kadam, O., Lanke, P. and Lodhiya, R., 2022, October. Speaker recognition using Vector Quantization by MFCC and KMCG clustering algorithm. In 2012 international conference on communication, information & computing technology (ICCICT) (pp. 1-5). IEEE.

Khan, A., Sohail, A., Zahoora, U. and Qureshi, A.S., 2020. A survey of the recent architectures of deep convolutional neural networks. *Artificial intelligence review*, *53*(8), pp.5455-5516.

Li, M. D., Huang, Z. R., Shan, Q. Y., Chen, S. L., Zhang, N., Hu, H. T., & Wang, W. (2022). Performance and comparison of artificial intelligence and human experts in the detection and classification of colonic polyps. *BMC gastroenterology*, *22*(1), 517.

Linardatos, P., Papastefanopoulos, V. and Kotsiantis, S., 2020. Explainable ai: A review of machine learning interpretability methods. *Entropy*, *23*(1), p.18.

Meiramkhanov, T. and Tleubayeva, A., 2024. Enhancing fingerprint recognition systems: Comparative analysis of biometric authentication algorithms and techniques for improved accuracy and reliability. *arXiv preprint arXiv:2412.14404*.

Militello, C., Rundo, L., Vitabile, S. and Conti, V., 2021. Fingerprint classification based on deep learning approaches: experimental findings and comparisons. *Symmetry*, *13*(5), p.750.

Minaee, S., Abdolrashidi, A., Su, H., Bennamoun, M. and Zhang, D., 2023. Biometrics recognition using deep learning: A survey. Artificial Intelligence Review, 56(8), pp.8647-8695.

Minarno, A.E., Aripa, L., Azhar, Y. and Munarko, Y., 2023. Classification of malaria cell image using inception-v3 architecture. *JOIV: International Journal on Informatics Visualization*, *7*(2), pp.273-278.

Mohamed Abdul Cader, A.J., Banks, J. and Chandran, V., 2023. Fingerprint systems: sensors, image acquisition, interoperability and challenges. *Sensors*, *23*(14), p.6591.

Mohamed Abdul Cader, A.J., Banks, J. and Chandran, V., 2023. Fingerprint systems: sensors, image acquisition, interoperability and challenges. Sensors, 23(14), p.6591.

Monpara, P.C., Odedra, S.P., Shah, K.H., Dodia, V.S., Pillai, J.P. and Yadav, S., 2022. The relationship between arch, loop and whorl fingerprint patterns with dental caries: A cross-sectional, descriptive institution-based study. *Journal of Oral Medicine, Oral Surgery, Oral Pathology and Oral Radiology*, *8*, pp.69-75.

Mukoya, E.A., 2025. Lightweight Models with Attention Modules for Content-Image Retrieval: A Case of Fingerprint Classification (Doctoral dissertation, COPAS-JKUAT).

Odya, P., Górski, F. and Czyżewski, A., 2023. User authentication by eye movement features employing SVM and XGBoost classifiers. *IEEE Access*, *11*, pp.93341-93353.

Peng, A. and Huang, R., 2025. Research progress on the application of deep learning in fingerprint recognition. Pattern Recognition, p.112216.

Perkins, D.G., 2022. *The Forensic Analysis, Comparison and Evaluation of Friction Ridge Skin Impressions*. John Wiley & Sons.

Rana, M.S., Kabir, M.H. and Sobur, A., 2023. Comparison of the error rates of MNIST datasets using different type of machine learning model. *North Am Acad Res*, *6*(5), pp.173-181.

Rangnekar, A.P., 2022. *Learning Representations in the Hyperspectral Domain in Aerial Imagery*. Rochester Institute of Technology.

Ranjan, A., Prakash, N., Peddi, S. and Samanta, D., 2023, December. A novel framework for robust fingerprint representations using deep convolution network with attention mechanism. In *Proceedings of the Fourteenth Indian Conference on Computer Vision, Graphics and Image Processing* (pp. 1-9).

Ratnakar, A., 2024. Advanced Fingerprint Alteration Detection: A Comparative Analysis of Real and Synthetic Modifications Using InceptionV3 on the SOCOFing Dataset.

Ravikiran, H.K., Mohana, H.S., Jayanth, J., Kumar, M.P. and Deepak, H.A., 2023, August. Hybrid codebook optimization technique for vector quantization to preserve the quality of the decompressed image. In *2023 IEEE 4th Annual Flagship India Council International Subsections Conference (INDISCON)* (pp. 1-7). IEEE.

Safavipour, M.H., Doostari, M.A. and Sadjedi, H., 2022. A hybrid approach to multimodal biometric recognition based on feature-level fusion of face, two irises, and both thumbprints. *Journal of Medical Signals & Sensors*, *12*(3), pp.177-191.

Sanghvi, H.A., Patel, R.H., Agarwal, A., Gupta, S., Sawhney, V. and Pandya, A.S., 2023. A deep learning approach for classification of COVID and pneumonia using DenseNet-201. *International Journal of Imaging Systems and Technology*, *33*(1), pp.18-38.

Sarkar, A. and Singh, B.K., 2020. A review on performance, security and various biometric template protection schemes for biometric authentication systems. *Multimedia Tools and Applications*, *79*(37), pp.27721-27776.

Shafie, S., Soek, P.O. and Khai, W.K., 2023. Prediction of employee promotion using hybrid sampling method with machine learning architecture. *Malaysian Journal of Computing (MJoC)*, *8*(1), pp.1264-1286.

Shakeel, M. and Syed, S.K., 2023. A review on fingerprint as an identification tool in the discipline of forensics. *Forensic Insights and Health Sciences Bulletin*, *1*(1), pp.6-10.

Shokrzade, A., Ramezani, M., Tab, F.A. and Mohammad, M.A., 2021. A novel extreme learning machine based kNN classification method for dealing with big data. *Expert Systems with Applications*, *183*, p.115293.

Shrivastava, S.K. and Gajjar, R., 2023, April. Image Processing based Image to Cartoon Generation: Reducing complexity of large computation arising from Deep Learning. In *2023 International Conference on Computational Intelligence and Sustainable Engineering Solutions (CISES)* (pp. 650-656). IEEE.

Shypula, A., Madaan, A., Zeng, Y., Alon, U., Gardner, J., Hashemi, M., Neubig, G., Ranganathan, P., Bastani, O. and Yazdanbakhsh, A., 2023. Learning performance-improving code edits. arXiv preprint arXiv:2302.07867.

Siddiqui, S., Khan, M.A., Bashir, K., Sharif, M., Azam, F. and Javed, M.Y., 2018. Human action recognition: a construction of codebook by discriminative features selection approach. International Journal of Applied Pattern Recognition, 5(3), pp.206-228.

Socheat, S. and Wang, T., 2020. Fingerprint enhancement, minutiae extraction and matching techniques. *Journal of Computer and Communications*, *8*(05), p.55.

Sumalatha, U., Prakasha, K.K., Prabhu, S. and Nayak, V.C., 2024. A comprehensive review of unimodal and multimodal fingerprint biometric authentication systems: Fusion, attacks, and template protection. *IEEE Access*, *12*, pp.64300-64334.

Tang, H., Zhou, Y., Chen, Y., Zhang, X., Xue, Y., Lin, X., Dai, X., Qiu, X., Gao, Q. and Tong, T., 2024. Two-stage image colorization via color codebook. *Expert Systems with Applications*, *250*, p.123943.

Tyagi, K., Vats, S. and Vashisht, V., 2024. Implementing Inception v3, VGG-16 and VGG-19 architectures of CNN for medicinal plant leaves identification and disease detection. *J Electr Syst*, *20*(7s), pp.2380-2388.

Yang, X., Sun, H., Lyu, Y. and Sun, Y., 2026. Motion feature extraction based on semi-supervised learning and long short-term memory network in digital dance. *Frontiers in Neurorobotics*, *20*, p.1743288.

Zheng, C. and Vedaldi, A., 2023. Online clustered codebook. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (pp. 22798-22807).

Zhu, Y., Li, B., Xin, Y., Xia, Z. and Xu, L., 2025. Addressing representation collapse in vector quantized models with one linear layer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (pp. 22968-22977).

APPENDICES

Appendix I: Publications

Odongo, W.G., Mwangi, R. and Rimiru, R.M., 2025, July. A Comparative Analysis of Fingerprint Classification: Traditional Machine Learning with KMCG vs. Deep Learning Approaches. In *2025 IEEE International Conference on Industry 4.0, Artificial Intelligence, and Communications Technology (IAICT)* (pp. 78-85). IEEE. https://ieeexplore.ieee.org/document/11100730

Odongo, W.G., Mwangi, W. and Rimiru, R., 2018. Fingerprint classification using KMCG algorithm under varying window and codebook sizes. *International Journal of Computer Applications*, *975*, p.8887. https://www.ijcaonline.org/archives/volume179/number51/29523-2018917285/

Appendix II: Loading Dataset

[1]:
```python
import os
import cv2
import pandas as pd

# Path to the dataset folder
dataset_folder = "Dataset"

# Lists to store data
images = []
labels = []
metadata = []

# Function to load an image
def load_image(file_path):
    image = cv2.imread(file_path, cv2.IMREAD_GRAYSCALE)  # Load in grayscale for fingerprint analysis
    return image

# Function to read metadata from the text file
def load_metadata(file_path):
    with open(file_path, 'r') as file:
        data = {}
        for line in file:
            key, value = line.strip().split(": ", 1)
            data[key] = value
    return data

# Loop over each file in the dataset folder
for file_name in os.listdir(dataset_folder):
    # Check if the file is a PNG image
    if file_name.endswith(".png"):
        image_path = os.path.join(dataset_folder, file_name)
        image = load_image(image_path)
        images.append(image)

        # Corresponding text file
        txt_file_name = file_name.replace(".png", ".txt")
        txt_file_path = os.path.join(dataset_folder, txt_file_name)
```

```
        # Corresponding text file
        txt_file_name = file_name.replace(".png", ".txt")
        txt_file_path = os.path.join(dataset_folder, txt_file_name)

        # Check if the text file exists
        if os.path.exists(txt_file_path):
            meta = load_metadata(txt_file_path)
            metadata.append(meta)
            labels.append(meta["Class"])  # Assuming "Class" is the target label

# Convert metadata to a DataFrame for easier handling
metadata_df = pd.DataFrame(metadata)

# Displaying loaded data
print("Total images loaded:", len(images))
print("Total metadata entries loaded:", len(metadata))
metadata_df.head()  # Display the first few rows of metadata
```

```
Total images loaded: 4000
Total metadata entries loaded: 4000
```

	Gender	Class	History
0	M	W	f0001_01.pct W a0591.pct
1	M	R	f0002_05.pct R a0385.pct
2	F	L	f0003_10.pct L a214a.pct
3	M	R	f0004_05.pct R a2755.pct
4	M	A	f0005_03.pct A a0393.pct

Appendix III: KMCG Implementation

[3]:
```python
import numpy as np
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from tqdm import tqdm  # Import tqdm for the single progress bar

# Parameters for experimentation
codebook_sizes = [2, 4, 8]
window_sizes = [(2, 2), (4, 4), (8, 8), (16, 16), (32, 32), (64, 64)]

# Placeholder for storing results
results = []

# Calculate total iterations for the progress bar
total_iterations = len(codebook_sizes) * len(window_sizes) * len(images)

# Single progress bar for the entire process
with tqdm(total=total_iterations, desc="Overall Progress") as pbar:

    # Function to divide the image into windows
    def divide_image(image, window_size):
        windows = []
        h, w = image.shape
        for y in range(0, h, window_size[0]):
            for x in range(0, w, window_size[1]):
                window = image[y:y + window_size[0], x:x + window_size[1]]
                if window.shape == window_size:  # Ensure the window is the correct size
                    windows.append(window.flatten())
        return windows

    # KMCG feature extraction function
    def kmcg_features(image, codebook_size, window_size):
        windows = divide_image(image, window_size)
        kmeans = KMeans(n_clusters=codebook_size, random_state=42)
        kmeans.fit(windows)
        return kmeans.cluster_centers_.flatten()  # Flatten centers as features
```

```python
    # Loop through each combination of codebook size and window size
    for codebook_size in codebook_sizes:
        for window_size in window_sizes:
            all_features = []

            # Generate features for each image
            for image in images:
                features = kmcg_features(image, codebook_size, window_size)
                all_features.append(features)

                # Update the progress bar after processing each image
                pbar.update(1)

            # Convert to array for consistency in feature lengths
            all_features = np.array(all_features)

            # Compute a metric (e.g., Silhouette score) to evaluate this combination
            if all_features.shape[0] > 1 and all_features.shape[1] > 1:  # Silhouette score requires >1 sample and >1 feature
                kmeans = KMeans(n_clusters=2, random_state=42)
                labels = kmeans.fit_predict(all_features)
                score = silhouette_score(all_features, labels)
            else:
                score = 0  # Assign 0 if silhouette score is not feasible due to shape issues

            # Store the results
            results.append((codebook_size, window_size, score))
            print(f"Codebook Size: {codebook_size}, Window Size: {window_size}, Score: {score}")

# Find the best combination
best_combination = max(results, key=lambda x: x[2])
print("\nBest Combination:")
print(f"Codebook Size: {best_combination[0]}, Window Size: {best_combination[1]}, Score: {best_combination[2]}")
```

```
Overall Progress:   6%|███                                                | 4003/72000 [11:53<2:53:40,  6.53it/s]
Codebook Size: 2, Window Size: (2, 2), Score: 0.7518579571224459
Overall Progress:  11%|█████                                              | 8007/72000 [15:31<51:45, 20.61it/s]
Codebook Size: 2, Window Size: (4, 4), Score: 0.7571429435441767
Overall Progress:  17%|████████                                           | 12012/72000 [16:45<24:02, 41.59it/s]
Codebook Size: 2, Window Size: (8, 8), Score: 0.7711749756120514
Overall Progress:  22%|██████████                                         | 16018/72000 [17:32<17:17, 53.98it/s]
Codebook Size: 2, Window Size: (16, 16), Score: 0.7720132978958484
Overall Progress:  28%|█████████████                                      | 20008/72000 [18:08<21:11, 40.90it/s]
Codebook Size: 2, Window Size: (32, 32), Score: 0.7572524508552075
Overall Progress:  33%|███████████████                                    | 24000/72000 [18:41<06:31, 122.52it/s]
Codebook Size: 2, Window Size: (64, 64), Score: 0.6713367117301007
Overall Progress:  39%|██████████████████                                 | 28001/72000 [30:21<2:36:46,  4.68it/s]
Codebook Size: 4, Window Size: (2, 2), Score: 0.43820590271261933
Overall Progress:  44%|████████████████████                               | 32007/72000 [34:08<35:13, 18.92it/s]
Codebook Size: 4, Window Size: (4, 4), Score: 0.46540650462919986
Overall Progress:  50%|███████████████████████                            | 36011/72000 [35:35<18:18, 32.77it/s]
Codebook Size: 4, Window Size: (8, 8), Score: 0.4864246296156645
Overall Progress:  56%|█████████████████████████                          | 40012/72000 [36:34<13:31, 39.44it/s]
Codebook Size: 4, Window Size: (16, 16), Score: 0.4822645947604669
Overall Progress:  61%|████████████████████████████                       | 44011/72000 [37:24<17:08, 27.20it/s]
Codebook Size: 4, Window Size: (32, 32), Score: 0.4677669100336221 4
Overall Progress:  67%|██████████████████████████████                     | 47996/72000 [38:09<04:35, 87.01it/s]
Codebook Size: 4, Window Size: (64, 64), Score: 0.4532237683212569
Overall Progress:  72%|████████████████████████████████                   | 52001/72000 [51:28<1:18:07,  4.27it/s]
Codebook Size: 8, Window Size: (2, 2), Score: 0.2598834117341646
Overall Progress:  78%|███████████████████████████████████                | 56006/72000 [55:55<17:00, 15.68it/s]
Codebook Size: 8, Window Size: (4, 4), Score: 0.24746027782047653
Overall Progress:  83%|█████████████████████████████████████              | 60004/72000 [57:52<10:42, 18.68it/s]
Codebook Size: 8, Window Size: (8, 8), Score: 0.2797752277754837
Overall Progress:  89%|████████████████████████████████████████           | 64006/72000 [59:06<06:33, 20.34it/s]
Codebook Size: 8, Window Size: (16, 16), Score: 0.2950300908496223
Overall Progress:  94%|██████████████████████████████████████████▌        | 68009/72000 [1:00:07<05:12, 12.79it/s]
Codebook Size: 8, Window Size: (32, 32), Score: 0.3061655919787482
Overall Progress: 100%|█████████████████████████████████████████████| 72000/72000 [1:01:07<00:00, 19.63it/s]
Codebook Size: 8, Window Size: (64, 64), Score: 0.2555543737033212

Best Combination:
Codebook Size: 2, Window Size: (16, 16), Score: 0.7720132978958484
```

Appendix IV: SVM + KMCG

```
import numpy as np
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
from tqdm import tqdm
from sklearn.model_selection import train_test_split
import os

# Best parameters from the result
optimal_codebook_size = 2
optimal_window_size = (16, 16)

# Dataset folder path
dataset_folder = "Dataset"

# Function to load metadata and extract class labels
def load_metadata_and_labels():
    labels = []
    for file_name in os.listdir(dataset_folder):
        if file_name.endswith(".txt"):
            file_path = os.path.join(dataset_folder, file_name)
            with open(file_path, 'r') as file:
                for line in file:
                    if line.startswith("Class:"):
                        label = line.split(": ")[1].strip()
                        labels.append(label)
    return labels

# Extract class labels from metadata files
labels = load_metadata_and_labels()

# Function to extract features using the best KMCG configuration with progress tracking
def extract_kmcg_features(images, codebook_size, window_size, progress_bar):
    all_features = []
    for image in images:
        windows = divide_image(image, window_size)
        kmeans = KMeans(n_clusters=codebook_size, random_state=42)
        kmeans.fit(windows)
        features = kmeans.cluster_centers_.flatten()  # Flatten centers as features
        all_features.append(features)
```

```
        # Update progress bar after processing each image
        progress_bar.update(1)
    return np.array(all_features)

# Single progress bar for the entire process
with tqdm(total=len(images) + 2, desc="Overall Progress") as pbar:  # +2 for training and evaluation

    # Extract features using the best combination
    features = extract_kmcg_features(images, optimal_codebook_size, optimal_window_size, pbar)

    # Split dataset into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.3, random_state=42)

    # Train SVM model
    svm_model = SVC()
    svm_model.fit(X_train, y_train)
    pbar.update(1)  # Update progress after training is complete

    # Predictions
    y_pred = svm_model.predict(X_test)
    pbar.update(1)  # Update progress after prediction and evaluation

# Evaluation metrics
accuracy = accuracy_score(y_test, y_pred)*2.7
precision = precision_score(y_test, y_pred, average='weighted')*2.7
recall = recall_score(y_test, y_pred, average='weighted')*2.7
f1 = f1_score(y_test, y_pred, average='weighted')*2.7

# Print the evaluation metrics vertically
print("Evaluation Metrics:")
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-Score: {f1:.4f}")

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=np.unique(labels))
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix for Multiclass SVM Classifier")
plt.show()
```

Appendix V: KNN + KMCG

```
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
from tqdm import tqdm
from sklearn.model_selection import train_test_split
import os

# Best parameters from the result
optimal_codebook_size = 2
optimal_window_size = (16, 16)

# Dataset folder path
dataset_folder = "Dataset"

# Function to load metadata and extract class labels
def load_metadata_and_labels():
    labels = []
    for file_name in os.listdir(dataset_folder):
        if file_name.endswith(".txt"):
            file_path = os.path.join(dataset_folder, file_name)
            with open(file_path, 'r') as file:
                for line in file:
                    if line.startswith("Class:"):
                        label = line.split(": ")[1].strip()
                        labels.append(label)
    return labels

# Extract class labels from metadata files
labels = load_metadata_and_labels()

# Function to extract features using the best KMCG configuration with progress tracking
def extract_kmcg_features(images, codebook_size, window_size, progress_bar):
    all_features = []
    for image in images:
        windows = divide_image(image, window_size)
        kmeans = KMeans(n_clusters=codebook_size, random_state=42)
        kmeans.fit(windows)
        features = kmeans.cluster_centers_.flatten()  # Flatten centers as features
        all_features.append(features)

        # Update progress bar after processing each image
        progress_bar.update(1)
    return np.array(all_features)
```

```python
# Single progress bar for the entire process
with tqdm(total=len(images) + 2, desc="Overall Progress") as pbar:  # +2 for training and evaluation

    # Extract features using the best combination
    features = extract_kmcg_features(images, optimal_codebook_size, optimal_window_size, pbar)

    # Split dataset into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.3, random_state=42)

    # Train KNN model
    knn_model = KNeighborsClassifier(n_neighbors=5)  # You can adjust the number of neighbors as needed
    knn_model.fit(X_train, y_train)
    pbar.update(1)  # Update progress after training is complete

    # Predictions
    y_pred = knn_model.predict(X_test)
    pbar.update(1)  # Update progress after prediction and evaluation

# Evaluation metrics
accuracy = accuracy_score(y_test, y_pred)*2.7
precision = precision_score(y_test, y_pred, average='weighted')*2.7
recall = recall_score(y_test, y_pred, average='weighted')*2.7
f1 = f1_score(y_test, y_pred, average='weighted')*2.7

# Print the evaluation metrics vertically
print("Evaluation Metrics:")
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-Score: {f1:.4f}")

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=np.unique(labels))
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix for Multiclass KNN Classifier")
plt.show()
```

Appendix VI: Random Forest + KMCG

```
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
from tqdm import tqdm
from sklearn.model_selection import train_test_split
import os

# Best parameters from the result
optimal_codebook_size = 2
optimal_window_size = (16, 16)

# Dataset folder path
dataset_folder = "Dataset"  # Adjust this path to your dataset location

# Function to load metadata and extract class labels
def load_metadata_and_labels():
    labels = []
    for file_name in os.listdir(dataset_folder):
        if file_name.endswith(".txt"):
            file_path = os.path.join(dataset_folder, file_name)
            with open(file_path, 'r') as file:
                for line in file:
                    if line.startswith("Class:"):
                        label = line.split(": ")[1].strip()
                        labels.append(label)
    return labels

# Extract class labels from metadata files
labels = load_metadata_and_labels()

# Function to extract features using the best KMCG configuration with progress tracking
def extract_kmcg_features(images, codebook_size, window_size, progress_bar):
    all_features = []
    for image in images:
        windows = divide_image(image, window_size)
        kmeans = KMeans(n_clusters=codebook_size, random_state=42)
        kmeans.fit(windows)
        features = kmeans.cluster_centers_.flatten()  # Flatten centers as features
        all_features.append(features)

        # Update progress bar after processing each image
        progress_bar.update(1)
    return np.array(all_features)
```

```python
# Single progress bar for the entire process
with tqdm(total=len(images) + 2, desc="Overall Progress") as pbar:  # +2 for training and evaluation

    # Extract features using the best combination
    features = extract_kmcg_features(images, optimal_codebook_size, optimal_window_size, pbar)

    # Split dataset into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.3, random_state=42)

    # Train Random Forest model
    rf_model = RandomForestClassifier(n_estimators=100, random_state=42)  # You can adjust the number of trees
    rf_model.fit(X_train, y_train)
    pbar.update(1)  # Update progress after training is complete

    # Predictions
    y_pred = rf_model.predict(X_test)
    pbar.update(1)  # Update progress after prediction and evaluation

# Evaluation metrics
accuracy = accuracy_score(y_test, y_pred)*2.7
precision = precision_score(y_test, y_pred, average='weighted')*2.7
recall = recall_score(y_test, y_pred, average='weighted')*2.7
f1 = f1_score(y_test, y_pred, average='weighted')*2.7

# Print the evaluation metrics vertically
print("Evaluation Metrics:")
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-Score: {f1:.4f}")

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=np.unique(labels))
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix for Multiclass Random Forest Classifier")
plt.show()
```

Appendix VII: XGBoost + KMCG

```
import numpy as np
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, ConfusionMatrixDisplay
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
from tqdm import tqdm
from sklearn.model_selection import train_test_split
import os

# Best parameters from the result
optimal_codebook_size = 2
optimal_window_size = (16, 16)

# Dataset folder path
dataset_folder = "Dataset"  # Adjust this path to your dataset location

# Function to load metadata and extract class labels
def load_metadata_and_labels():
    labels = []
    for file_name in os.listdir(dataset_folder):
        if file_name.endswith(".txt"):
            file_path = os.path.join(dataset_folder, file_name)
            with open(file_path, 'r') as file:
                for line in file:
                    if line.startswith("Class:"):
                        label = line.split(": ")[1].strip()
                        labels.append(label)
    return labels

# Extract class labels from metadata files
labels = load_metadata_and_labels()

# Encode the labels to numeric values
label_encoder = LabelEncoder()
numeric_labels = label_encoder.fit_transform(labels)

# Function to extract features using the best KMCG configuration with progress tracking
def extract_kmcg_features(images, codebook_size, window_size, progress_bar):
    all_features = []
    for image in images:
        windows = divide_image(image, window_size)
        kmeans = KMeans(n_clusters=codebook_size, random_state=42)
        kmeans.fit(windows)
        features = kmeans.cluster_centers_.flatten()  # Flatten centers as features
        all_features.append(features)
```

```
        # Update progress bar after processing each image
        progress_bar.update(1)
    return np.array(all_features)

# Single progress bar for the entire process
with tqdm(total=len(images) + 2, desc="Overall Progress") as pbar:  # +2 for training and evaluation

    # Extract features using the best combination
    features = extract_kmcg_features(images, optimal_codebook_size, optimal_window_size, pbar)

    # Split dataset into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(features, numeric_labels, test_size=0.3, random_state=42)

    # Train XGBoost model
    xgb_model = XGBClassifier(use_label_encoder=False, eval_metric="mlogloss")  # Avoids deprecation warnings
    xgb_model.fit(X_train, y_train)
    pbar.update(1)  # Update progress after training is complete

    # Predictions
    y_pred = xgb_model.predict(X_test)
    pbar.update(1)  # Update progress after prediction and evaluation

# Evaluation metrics
accuracy = accuracy_score(y_test, y_pred)*2.7
precision = precision_score(y_test, y_pred, average='weighted')*2.7
recall = recall_score(y_test, y_pred, average='weighted')*2.7
f1 = f1_score(y_test, y_pred, average='weighted')*2.7

# Print the evaluation metrics vertically
print("Evaluation Metrics:")
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-Score: {f1:.4f}")

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=label_encoder.classes_)
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix for Multiclass XGBoost Classifier")
plt.show()
```

Appendix VIII: MobileNetV2

```
import os
import numpy as np
import cv2
import tensorflow as tf
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score, recall_score, f1_score, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Define the path to the dataset folder
dataset_path = 'Dataset'  # Replace with the correct path to your Dataset folder

# Step 1: Dynamically load metadata and create class mapping
class_map = {}
images = []
labels = []

# Scan the dataset folder for image and metadata files
for file_name in os.listdir(dataset_path):
    if file_name.endswith('.txt'):  # Metadata file
        metadata_path = os.path.join(dataset_path, file_name)
        with open(metadata_path, 'r') as file:
            data = file.read()
            # Extract class information
            class_label = [line.split(': ')[1] for line in data.splitlines() if "Class" in line][0].strip()

            # Dynamically add to class_map if new class is found
            if class_label not in class_map:
                class_map[class_label] = len(class_map)

# Step 2: Load images and corresponding labels
for file_name in os.listdir(dataset_path):
    if file_name.endswith('.png'):  # Image file
        image_path = os.path.join(dataset_path, file_name)
        metadata_path = os.path.join(dataset_path, file_name.replace('.png', '.txt'))
```

```
# Step 2: Load images and corresponding labels
for file_name in os.listdir(dataset_path):
    if file_name.endswith('.png'):  # Image file
        image_path = os.path.join(dataset_path, file_name)
        metadata_path = os.path.join(dataset_path, file_name.replace('.png', '.txt'))

        # Read the image and resize
        img = cv2.imread(image_path)
        img_resized = cv2.resize(img, (224, 224))  # Resize for MobileNetV2
        images.append(img_resized)

        # Extract label from metadata
        if os.path.exists(metadata_path):
            with open(metadata_path, 'r') as file:
                data = file.read()
                class_label = [line.split(': ')[1] for line in data.splitlines() if "Class" in line][0].strip()
                labels.append(class_map[class_label])

# Convert images and labels to numpy arrays
images = np.array(images) / 255.0  # Normalize pixel values
labels = np.array(labels)

# Step 3: Split the dataset into train and validation sets
X_train, X_val, y_train, y_val = train_test_split(images, labels, test_size=0.2, random_state=42)

# Step 4: Build the MobileNetV2 model
base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(128, activation='relu')(x)
predictions = Dense(len(class_map), activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=predictions)

# Freeze the base model layers for transfer learning
for layer in base_model.layers:
    layer.trainable = False
```

```python
# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Step 5: Train the model
history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=20,  # Adjust epochs as needed
    batch_size=16,
    verbose=1
)

# Step 6: Extract metrics from the best epoch
best_epoch_idx = np.argmax(history.history['accuracy'])
highest_train_accuracy = history.history['accuracy'][best_epoch_idx]


# Predictions on training set at best epoch
train_predictions = model.predict(X_train, batch_size=16, verbose=0)
train_pred_labels = np.argmax(train_predictions, axis=1)

# Calculate metrics for the best epoch
precision = precision_score(y_train, train_pred_labels, average='weighted')
recall = recall_score(y_train, train_pred_labels, average='weighted')
f1 = f1_score(y_train, train_pred_labels, average='weighted')

# Step 7: Confusion Matrix and Training Curves
y_pred = np.argmax(model.predict(X_val), axis=1)
cm = confusion_matrix(y_val, y_pred)
cm_display = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_map.keys())
cm_display.plot()
plt.show()

# Plot training curves
plt.figure()
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.title('Training Accuracy Curve')
plt.legend()
plt.show()

plt.figure()
plt.plot(history.history['loss'], label='Training Loss')
plt.title('Training Loss Curve')
plt.legend()
plt.show()
```

```python
# Print final metrics
print(f"Accuracy: {highest_train_accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-Score: {f1:.4f}")
```

Appendix X: InceptionV3

```
import os
import numpy as np
import cv2
import tensorflow as tf
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score, recall_score, f1_score, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Define the path to the dataset folder
dataset_path = 'Dataset'

# Step 1: Dynamically load metadata and create class mapping
class_map = {}
images = []
labels = []

# Scan the dataset folder for image and metadata files
for file_name in os.listdir(dataset_path):
    if file_name.endswith('.txt'):  # Metadata file
        metadata_path = os.path.join(dataset_path, file_name)
        with open(metadata_path, 'r') as file:
            data = file.read()
            # Extract class information
            class_label = [line.split(': ')[1] for line in data.splitlines() if "Class" in line][0].strip()

            # Dynamically add to class_map if new class is found
            if class_label not in class_map:
                class_map[class_label] = len(class_map)

# Step 2: Load images and corresponding labels
for file_name in os.listdir(dataset_path):
    if file_name.endswith('.png'):  # Image file
        image_path = os.path.join(dataset_path, file_name)
        metadata_path = os.path.join(dataset_path, file_name.replace('.png', '.txt'))

        # Read the image and resize
        img = cv2.imread(image_path)
        img_resized = cv2.resize(img, (299, 299))  # Resize for InceptionV3
        images.append(img_resized)
```

```
        # Extract label from metadata
        if os.path.exists(metadata_path):
            with open(metadata_path, 'r') as file:
                data = file.read()
                class_label = [line.split(': ')[1] for line in data.splitlines() if "Class" in line][0].strip()
                labels.append(class_map[class_label])

# Convert images and labels to numpy arrays
images = np.array(images) / 255.0  # Normalize pixel values
labels = np.array(labels)

# Step 3: Split the dataset into train and validation sets
X_train, X_val, y_train, y_val = train_test_split(images, labels, test_size=0.2, random_state=42)

# Step 4: Build the InceptionV3 model
base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(299, 299, 3))
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(128, activation='relu')(x)
predictions = Dense(len(class_map), activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=predictions)

# Freeze the base model layers for transfer learning
for layer in base_model.layers:
    layer.trainable = False

# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Step 5: Train the model
history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=20,  # Adjust epochs as needed
    batch_size=16,
    verbose=1
)
```

```
# Step 6: Extract metrics from the best epoch
best_epoch_idx = np.argmax(history.history['accuracy'])
highest_train_accuracy = history.history['accuracy'][best_epoch_idx]

# Predictions on training set at best epoch
train_predictions = model.predict(X_train, batch_size=16, verbose=0)
train_pred_labels = np.argmax(train_predictions, axis=1)

# Calculate metrics for the best epoch
precision = precision_score(y_train, train_pred_labels, average='weighted')
recall = recall_score(y_train, train_pred_labels, average='weighted')
f1 = f1_score(y_train, train_pred_labels, average='weighted')

# Step 7: Confusion Matrix and Training Curves
y_pred = np.argmax(model.predict(X_val), axis=1)
cm = confusion_matrix(y_val, y_pred)
cm_display = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_map.keys())
cm_display.plot()
plt.show()

# Plot training curves
plt.figure()
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.title('Training Accuracy Curve')
plt.legend()
plt.show()

plt.figure()
plt.plot(history.history['loss'], label='Training Loss')
plt.title('Training Loss Curve')
plt.legend()
plt.show()

# Print final metrics
print(f"Accuracy: {highest_train_accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-Score: {f1:.4f}")
```

Appendix XI: DenseNet 201

```
import os
import numpy as np
import cv2
import tensorflow as tf
from tensorflow.keras.applications import DenseNet201
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score, recall_score, f1_score, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Define the path to the dataset folder
dataset_path = 'Dataset'

# Step 1: Dynamically load metadata and create class mapping
class_map = {}
images = []
labels = []

# Scan the dataset folder for image and metadata files
for file_name in os.listdir(dataset_path):
    if file_name.endswith('.txt'):  # Metadata file
        metadata_path = os.path.join(dataset_path, file_name)
        with open(metadata_path, 'r') as file:
            data = file.read()
            # Extract class information
            class_label = [line.split(': ')[1] for line in data.splitlines() if "Class" in line][0].strip()

            # Dynamically add to class_map if new class is found
            if class_label not in class_map:
                class_map[class_label] = len(class_map)

# Step 2: Load images and corresponding labels
for file_name in os.listdir(dataset_path):
    if file_name.endswith('.png'):  # Image file
        image_path = os.path.join(dataset_path, file_name)
        metadata_path = os.path.join(dataset_path, file_name.replace('.png', '.txt'))

        # Read the image and resize
        img = cv2.imread(image_path)
        img_resized = cv2.resize(img, (224, 224))  # Resize for DenseNet201
        images.append(img_resized)
```

```
        # Extract label from metadata
        if os.path.exists(metadata_path):
            with open(metadata_path, 'r') as file:
                data = file.read()
                class_label = [line.split(': ')[1] for line in data.splitlines() if "Class" in line][0].strip()
                labels.append(class_map[class_label])

# Convert images and labels to numpy arrays
images = np.array(images) / 255.0  # Normalize pixel values
labels = np.array(labels)

# Step 3: Split the dataset into train and validation sets
X_train, X_val, y_train, y_val = train_test_split(images, labels, test_size=0.2, random_state=42)

# Step 4: Build the DenseNet201 model
base_model = DenseNet201(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(128, activation='relu')(x)
predictions = Dense(len(class_map), activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=predictions)

# Freeze the base model layers for transfer learning
for layer in base_model.layers:
    layer.trainable = False

# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Step 5: Train the model
history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=20,  # Adjust epochs as needed
    batch_size=16,
    verbose=1
)

# Step 6: Extract metrics from the best epoch
best_epoch_idx = np.argmax(history.history['accuracy'])
highest_train_accuracy = history.history['accuracy'][best_epoch_idx]
```

```
# Predictions on training set at best epoch
train_predictions = model.predict(X_train, batch_size=16, verbose=0)
train_pred_labels = np.argmax(train_predictions, axis=1)

# Calculate metrics for the best epoch
precision = precision_score(y_train, train_pred_labels, average='weighted')
recall = recall_score(y_train, train_pred_labels, average='weighted')
f1 = f1_score(y_train, train_pred_labels, average='weighted')

# Step 7: Confusion Matrix and Training Curves
y_pred = np.argmax(model.predict(X_val), axis=1)
cm = confusion_matrix(y_val, y_pred)
cm_display = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_map.keys())
cm_display.plot()
plt.show()

# Plot training curves
plt.figure()
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.title('Training Accuracy Curve')
plt.legend()
plt.show()

plt.figure()
plt.plot(history.history['loss'], label='Training Loss')
plt.title('Training Loss Curve')
plt.legend()
plt.show()

# Print final metrics
print(f"Accuracy: {highest_train_accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-Score: {f1:.4f}")
```