FINGERPRINT IDENTIFICATION WITH FUSION OF GABOR AND MINUTIAE FEATURES USING BPNN CLASSIFIER

A thesis submitted in partial fulfillment of the
requirement for the award of the degree of

Bachelor of Technology
in
Electronics & Communication Engineering

by
J.LAKSHMI BHAVANI(Y20EC075)
K.KALYAN(Y20EC079)
L.JAGADEESWARA RAO(Y20EC109)

Under the guidance of
Dr.P.P.S.Subhashini M.Tech., Ph.D
Associate Professor in ECE

Department of Electronics & Communication Engineering
R.V.R. & J.C. COLLEGE OF ENGINEERING
(Autonomous)
Approved by AICTE :: Affiliated to Acharya Nagarjuna University
Chowdavaram, Guntur - 522019, Andhra Pradesh, India

2024

Department of Electronics & Communication Engineering

CERTIFICATE

This is to certify that the thesis report entitled **"FINGERPRINT IDENTIFICATION WITH FUSION OF GABOR AND MINUTIAE FEATURES USING BPNN CLASSIFIER"** that is being submitted by **"J.LAKSHMI BHAVANI (Y20EC075), K.KALYAN (Y20EC079), L.JAGADEESWARA RAO (Y20EC109)"** in partial fulfillment for the award of the Degree of **Bachelor of Technology** in **Electronics & Communication Engineering** to R.V.R J.C College of Engineering (Autonomous) is a record of bonafide work carried out by them under my guidance and supervision. The results embodied in this project report have not been submitted to any other University or Institute for the award of any degree or diploma.

Date:

Signature of Guide

Dr.P.P.S.Subhashini M.Tech., Ph.D

Associate Professor in ECE

Signature of HOD

Dr.T.Ranga Babu M.Tech., Ph.D

Professor & Head

Abstract

Fingerprint identification is a biometric technology that aims to automatically identify or verify the identification of individuals based on their unique fingerprints. Fingerprint identification systems are widely used in various applications, including forensics and biometric identification for security purposes.The fingerprint identification system consists of two modules Training and Testing

Feature extraction is important in fingerprint identification. It uses methods like Gabor and Minutiae. Gabor filters show texture and direction, while minutiae focus on points. Combining Gabor with minutiae improves matching accuracy, the systems become very good at matching fingerprints accurately, even in tough situations.. Principal Component Analysis (PCA) selects features by prioritizing high kurtosis values, enhancing system reliability.

The Back Propagation Neural Network (BPNN) is often used in machine learning to group things. It uses a Sigmoid function to make decisions. Another tool, the Levenberg-Marquardt (LM) algorithm, helps solve tough math problems, especially in training neural networks. This project aims to make a smart system that finds fingerprints automatically. It combines Gabor and minutiae features and uses a BPNN classifier. The goal is to identify fingerprints accurately.

Table of contents

List of Tables

List of Figures

1

INTRODUCTION

1.1 Pattern Recognition

Pattern recognition is the process of identifying and interpreting regularities or patterns within data. It involves recognizing meaningful structures, relationships, or features in raw data and making sense of them to understand or categorize the underlying information. Pattern recognition is a fundamental aspect of human cognition, and it plays a crucial role in various fields, including computer science, psychology, biology, and engineering.In computer science and machine learning, pattern recognition refers to the development of algorithms and techniques that enable computers to automatically detect patterns in data and make predictions or decisions based on those patterns. These algorithms analyze input data, extract relevant features, and classify or interpret the data into meaningful categories or groups.

Pattern recognition is used in a wide range of applications, including image and speech recognition, natural language processing, medical diagnosis, financial forecasting, and anomaly detection. It involves several key steps, including data acquisition, preprocessing, feature extraction, pattern classification, recognition, and interpretation.

Overall, pattern recognition is a multidisciplinary field that continues to evolve, driving innovations in artificial intelligence, robotics, healthcare, and many other domains. Its applications are vast and diverse, offering solutions to real-world problems and pushing the boundaries of what's possible in understanding and analyzing data.

1.2 Fingerprint Recognition

Fingerprint identification relies on pattern recognition to analyze and match the unique ridge structures present in an individual's fingerprints. Initially, high-resolution fingerprint images are acquired and processed to remove noise and enhance clarity. Feature extraction algorithms then identify key characteristics such as minutiae points and ridge patterns, which are used to create a compact template representing the fingerprint. During the matching phase, this template is compared with stored templates in a database using various techniques to determine a potential match. The system evaluates the similarity scores and

makes a decision regarding the fingerprint's identity based on predefined thresholds. Fingerprint recognition systems find extensive application in law enforcement, access control, and mobile devices due to their reliability and non-intrusive nature. Continuous advancements in pattern recognition techniques contribute to improving the accuracy and efficiency of fingerprint identification systems, ensuring their effectiveness in various real-world scenarios[1].

1.3 Neural Networks

A neural network is a computational model inspired by the human brain's neural structure, comprising interconnected nodes called neurons organized into layers. These layers include an input layer receiving data, hidden layers performing computations, and an output layer producing results. Through a process called backpropagation, neural networks are trained by adjusting connection weights to minimize the difference between predicted and actual outputs, using labeled training examples. They excel at learning complex patterns, making them valuable for tasks like image recognition and predictive modeling. Deep learning, a subset of neural networks with multiple hidden layers, enables hierarchical data representation learning, leading to cutting-edge performance across various domains.

In this project, we are using a Backpropagation Neural Network (BPNN) as the core neural network model. BPNN is a type of artificial neural network that employs the backpropagation algorithm to train the network by adjusting the weights of connections between neurons. This allows the network to learn from labeled training data and improve its ability to make accurate predictions or classifications.

The BPNN is being applied to a specific task or problem domain, such as image recognition, time series prediction, or classification. The network is trained using a dataset relevant to the problem, and its performance is evaluated based on metrics such as accuracy, precision, or mean squared error. Additionally, the project may involve fine-tuning the network architecture, selecting appropriate activation functions, and optimizing hyperparameters to improve its performance [4].

2

FEATURE ENGINEERING

2.1 Feature Extraction

Feature extraction is a part of the dimensionality reduction process, in which, an initial set of the raw data is divided and reduced to more manageable groups. So when you want to process it will be easier. The most important characteristic of these large data sets is that they have a large number of variables. These variables require a lot of computing resources to process. So Feature extraction helps to get the best feature from those big data sets by selecting and combining variables into features, thus, effectively reducing the amount of data. These features are easy to process, but still able to describe the actual data set with accuracy and originality.

2.1.1 Gabor filter

A Gabor filter is a linear filter used in image processing for edge detection, texture classification, feature extraction and disparity estimation. It is a bandpass filter, i.e. it passes frequencies in a certain band and attenuates the other frequencies outside such band. A Gabor filter is a Gaussian modulated by a plane wave.

Gabor filters are complex sinusoidal waveforms modulated by a Gaussian function. These filters are tuned to specific orientations and frequencies, allowing them to respond selectively to different patterns in the image. In fingerprint identification, Gabor filters are designed to capture the ridge structure and minutiae points, that is shown in Equation 2.1.

$$G(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \exp\left(i(2\pi\frac{x'}{\lambda} + \psi)\right) \tag{2.1}$$

Where $G(x, y; \lambda, \theta, \psi, \sigma, \gamma)$ is the complex-valued Gabor filter response at position (x, y). $x' = x\cos(\theta) + y\sin(\theta)$ and $y' = -x\sin(\theta) + y\cos(\theta)$ are the rotated coordinates. λ is the wavelength (spatial frequency) of the sinusoidal factor. θ is the orientation of the normal to the parallel stripes of the Gabor function. ψ is the phase offset. σ is the standard deviation of the Gaussian envelope. γ is the spatial aspect ratio, controlling the ellipticity of the support of the Gabor function.

Orientation and Frequency: In the context of fingerprint identification, Gabor filters

are usually designed to cover a range of orientations and frequencies that are relevant to the fingerprint ridge patterns. By varying the orientation and frequency parameters of the Gabor filters, different aspects of the fingerprint texture can be captured.

Feature Extraction: After convolving the fingerprint image with the Gabor filter bank, the resulting response images contain information about the texture and ridge structure at different scales and orientations. From these response images, statistical features are extracted to characterize the fingerprint. Commonly used features include mean, variance, energy, entropy, and local binary patterns (LBP) computed from the filtered images

Texture Representation: Gabor features provide an effective representation of the texture present in fingerprints. The combination of Gabor filters covering multiple orientations and frequencies allows for a comprehensive characterization of the fingerprint's texture, including ridge patterns, ridge endings, bifurcations, and other minutiae. Robustness: Gabor features are robust to variations in fingerprint images caused by factors such as changes in illumination, sensor noise, finger pressure, and skin conditions. This robustness is crucial for reliable fingerprint identification in real-world scenarios where image quality may vary [3].

2.1.2 Minutiae Features

Minutiae points are the major features of a fingerprint image and are used in the matching of fingerprints. These minutiae points are used to determine the uniqueness of a fingerprint image. A good quality fingerprint image can have 25 to 80 minutiae.

It is the most widely used technique of fingerprint representation and its configuration is highly distinctive. It is more accurate compared to other correlation based systems and the template size is smaller in minutiae based fingerprint representation. In this system, two fingerprints match if their minutiae points match. Minutiae based fingerprint technique is the backbone of most currently available fingerprint recognition products.

Compared to other fingerprint features, the minutia point features having corresponding orientation maps are distinct enough to distinguish between fingerprints robustly. Fingerprint representation using minutiae feature reduces the complex issue of fingerprint recognition to an issue of point pattern matching [9].

Since the original image cannot be reconstructed using only the minutiae information, the minutiae based fingerprint identification systems can also assist privacy issues and the minutiae are actually sufficient enough to prove finger individuality. In terms of contrast, image resolution and global distortion the minutiae are more stable and robust in relation to other fingerprint matching schemes.

Minutiae Extraction Process:

- Preprocessing: The captured images undergo preprocessing steps like noise removal, image enhancement, and binarization to improve the quality of the fingerprint image.
- Ridge Detection: The ridge structure of the fingerprint is detected using techniques like ridge thinning or skeletonization to identify the ridge lines.
- Minutiae Detection: Minutiae points are detected by identifying ridge endings (points where a ridge terminates) and ridge bifurcations (points where a ridge splits into two branches). Minutiae Representation: Detected minutiae are represented by their coordinates, type (ending or bifurcation), and orientation.

Types of Minutiae:

- Ridge Endings: Points where a ridge ends abruptly.
- Ridge Bifurcations: Points where a ridge splits into two branches.

2.2 Feature Extraction followed by Feature Selection

Feature selection is a critical step in machine learning and data analysis, where the goal is to identify and retain the most relevant and informative features from a dataset while discarding irrelevant or redundant ones.

2.2.1 Principal Component Analysis(PCA)

Principal component analysis (PCA) is a fundamental multivariate data analysis method that is encountered in a variety of areas in neural networks, signal processing, and

machine learning. It is an unsupervised method for reducing the dimensionality of the existing data set and extracting important information.PCA does not use any output information; the criterion to be maximized is the variance [1].

PCA can be applied to economically represent the input digit images by projecting them onto a low-dimensional space constituted by a small number of basis. These basis images or the "eigendigits" are derived by finding the most significant eigenvectors of the pixel- wise covariance matrix, after mean-centering the data for each attribute. After projection, we use the 1-NN classifier to classify the digit in the low dimensional space. PCA reduces the dimensions of the dataset from 784 to a lower value for ease of computation[2]. Consider a set of n 2-D training images of size MN. Each digit image is represented as a 1-D column vector I_i, by concatenating each row into a long vector, where lin is in an MN-dimensional space. The set's average is determined by Equation 2.2.

$$m = \frac{1}{n}\sum_{i=1} I_i \tag{2.2}$$

By the vector $x = I_i$ - m, i = 1,..., n, each digit deviates from the average. The shifted digits are put on a matrix of dimension MNn, $X = [x_1 x_2 ... x_n]$. The training image set's covariance matrix C_x, is denoted by Equation 2.3.

$$C_x = XX^T \tag{2.3}$$

To solve the eigenvalue problem, Equation 2.4 required.

$$C_x U = UA \tag{2.4}$$

where A is a digonal matrix defined by the eigen values λ of the matrix C_x, that is $\lambda = diag[\lambda_1,\lambda_2,...,\lambda_{MN}]$ and U is the associated eigenvectors represent the new digit space. There are MN possible projections of he image vector x, where $diag[\lambda_1,\lambda_2,...,\lambda_{MN}]$ is a diagonal matrix defined by the eigenvalues of the matrix Cx and U is the corresponding eigenvector. The new digit space is now represented by these eigenvectors. The picture

vector can be projected in MN in different ways by using Equation 2.5.

$$y_j = U_j^T x, j = 1, ...MN \tag{2.5}$$

where the U_j are the eigenvectors of the covariance matrix C_X, and the y_j are the projections of x and called the principal components (also known as eigen digits). The original image vector x may be reconstructed exactly from the projections y_j by Equation 2.6 and Equation 2.7.

$$y = [y_1, y_2, y_3, ..., y_{MN}] \tag{2.6}$$

$$x = Uy = \sum_{j=1}^{MN} u_j y_j \tag{2.7}$$

The dimensionality can be reduced by selecting the first n (less than MN) eigenvectors that have large variances and discarding the remaining ones that have small variances. One may then approximate the image vector x by truncating the expansion of Equation 2.7 after m terms as follows as in Equation 2.8.

$$\hat{x} = \sum_{j=1}^{\hat{n}} u_j y_j \tag{2.8}$$

Therefore, a few numbers of eigenvectors provide sufficient information for image cod-ing and digit recognition [1].

Along with that, One such method is using kurtosis, which measures the ”tailedness” or the degree of peakedness of a distribution. kurtosis can be used to assess the shape of the distribution of principal components and prioritize the selection of those with higher kurtosis values. Here’s the formula for calculating kurtosis shows the Equation 2.9.

$$\text{Kurt}(x) = E\left[\left(\frac{x-\mu}{\sigma}\right)^4\right] \tag{2.9}$$

3

ARTIFICIAL NEURAL NETWORK

3.1 Introduction

Artificial Neural Networks (ANNs) have emerged as a powerful paradigm in machine learning, transforming the landscape of artificial intelligence research and applications. ANNs are composed of interconnected nodes, or neurons, organized into layers, with each neuron performing simple computations on input data and transmitting signals to neurons in the next layer. The architecture of ANNs can vary widely, ranging from simple feedforward networks to complex recurrent and convolutional architectures tailored for specific tasks. Deep Learning, a subset of ANNs, has gained prominence for its ability to learn hierarchical representations of data through multiple layers of abstraction, enabling the modeling of complex relationships and patterns in data. ANNs have demonstrated remarkable success across diverse domains, including computer vision, natural language processing, speech recognition, robotics, healthcare, and more. Their ability to learn from large-scale datasets, adapt to various problem domains, and generalize to unseen data makes them a fundamental tool in modern AI research and industry applications. Ongoing research in ANNs focuses on improving model architectures, training algorithms, and interpretability, further advancing the capabilities and understanding of these powerful computational models.

3.2 Biological Neuron

The brain is principally composed of about 10 billion neurons; each connected to about 10,000 other neurons. Each neuron consists of a cell body or soma where a cell nucleus is located as shown in Figure 3.1. A tree-like nerve fiber called dendrites is associated with the cell body and they receive signals from other neurons. From the cell body, a single long tiber called an axon is extended, which branches into stands and sub-strands in connection to many other neurons, and a small space formed here is said to be a synaptic junction or synapse associated with other neurons. Transmission of signal from one cell to another at a synapse. The axon of a typical neuron leads to a few thousand synapses associated with other neurons. Transmission of signal from one cell to another at a synapse is a complex chemical process in which some substances are released. Their effect is to raise

or lower the electrical potential inside the body of the receiving cell.

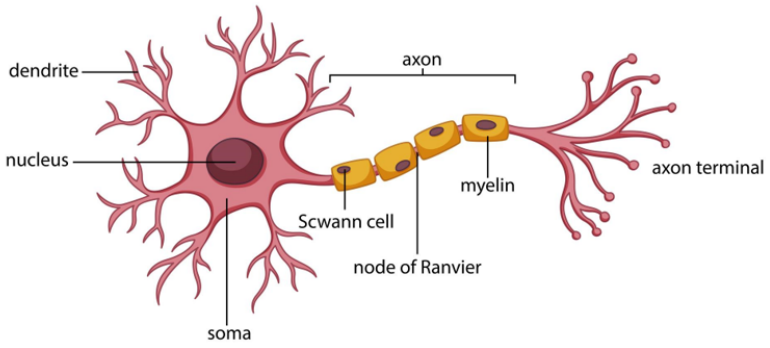


Figure 3.1: Biological Neuron

This potential reaches a thresh- hold and electric activity in the form of short pulses is generated when this happens The cell is said to have fired.

Each neuron receives electrochemical inputs from other neurons at the dendrites, if the sum of these electrical inputs is sufficiently powerful to activate the neuron, it transmits an electrochemical signal along the axon and passes the signal to the other neurons whose dendrites are attached at any of the axon terminals. These attached neurons may then fire. It is important to note that a neuron fires only if the total signal received at the cell body exceeds a certain level. The neuron either fires or it doesn't, there aren't different grades of firing. So, our entire brain is composed of these interconnected electrochemically transmitting neurons. From a very large number of extremely simple processing units(each performing a weighted sum of its inputs, and then firing a binary signal if the total input exceeds a certain level) the brain manages to perform extremely complex tasks. This is the model on which artificial networks are based. Thus far, artificial neural networks haven't even come close to modeling the complexity of the brain, but they have shown to be good at programs that are easy for humans but difficult for a traditional computer, such as image recognition and prediction based on past knowledge [6].

3.3 Types of Learning

All learning methods used for adaptive neural networks can be classified into two

major categories one is Supervised learning incorporates an external teacher so that each out- put unit is told what its desired response to input signals ought to be and unsupervised learning. During the learning process, global information may be required. Paradigms of Supervised learning include error-correction learning and stochastic learning. An important Issue concerning supervised learning is the problem of error convergence,i.e.the minimization of error between the desired and computed unit values. The aim is to determine a set of weights that minimizes the error. One well-known method, which is common to many learning paradigms, is the least mean square convergence. Another one is unsupervised learning. Uses no external teacher and is based upon only local information. It is also referred to as self - organization, in the sense that it self-organizes data presented to the network and detects their emergent collective properties. Paradigms of unsupervised learning are Hebbian learning and competitive learning from human neurons to artificial Neurons another aspect of learning concerns the distinction or not of a separate phase, during which the network is trained, and a subsequent operation phase. A neural network learns offline if the learning phase and the operation phase of distinct. A neural network learns online if it learns and operates at the same time. Usually, supervised learning is performed offline, whereas unsupervised learning is performed online [6].

3.3.1 Supervised Learning

Supervised learning is a foundational approach in machine learning where models are trained on labeled datasets, mapping input data to corresponding output labels. The process involves iteratively adjusting model parameters to minimize the error between predicted outputs and true labels. Classification tasks involve categorizing data into predefined classes, while regression tasks predict numerical values. Applications of supervised learning range from spam detection and image recognition to stock price prediction and medical diagnosis. It remains a cornerstone of machine learning, facilitating advancements in various domains by enabling accurate predictions and classifications based on labeled training data.

3.3.2 Unsupervised Learning

Unsupervised learning is a branch of machine learning where models are trained on unlabeled data, aiming to discover hidden patterns, structures, or relationships within the data. Unlike supervised learning, there are no predefined output labels, and the model learns to extract meaningful information solely from the input data. Common techniques in unsupervised learning include clustering, where similar data points are grouped together, and dimensionality reduction, which aims to simplify data while preserving important features. Unsupervised learning has diverse applications such as customer segmentation, anomaly detection, and data compression. It plays a crucial role in exploratory data analysis and can uncover valuable insights from large and complex datasets [6].

3.4 Major Components of ANN

Major components of ANN are weight Factor, summation function, transfer function, output function, Error function and back propagated values, learning function, and learning rates. Figure 3.2 presents the general architecture of the neural network that is similar to the biological neuron.

3.4.1 Weight Vector:

A neuron usually receives many simultaneous inputs. Each input has its own relative weight, which gives the input the impact that it needs on the processing elements summation function. The weights perform the same type of function, as do the varying synaptic Strengths of biological neurons.

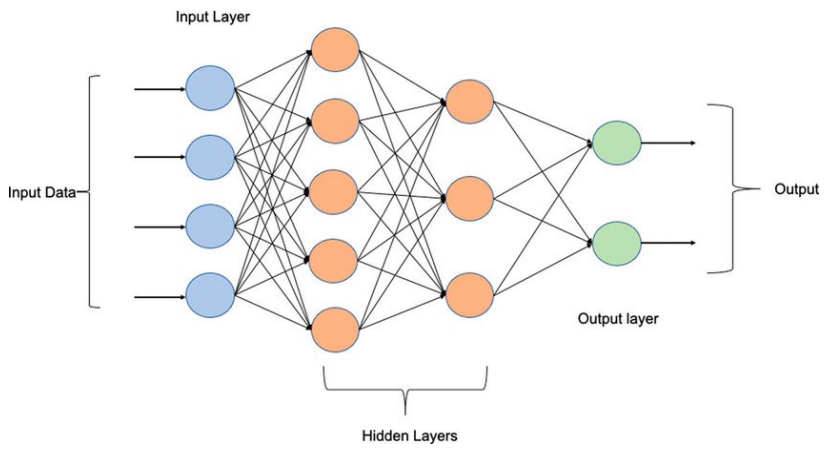


Figure 3.2: Multi-Layer Neural Network

In both cases, some inputs are made more important than others so that they have a greater effect on the processing element as they combine to produce a neural response. Weights are adaptive coefficients within the network that determine the intensity of the input signal as registered by the artificial neuron. They are a measure of an input's connection strength. These strengths can be modified in response to various training sets and according to a network's specific topology or through its learning rules [5].

3.4.2 Summation Factor:

The summation factor in the weighted sum of all inputs refers to the process of combining input values with corresponding weights in a linear model. In many machine learning algorithms, including artificial neural networks, the output of a neuron or node is calculated by taking the weighted sum of its inputs, where each input value is multiplied by a corresponding weight. The summation factor represents the aggregation of these weighted inputs, typically followed by the application of an activation function to produce the final output of the neuron. This process allows the model to assign different levels of importance to each input feature, with the weights determining the degree of influence that each feature has on the neuron's output. Adjusting the weights during the training process enables the model to learn and adapt to the patterns in the data, optimizing its performance for the task at hand [5].

3.4.3 Transfer Function:

A transfer function, also known as an activation function in the context of neural networks, is a mathematical function that operates on the input of a neuron and determines the neuron's output. In neural networks, the transfer function introduces nonlinearity to the model, allowing it to learn complex patterns and relationships in the data. There are several types of transfer functions used in neural networks, each with its characteristics and advantages:

1. Sigmoidal Function:

The sigmoid function can be configured with two parameters: Ti describes the shift of

the function with the absolute value-Ti: the parameter c is a measure for the steepness. Given that c is larger than zero, the function is on the whole domain monotonically increasing continuously and differentiable. For this reason, it is particularly important for networks applying backpropagation algorithms. For larger values of c, the fermi function approximates a Heaviside function. The plot is shown in Figure 3.3 and is given by Equation 3.1 [5].

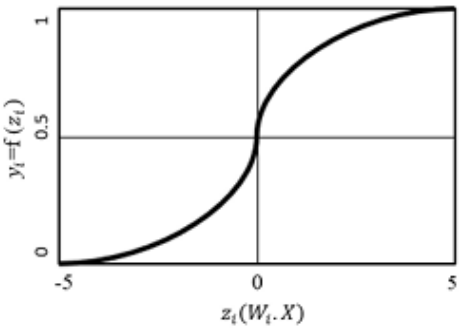


Figure 3.3: Sigmoidal Function

$$y_i = f(z_i, T_i, c) = \frac{1}{1 + e^(z_i + T_i)} \tag{3.1}$$

2. Rayleigh Function:

The Rayleigh fading model used in wireless communications, which models the variation in signal strength over a wireless channel due to multipath propagation. In this context, the Rayleigh fading model assumes that the magnitude of the received signal follows a Rayleigh distribution. In wireless communications, the Rayleigh fading model is often used in scenarios where there is no dominant line-of-sight path between the transmitter and the receiver, such as in urban environments with many obstacles. The Rayleigh fading model assumes that the magnitude of the received signal can be modeled as a random variable with a Rayleigh distribution as shown in Figure 3.4 and Equation 3.2. The probability density function (PDF) of the Rayleigh distribution is given by:

$$f(x;\sigma) = \frac{x}{\sigma^2} e^{\frac{-x^2}{2\sigma^2}} \tag{3.2}$$

3. Softmax Function:

The softmax function is a widely used activation function in machine learning, par-

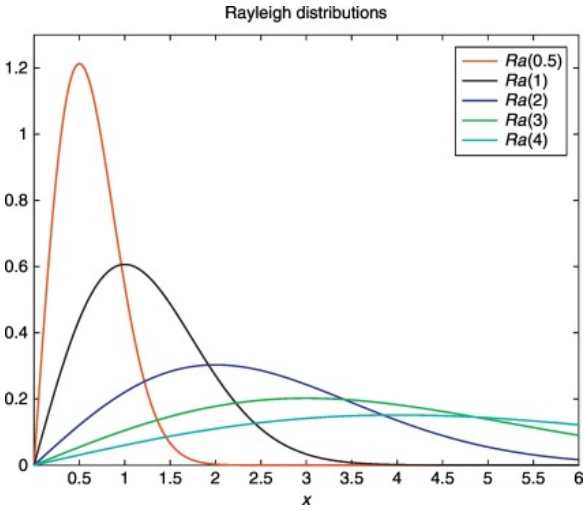


Figure 3.4: Rayleigh Function

ticularly in classification problems. It is used to convert a vector of real numbers into a probability distribution. Given a vector z of real numbers, the softmax function softmax(z) is defined element-wise as shown in Figure 3.5 and Equation 3.3 [6].

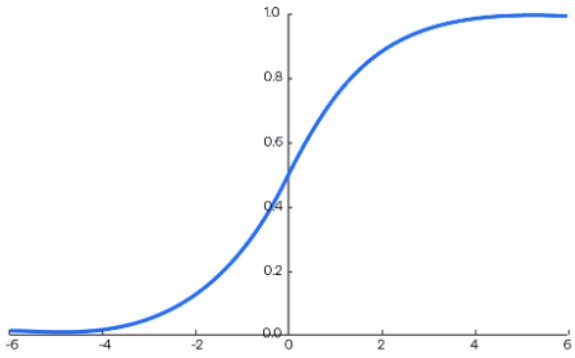


Figure 3.5: Softmax Function

$$Softmax(z_i) = \frac{e_i^z}{\sum_{j=1}^{N} e^z j} \tag{3.3}$$

where, Zi is the i^{th} element of the input vector z, N is the total number of elements in z, e is the base of the natural logarithm z. The softmax function essentially normalizes the input vector z into a probability distribution, ensuring that the resulting values are non-negative and sum up to 1. This makes it useful for multi-class classification tasks where the model needs to output probabilities for each class [5].

4. Step Function:

The Heaviside function (as temporal function known as the 'step function') in Figure 3.6 is used to model the classic All-or-none behavior. It resembles a ramp function as shown in Figure 3.6 changes the function value abruptly when a threshold value T is reached and is given by Equation 3.4 [5].

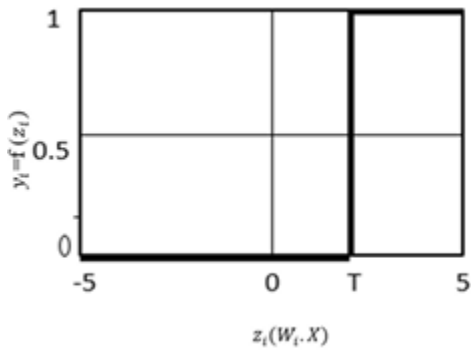


Figure 3.6: Step Function

$$x_i = f(z_i) = f(x) = \begin{cases} 0 & , \ z_i < T \\ 1 & , \ z_i \geq T \end{cases} \tag{3.4}$$

3.4.4 Output Function:

Each processing element is allowed one output signal, which it may output to hundreds of neurons. This is just like the biological neuron, where there are many inputs and only one output action. Normally, the output is directly equivalent to the transfer function's result. Some network topologies, however, modify the transfer result to incorporate competition among neighboring processing elements. Neurons are allowed to compete with each other, inhibiting processing elements unless they have great strength. Competition can occur at one or both two levels. First, competition determines which artificial neuron will be active, or provides an output. Second, competitive inputs help determine which processing element will participate in the learning or adaptation process [4].

3.4.5 Error Function:

In most learning networks the difference between the current output and the desired

output is calculated. This error is then transformed by the error function to match partic ular network architecture. The most basic architectures use this error directly, but some square the error to fit their specific purposes. The artificial neuron's error is then typically propagated into the learning function of another processing element. This error term is sometimes called the current error.

The current error is typically propagated backward to a previous layer. Yet, this back propagated value can be either the current error, the current error scaled in some manner, or some other desired output depending on the network type. Normally, this back-propagated value, after being scaled by the learning function, is multiplied against each of the incoming connection weights to modify them before the next learning cycle [4].

3.4.6 Learning Function:

The purpose of the learning function is to modify the variable connection weights on the inputs of each processing element according to some neural-based algorithm This process of changing the weights of the input connections to achieve some desired result could also be called the adaptation function, as well as the learning mode. There are two types of learning: supervised and unsupervised. Supervised learning requires a teacher. The teacher may be a training set of data or an observer who grades the performance of the network results. Either way, having a teacher is learning by reinforcement. When there is no external teacher, the system must organize itself by some internal criteria designed into the network. This is learning by doing [4].

3.4.7 Learning Rates:

The rate at which ANNs learn depends upon several controllable factors. In selecting the approach there are many trade-offs to consider. Obviously, a slower rate means a lot more time is spent in accomplishing offline learning to produce an adequately trained sys tem. With the faster learning rates, however, the network may not be able to make the fine discriminations possible with a system that learns more slowly. Researchers are working on producing the best of both worlds.

Generally, several factors besides time have to be considered when discussing the Line

training task, which is often described as ”tiresome: Network complexity size, paradigm selection, architecture, type of learning rule or rules employed, and desired accuracy must all be considered. These factors play a significant role in determining how long it will take to train a network. Changing any one of these factors may either extend the training time to an unreasonable length or even result in an unacceptable accuracy [4].

3.5 Applications of Neural Networks

- Image and Speech Recognition:Neural networks are used to recognize and classify images and speechpatterns, such as identifying faces, objects, and speech commands.
- Financial Forecasting: Neural networks are used to predict stock prices, exchange rates, and other financial data.
- Medical Diagnosis: Neural networks are used to analyze medical data, including medical images, and assist with the diagnosis of diseases and medical conditions.
- Gaming: Neural networks are used in the development of artificial intelligence for gaming, including game playing and opponent modeling.
- Marketing: Neural networks are used to analyze consumer behavior and preferences, and assist with targeted marketing and advertising
- Predictive Maintenance: Neural networks are used to predict equipment failures and maintenance needs in industries such as manufacturing and transportation.
- Fraud Detection: Neural networks are used to detect fraudulent activities in banking and finance.
- Autonomous Vehicles: Neural networks are used in the development of autonomous vehicles, including object detection, navigation, and control [4].

4

CONVOLUTIONAL NEURAL NETWORK

4.1 Introduction

Convolutional Neural Networks (CNNs) stand as a cornerstone in the realm of deep learning, revolutionizing various fields, particularly computer vision, with their unparalleled ability to comprehend and extract intricate patterns from visual data. CNNs have garnered significant attention due to their exceptional capacity to process complex image-based information. At its core, a CNN mimics the visual cortex's architecture, composed of multiple layers that progressively learn and abstract hierarchical features from raw pixel inputs. This hierarchical feature extraction enables CNNs to automatically detect essential details like edges, textures, and shapes, making them highly adept at tasks such as image classification, object detection, and image segmentation.

Constructing and training Convolutional Neural Networks (CNNs) is increasingly accessible and various tools and frameworks available in the field of deep learning. These frameworks offer pre-defined layers, training functions, and visualization tools, streamlining the process. Additionally, the versatility of these frameworks allows for efficient data preprocessing, augmentation, and the integration of custom architectures. This accessibility empowers researchers and practitioners to tailor CNNs to a diverse range of applications. CNNs play a crucial role in advancing computer vision capabilities, driving breakthroughs in fields ranging from medical diagnostics to autonomous vehicles. Their profound impact on modern technological landscapes underscores the significance of their continued development and application.

The network architecture can vary depending on the types and numbers of layers included. The types and number of layers included depends on the particular application or data. For example, classification networks typically have a softmax layer and a classification layer, whereas regression networks must have a regression layer at the end of the network. A smaller network with only one or two convolutional layers might be sufficient to learn on a small number of grayscale image data. On the other hand, for more complex data with millions of colored images, you might need a more complicated network with multiple convolutional and fully connected layers [9].

4.2 Image Input Layer

The Image Input Layer plays a pivotal role in seamlessly integrating image data into various deep learning workflows. Serving as the entry point of the neural network architecture, this layer facilitates the ingestion of image data with diverse dimensions and formats. It accommodates the preprocessing and normalization steps necessary for model training, enabling the network to efficiently learn essential features. Additionally, the Image Input Layer offers flexibility in handling augmented datasets, contributing to improved model robustness. With its user-friendly interface and compatibility with various neural network architectures, this layer significantly simplifies the process of incorporating image data into deep learning pipelines [9].

4.3 Convolutional Layer

A 2-D convolutional layer applies sliding convolutional filters to 2-D input. Create a 2-D convolutional layer using convolution2dLayer. The convolutional layer consists of various components.

- Filters and Stride: A convolutional layer consists of neurons that connect to subregions of the input images or the outputs of the previous layer. The layer learns the features localized by these regions while scanning through an image. When creating a layer using the convolution2dLayer function, you can specify the size of these regions using the filter Size input argument.

  For each region, the trainNetwork function computes a dot product of the weights and the input, and then adds a bias term. A set of weights that is applied to a region in the image is called a filter. The filter moves along the input image vertically and horizontally, repeating the same computation for each region. In other words, the filter convolves the input. This image shows a 3-by-3 filter scanning through the input. The lower map represents the input and the upper map represents the output [14].

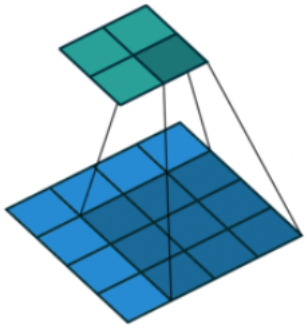


Figure 4.1: Filters and Stride

The step size with which the filter moves is called a stride. You can specify the step size with the Stride name-value pair argument. The local regions that the neurons connect to can overlap depending on the filterSize and 'Stride' values as shown in Figure 4.1. This image shows a 3-by-3 filter scanning through the input with a stride of 2. The lower map represents the input and the upper map represents the output.

The number of weights in a filter is h * w * c, where h is the height, and w is the width of the filter, respectively, and c is the number of channels in the input. For example, if the input is a color image, the number of color channels is 3. The number of filters determines the number of channels in the output of a convolutional layer [9]. Specify the number of filters using the Num Filters argument with the convolution2dLayer function as shown in Figure 4.2 .

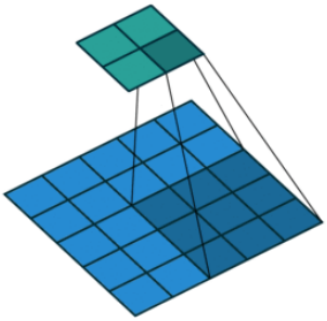


Figure 4.2: Filters and Stride

- Dilated Convolution: A dilated convolution is a convolution in which the filters are expanded by spaces inserted between the elements of the filter. Specify the dilation factor using the 'DilationFactor' property. Use dilated convolutions to increase the

receptive field of the layer without increasing the number of parameters or computation.

The layer expands the filters by inserting zeros between each filter element. The dilation factor determines the step size for sampling the input or equivalently the up-sampling factor of the filter. It corresponds to an effective filter size of Filter Size –1. Dilation Factor +1. For example, a 3-by-3 filter with the dilation factor [2 2] is equivalent to a 5-by-5 filter with zeros between the elements as shown in Figure 4.3.

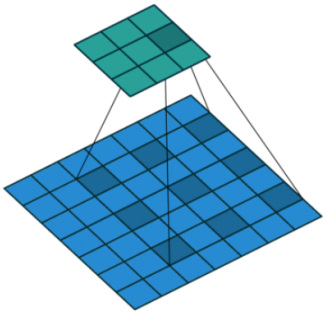


Figure 4.3: Dilated Convolutin

This image shows a 3-by-3 filter dilated by a factor of two scanning through the input. The lower map represents the input and the upper map represents the output [9].

- Feature Maps: As a filter moves along the input, it uses the same set of weights and the same bias for the convolution, forming a feature map. Each feature map is the result of a convolution using a different set of weights and a different bias. Hence, the number of feature maps is equal to the number of filters. The total number of parameters in a convolutional layer is ((h*w*c + 1) *Number of Filters), where 1 is the bias.

- Padding: You can also apply padding to input image borders vertically and horizontally using the 'Padding' name-value pair argument. Padding is values appended to the borders of the input to increase its size. By adjusting the padding, you can control the output size of the layer.

This image shows a 3-by-3 filter scanning through the input with padding of size

1. The lower map represents the input and the upper map represents the output as shown in Figure 4.4.

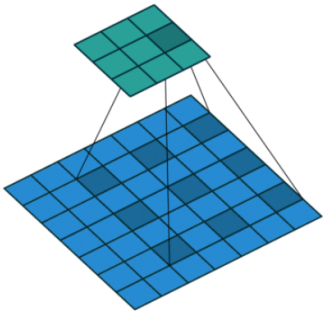


Figure 4.4: Padding

- Output Size: The output height and width of a convolutional layer is (Input Size – ((Filter Size – 1) *Dilation Factor + 1) + 2*Padding)/Stride + 1. This value must be an integer for the whole image to be fully covered. If the combination of these options does not lead the image to be fully covered, the software by default ignores the remaining part of the image along the right and bottom edges in the convolution.

- Number of Neurons: The product of the output height and width gives the total number of neurons in a feature map, say Map Size. The total number of neurons (output size) in a convolutional layer is Map Size*Number of Filters [12].

  For example, suppose that the input image is a 32-by-32-by-3 color image. For a convolutional layer with eight filters and a filter size of 5-by-5, the number of weights per filter is 5 * 5 * 3 = 75, and the total number of parameters in the layer is (75 + 1) * 8 = 608. If the stride is 2 in each direction and padding of size 2 is specified, then each feature map is 16-by-16. This is because (32 – 5 + 2 * 2)/2 + 1 = 16.5, and some of the outermost padding to the right and bottom of the image is discarded. Finally, the total number of neurons in the layer is 16 * 16 * 8 = 2048.

  Usually, the results from these neurons pass through some form of nonlinearity, such as rectified linear units (ReLU).

- Learning Parameters: You can adjust the learning rates and regularization options for the layer using name-value pair arguments while defining the convolutional layer. If you choose not to specify these options, then trainNetwork uses the global training options defined with the training Options function.

- Number of Layers: A convolutional neural network can consist of one or multiple convolutional layers. The number of convolutional layers depends on the amount and complexity of the data [9].

4.4 Batch Normalization Layer

Create a batch normalization layer using batchNormalizationLayer. A batch normalization layer normalizes a mini-batch of data across all observations for each channel independently. To speed up training of the convolutional neural network and reduce the sensitivity to network initialization, use batch normalization layers between convolutional layers and nonlinearities, such as ReLU layers. The layer first normalizes the activations of each channel by subtracting the mini-batch mean and dividing by the mini-batch standard deviation. Then, the layer shifts the input by a learnable offset and scales it by a learnable scale factor α, β, γ are themselves learnable parameters that are updated during network training.

Batch normalization layers normalize the activations and gradients propagating through a neural network, making network training an easier optimization problem. To take full advantage of this fact, you can try increasing the learning rate. Since the optimization problem is easier, the parameter updates can be larger and the network can learn faster. You can also try reducing the L2 and dropout regularization.

With batch normalization layers, the activations of a specific image during training depend on which images happen to appear in the same mini-batch. To take full advantage of this regularizing effect, try shuffling the training data before every training epoch. To specify how often to shuffle the data during training, use the 'Shuffle' name-value pair argument of training Options [8].

4.5 ReLU Layer

Create a ReLU layer using reluLayer. A ReLU layer performs a threshold operation to each element of the input, where any value less than zero is set to zero. Convolutional and batch normalization layers are usually followed by a nonlinear activation function such as a rectified linear unit (ReLU), specified by a ReLU layer. A ReLU layer performs a threshold operation to each element, where any input value less than zero is set to zero, that is, as shown in Equation 4.1.

$$f(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{4.1}$$

The ReLU layer does not change the size of its input. There are other nonlinear activation layers that perform different operations and can improve the network accuracy for some applications. For a list of activation layers, see Activation Layers [8].

4.6 Max and Average Pooling Layers

A 2-D max pooling layer performs downsampling by dividing the input into rectangular pooling regions, then computing the maximum of each region. Create a max pooling layer using maxPooling2dLayer. A 2-D average pooling layer performs downsampling by dividing the input into rectangular pooling regions, then computing the average of each region. Create an average pooling layer using averagePooling2dLayer. Pooling layers follow the convolutional layers for down-sampling, hence, reducing the number of connections to the following layers. They do not perform any learning themselves, but reduce the number of parameters to be learned in the following layers. They also help reduce overfitting.

A max pooling layer returns the maximum values of rectangular regions of its input. The size of the rectangular regions is determined by the poolSize argument of maxPoolingLayer. For example, if poolSize is [2 3], then the layer returns the maximum value in regions of height 2 and width 3.

An average pooling layer outputs the average values of rectangular regions of its input.

The size of the rectangular regions is determined by the poolSize argument of averagePoolingLayer. For example, if poolSize is [2 3], then the layer returns the average value of regions of height 2 and width 3. Pooling layers scan through the input horizontally and vertically in step sizes you can specify using the 'Stride' name-value pair argument. If the pool size is smaller than or equal to the stride, then the pooling regions do not overlap.

For nonoverlapping regions (Pool Size and Stride are equal), if the input to the pooling layer is n-by-n, and the pooling region size is h-by-h, then the pooling layer down-samples the regions by h. That is, the output of a max or average pooling layer for one channel of a convolutional layer is n/h-by-n/h. For overlapping regions, the output of a pooling layer is (Input Size – Pool Size + 2*Padding)/Stride + 1 [8].

4.7 Fully Connected Layer

Create a fully connected layer using fullyConnectedLayer. A fully connected layer multiplies the input by a weight matrix and then adds a bias vector. The convolutional (and down-sampling) layers are followed by one or more fully connected layers. As the name suggests, all neurons in a fully connected layer connect to all the neurons in the previous layer. This layer combines all of the features (local information) learned by the previous layers across the image to identify the larger patterns. For classification problems, the last fully connected layer combines the features to classify the images. This is the reason that the output Size argument of the last fully connected layer of the network is equal to the number of classes of the data set. For regression problems, the output size must be equal to the number of response variables [8].

4.8 Output Layers

Softmax and Classification Layers: A softmax layer applies a softmax function to the input. Create a softmax layer using SoftMax Layer. A classification layer computes the cross-entropy loss for classification and weighted classification tasks with mutually exclusive classes. Create a classification layer using classificationLayer. For classification

problems, a softmax layer and then a classification layer usually follows the final fully connected layer as shown in Equation 4.2 and 4.3. The output unit activation function is the softmax function:

$$y_r(x) = \frac{exp(a_r(x))}{\sum_{j=1}^{k} exp(a_j(x))} \tag{4.2}$$

Here,The softmax function takes as input a vector x of k real numbers.

Were,

$$0 \leq y_r \leq 1 and \sum_{j=1}^{k} y_j = 1 \tag{4.3}$$

The softmax function is the output unit activation function after the last fully connected layer for multi-class classification problems as shown in Equation 4.4.

$$P(c_r|x,\theta) = \frac{P(x,\theta|c_r)P(c_r)}{\sum_{j=1}^{k} P(x,\theta|c_j)P(c_j)} \tag{4.4}$$

Where, the conditional probability of the sample given class r, and $P(c_r)$ is the class prior probability as shown in Equation 4.5 and 4.6.

$$0 \leq P(c_r|x\theta) \leq 1 and \sum_{j=1}^{k} P(c_j|x,\theta) = 1 \tag{4.5}$$

$$loss = -\frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{k} w_i t_{ni} \log y_{ni} \tag{4.6}$$

where $N$is the number of samples, k is the number of classes, w_i is the weight for class i, t_{ni} is the indicator that the nth sample belongs to the ith class, and y_{ni} is the output for sample n for class i, which in this case, is the value from the softmax function. In other words, yni is the probability that the network associates the nth input with class i [8].

Disadvantages:

Using a CNN for fingerprint ID needs lots of computer time.Neural networks need lots of data for good training, making it tough for fingerprint datasets with various patterns. CNNs are like black boxes, hard to understand how they work.

5

BACKPROPAGATION NEURAL NETWORK

5.1 Introduction

Backpropagation Neural Network (BPNN) is a widely used type of artificial neural network (ANN) for supervised learning tasks. It has gained popularity due to its ability to learn complex patterns from data and make accurate predictions. BPNN is a feedforward neural network, which means that information flows in one direction, from the input layer through the hidden layers to the output layer. BPNN is known for its capability to approximate non-linear functions, making it suitable for a wide range of applications, such as image recognition, speech recognition, naturallanguage processing, financial prediction, and medical diagnosis. It can handle both regression and classification problems, making it versatile for various domains. One of the key reasons behind the popularity of BPNN is its ability to learn from labeled data during thetraining process. It can adjust the weights and biases of the network based on the error or loss between the predicted outputs and the actual outputs, using the backpropagation algorithm. BPNN enables the network to continuously update its parameters and improve its performance iteratively. Another advantage of BPNN is its ability to automatically extract relevant features from the input data without relying on explicit feature engineering. This allows it to capture complex patterns in the data, which may not be easily discernible through traditional approaches. BPNN can also handle high-dimensional data effectively, making it suitable for tasks that involve large datasets with multiple input features.

Furthermore, BPNN offers flexibility in terms of architecture, activation functions, and learning strategies, allowing users to customize the network based on their specific requirements. It can be trained using different optimization algorithms, such as gradient descent,and can incorporate various regularization techniques, such as dropout and weight decay,to improve generalization performance. Overall, BPNN is a powerful and popular type of ANN that has been widely used in supervised learning tasks due to its ability to learn complex patterns from data, make accurate predictions, and its flexibility in architecture and customization.

5.2 BPNN Architecture

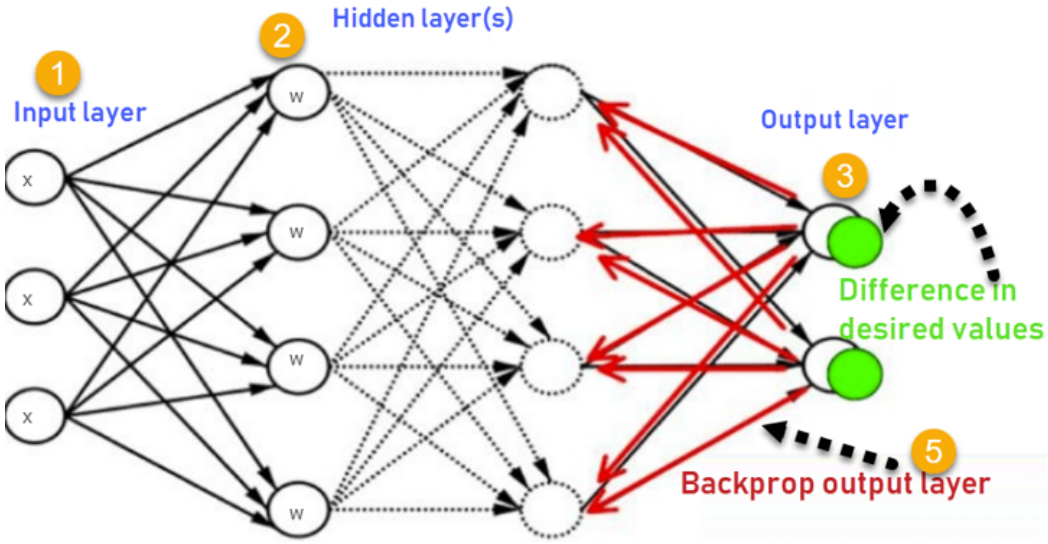


Figure 5.1: BPNN Architecture

As shown in Figure 5.1 shows the architecture of a Backpropagation Neural Network (BPNN), which is also known as a Multi-Layer Perceptron (MLP), consists of multiple layers of interconnected nodes, including an input layer, one or more hidden layers, and an output layer.

- Input Layer: The input layer is the first layer in the BPNN architecture and receives the input data for the network. Each node in the input layer represents a feature or attribute of the input data, and the number of nodes in the input layer depends on the dimensionality of the input data.

- Hidden Layers: The hidden layers are the intermediate layers between the input and output layers. They consist of one or more layers of nodes, and the number of hidden layers and the number of nodes in each hidden layer can vary depending on the complexity of the problem being solved. Hidden layers are responsible for processing the input data and extracting relevant features for making predictions.

- Output Layer: The output layer is the final layer in the BPNN architecture and produces the predicted output or the decision of the network. The number of nodes in the output layer depends on the number of classes or targets in the problem being

solved.Each node in the output layer represents a class or target label, and the activation of the nodes in the output layer determines the predicted output.

The connections between the nodes in the layers are represented by weights, which are learned during the training process. Each connection is associated with a weight that determines the strength of the connection. Additionally, each node in the network, including the input, hidden, and output layers, is associated with a bias, which is an additional adjustable parameter used to adjust the activation of the node. The architecture of BPNN allows for complex non-linear mappings between input and output data, which enables the network to learn complex patterns in the data and make predictions on unseen data. The activation function, which is applied to the output of each node, determines the activation level of the node and introduces non-linearity into the network. The commonly used sigmoid function or other activation functions such as ReLU,tanh, etc., are applied to the nodes in the hidden and output layers to introduce non-linearity into the network and enable it to model complex patterns in the data. Overall, the architecture of BPNN with multiple layers, interconnected nodes, weights, biases, and activation functions allows for the network to learn from data and make predictions on unseen data, making it a powerful tool for a wide range of supervised learning tasks [4].

5.3 Training BPNN

Training a Backpropagation Neural Network (BPNN) involves several key steps, including forward propagation, error computation, backward propagation, and weight update using an optimization algorithm.

- Forward Propagation: In this step, the input data is fed into the network, and the activations of the nodes in the hidden and output layers are computed using the sigmoid function. The weighted sum of inputs for each neuron in the hidden and output layers is calculated and then passed through functions such as sigmoid, tanh, and ReLU, as well as various optimization algorithms like stochastic gradient descent (SGD) and Levenberg-Marquardt. These activations are then used as inputs for the next layer

in the forward propagation process until the output layer is reached, and the final predictions are obtained.

- Error Computation: After obtaining the predicted outputs from the forward propagation step, the error or loss between the predicted outputs and the actual outputs (targets) is computed. This error measure quantifies the difference between the predicted and actual outputs and serves as a measure of how well the network is performing.

- Backward Propagation: In this step, the error computed in the previous step is propagated backward through the network to update the weights. The gradient of the error with respect to the activations of the nodes in the output layer is calculated, and then the gradient is backpropagated through the hidden layers, layer by layer, using the chain rule of calculus. The gradient represents the direction and magnitude of the steepest increase in the error, and it is used to update the weights in the opposite direction of the gradient in order to minimize the error.

- Weight Update: The weights of the network are updated using an optimization algorithm, such as gradient descent or one of its variants, based on the gradient computed in the backward propagation step. The weights are adjusted in the direction of the negative gradient, scaled by a learning rate, which determines the step size of the weight update. The learning rate is a hyperparameter that needs to be carefully chosen to balance the trade-off between convergence speed and stability.

- Multiple Epochs: Training BPNN typically involves repeating the forward and backward propagation steps for multiple epochs. An epoch refers to a complete pass through the entire training dataset. During each epoch, the network processes the training data, computes the error, updates the weights, and iteratively improves its performance. Repeating the process for multiple epochs allows the network to optimize the weights and improve its accuracy over time.

It's important to note that the training process of BPNN is an iterative and computationally intensive task that requires careful tuning of hyperparameters, such as learning rate,

batch size, and number of epochs, to achieve optimal results. Additionally, proper handling of overfitting, validation, and regularization techniques may also be necessary to ensure the network generalizes well to unseen data [4].

5.4 Levenberg-Marquardt optimization (LM)

Levenberg-Marquardt (LM) is a widely used optimization algorithm for solving nonlinear least squares problems, often employed in training neural networks. It's an iterative method that combines aspects of both steepest descent and Gauss-Newton methods. The main idea is to adjust the parameters (weights and biases in the context of neural networks) in a way that minimizes a specified error function. In the realm of fitting a parameterized mathematical model to a dataset, least squares problems emerge as the objective involves minimizing the sum of squared differences of the errors between the model function and a set of data points. If a model is linear in its parameters, the least square objective is quadratic in the parameters. This objective may be minimized with respect to the parameters in one step via the solution to a linear matrix equation. When dealing with nonlinear fit functions, solving least squares problems requires iterative methods. These algorithms iteratively refine the model parameters to minimize the sum of squared errors between the model and data points. The Levenberg-Marquardt algorithm integrates two numerical minimization approaches: gradient descent and Gauss-Newton methods. In gradient descent, parameters are adjusted in the direction of steepest descent to reduce the sum of squared errors. Conversely, the Gauss-Newton method treats the least square function as locally quadratic in parameters, aiming to minimize this quadratic. The Levenberg-Marquardt algorithm behaves more like gradient descent when parameters are distant from their optimal value and shifts towards Gauss-Newton when parameters approach optimality.

LM Algorithm

- **Initialization**: Start with an initial guess for the parameters (weights and biases in the case of neural networks). Choose an initial value for the damping parameter (). Define the error function to be minimized (e.g. Mean Square Error).

- **Evaluation**: Compute the error function (e.g., Mean Square Error) using the current parameter values. Calculate the Jacobian matrix, which contains the partial derivatives of the outputs with respect to the parameters. This matrix helps in approximating the local gradient of the error function. The objective function represents the error we want to minimize. In the context of neural networks, it's often the mean square error (MSE) between the predicted output and the actual output.

  Objective Function as shown in Equation 5.1.

$$J = \frac{1}{2m}\sum_{i=1}^{m}(\hat{y_i} - y_i)^2 \tag{5.1}$$

- **Parameter Update Rule**: Update the parameters using the LM update rule, which combines aspects of gradient descent and Gauss-Newton methods. Adjust the damping parameter λ to control the step size. If the error decreases, decrease λ to allow larger steps; if the error increases, increase λ to take smaller steps. The update rule is given by as shown in Equation 5.2.

$$[J^T W J + \lambda I]h_{im} = J^T W(y - \hat{y}) \tag{5.2}$$

  where small values of the damping parameter result in a Gauss-Newton update and large values of result in a gradient descent update. The damping parameter is initialized to be large so that first updates are small steps in the steepest-descent direction. If any iteration happens to result in a worse approximation $(2(p + hlm) > 2(p))$, then is increased.

- **Convergence Check**:

  Check for convergence criteria, such as:i. Whether the change in the error function is below a specified threshold. ii. Whether the change in the parameters is below a specified threshold.iii. Maximum number of iterations reached.

- **Iteration**: If convergence criteria are not met, repeat steps 2-4 until convergence is achieved. Otherwise, stop the algorithm and consider the current parameter values as

the optimized solution. Throughout these phases, the algorithm iteratively refines the parameter values to minimize the error function. By combining gradient descent with the Gauss-Newton method and adjusting the damping parameter, LM strikes a balance between convergence speed and stability, making it effective for optimizing nonlinear least squares problems, including neural network training [4].

5.5 BPNN Algorithm

The BPNN (Backpropagation Neural Network) algorithm is a popular supervised learning algorithm used for training feedforward artificial neural networks. It is widely used for tasks such as regression and classification. The algorithm consists of several steps:

Initialize Weights and Biases: The weights and biases of the network are initialized with small random values or predefined values.

Forward Propagation: Feed forward the inputs through the network to calculate the output using as shown in Equations 5.3 and 5.4.

$$z^{l+1} = w^{l+1}a^l + b^{l+1} \tag{5.3}$$

$$a^{l+1} = g(z^l + 1) \tag{5.4}$$

where w^{l+1} is the weight matrix connecting layer l to layer $l+1$, $b^{l+1}$is the bias vector for layer $l+1$, a^l is the activation vector for layer l, $g(z^l+1)$ is the activation function, and z^{l+1} is the weighted input to layer $l+1$.

- Compute Error/Loss: Calculate the error between the predicted output and the true output by using as shown in Equation 5.5.

$$E = \frac{1}{2}(y - \hat{y})^2 \tag{5.5}$$

where E is the error, y is the true output and $\hat{y}$ is the predicted output.

- Backward Propagation: Calculate the gradient of the loss function with respect to the

weights and biases by using Equation 5.6 and Equation 5.7.

$$\frac{\partial E}{\partial w^{l+1}} = \delta^{l+1}(a^l)^T \tag{5.6}$$

$$\frac{\partial E}{\partial b^{l+1}} = \delta^{l+1} \tag{5.7}$$

where δ^{l+1} is the error vector for layer $l+1$. Propagate the error back through the network to calculate the error vectors for each layer by using Equation 5.8 and Equation 5.9.

$$\delta^l = (y - \hat{y}) \cdot g'(z^L) \tag{5.8}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1} \cdot g'(z^l) \tag{5.9}$$

where L represents the output layer of the neural network, while l represents any of the hidden layers of the network. denotes element-wise multiplication and g(.) is the derivative of the activation function.

- Weight Updates: Update the weights and biases using gradient descent as shown in Equation 5.10 and Equation 5.11.

$$w^{l+1} = w^{l+1} - \eta \frac{\partial E}{\partial w^{l+1}} \tag{5.10}$$

$$b^{l+1} = b^{l+1} - \eta \frac{\partial E}{\partial b^{l+1}} \tag{5.11}$$

- Repeat: Steps 2-5 are repeated for multiple epochs, where each epoch corresponds to one complete pass through the entire dataset. This allows the network to iteratively adjust the weights and biases to minimize the error and improve its performance.

- Termination: The training process is terminated based on a stopping criterion, such as reaching a certain number of epochs, achieving a desired level of accuracy, or observing no significant improvement in the error [4].

6

DESIGN APPROACH

6.1 Introduction

The proposed methodology for fingerprint identification involves several sequential steps to enhance accuracy through the fusion of Gabor and Minutiae features using a Backpropagation Neural Network (BPNN) classifier. Initially, a dataset of fingerprint images is collected, followed by a preprocessing stage. This includes resizing images and applying morphological operations, such as dilation, erosion, and opening, to improve the quality and clarity of the features. Gabor features, capturing texture information, and minutiae extraction, identifying specific ridge characteristics, are then extracted. The fusion of Gabor and minutiae features aims to create a comprehensive representation of fingerprint patterns. To manage the high-dimensional feature space, Principal Component Analysis (PCA) is applied for dimensionality reduction. Subsequently, the reduced feature set along with their corresponding labels are loaded for the final classification step. A feed-forward BPNN is created to train the model to classify fingerprints into distinct categories, such as arches, left loops, right loops, tented arches, and whorls. The accuracy of the classification results is evaluated, demonstrating the effectiveness of the proposed methodology in accurately identifying and categorizing various fingerprint patterns.

6.2 Pre-Processing

Preprocessing plays a crucial role in improving the quality of fingerprint images and facilitating accurate feature extraction. It begins with input fingerprint images, which may vary in size and quality. Resizing these images to a standardized dimension not only ensures consistency but also reduces computational complexity. Morphological operations are then applied to refine the fingerprint ridges and valleys, enhancing their clarity and continuity. Next, Gabor filters are utilized to extract both phase and texture features from the fingerprint image. Gabor phase features capture the orientation and direction of ridges, while Gabor texture features characterize the fine details and patterns within the ridges. These Gabor features provide a robust representation of the fingerprint's unique characteristics, enabling more accurate identification.By combining both phase and texture features, the

preprocessing stage aims to create a comprehensive fingerprint representation that captures essential details while minimizing noise and irrelevant information. Overall, preprocessing lays the foundation for effective feature extraction and classification, ultimately improving the accuracy and reliability of fingerprint identification systems as shown in Figure 6.1.

6.3 Block Diagram

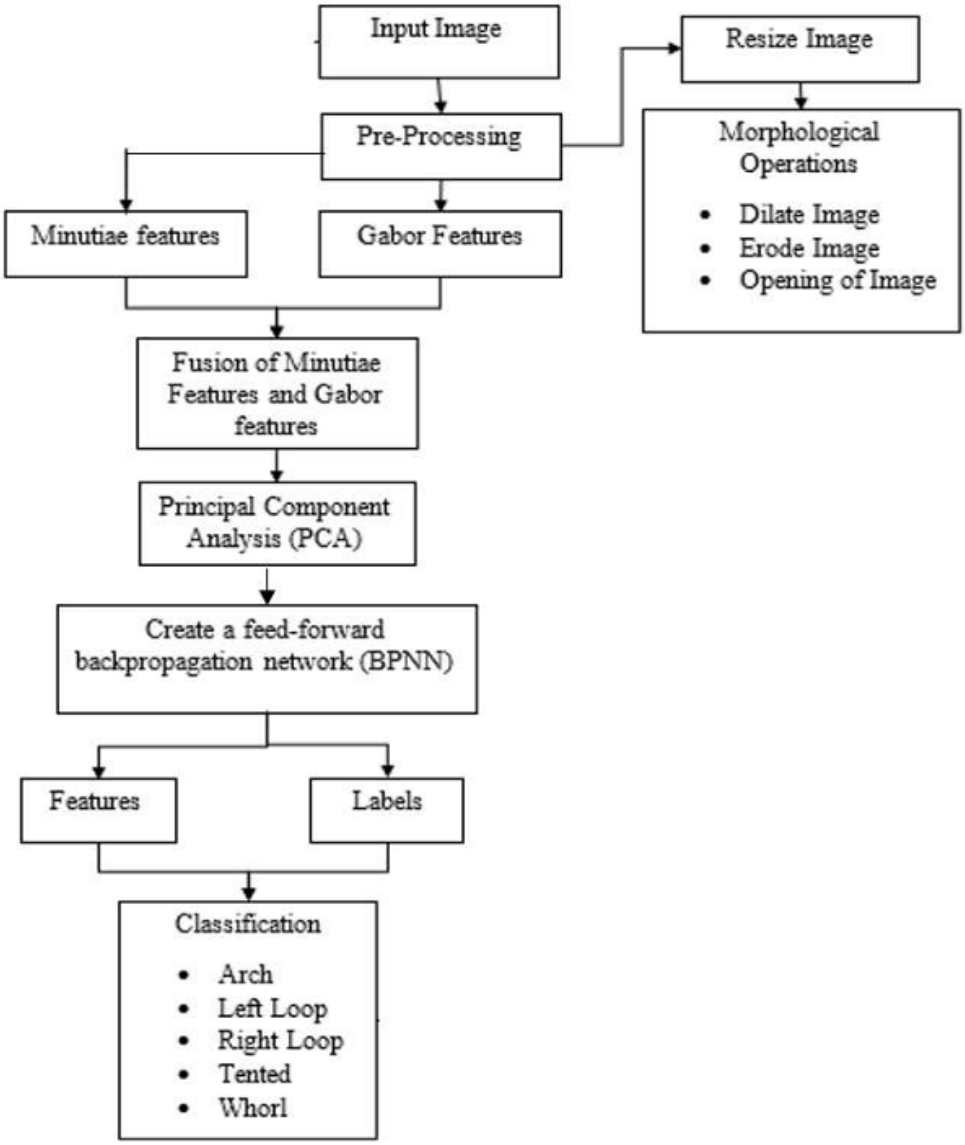


Figure 6.1: Proposed Method Block Diagram

6.4 Image Resize

Image resizing is a crucial preprocessing step in computer vision and image processing tasks, particularly when dealing with datasets containing images of varying sizes. When images are resized to a standard size, such as 227x227 pixels, it ensures consistency in the input dimensions across the dataset, which is essential for training deep learning models like convolutional neural networks, Back Propagation Neural Network etc..,. Resizing involves altering the dimensions of an image while preserving its aspect ratio, which means scaling it proportionally to fit within the specified dimensions. In the context of 227x227 resizing, images are adjusted to fit within a square frame with each side measuring 227 pixels. This standardization facilitates efficient model training and inference by reducing computational overhead and memory requirements. However, it's important to consider the potential loss of information or distortion that may occur during resizing, which can impact the performance of the model, especially in tasks where fine details are crucial. Therefore, choosing an appropriate resizing technique and understanding its effects on the data are vital aspects of preprocessing pipelines in image-based machine learning applications [4].

6.5 Morphological Operations

Morphological operations are fundamental techniques in image processing that manipulate the shape or morphology of objects within an image. These operations are particularly useful for tasks such as noise reduction, edge detection, and object segmentation.Morphological operations are commonly employed in fingerprint image processing for tasks such as fingerprint enhancement, ridge extraction, and minutiae detection. The two primary morphological operations are erosion and dilation.

- **Erosion:** This operation can be used to remove noise and thin out the ridges of the fingerprint. By applying erosion, you can reduce the thickness of the ridges while preserving their structure. This can help in obtaining clearer ridge patterns.

- **Dilation:** Dilation can be used to thicken the ridges and close small gaps in the fingerprint

image. This operation helps in enhancing the ridge structures and making them more prominent.

- **Opening:** Opening operation, which combines erosion followed by dilation, can be used to remove small noise while preserving the ridge structures

By appropriately applying these morphological operations, you can preprocess fingerprint images to enhance their quality, extract relevant features, and facilitate fingerprint recognition and matching tasks.

6.6 Gabor Features

Gabor features serve as pivotal elements in fingerprint identification systems, harnessing complex sinusoidal waveforms modulated by Gaussian envelopes to extract texture patterns from fingerprint images. Through a filter bank, each Gabor filter is meticulously tuned to specific frequencies and orientations, facilitating the capture of localized texture information at varying scales and directions. Following the application of this filter bank, Gabor features are derived from the filtered images, encapsulating distinctive texture characteristics crucial for fingerprint identification. Subsequently, statistical measures such as mean, variance, energy, and entropy are computed from these features, enabling efficient texture analysis. During the identification process, the extracted Gabor features from a query fingerprint are compared with those from stored fingerprints in the database, utilizing similarity measures like Euclidean distance or cosine similarity for feature matching. Ultimately, Gabor features play a pivotal role in accurately identifying individuals through their fingerprints, ensuring robustness and reliability in fingerprint identification systems [13].

6.7 Minutiae Features

Minutiae features are key elements in fingerprint recognition systems, representing specific points where ridges in a fingerprint terminate, bifurcate, or encounter singularities like ridge endings and ridge bifurcations. These minutiae points serve as unique identifiers for fingerprint matching and verification. Minutiae extraction involves detecting these points

within the fingerprint image, typically achieved through algorithms that analyze local ridge structures and orientations. The extracted minutiae features are then stored in a database alongside additional information such as their coordinates, ridge orientations, and types (e.g., ending or bifurcation). During fingerprint matching, the presence and spatial arrangement of minutiae features are compared between the query fingerprint and the reference fingerprints in the database, employing techniques such as alignment, rotation, and translation to achieve accurate matching. Minutiae features are highly reliable for fingerprint recognition due to their uniqueness and robustness against variations in fingerprint quality and acquisition conditions [13].

6.8 Fused Features of Gabour and Minutiae Features

The fusion of Gabor and minutiae features combines the strengths of both approaches to enhance the performance and robustness of fingerprint recognition systems. By integrating texture information captured by Gabor features with the distinctive minutiae points, the fused feature set provides a comprehensive representation of the fingerprint, encompassing both global and local characteristics. This fusion approach leverages the discriminative power of Gabor features in capturing texture patterns and the unique identifiers offered by minutiae points. During feature extraction, Gabor features are extracted from the fingerprint image using a filter bank, while minutiae points are detected through ridge analysis. The two feature sets are then combined, either by concatenating them into a single feature vector or employing more sophisticated fusion techniques such as feature level fusion or decision level fusion. This fused feature representation offers improved performance in fingerprint matching tasks, enhancing accuracy, and robustness by capturing complementary information from both texture and structural characteristics of fingerprints. Fusing Gabor and minutiae features enables fingerprint recognition systems to achieve higher accuracy rates and better resilience against variations in fingerprint quality and environmental conditions.

6.9 Feature Selection Through PCA

Feature selection through Principal Component Analysis (PCA) involves transforming the original feature space into a new set of orthogonal variables known as principal components (PCs), which capture the maximum variance in the data. By retaining only a subset of these principal components, typically determined by the cumulative explained variance, dimensionality reduction is achieved while preserving most of the information present in the dataset. In parallel, kurtosis, a statistical measure describing the distribution's tail extremity, is utilized to evaluate the importance of individual features based on their distribution shapes. Features exhibiting high kurtosis values tend to possess more discriminative information or outliers. Integrating PCA with kurtosis-based feature selection allows for the identification and retention of features that contribute significantly to the dataset's variance and discrimination power while discarding less relevant or noisy features. This combined approach enhances the robustness and efficiency of feature selection processes, making it particularly effective in tasks such as fingerprint recognition systems where extracting meaningful and discriminative features is crucial for accurate identification.

6.10 Create a Feed-Forward Backpropagation Network

In the context of fingerprint classification, a feedforward backpropagation neural network (BPNN) with transig (sigmoid) activation function, comprising one input node, two hidden layers, and five output nodes, facilitates the categorization of fingerprint types such as left loop, right loop, whorl, tented, and arch. This architecture is designed to process fingerprint images or feature vectors, with the input node representing the features extracted from each fingerprint. The network's hidden layers, each with multiple nodes, serve as intermediary processing units that learn hierarchical representations of the input data. The transig activation function, commonly known as the sigmoid function, introduces nonlinearity to the network, enabling it to capture complex relationships between input features and output classes. The five output nodes correspond to the different fingerprint types, with each node outputting a probability score indicating the likelihood of the input fingerprint

belonging to that particular type. During training, the network undergoes backpropagation, where the error between the predicted and actual outputs is propagated backwards through the network, and the weights and biases are adjusted accordingly using gradient descent optimization. With 300 epochs of training, the network iteratively learns to minimize the classification error, thereby improving its ability to accurately classify fingerprints into the predefined categories [4].

6.11 Optimization Through Levenberg-Marquardt algorithm

The Levenberg-Marquardt algorithm (LM algorithm) is an optimization method used for solving non-linear least squares problems. This algorithm combines aspects of both gradient descent and Gauss-Newton methods to efficiently find the minimum of a sum of squared residuals. The LM algorithm iteratively adjusts model parameters to minimize the difference between observed and predicted values. It dynamically adjusts the step size during optimization, using a combination of gradient information and a damping parameter to control the update process. This damping parameter ensures robustness and stability, preventing large step sizes that may lead to divergence. The algorithm starts with an initial guess for the parameters and iterates until convergence is achieved or a predefined stopping criterion is met. During each iteration, it computes the gradient and the Hessian matrix (approximating the second derivative of the objective function) to determine the direction and magnitude of the parameter updates. The damping parameter balances between the steepest descent direction and the Gauss-Newton direction, effectively adapting to the local curvature of the objective function surface.

7

RESULTS

7.1 Data set

NIST(National Institute of Standards and Technology) is a well-known dataset of Fingerprint Images, often used as a benchmark for image classification tasks.The dataset consists of a total of 125 images, of which 25 are for whorl, 25 are for right loop, 25 are for left loop, 25 are for arch, and 25 are for tented, which are black and white images. Out of these, 70% is used for training the classifier, and 30% is used for testing. Each image in the NIST dataset with dimensions of different pixels.

7.2 Input Image

The images are labeled with the corresponding images of different types of input images are represented in Figure 7.1. Each image in the NIST dataset is represented as a matrix of grayscale values, where each element of the matrix represents the grayscale value of the corresponding pixel in the image. The grayscale values range from 0 to 255, where 0 represents black and 255 represents white as shown in below Figures (7.1) (7.2) (7.3) (7.4) (7.5) (7.6) (7.7) (7.8) (7.9) and (7.10). [11]

Figure 7.1: Whorl Input Image

Figure 7.2: Arch Input Image

Figure 7.3: Left Loop Image

Figure 7.4: Right Loop Input Image

Figure 7.5: Tented Input Image

image | Resize_image | dilated_image | eroded_image | phase | minutiae_features

374x388 uint8

	163	164	165	166	167	168	169	170	171	172	173	174	175
1	254	254	254	254	254	254	254	254	254	254	254	254	254
2	254	254	254	254	254	254	254	254	254	254	254	254	254
3	254	254	254	254	254	254	254	254	254	254	254	254	254
4	254	254	254	254	254	254	254	254	254	254	254	254	254
5	254	254	254	254	254	254	254	254	254	254	254	254	254
6	254	254	254	254	254	254	254	254	254	254	254	254	254
7	254	254	254	254	254	254	254	254	254	254	254	254	254
8	254	254	254	254	254	254	254	254	254	254	254	254	254
9	254	254	254	254	254	254	254	254	254	254	254	254	254
10	254	254	254	254	254	254	254	254	254	254	254	254	254
11	254	254	254	254	254	254	254	254	254	254	254	254	254
12	254	254	254	254	254	254	254	254	254	254	254	254	254
13	254	254	254	254	254	254	254	254	254	254	254	254	254
14	254	254	254	254	254	254	254	254	254	254	254	254	254
15	254	254	254	254	254	254	254	254	254	254	254	254	254
16	254	254	254	254	254	254	254	254	254	254	254	254	254
17	254	254	254	254	254	254	254	254	254	254	254	254	254
18	254	254	254	254	254	254	254	254	254	254	254	254	254
19	254	254	254	254	254	254	254	254	254	254	254	254	254
20	254	254	254	251	249	254	254	254	254	254	254	254	254
21	249	232	202	176	177	203	235	234	221	230	237	233	234
22	240	219	193	169	160	179	226	245	234	240	251	254	254
23	184	175	178	179	182	191	214	235	239	246	254	254	254
24	228	222	226	236	245	253	254	254	254	254	254	254	254

Figure 7.6: 374x388 Pixel Image of Whorl Input Image

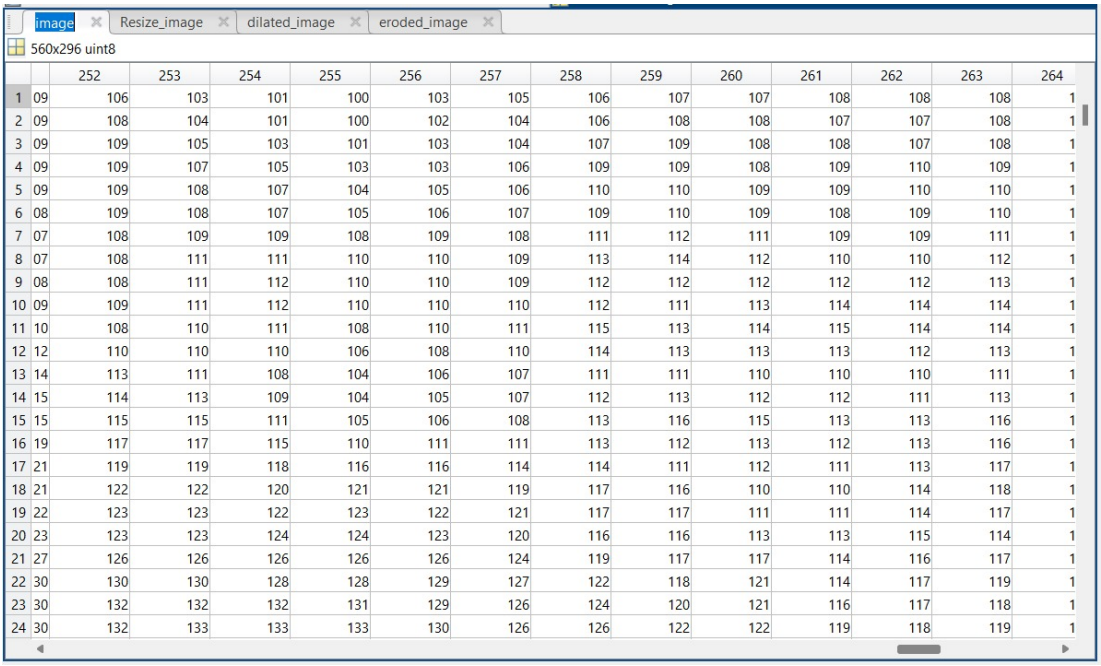


image | Resize_image | dilated_image | eroded_image

560x296 uint8

		252	253	254	255	256	257	258	259	260	261	262	263	264
1	09	106	103	101	100	103	105	106	107	107	108	108	108	1
2	09	108	104	101	100	102	104	106	108	108	107	107	108	1
3	09	109	105	103	101	103	104	107	109	108	108	107	108	1
4	09	109	107	105	103	103	106	109	109	108	109	110	109	1
5	09	109	108	107	104	105	106	110	110	109	109	110	110	1
6	08	109	108	107	105	106	107	109	110	109	108	109	110	1
7	07	108	109	109	108	109	108	111	112	111	109	109	111	1
8	07	108	111	111	110	110	109	113	114	112	110	110	112	1
9	08	108	111	112	110	110	109	112	112	112	112	112	113	1
10	09	109	111	112	110	110	110	112	111	113	114	114	114	1
11	10	108	110	111	108	110	111	115	113	114	115	114	114	1
12	12	110	110	110	106	108	110	114	113	113	113	112	113	1
13	14	113	111	108	104	106	107	111	111	110	110	110	111	1
14	15	114	113	109	104	105	107	112	113	112	112	111	113	1
15	15	115	115	111	105	106	108	113	116	115	113	113	116	1
16	19	117	117	115	110	111	111	113	112	113	112	113	116	1
17	21	119	119	118	116	116	114	114	111	112	111	113	117	1
18	21	122	122	120	121	121	119	117	116	110	110	114	118	1
19	22	123	123	122	123	122	121	117	117	111	111	114	117	1
20	23	123	123	124	124	123	120	116	116	113	113	115	114	1
21	27	126	126	126	126	126	124	119	117	117	114	116	117	1
22	30	130	130	128	128	129	127	122	118	121	114	117	119	1
23	30	132	132	132	131	129	126	124	120	121	116	117	118	1
24	30	132	133	133	133	130	126	126	122	122	119	118	119	1

Figure 7.7: 560x296 Pixel Image of Arch Input Image

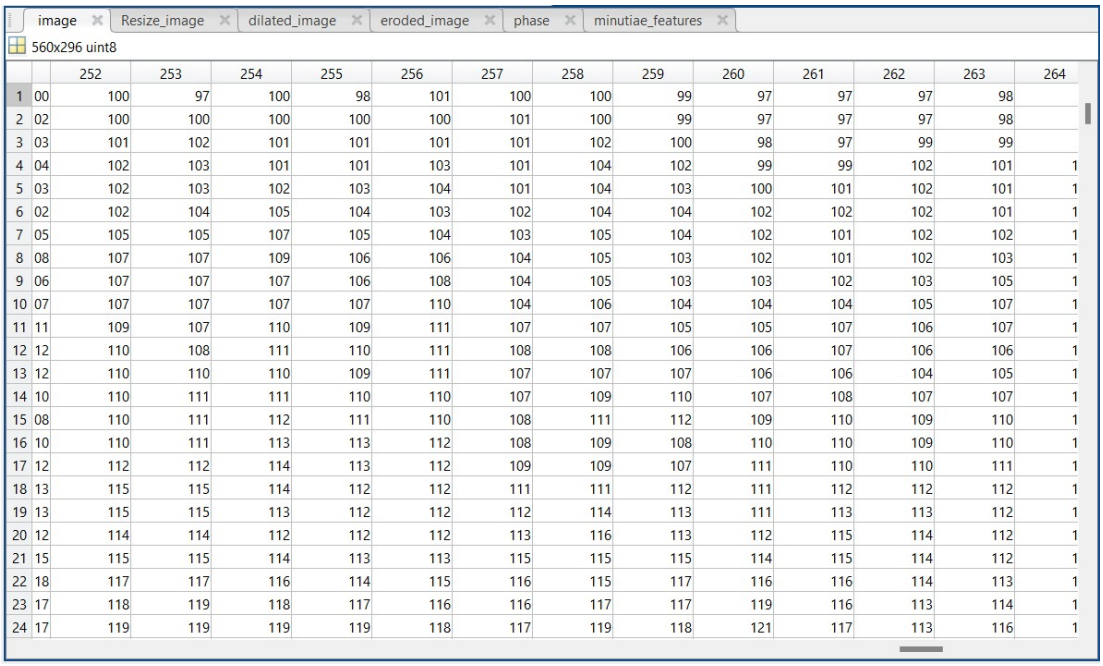


image | Resize_image | dilated_image | eroded_image | phase | minutiae_features

560x296 uint8

		252	253	254	255	256	257	258	259	260	261	262	263	264
1	00	100	97	100	98	101	100	100	99	97	97	97	98	
2	02	100	100	100	100	100	101	100	99	97	97	97	98	
3	03	101	102	101	101	101	101	102	100	98	97	99	99	
4	04	102	103	101	101	103	101	104	102	99	99	102	101	1
5	03	102	103	102	103	104	101	104	103	100	101	102	101	1
6	02	102	104	105	104	103	102	104	104	102	102	102	101	1
7	05	105	105	107	105	104	103	105	104	102	101	102	102	1
8	08	107	107	109	106	106	104	105	103	102	101	102	103	1
9	06	107	107	107	106	108	104	105	103	103	102	103	105	1
10	07	107	107	107	107	110	104	106	104	104	104	105	107	1
11	11	109	107	110	109	111	107	107	105	105	107	106	107	1
12	12	110	108	111	110	111	108	108	106	106	107	106	106	1
13	12	110	110	110	109	111	107	107	107	106	106	104	105	1
14	10	110	111	111	110	110	107	109	110	107	108	107	107	1
15	08	110	111	112	111	110	108	111	112	109	110	109	110	1
16	10	110	111	113	113	112	108	109	108	110	110	109	110	1
17	12	112	112	114	113	112	109	109	107	111	110	110	111	1
18	13	115	115	114	112	112	111	111	112	111	112	112	112	1
19	13	115	115	113	112	112	112	114	113	111	113	113	112	1
20	12	114	114	112	112	112	113	116	113	112	115	114	112	1
21	15	115	115	114	113	113	115	115	115	114	115	114	112	1
22	18	117	117	116	114	115	116	115	117	116	116	114	113	1
23	17	118	119	118	117	116	116	117	117	119	116	113	114	1
24	17	119	119	119	119	118	117	119	118	121	117	113	116	1

Figure 7.8: 560x296 Pixel Image of Left Loop Input Image

image | Resize_image | dilated_image | eroded_image | phase | minutiae_features

374x388 uint8

	163	164	165	166	167	168	169	170	171	172	173	174	175
1	254	247	207	155	115	94	85	80	88	105	129	169	220
2	254	254	252	232	201	163	123	103	100	96	95	116	160
3	254	254	254	254	239	205	162	132	117	105	97	104	132
4	254	254	254	254	254	254	237	206	166	127	100	89	91
5	243	254	254	254	254	254	253	236	204	160	117	91	85
6	185	222	241	250	254	254	254	254	252	226	180	132	108
7	148	183	205	221	238	252	254	254	254	245	212	168	141
8	108	110	116	130	153	188	226	254	254	254	254	243	225
9	110	100	99	102	115	143	188	234	254	254	254	254	246
10	167	130	108	95	86	93	125	173	214	241	254	254	254
11	208	170	143	122	99	89	105	139	174	207	235	254	254
12	254	250	232	192	144	107	87	84	96	120	160	208	245
13	254	254	253	231	185	132	93	76	77	90	121	168	213
14	254	254	254	254	240	198	147	111	96	99	108	121	143
15	241	254	254	254	254	230	189	153	136	132	130	126	123
16	220	249	254	254	254	254	253	244	239	236	228	201	153
17	208	228	235	240	251	254	254	254	254	254	253	230	182
18	231	202	176	175	200	234	254	254	254	254	254	250	216
19	252	221	177	149	156	191	230	254	254	254	254	254	246
20	254	244	204	154	126	133	166	204	235	254	254	254	254
21	254	254	235	198	158	131	128	150	187	222	247	254	254
22	254	254	254	248	223	176	129	110	121	148	191	236	254
23	254	254	254	254	249	212	167	139	128	127	153	203	245
24	254	254	254	254	254	253	243	226	195	158	142	162	194

Figure 7.9: 374x388 Pixel Image of Right Loop Input Image

image

560x296 uint8

	89	90	91	92	93	94	95	96	97	98	99	100	101
1	49	54	62	71	78	85	82	74	68	70	75	82	89
2	62	66	73	82	88	92	90	84	78	81	86	93	98
3	75	79	85	92	97	98	97	93	89	92	96	101	105
4	89	94	97	100	104	105	103	100	99	102	103	106	108
5	98	101	103	104	107	107	104	102	100	102	103	104	105
6	103	106	106	107	110	106	103	101	97	98	99	99	99
7	103	105	105	103	103	100	96	91	89	88	89	90	90
8	101	101	101	99	95	93	87	81	80	78	78	79	79
9	90	89	90	88	85	79	74	66	67	67	66	66	67
10	77	76	78	77	74	68	63	55	59	61	61	61	62
11	61	62	66	66	64	59	56	52	58	63	67	69	70
12	52	55	61	65	64	60	60	60	66	73	79	82	82
13	47	51	61	68	70	67	71	74	79	87	94	96	96
14	54	58	66	75	78	78	83	88	91	97	103	105	105
15	63	66	72	82	86	89	94	101	102	106	109	112	112
16	77	79	82	88	91	98	103	107	106	107	108	108	106
17	91	91	92	95	97	105	108	109	106	105	104	101	98
18	105	104	104	105	106	107	106	105	100	97	95	89	83
19	106	105	105	108	107	105	101	96	87	82	77	71	67
20	101	99	101	107	102	100	93	83	70	61	55	50	50
21	86	83	86	92	91	90	82	70	59	53	52	52	57
22	69	66	71	77	81	82	74	61	51	49	53	60	68
23	56	56	64	74	86	89	83	71	64	64	69	77	87

Figure 7.10: 560x296 Pixel Image of Tented Input Image

7.3 Resized Images

Resizing an image involves changing its dimensions, either increasing or decreasing its size.This process is commonly performed in image processing for various reasons, such as preparing data for a specific algorithm, sizes, or reducing computational complexity. Resized and pixel image as shown in Figure (7.11)(7.12)(7.13)(7.14)(7.15) and (7.16).

Figure 7.11: Resized Images

image | Resize_image | dilated_image | eroded_image | phase | minutiae_features

227x227 uint8

	65	66	67	68	69	70	71	72	73	74	75	76	77
41	177	205	216	204	207	222	244	255	255	255	252	240	199
42	221	246	254	255	255	255	251	244	247	225	194	173	138
43	252	255	255	255	248	226	183	153	152	143	134	166	194
44	255	255	254	249	206	142	146	143	111	140	169	218	250
45	254	235	209	179	143	125	166	206	210	227	237	254	242
46	224	156	137	128	127	173	219	247	250	254	255	255	222
47	168	137	126	143	203	242	252	211	194	237	255	254	247
48	164	172	186	209	254	255	254	216	204	225	239	207	175
49	208	226	251	255	254	242	230	221	178	167	217	166	115
50	250	255	254	251	215	149	121	124	122	136	206	223	202
51	252	231	190	185	144	86	98	123	165	190	219	254	245
52	170	171	140	129	140	138	191	225	235	236	223	192	159
53	114	159	168	185	228	236	244	218	175	151	139	150	147
54	178	210	229	243	236	204	173	132	129	156	178	219	236
55	255	244	206	173	147	114	118	141	160	223	253	241	214
56	219	182	135	124	131	148	188	230	246	242	205	149	113
57	146	169	192	203	212	241	255	252	235	186	139	115	118
58	170	225	255	255	254	252	247	202	159	137	176	202	200
59	242	254	243	215	196	208	228	200	199	199	237	210	162
60	237	218	179	148	152	187	240	247	255	255	241	161	141
61	182	189	159	161	183	222	253	255	255	255	237	189	185
62	186	228	221	240	248	252	246	232	212	198	158	110	100
63	239	254	244	249	237	199	164	148	132	126	109	97	89
64	251	203	173	214	172	144	133	117	126	157	178	195	201

Figure 7.12: 277x277 Pixel Image of whorl Resized Image

image | Resize_image | dilated_image | eroded_image

227x227 uint8

	150	151	152	153	154	155	156	157	158	159	160	161	162
1	116	117	118	120	118	116	117	119	120	119	116	116	118
2	122	122	121	121	121	119	119	121	120	119	117	118	120
3	126	125	124	125	125	123	121	121	121	118	117	119	121
4	126	127	128	127	125	126	125	123	120	119	122	122	123
5	129	129	128	128	126	127	128	126	123	124	124	125	124
6	132	130	129	128	126	126	127	127	126	126	129	128	128
7	134	134	134	132	130	127	125	127	126	127	130	131	132
8	133	134	133	132	133	130	130	130	129	131	131	132	132
9	134	135	134	133	135	134	134	134	134	134	132	132	133
10	137	136	135	135	136	136	136	137	138	137	134	135	134
11	140	138	139	136	138	139	138	138	138	139	140	139	137
12	138	140	141	142	143	141	138	137	139	140	140	142	141
13	140	140	142	143	145	143	141	141	139	139	139	141	142
14	144	143	143	144	145	144	145	145	145	143	140	140	141
15	146	145	145	147	146	144	146	147	147	146	146	145	144
16	146	147	147	148	148	147	147	144	144	146	146	147	146
17	148	148	148	149	150	149	149	146	146	147	146	148	148
18	149	148	149	149	151	151	150	148	150	151	149	150	149
19	154	152	151	151	151	149	148	152	153	150	150	151	150
20	154	151	150	152	150	150	149	151	152	151	152	152	152
21	153	152	151	151	150	152	154	153	152	153	153	153	151
22	145	150	151	151	152	153	154	156	155	155	154	154	154
23	123	140	145	147	149	150	152	154	153	152	152	154	156
24	123	137	147	145	147	149	150	150	149	148	146	147	153

Figure 7.13: 277x277 Pixel Image of Arch Resized Image

image | Resize_image | dilated_image | eroded_image | phase | minutiae_features

227x227 uint8

	150	151	152	153	154	155	156	157	158	159	160	161	162
1	109	110	109	108	106	106	107	105	106	106	106	105	107
2	111	110	110	110	111	111	111	109	110	109	111	110	108
3	112	112	111	113	116	115	112	110	112	112	112	112	112
4	114	115	116	117	115	117	114	112	113	114	115	113	114
5	117	117	116	116	115	116	116	114	112	113	116	116	115
6	119	117	117	116	115	116	115	115	112	114	116	117	116
7	121	122	120	119	118	117	117	117	115	116	117	118	117
8	124	122	123	122	121	120	120	119	118	117	120	120	121
9	126	125	124	125	123	123	122	119	118	118	122	122	123
10	127	125	126	127	126	125	124	122	123	124	125	125	125
11	129	128	129	126	126	126	127	125	127	129	128	127	126
12	129	129	130	130	132	131	129	128	129	131	130	128	127
13	130	131	133	132	134	133	131	133	131	132	131	129	129
14	133	134	134	134	133	131	132	135	134	131	133	132	131
15	136	135	134	134	132	130	133	136	134	135	135	133	134
16	135	137	137	135	133	133	137	137	134	136	134	136	134
17	136	138	140	138	138	137	137	137	134	135	137	136	136
18	138	140	140	140	140	140	139	136	132	135	139	138	138
19	141	142	143	142	141	141	139	139	137	137	139	136	135
20	142	145	146	143	143	144	142	140	141	140	142	139	139
21	145	144	147	145	145	146	145	144	144	144	144	141	142
22	145	145	148	147	147	147	147	146	145	146	145	144	143
23	147	147	148	148	148	147	146	147	146	145	145	145	141
24	148	149	148	146	147	148	147	149	149	148	147	147	145

Figure 7.14: 277x277 Pixel Image of Left Loop Resized Image

image | Resize_image

227x227 uint8

	83	84	85	86	87	88	89	90	91	92	93	94	95
25	124	131	134	121	108	101	106	115	114	110	112	112	116
26	77	95	109	87	63	53	54	60	60	61	67	68	71
27	62	83	100	95	81	66	55	46	46	53	62	67	72
28	113	125	133	135	130	119	106	96	96	104	114	118	124
29	129	124	123	119	114	110	103	97	97	100	102	104	110
30	77	70	69	61	56	57	56	55	57	58	54	60	67
31	67	68	72	70	66	71	84	87	92	91	90	98	107
32	123	125	126	127	125	127	139	143	145	144	146	145	141
33	126	131	131	132	133	129	122	118	121	126	123	112	96
34	68	76	82	82	76	71	61	57	64	76	84	77	71
35	58	61	68	66	58	61	63	68	80	99	120	121	118
36	111	112	112	108	106	115	121	127	137	142	149	151	149
37	149	141	129	121	123	124	118	120	124	118	110	107	111
38	126	95	70	65	79	80	60	63	77	84	71	68	79
39	98	73	56	55	73	82	68	78	103	121	117	125	138
40	134	120	108	104	108	116	123	137	146	150	152	157	158
41	140	134	127	120	112	107	103	102	102	100	104	120	125
42	80	79	77	75	74	63	63	65	68	71	78	93	105
43	81	84	95	111	114	114	123	127	125	129	135	139	144
44	144	145	149	154	151	150	154	156	157	153	148	141	136
45	141	138	125	111	99	100	108	114	113	99	89	82	73
46	82	84	77	63	64	84	103	107	101	90	78	73	69
47	81	96	113	119	123	141	152	149	143	141	134	128	124

Figure 7.15: 277x277 Pixel Image of Right Loop Resized Image

image | Resize_image | dilated_image | eroded_image | phase | minutiae_features

227x227 uint8

	65	66	67	68	69	70	71	72	73	74	75	76	77
41	209	206	206	204	195	201	206	207	206	206	206	200	200
42	210	206	205	207	205	207	209	209	207	209	207	201	199
43	209	208	207	209	208	209	212	212	209	210	205	199	203
44	208	209	209	209	209	209	211	210	208	209	205	203	205
45	211	209	209	211	209	208	208	205	206	209	205	207	206
46	212	210	211	212	211	211	209	204	204	206	203	206	210
47	210	207	208	211	209	209	209	210	211	211	208	207	210
48	211	202	205	210	209	211	210	211	213	215	214	212	209
49	209	211	213	211	211	216	214	212	216	215	212	212	211
50	210	213	215	214	211	213	215	215	216	216	214	213	216
51	215	214	215	216	216	216	217	216	215	217	215	217	218
52	215	212	215	218	216	214	214	212	211	215	214	218	218
53	212	215	218	217	214	215	216	218	218	214	217	218	218
54	217	218	218	219	216	216	219	219	222	219	219	218	220
55	218	219	217	215	217	217	218	220	218	220	220	215	219
56	220	220	221	218	216	216	218	216	213	215	219	213	214
57	222	220	220	220	219	217	218	215	212	211	220	222	219
58	222	216	214	218	220	220	220	219	220	221	222	221	220
59	222	220	219	219	215	219	219	219	222	220	222	221	219
60	215	216	214	215	213	216	217	221	223	223	225	224	223
61	219	219	216	220	222	221	222	222	223	224	225	224	222
62	221	224	225	218	208	220	226	224	225	223	225	226	224
63	225	225	223	218	214	222	226	227	227	224	226	226	226
64	223	222	224	222	226	225	224	223	225	224	224	223	226

Figure 7.16: 277x277 Pixel Image of Tented Resized Image

7.4 Dilated Images

It was mainly used to grow an object in size by extracting the outer boundaries of the given fingerprint image. Dilation continuously filled the missing pixel of a broken ridge as it added pixels at the boundary of the objects.Dilated and pixel images as shown in Figure (7.17)(7.18)(7.19)(7.20)(7.21) and (7.22).

Figure 7.17: Dilated Image

image | Resize_image | dilated_image | eroded_image | phase | minutiae_features

227x227 uint8

	65	66	67	68	69	70	71	72	73	74	75	76	77
41	246	254	255	255	255	255	255	255	255	255	255	255	255
42	255	255	255	255	255	255	255	255	255	255	255	252	240
43	255	255	255	255	255	255	255	251	247	247	225	250	250
44	255	255	255	255	255	248	226	210	227	237	254	254	254
45	255	255	255	254	249	219	247	250	254	255	255	255	255
46	255	254	235	209	242	252	252	252	254	255	255	255	255
47	254	224	209	254	255	255	255	254	254	255	255	255	255
48	243	251	255	255	255	255	255	254	237	255	255	255	254
49	255	255	255	255	255	255	255	254	225	239	239	239	223
50	255	255	255	255	255	254	242	230	221	219	254	254	254
51	255	255	255	254	251	215	225	235	236	236	254	254	254
52	255	252	231	228	236	244	244	244	236	236	254	254	254
53	213	229	243	243	243	244	244	244	236	236	236	236	249
54	255	255	244	243	243	244	244	244	223	253	253	253	249
55	255	255	244	243	243	236	230	246	246	253	253	253	249
56	255	255	244	212	241	255	255	255	252	253	253	253	241
57	236	255	255	255	255	255	255	255	252	246	242	205	202
58	254	255	255	255	255	255	255	255	252	237	237	237	210
59	254	255	255	255	255	254	252	255	255	255	255	241	210
60	254	254	254	243	222	253	255	255	255	255	255	241	214
61	252	237	240	248	252	253	255	255	255	255	255	241	214
62	254	254	254	249	252	253	255	255	255	255	255	237	214
63	255	254	254	249	252	252	252	246	232	212	198	201	201
64	255	254	254	249	249	237	199	181	235	252	255	255	255

Figure 7.18: 277x277 Pixel Image of Whorl Dilated Image

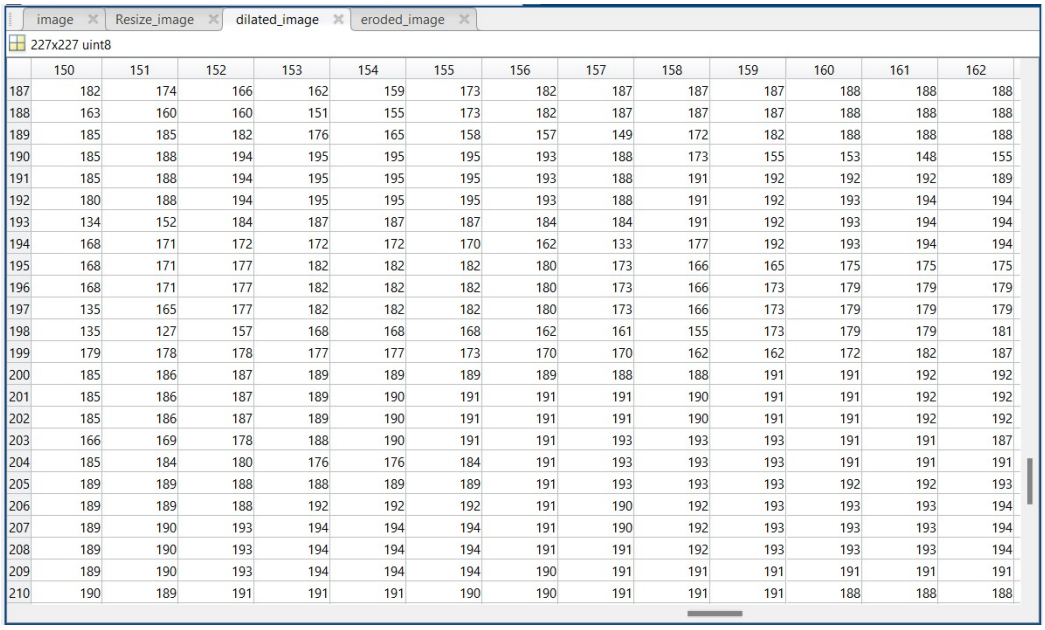
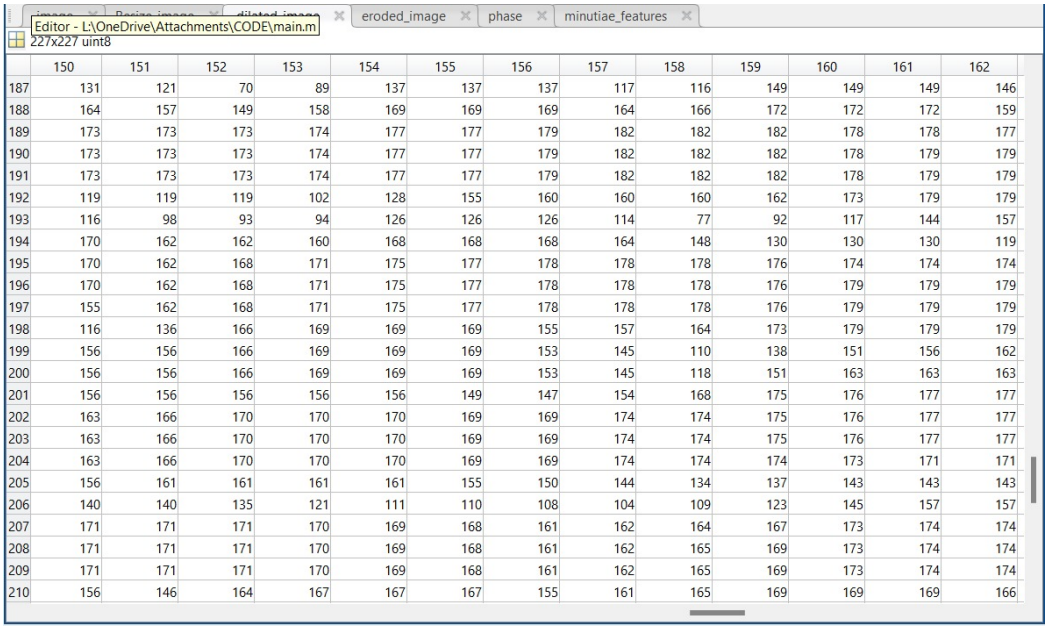


image | Resize_image | dilated_image | eroded_image

227x227 uint8

	150	151	152	153	154	155	156	157	158	159	160	161	162
187	182	174	166	162	159	173	182	187	187	187	188	188	188
188	163	160	160	151	155	173	182	187	187	187	188	188	188
189	185	185	182	176	165	158	157	149	172	182	188	188	188
190	185	188	194	195	195	195	193	188	173	155	153	148	155
191	185	188	194	195	195	195	193	188	191	192	192	192	189
192	180	188	194	195	195	195	193	188	191	192	193	194	194
193	134	152	184	187	187	187	184	184	191	192	193	194	194
194	168	171	172	172	172	170	162	133	177	192	193	194	194
195	168	171	177	182	182	182	180	173	166	165	175	175	175
196	168	171	177	182	182	182	180	173	166	173	179	179	179
197	135	165	177	182	182	182	180	173	166	173	179	179	179
198	135	127	157	168	168	168	162	161	155	173	179	179	181
199	179	178	178	177	177	173	170	170	162	162	172	182	187
200	185	186	187	189	189	189	189	188	188	191	191	192	192
201	185	186	187	189	190	191	191	191	190	191	191	192	192
202	185	186	187	189	190	191	191	191	190	191	191	192	192
203	166	169	178	188	190	191	191	193	193	193	191	191	187
204	185	184	180	176	176	184	191	193	193	193	191	191	191
205	189	189	188	188	189	189	191	193	193	193	192	192	193
206	189	189	188	192	192	192	191	190	192	193	193	193	194
207	189	190	193	194	194	194	191	190	192	193	193	193	194
208	189	190	193	194	194	194	191	191	192	193	193	193	194
209	189	190	193	194	194	194	190	191	191	191	191	191	191
210	190	189	191	191	191	190	190	191	191	191	188	188	188

Figure 7.19: 277x277 Pixel Image of Arch Dilated Image


Editor - L:\OneDrive\Attachments\CODE\main.m | eroded_image | phase | minutiae_features

227x227 uint8

	150	151	152	153	154	155	156	157	158	159	160	161	162
187	131	121	70	89	137	137	137	117	116	149	149	149	146
188	164	157	149	158	169	169	169	164	166	172	172	172	159
189	173	173	173	174	177	177	179	182	182	182	178	178	177
190	173	173	173	174	177	177	179	182	182	182	178	179	179
191	173	173	173	174	177	177	179	182	182	182	178	179	179
192	119	119	119	102	128	155	160	160	160	162	173	179	179
193	116	98	93	94	126	126	126	114	77	92	117	144	157
194	170	162	162	160	168	168	168	164	148	130	130	130	119
195	170	162	168	171	175	177	178	178	178	176	174	174	174
196	170	162	168	171	175	177	178	178	178	176	179	179	179
197	155	162	168	171	175	177	178	178	178	176	179	179	179
198	116	136	166	169	169	169	155	157	164	173	179	179	179
199	156	156	166	169	169	169	153	145	110	138	151	156	162
200	156	156	166	169	169	169	153	145	118	151	163	163	163
201	156	156	156	156	156	149	147	154	168	175	176	177	177
202	163	166	170	170	170	169	169	174	174	175	176	177	177
203	163	166	170	170	170	169	169	174	174	175	176	177	177
204	163	166	170	170	170	169	169	174	174	174	173	171	171
205	156	161	161	161	161	155	150	144	134	137	143	143	143
206	140	140	135	121	111	110	108	104	109	123	145	157	157
207	171	171	171	170	169	168	161	162	164	167	173	174	174
208	171	171	171	170	169	168	161	162	165	169	173	174	174
209	171	171	171	170	169	168	161	162	165	169	173	174	174
210	156	146	164	167	167	167	155	161	165	169	169	169	166

Figure 7.20: 277x277 Pixel Image of Left Loop Dilated Image

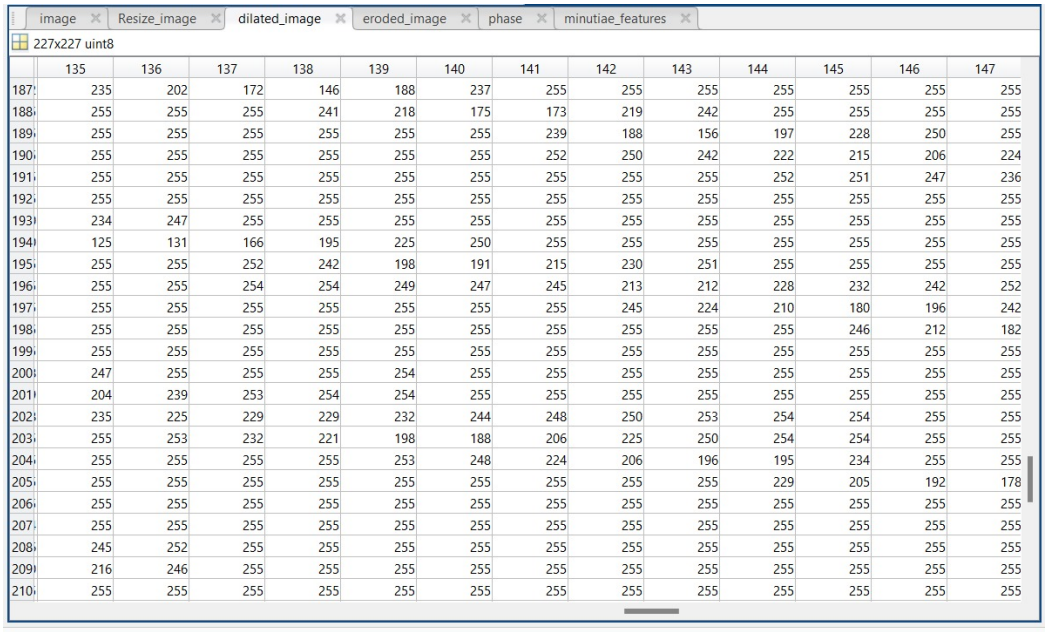


image | Resize_image | dilated_image | eroded_image | phase | minutiae_features

227x227 uint8

	135	136	137	138	139	140	141	142	143	144	145	146	147
187	235	202	172	146	188	237	255	255	255	255	255	255	255
188	255	255	255	241	218	175	173	219	242	255	255	255	255
189	255	255	255	255	255	255	239	188	156	197	228	250	255
190	255	255	255	255	255	255	252	250	242	222	215	206	224
191	255	255	255	255	255	255	255	255	255	252	251	247	236
192	255	255	255	255	255	255	255	255	255	255	255	255	255
193	234	247	255	255	255	255	255	255	255	255	255	255	255
194	125	131	166	195	225	250	255	255	255	255	255	255	255
195	255	255	252	242	198	191	215	230	251	255	255	255	255
196	255	255	254	254	249	247	245	213	212	228	232	242	252
197	255	255	255	255	255	255	255	245	224	210	180	196	242
198	255	255	255	255	255	255	255	255	255	255	246	212	182
199	255	255	255	255	255	255	255	255	255	255	255	255	255
200	247	255	255	255	254	255	255	255	255	255	255	255	255
201	204	239	253	254	254	255	255	255	255	255	255	255	255
202	235	225	229	229	232	244	248	250	253	254	254	255	255
203	255	253	232	221	198	188	206	225	250	254	254	255	255
204	255	255	255	255	253	248	224	206	196	195	234	255	255
205	255	255	255	255	255	255	255	255	255	229	205	192	178
206	255	255	255	255	255	255	255	255	255	255	255	255	255
207	255	255	255	255	255	255	255	255	255	255	255	255	255
208	245	252	255	255	255	255	255	255	255	255	255	255	255
209	216	246	255	255	255	255	255	255	255	255	255	255	255
210	255	255	255	255	255	255	255	255	255	255	255	255	255

Figure 7.21: 277x277 Pixel Image of Right Loop Dilated Image

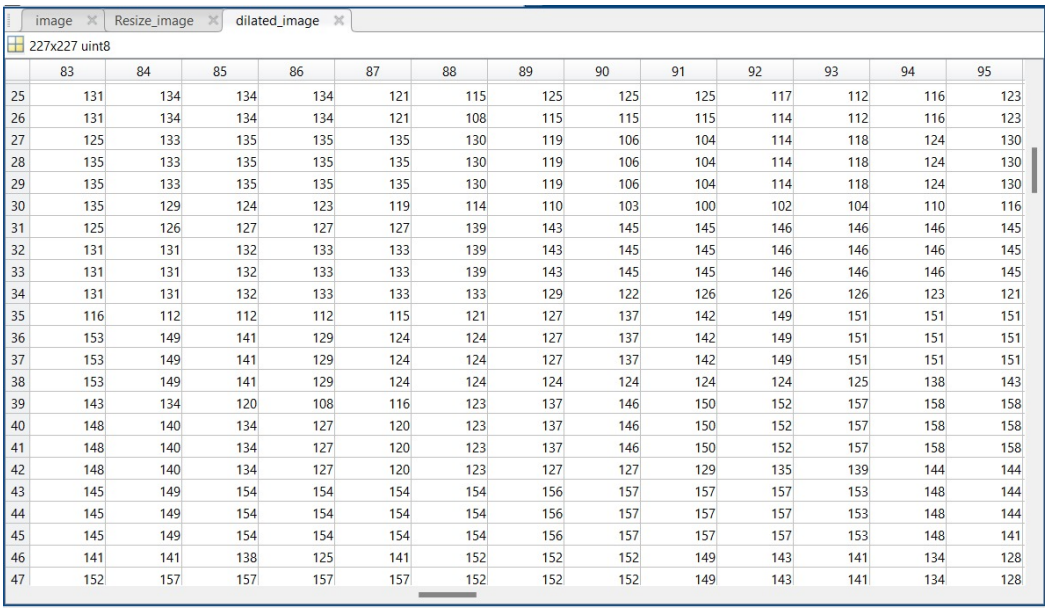


image | Resize_image | dilated_image

227x227 uint8

	83	84	85	86	87	88	89	90	91	92	93	94	95
25	131	134	134	134	121	115	125	125	125	117	112	116	123
26	131	134	134	134	121	108	115	115	115	114	112	116	123
27	125	133	135	135	135	130	119	106	104	114	118	124	130
28	135	133	135	135	135	130	119	106	104	114	118	124	130
29	135	133	135	135	135	130	119	106	104	114	118	124	130
30	135	129	124	123	119	114	110	103	100	102	104	110	116
31	125	126	127	127	127	139	143	145	145	146	146	146	145
32	131	131	132	133	133	139	143	145	145	146	146	146	145
33	131	131	132	133	133	139	143	145	145	146	146	146	145
34	131	131	132	133	133	133	129	122	126	126	126	123	121
35	116	112	112	112	115	121	127	137	142	149	151	151	151
36	153	149	141	129	124	124	127	137	142	149	151	151	151
37	153	149	141	129	124	124	127	137	142	149	151	151	151
38	153	149	141	129	124	124	124	124	124	124	125	138	143
39	143	134	120	108	116	123	137	146	150	152	157	158	158
40	148	140	134	127	120	123	137	146	150	152	157	158	158
41	148	140	134	127	120	123	137	146	150	152	157	158	158
42	148	140	134	127	120	123	127	127	129	135	139	144	144
43	145	149	154	154	154	154	156	157	157	157	153	148	144
44	145	149	154	154	154	154	156	157	157	157	153	148	144
45	145	149	154	154	154	154	156	157	157	157	153	148	141
46	141	141	138	125	141	152	152	152	149	143	141	134	128
47	152	157	157	157	157	152	152	152	149	143	141	134	128

Figure 7.22: 277x277 Pixel Image of Tented Dilated Image

7.5 Eroded Images

This was used as the complement operation of dilation to smooth the fingerprint image after dilation. It caused to loss of size by extracting inner boundaries of a fingerprint edge.Therefore, it can be used to remove the noisy connections between the two objects.Eroded and pixel images as shown in Figure (7.23)(7.24)(7.25)(7.26)(7.27) and (7.28)

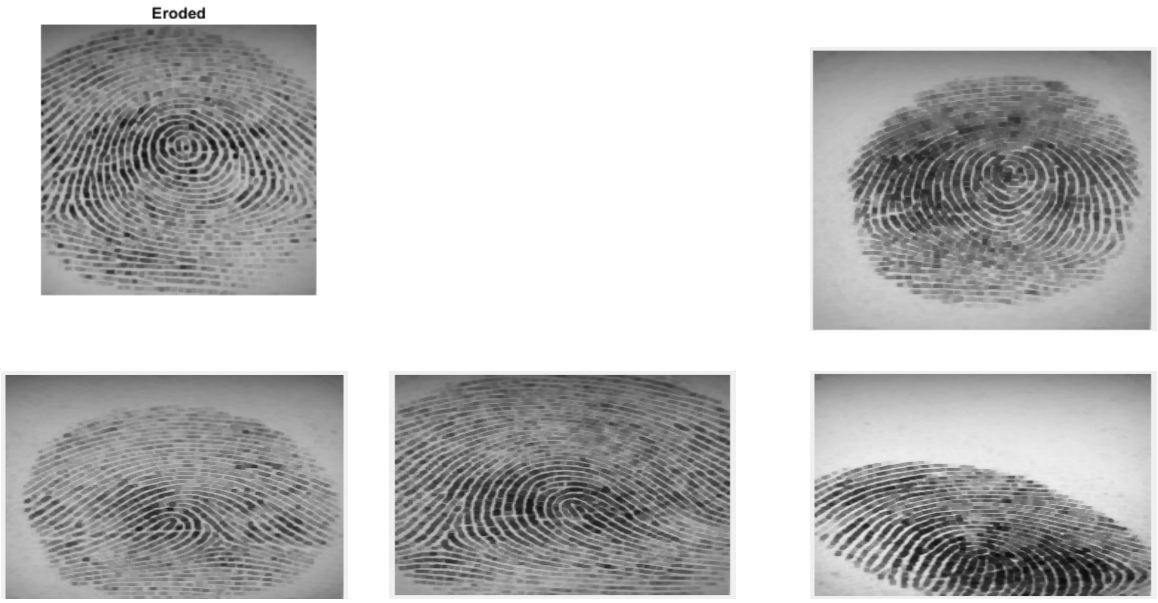
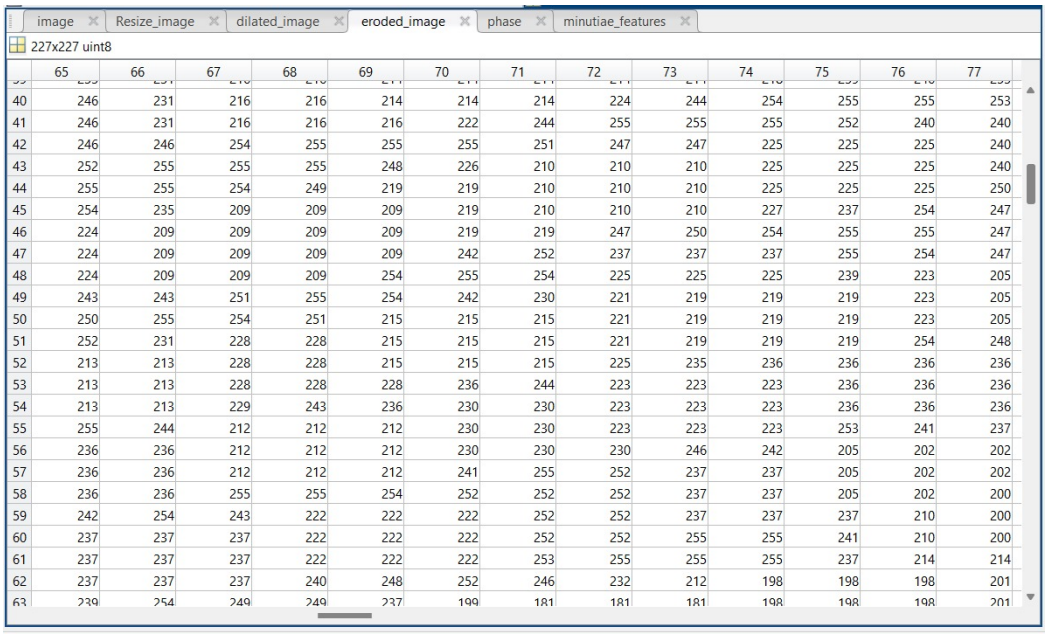
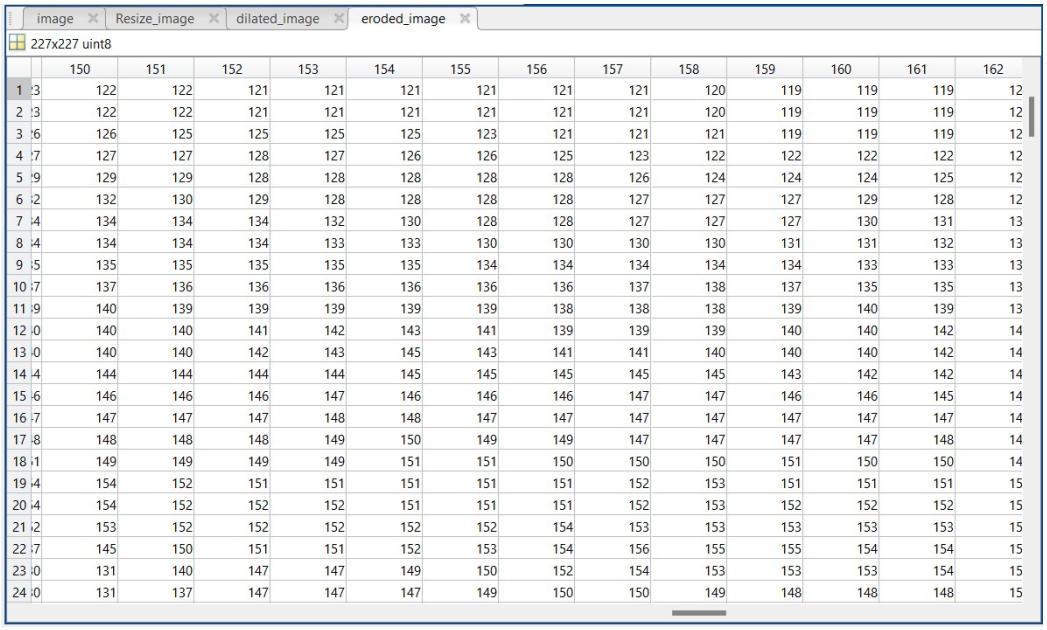


Figure 7.23: Eroded Images



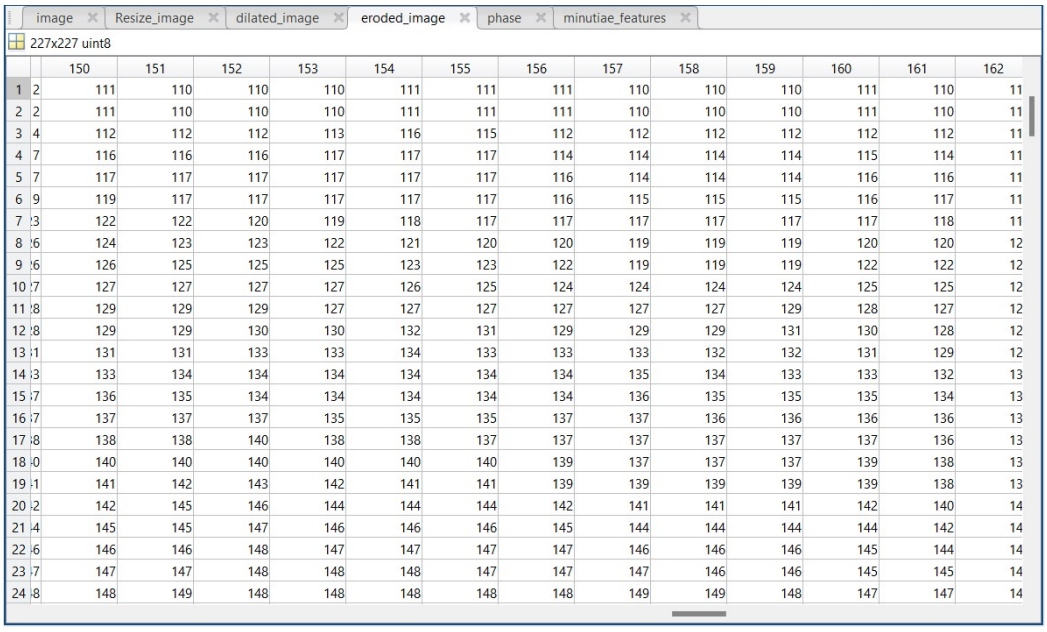
	65	66	67	68	69	70	71	72	73	74	75	76	77
40	246	231	216	216	214	214	214	224	244	254	255	255	253
41	246	231	216	216	216	222	244	255	255	255	252	240	240
42	246	246	254	255	255	255	251	247	247	225	225	225	240
43	252	255	255	255	248	226	210	210	210	225	225	225	240
44	255	255	254	249	219	219	210	210	210	225	225	225	250
45	254	235	209	209	209	219	210	210	210	227	237	254	247
46	224	209	209	209	209	219	219	247	250	254	255	255	247
47	224	209	209	209	209	242	252	237	237	237	255	254	247
48	224	209	209	209	254	255	254	225	225	225	239	223	205
49	243	243	251	255	254	242	230	221	219	219	219	223	205
50	250	255	254	251	215	215	215	221	219	219	219	223	205
51	252	231	228	228	215	215	215	221	219	219	219	254	248
52	213	213	228	228	215	215	215	225	235	236	236	236	236
53	213	213	228	228	228	236	244	223	223	223	236	236	236
54	213	213	229	243	236	230	230	223	223	223	236	236	236
55	255	244	212	212	212	230	230	223	223	223	253	241	237
56	236	236	212	212	212	230	230	230	246	242	205	202	202
57	236	236	212	212	212	241	255	252	237	237	205	202	202
58	236	236	255	255	254	252	252	252	237	237	205	202	200
59	242	254	243	222	222	222	252	252	237	237	237	210	200
60	237	237	237	222	222	222	252	252	255	255	241	210	200
61	237	237	237	222	222	222	253	255	255	255	237	214	214
62	237	237	237	240	248	252	246	232	212	198	198	198	201
63	239	254	249	249	237	199	181	181	181	198	198	198	201

Figure 7.24: 277x277 Pixel Image of Whorl Eroded Image



		150	151	152	153	154	155	156	157	158	159	160	161	162
1	3	122	122	121	121	121	121	121	121	120	119	119	119	12
2	3	122	122	121	121	121	121	121	121	120	119	119	119	12
3	6	126	125	125	125	125	123	121	121	121	119	119	119	12
4	7	127	127	128	127	126	126	125	123	122	122	122	122	12
5	9	129	129	128	128	128	128	128	126	124	124	124	125	12
6	2	132	130	129	128	128	128	128	127	127	127	129	128	12
7	4	134	134	134	132	130	128	128	127	127	127	130	131	13
8	4	134	134	134	133	133	130	130	130	130	131	131	132	13
9	5	135	135	135	135	135	134	134	134	134	134	133	133	13
10	7	137	136	136	136	136	136	136	137	138	137	135	135	13
11	9	140	139	139	139	139	139	138	138	138	139	140	139	13
12	0	140	140	141	142	143	141	139	139	139	140	140	142	14
13	0	140	140	142	143	145	143	141	141	140	140	140	142	14
14	4	144	144	144	144	145	145	145	145	145	143	142	142	14
15	6	146	146	146	147	146	146	146	147	147	146	146	145	14
16	7	147	147	147	148	148	147	147	147	147	147	147	147	14
17	8	148	148	148	149	150	149	149	147	147	147	147	148	14
18	1	149	149	149	149	151	151	150	150	150	151	150	150	14
19	4	154	152	151	151	151	151	151	152	153	151	151	151	15
20	4	154	152	152	152	151	151	151	152	153	152	152	152	15
21	2	153	152	152	152	152	152	154	153	153	153	153	153	15
22	7	145	150	151	151	152	153	154	156	155	155	154	154	15
23	0	131	140	147	147	149	150	152	154	153	153	153	154	15
24	0	131	137	147	147	147	149	150	150	149	148	148	148	15

Figure 7.25: 277x277 Pixel Image of Arch Eroded Image



		150	151	152	153	154	155	156	157	158	159	160	161	162
1	2	111	110	110	110	111	111	111	110	110	110	111	110	11
2	2	111	110	110	110	111	111	111	110	110	110	111	110	11
3	4	112	112	112	113	116	115	112	112	112	112	112	112	11
4	7	116	116	116	117	117	117	114	114	114	114	115	114	11
5	7	117	117	117	117	117	117	116	114	114	114	116	116	11
6	9	119	117	117	117	117	117	116	115	115	115	116	117	11
7	3	122	122	120	119	118	117	117	117	117	117	117	118	11
8	6	124	123	123	122	121	120	120	119	119	119	120	120	12
9	6	126	125	125	125	123	123	122	119	119	119	122	122	12
10	7	127	127	127	127	126	125	124	124	124	124	125	125	12
11	8	129	129	129	127	127	127	127	127	127	129	128	127	12
12	8	129	129	130	130	132	131	129	129	129	131	130	128	12
13	1	131	131	133	133	134	133	133	133	132	132	131	129	12
14	3	133	134	134	134	134	134	134	135	134	133	133	132	13
15	7	136	135	134	134	134	134	134	136	135	135	135	134	13
16	7	137	137	137	135	135	135	137	137	136	136	136	136	13
17	8	138	138	140	138	138	137	137	137	137	137	137	136	13
18	0	140	140	140	140	140	140	139	137	137	137	139	138	13
19	1	141	142	143	142	141	141	139	139	139	139	139	138	13
20	2	142	145	146	144	144	144	142	141	141	141	142	140	14
21	4	145	145	147	146	146	146	145	144	144	144	144	142	14
22	6	146	146	148	147	147	147	147	146	146	146	145	144	14
23	7	147	147	148	148	148	147	147	147	146	146	145	145	14
24	8	148	149	148	148	148	148	148	149	149	148	147	147	14

Figure 7.26: 277x277 Pixel Image of Left Loop Eroded Image

image | Resize_image | dilated_image | eroded_image

227x227 uint8

	83	84	85	86	87	88	89	90	91	92	93	94	95
25	124	131	134	121	108	108	108	115	114	112	112	112	116
26	113	125	133	121	108	108	106	104	104	104	112	112	116
27	113	125	133	121	108	108	106	104	104	104	112	112	116
28	113	125	133	135	130	119	106	104	104	104	114	118	124
29	129	124	123	119	114	110	103	100	100	100	102	104	110
30	123	124	123	119	114	110	103	100	100	100	102	104	110
31	123	124	123	119	114	110	103	100	100	100	102	104	110
32	123	125	126	127	127	127	139	143	145	145	146	145	141
33	128	131	131	132	133	129	122	122	122	126	123	121	118
34	112	112	112	112	112	115	121	122	122	126	123	121	118
35	112	112	112	112	112	115	121	122	122	126	123	121	118
36	112	112	112	112	112	115	121	127	137	142	149	151	150
37	149	141	129	124	124	124	124	124	124	124	124	125	138
38	134	120	108	108	108	116	123	124	124	124	124	125	138
39	134	120	108	108	108	116	123	124	124	124	124	125	138
40	134	120	108	108	108	116	123	137	146	150	152	157	158
41	140	134	127	120	120	120	123	127	127	129	135	139	144
42	140	134	127	120	120	120	123	127	127	129	135	139	144
43	140	134	127	120	120	120	123	127	127	129	135	139	144
44	144	145	149	154	154	154	154	156	157	153	148	141	141
45	141	138	125	125	125	141	152	149	143	141	134	128	127
46	141	138	125	125	125	141	152	149	143	141	134	128	127
47	141	138	125	125	125	141	152	149	143	141	134	128	127

Figure 7.27: 277x277 Pixel Image of Right Loop Eroded Image

image | Resize_image | dilated_image | eroded_image | phase | minutiae_features

227x227 uint8

	65	66	67	68	69	70	71	72	73	74	75	76	77
41	209	206	206	204	204	204	206	207	206	206	206	204	204
42	210	207	207	207	207	207	209	209	209	209	207	204	204
43	210	209	209	209	209	209	212	212	210	210	207	204	204
44	210	209	209	209	209	209	211	210	209	209	207	207	207
45	211	211	211	211	211	211	211	210	209	209	207	207	207
46	212	212	212	212	211	211	211	210	209	209	209	209	210
47	212	212	212	212	211	211	211	211	211	211	210	210	210
48	212	212	212	212	211	211	211	211	213	215	214	213	213
49	212	212	213	213	213	216	216	216	216	215	214	214	214
50	215	215	215	215	215	216	216	216	216	216	216	216	216
51	215	215	215	216	216	216	217	217	217	217	217	217	218
52	215	215	215	218	217	217	217	217	217	217	217	218	218
53	217	217	218	218	217	217	217	218	218	218	218	218	218
54	218	218	218	219	219	219	219	219	222	219	219	219	220
55	219	219	219	219	219	219	219	220	220	220	220	220	220
56	220	220	221	219	219	219	219	220	220	220	220	220	220
57	222	220	220	220	219	219	219	220	220	220	220	222	221
58	222	220	220	220	220	220	220	220	220	221	222	222	221
59	222	220	220	220	220	220	220	220	222	222	222	222	222
60	222	220	220	220	220	220	220	221	223	223	225	224	223
61	222	220	220	220	222	222	222	222	223	224	225	224	223
62	224	224	225	222	222	222	226	225	225	225	225	226	226
63	225	225	225	222	222	222	226	227	227	226	226	226	226
64	225	225	225	225	226	226	226	226	226	226	226	226	226

Figure 7.28: 277x277 Pixel Image of Tented Eroded Image

7.6 Gabor Features Images

The Gabor filter to analyze an image. Gabor filters are commonly used in image processing for texture analysis and feature extraction. Through a filter bank, each Gabor filter is meticulously tuned to specific frequencies and orientations, facilitating the capture of localized texture information at varying scales and directions .Gabor and pixel images as shown in Figure (7.29)(7.30)(7.31)(7.32)(7.33) and (7.34).

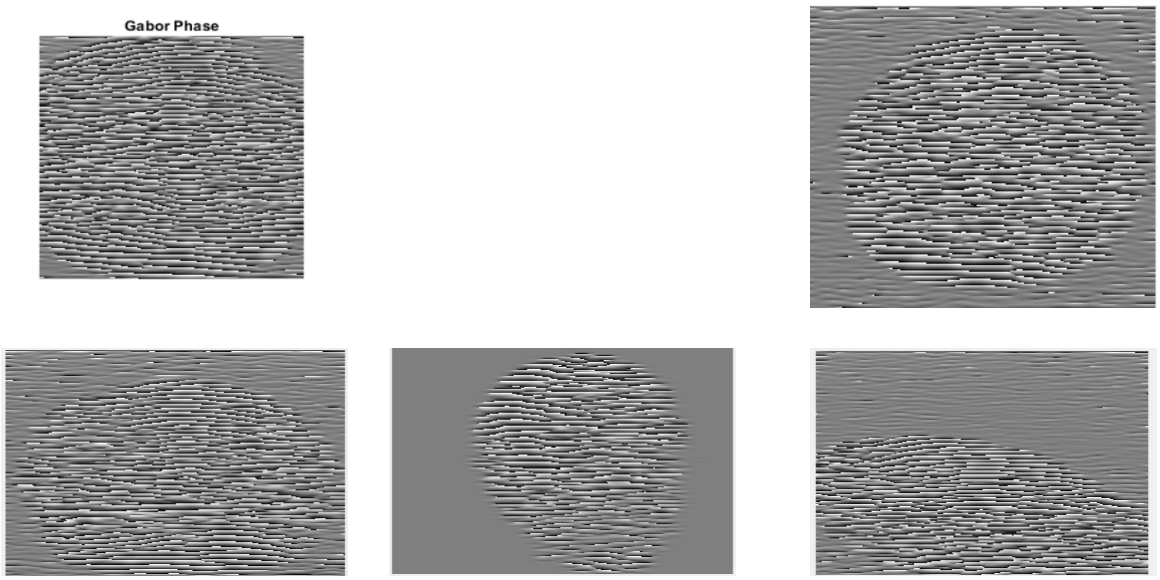


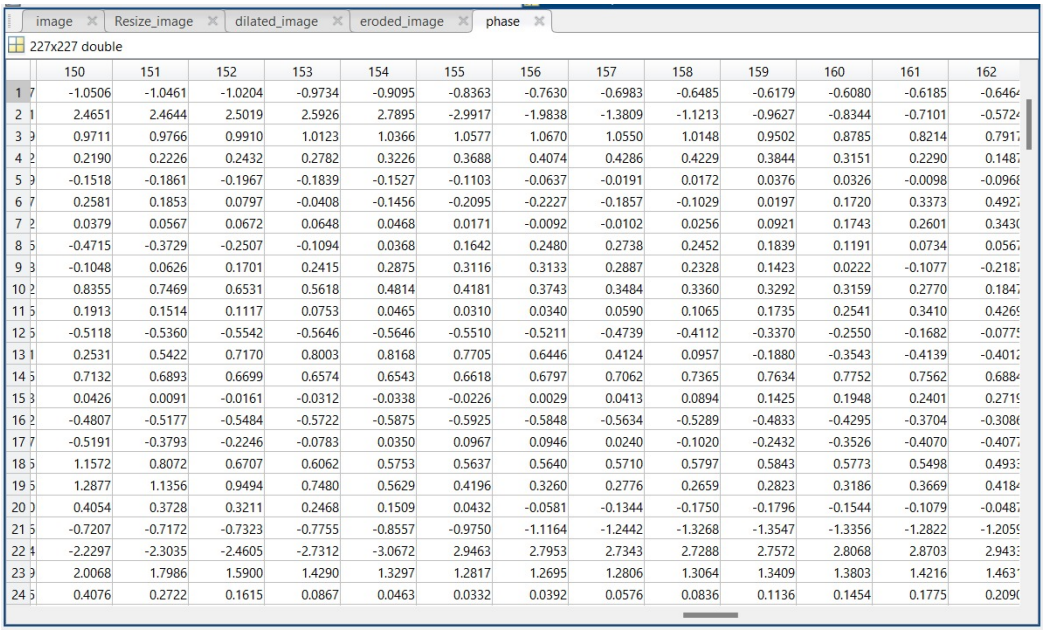
Figure 7.29: Gabor Features Images

image | Resize_image | dilated_image | eroded_image | phase | minutiae_features

227x227 double

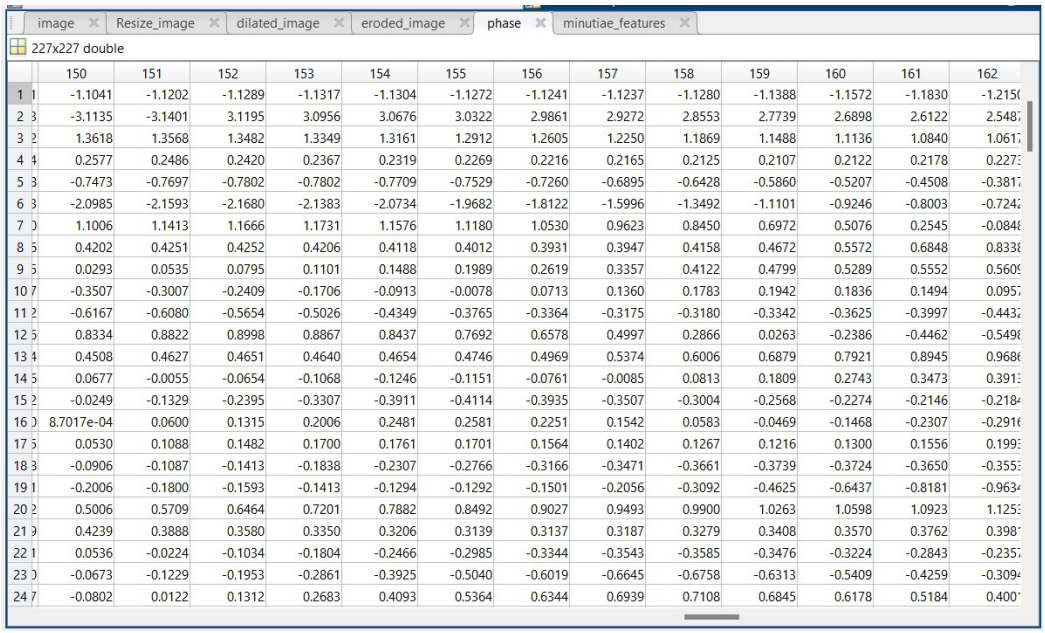
	65	66	67	68	69	70	71	72	73	74	75	76	77
41	2.2557	1.9345	1.6840	1.4149	1.0797	0.6810	0.2977	0.0055	-0.1879	-0.3024	-0.3527	-0.3457	-0.2887
42	1.0394	0.9715	0.8972	0.8105	0.7034	0.5648	0.3808	0.1395	-0.1495	-0.4364	-0.6607	-0.8022	-0.8795
43	0.2474	0.1805	0.1148	0.0470	-0.0265	-0.1099	-0.2097	-0.3358	-0.5054	-0.7440	-1.0692	-1.4366	-1.7496
44	-0.5407	-0.6360	-0.7255	-0.8136	-0.9063	-1.0124	-1.1479	-1.3466	-1.6815	-2.2367	-2.8278	3.0865	2.8824
45	-1.4358	-1.5764	-1.7232	-1.8981	-2.1534	-2.6260	2.8613	2.2550	1.9654	1.8148	1.7241	1.6615	1.6112
46	-2.8056	-3.0802	2.7122	1.9200	1.3229	1.0250	0.8608	0.7573	0.6861	0.6346	0.5952	0.5627	0.5329
47	1.5105	1.3323	0.9599	0.4626	0.0839	-0.1377	-0.2708	-0.3591	-0.4230	-0.4722	-0.5121	-0.5462	-0.5774
48	0.7959	0.8708	0.8651	0.6655	-0.1656	-1.0965	-1.3947	-1.5209	-1.5965	-1.6514	-1.6957	-1.7337	-1.7682
49	0.4293	0.5466	0.6338	0.7149	0.8414	1.2140	2.4245	3.0442	-3.0918	-3.0506	-3.0441	-3.0519	-3.0669
50	-0.0724	-0.0296	0.0157	0.0834	0.2095	0.4604	0.8969	1.3682	1.6640	1.8127	1.8857	1.9217	1.9385
51	-0.8841	-0.8574	-0.8110	-0.7244	-0.5628	-0.2879	0.0720	0.3856	0.5841	0.6954	0.7582	0.7966	0.8233
52	-2.0824	-2.0292	-1.9548	-1.8194	-1.5487	-1.0892	-0.6403	-0.3902	-0.2777	-0.2276	-0.2011	-0.1806	-0.1589
53	2.2754	2.3308	2.3133	2.2152	1.9830	1.4229	0.4150	-0.2255	-0.5426	-0.7328	-0.8512	-0.9197	-0.9541
54	1.0090	1.0086	0.9554	0.8496	0.6959	0.5120	0.3249	0.1509	-0.0202	-0.2231	-0.5033	-0.8701	-1.2363
55	0.1536	0.1186	0.0554	-0.0441	-0.1862	-0.3656	-0.5587	-0.7395	-0.9029	-1.0673	-1.2657	-1.5358	-1.8917
56	-0.6398	-0.7053	-0.7778	-0.8849	-1.0766	-1.4146	-1.8520	-2.2097	-2.4538	-2.6528	-2.8628	-3.1106	2.8936
57	-0.1866	0.5244	0.9981	1.1587	1.2466	1.3383	1.4494	1.5646	1.6532	1.6891	1.6625	1.5776	1.4492
58	0.5933	0.5792	0.5363	0.4864	0.4494	0.4443	0.4848	0.5692	0.6713	0.7510	0.7824	0.7613	0.6954
59	0.1242	0.1200	0.1426	0.2246	0.4029	0.6661	0.8977	1.0079	1.0107	0.9472	0.8493	0.7383	0.6279
60	-0.0386	0.1519	0.3780	0.5650	0.6720	0.7053	0.6828	0.6180	0.5176	0.3858	0.2273	0.0518	-0.1256
61	-0.0836	-0.0226	-0.0050	-0.0237	-0.0685	-0.1322	-0.2115	-0.3068	-0.4216	-0.5619	-0.7339	-0.9391	-1.1667
62	-0.4191	-0.5870	-0.7455	-0.8927	-1.0286	-1.1554	-1.2771	-1.3998	-1.5317	-1.6840	-1.8702	-2.1023	-2.3774
63	-1.1540	-1.4464	-1.7277	-1.9794	-2.1958	-2.3809	-2.5439	-2.6955	-2.8476	-3.0136	3.0752	2.8400	2.5649
64	-2.6438	-2.9342	3.0915	2.8639	2.6648	2.4908	2.3364	2.1946	2.0578	1.9175	1.7651	1.5920	1.3927

Figure 7.30: 277x277 Pixel Image of Whorl Gabor Feature Image



		150	151	152	153	154	155	156	157	158	159	160	161	162
1	7	-1.0506	-1.0461	-1.0204	-0.9734	-0.9095	-0.8363	-0.7630	-0.6983	-0.6485	-0.6179	-0.6080	-0.6185	-0.646
2	1	2.4651	2.4644	2.5019	2.5926	2.7895	-2.9917	-1.9838	-1.3809	-1.1213	-0.9627	-0.8344	-0.7101	-0.572
3	9	0.9711	0.9766	0.9910	1.0123	1.0366	1.0577	1.0670	1.0550	1.0148	0.9502	0.8785	0.8214	0.791
4	2	0.2190	0.2226	0.2432	0.2782	0.3226	0.3688	0.4074	0.4286	0.4229	0.3844	0.3151	0.2290	0.148
5	9	-0.1518	-0.1861	-0.1967	-0.1839	-0.1527	-0.1103	-0.0637	-0.0191	0.0172	0.0376	0.0326	-0.0098	-0.096
6	7	0.2581	0.1853	0.0797	-0.0408	-0.1456	-0.2095	-0.2227	-0.1857	-0.1029	0.0197	0.1720	0.3373	0.492
7	2	0.0379	0.0567	0.0672	0.0648	0.0468	0.0171	-0.0092	-0.0102	0.0256	0.0921	0.1743	0.2601	0.343
8	5	-0.4715	-0.3729	-0.2507	-0.1094	0.0368	0.1642	0.2480	0.2738	0.2452	0.1839	0.1191	0.0734	0.056
9	3	-0.1048	0.0626	0.1701	0.2415	0.2875	0.3116	0.3133	0.2887	0.2328	0.1423	0.0222	-0.1077	-0.218
10	2	0.8355	0.7469	0.6531	0.5618	0.4814	0.4181	0.3743	0.3484	0.3360	0.3292	0.3159	0.2770	0.184
11	5	0.1913	0.1514	0.1117	0.0753	0.0465	0.0310	0.0340	0.0590	0.1065	0.1735	0.2541	0.3410	0.426
12	5	-0.5118	-0.5360	-0.5542	-0.5646	-0.5646	-0.5510	-0.5211	-0.4739	-0.4112	-0.3370	-0.2550	-0.1682	-0.077
13	1	0.2531	0.5422	0.7170	0.8003	0.8168	0.7705	0.6446	0.4124	0.0957	-0.1880	-0.3543	-0.4139	-0.401
14	5	0.7132	0.6893	0.6699	0.6574	0.6543	0.6618	0.6797	0.7062	0.7365	0.7634	0.7752	0.7562	0.688
15	3	0.0426	0.0091	-0.0161	-0.0312	-0.0338	-0.0226	0.0029	0.0413	0.0894	0.1425	0.1948	0.2401	0.271
16	2	-0.4807	-0.5177	-0.5484	-0.5722	-0.5875	-0.5925	-0.5848	-0.5634	-0.5289	-0.4833	-0.4295	-0.3704	-0.308
17	7	-0.5191	-0.3793	-0.2246	-0.0783	0.0350	0.0967	0.0946	0.0240	-0.1020	-0.2432	-0.3526	-0.4070	-0.407
18	5	1.1572	0.8072	0.6707	0.6062	0.5753	0.5637	0.5640	0.5710	0.5797	0.5843	0.5773	0.5498	0.493
19	5	1.2877	1.1356	0.9494	0.7480	0.5629	0.4196	0.3260	0.2776	0.2659	0.2823	0.3186	0.3669	0.418
20	0	0.4054	0.3728	0.3211	0.2468	0.1509	0.0432	-0.0581	-0.1344	-0.1750	-0.1796	-0.1544	-0.1079	-0.048
21	5	-0.7207	-0.7172	-0.7323	-0.7755	-0.8557	-0.9750	-1.1164	-1.2442	-1.3268	-1.3547	-1.3356	-1.2822	-1.205
22	4	-2.2297	-2.3035	-2.4605	-2.7312	-3.0672	2.9463	2.7953	2.7343	2.7288	2.7572	2.8068	2.8703	2.943
23	9	2.0068	1.7986	1.5900	1.4290	1.3297	1.2817	1.2695	1.2806	1.3064	1.3409	1.3803	1.4216	1.463
24	5	0.4076	0.2722	0.1615	0.0867	0.0463	0.0332	0.0392	0.0576	0.0836	0.1136	0.1454	0.1775	0.209

Figure 7.31: 277x277 Pixel Image of Arch Gabor Feature Image



		150	151	152	153	154	155	156	157	158	159	160	161	162
1	1	-1.1041	-1.1202	-1.1289	-1.1317	-1.1304	-1.1272	-1.1241	-1.1237	-1.1280	-1.1388	-1.1572	-1.1830	-1.215
2	3	-3.1135	-3.1401	3.1195	3.0956	3.0676	3.0322	2.9861	2.9272	2.8553	2.7739	2.6898	2.6122	2.548
3	2	1.3618	1.3568	1.3482	1.3349	1.3161	1.2912	1.2605	1.2250	1.1869	1.1488	1.1136	1.0840	1.061
4	4	0.2577	0.2486	0.2420	0.2367	0.2319	0.2269	0.2216	0.2165	0.2125	0.2107	0.2122	0.2178	0.227
5	3	-0.7473	-0.7697	-0.7802	-0.7802	-0.7709	-0.7529	-0.7260	-0.6895	-0.6428	-0.5860	-0.5207	-0.4508	-0.381
6	3	-2.0985	-2.1593	-2.1680	-2.1383	-2.0734	-1.9682	-1.8122	-1.5996	-1.3492	-1.1101	-0.9246	-0.8003	-0.724
7	0	1.1006	1.1413	1.1666	1.1731	1.1576	1.1180	1.0530	0.9623	0.8450	0.6972	0.5076	0.2545	-0.084
8	5	0.4202	0.4251	0.4252	0.4206	0.4118	0.4012	0.3931	0.3947	0.4158	0.4672	0.5572	0.6848	0.833
9	5	0.0293	0.0535	0.0795	0.1101	0.1488	0.1989	0.2619	0.3357	0.4122	0.4799	0.5289	0.5552	0.560
10	7	-0.3507	-0.3007	-0.2409	-0.1706	-0.0913	-0.0078	0.0713	0.1360	0.1783	0.1942	0.1836	0.1494	0.095
11	2	-0.6167	-0.6080	-0.5654	-0.5026	-0.4349	-0.3765	-0.3364	-0.3175	-0.3180	-0.3342	-0.3625	-0.3997	-0.443
12	5	0.8334	0.8822	0.8998	0.8867	0.8437	0.7692	0.6578	0.4997	0.2866	0.0263	-0.2386	-0.4462	-0.549
13	4	0.4508	0.4627	0.4651	0.4640	0.4654	0.4746	0.4969	0.5374	0.6006	0.6879	0.7921	0.8945	0.968
14	5	0.0677	-0.0055	-0.0654	-0.1068	-0.1246	-0.1151	-0.0761	-0.0085	0.0813	0.1809	0.2743	0.3473	0.391
15	2	-0.0249	-0.1329	-0.2395	-0.3307	-0.3911	-0.4114	-0.3935	-0.3507	-0.3004	-0.2568	-0.2274	-0.2146	-0.218
16	0	8.7017e-04	0.0600	0.1315	0.2006	0.2481	0.2581	0.2251	0.1542	0.0583	-0.0469	-0.1468	-0.2307	-0.291
17	5	0.0530	0.1088	0.1482	0.1700	0.1761	0.1701	0.1564	0.1402	0.1267	0.1216	0.1300	0.1556	0.199
18	3	-0.0906	-0.1087	-0.1413	-0.1838	-0.2307	-0.2766	-0.3166	-0.3471	-0.3661	-0.3739	-0.3724	-0.3650	-0.355
19	1	-0.2006	-0.1800	-0.1593	-0.1413	-0.1294	-0.1292	-0.1501	-0.2056	-0.3092	-0.4625	-0.6437	-0.8181	-0.963
20	2	0.5006	0.5709	0.6464	0.7201	0.7882	0.8492	0.9027	0.9493	0.9900	1.0263	1.0598	1.0923	1.125
21	9	0.4239	0.3888	0.3580	0.3350	0.3206	0.3139	0.3137	0.3187	0.3279	0.3408	0.3570	0.3762	0.398
22	1	0.0536	-0.0224	-0.1034	-0.1804	-0.2466	-0.2985	-0.3344	-0.3543	-0.3585	-0.3476	-0.3224	-0.2843	-0.235
23	0	-0.0673	-0.1229	-0.1953	-0.2861	-0.3925	-0.5040	-0.6019	-0.6645	-0.6758	-0.6313	-0.5409	-0.4259	-0.309
24	7	-0.0802	0.0122	0.1312	0.2683	0.4093	0.5364	0.6344	0.6939	0.7108	0.6845	0.6178	0.5184	0.400

Figure 7.32: 277x277 Pixel Image of Left Loop Gabor Feature Image

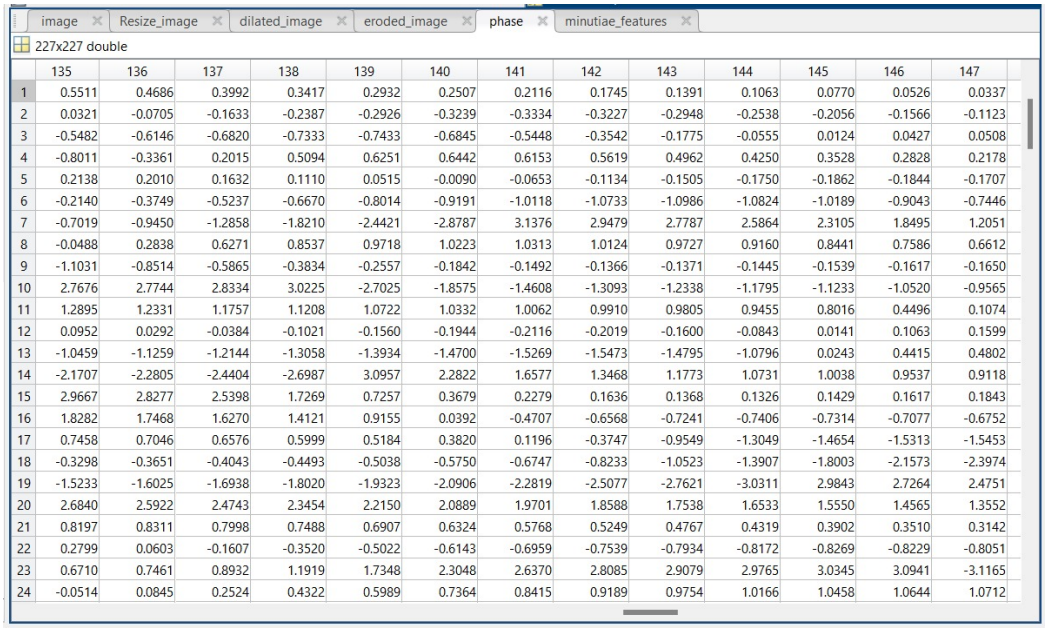


image | Resize_image | dilated_image | eroded_image | phase | minutiae_features

227x227 double

	135	136	137	138	139	140	141	142	143	144	145	146	147
1	0.5511	0.4686	0.3992	0.3417	0.2932	0.2507	0.2116	0.1745	0.1391	0.1063	0.0770	0.0526	0.0337
2	0.0321	-0.0705	-0.1633	-0.2387	-0.2926	-0.3239	-0.3334	-0.3227	-0.2948	-0.2538	-0.2056	-0.1566	-0.1123
3	-0.5482	-0.6146	-0.6820	-0.7333	-0.7433	-0.6845	-0.5448	-0.3542	-0.1775	-0.0555	0.0124	0.0427	0.0508
4	-0.8011	-0.3361	0.2015	0.5094	0.6251	0.6442	0.6153	0.5619	0.4962	0.4250	0.3528	0.2828	0.2178
5	0.2138	0.2010	0.1632	0.1110	0.0515	-0.0090	-0.0653	-0.1134	-0.1505	-0.1750	-0.1862	-0.1844	-0.1707
6	-0.2140	-0.3749	-0.5237	-0.6670	-0.8014	-0.9191	-1.0118	-1.0733	-1.0986	-1.0824	-1.0189	-0.9043	-0.7446
7	-0.7019	-0.9450	-1.2858	-1.8210	-2.4421	-2.8787	3.1376	2.9479	2.7787	2.5864	2.3105	1.8495	1.2051
8	-0.0488	0.2838	0.6271	0.8537	0.9718	1.0223	1.0313	1.0124	0.9727	0.9160	0.8441	0.7586	0.6612
9	-1.1031	-0.8514	-0.5865	-0.3834	-0.2557	-0.1842	-0.1492	-0.1366	-0.1371	-0.1445	-0.1539	-0.1617	-0.1650
10	2.7676	2.7744	2.8334	3.0225	-2.7025	-1.8575	-1.4608	-1.3093	-1.2338	-1.1795	-1.1233	-1.0520	-0.9565
11	1.2895	1.2331	1.1757	1.1208	1.0722	1.0332	1.0062	0.9910	0.9805	0.9455	0.8016	0.4496	0.1074
12	0.0952	0.0292	-0.0384	-0.1021	-0.1560	-0.1944	-0.2116	-0.2019	-0.1600	-0.0843	0.0141	0.1063	0.1599
13	-1.0459	-1.1259	-1.2144	-1.3058	-1.3934	-1.4700	-1.5269	-1.5473	-1.4795	-1.0796	0.0243	0.4415	0.4802
14	-2.1707	-2.2805	-2.4404	-2.6987	3.0957	2.2822	1.6577	1.3468	1.1773	1.0731	1.0038	0.9537	0.9118
15	2.9667	2.8277	2.5398	1.7269	0.7257	0.3679	0.2279	0.1636	0.1368	0.1326	0.1429	0.1617	0.1843
16	1.8282	1.7468	1.6270	1.4121	0.9155	0.0392	-0.4707	-0.6568	-0.7241	-0.7406	-0.7314	-0.7077	-0.6752
17	0.7458	0.7046	0.6576	0.5999	0.5184	0.3820	0.1196	-0.3747	-0.9549	-1.3049	-1.4654	-1.5313	-1.5453
18	-0.3298	-0.3651	-0.4043	-0.4493	-0.5038	-0.5750	-0.6747	-0.8233	-1.0523	-1.3907	-1.8003	-2.1573	-2.3974
19	-1.5233	-1.6025	-1.6938	-1.8020	-1.9323	-2.0906	-2.2819	-2.5077	-2.7621	-3.0311	2.9843	2.7264	2.4751
20	2.6840	2.5922	2.4743	2.3454	2.2150	2.0889	1.9701	1.8588	1.7538	1.6533	1.5550	1.4565	1.3552
21	0.8197	0.8311	0.7998	0.7488	0.6907	0.6324	0.5768	0.5249	0.4767	0.4319	0.3902	0.3510	0.3142
22	0.2799	0.0603	-0.1607	-0.3520	-0.5022	-0.6143	-0.6959	-0.7539	-0.7934	-0.8172	-0.8269	-0.8229	-0.8051
23	0.6710	0.7461	0.8932	1.1919	1.7348	2.3048	2.6370	2.8085	2.9079	2.9765	3.0345	3.0941	-3.1165
24	-0.0514	0.0845	0.2524	0.4322	0.5989	0.7364	0.8415	0.9189	0.9754	1.0166	1.0458	1.0644	1.0712

Figure 7.33: 277x277 Pixel Image of Right Loop Gabor Feature Image

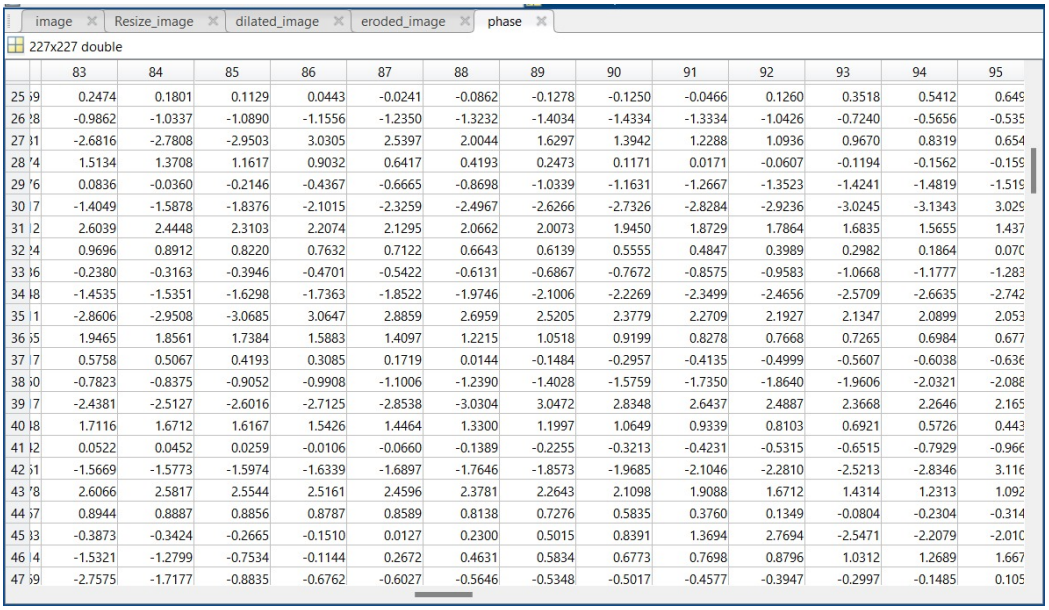


image | Resize_image | dilated_image | eroded_image | phase

227x227 double

	83	84	85	86	87	88	89	90	91	92	93	94	95
25	0.2474	0.1801	0.1129	0.0443	-0.0241	-0.0862	-0.1278	-0.1250	-0.0466	0.1260	0.3518	0.5412	0.649
26	-0.9862	-1.0337	-1.0890	-1.1556	-1.2350	-1.3232	-1.4034	-1.4334	-1.3334	-1.0426	-0.7240	-0.5656	-0.535
27	-2.6816	-2.7808	-2.9503	3.0305	2.5397	2.0044	1.6297	1.3942	1.2288	1.0936	0.9670	0.8319	0.654
28	1.5134	1.3708	1.1617	0.9032	0.6417	0.4193	0.2473	0.1171	0.0171	-0.0607	-0.1194	-0.1562	-0.159
29	0.0836	-0.0360	-0.2146	-0.4367	-0.6665	-0.8698	-1.0339	-1.1631	-1.2667	-1.3523	-1.4241	-1.4819	-1.519
30	-1.4049	-1.5878	-1.8376	-2.1015	-2.3259	-2.4967	-2.6266	-2.7326	-2.8284	-2.9236	-3.0245	-3.1343	3.029
31	2.6039	2.4448	2.3103	2.2074	2.1295	2.0662	2.0073	1.9450	1.8729	1.7864	1.6835	1.5655	1.437
32	0.9696	0.8912	0.8220	0.7632	0.7122	0.6643	0.6139	0.5555	0.4847	0.3989	0.2982	0.1864	0.070
33	-0.2380	-0.3163	-0.3946	-0.4701	-0.5422	-0.6131	-0.6867	-0.7672	-0.8575	-0.9583	-1.0668	-1.1777	-1.283
34	-1.4535	-1.5351	-1.6298	-1.7363	-1.8522	-1.9746	-2.1006	-2.2269	-2.3499	-2.4656	-2.5709	-2.6635	-2.742
35	-2.8606	-2.9508	-3.0685	3.0647	2.8859	2.6959	2.5205	2.3779	2.2709	2.1927	2.1347	2.0899	2.053
36	1.9465	1.8561	1.7384	1.5883	1.4097	1.2215	1.0518	0.9199	0.8278	0.7668	0.7265	0.6984	0.677
37	0.5758	0.5067	0.4193	0.3085	0.1719	0.0144	-0.1484	-0.2957	-0.4135	-0.4999	-0.5607	-0.6038	-0.636
38	-0.7823	-0.8375	-0.9052	-0.9908	-1.1006	-1.2390	-1.4028	-1.5759	-1.7350	-1.8640	-1.9606	-2.0321	-2.088
39	-2.4381	-2.5127	-2.6016	-2.7125	-2.8538	-3.0304	3.0472	2.8348	2.6437	2.4887	2.3668	2.2646	2.165
40	1.7116	1.6712	1.6167	1.5426	1.4464	1.3300	1.1997	1.0649	0.9339	0.8103	0.6921	0.5726	0.443
41	0.0522	0.0452	0.0259	-0.0106	-0.0660	-0.1389	-0.2255	-0.3213	-0.4231	-0.5315	-0.6515	-0.7929	-0.966
42	-1.5669	-1.5773	-1.5974	-1.6339	-1.6897	-1.7646	-1.8573	-1.9685	-2.1046	-2.2810	-2.5213	-2.8346	3.116
43	2.6066	2.5817	2.5544	2.5161	2.4596	2.3781	2.2643	2.1098	1.9088	1.6712	1.4314	1.2313	1.092
44	0.8944	0.8887	0.8856	0.8787	0.8589	0.8138	0.7276	0.5835	0.3760	0.1349	-0.0804	-0.2304	-0.314
45	-0.3873	-0.3424	-0.2665	-0.1510	0.0127	0.2300	0.5015	0.8391	1.3694	2.7694	-2.5471	-2.2079	-2.010
46	-1.5321	-1.2799	-0.7534	-0.1144	0.2672	0.4631	0.5834	0.6773	0.7698	0.8796	1.0312	1.2689	1.667
47	-2.7575	-1.7177	-0.8835	-0.6762	-0.6027	-0.5646	-0.5348	-0.5017	-0.4577	-0.3947	-0.2997	-0.1485	0.105

Figure 7.34: 277x277 Pixel Image of Tented Gabor Feature Image

7.7 Minutiae Features Images

The most important step in automatic fingerprint matching is reliably extract minutiae from captured fingerprint images. Minutiae points are the major features of a fingerprint image and are used in the matching fingerprints. It finds the uniqueness of image.Minutiae and pixel images as shown in Figure (7.35)(7.36)(7.37)(7.38)(7.39) and (7.40).

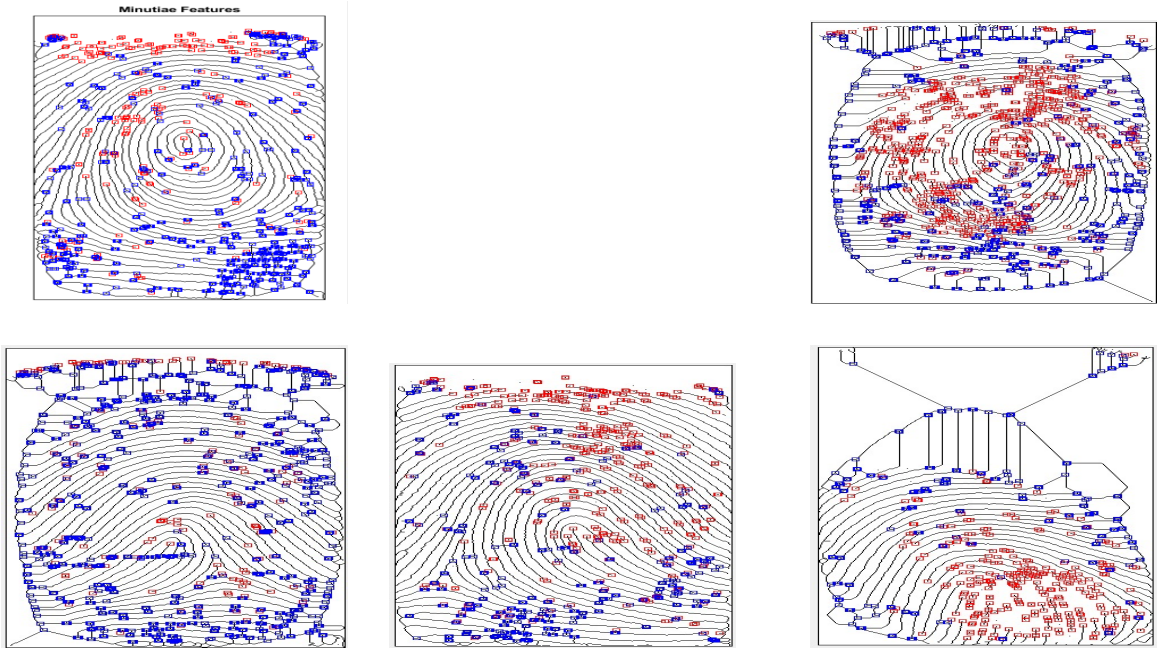


Figure 7.35: Minutiae Features Images

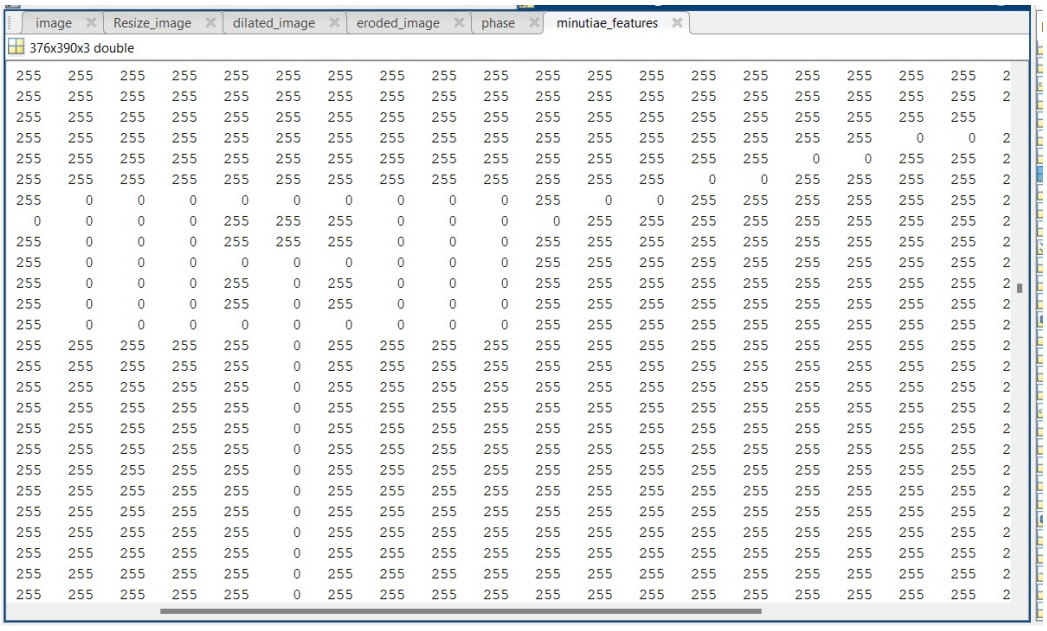


Figure 7.36: 376x390x3 Pixel Image of Whorl Minutiae Feature Image

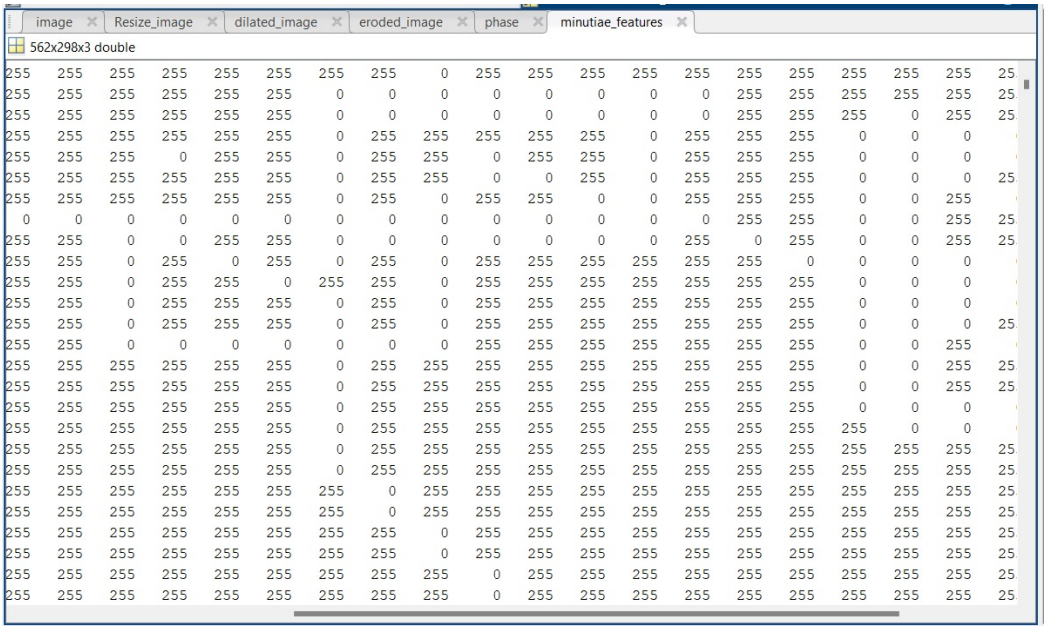


Figure 7.37: 562x298x3 Pixel Image of Arch Minutiae Feature Image

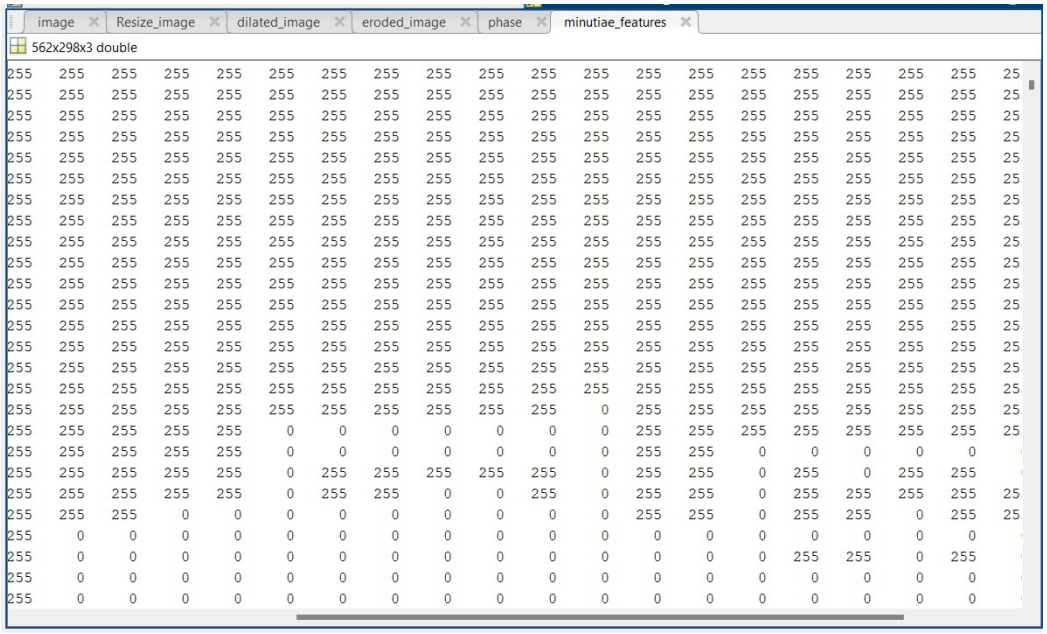


Figure 7.38: 562x298x3 Pixel Image of Left Loop Minutiae Feature Image

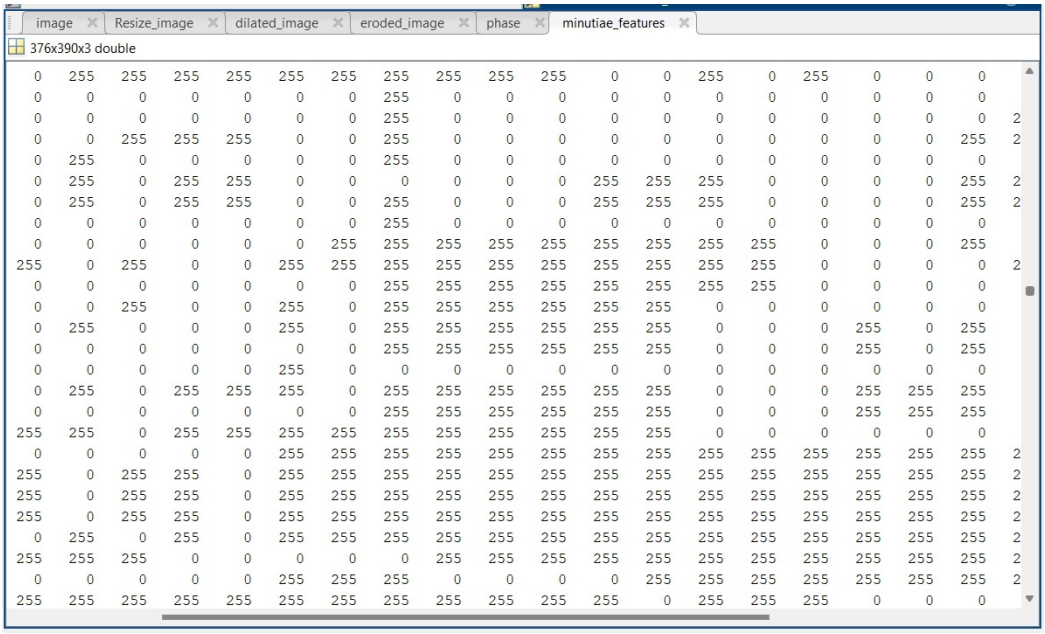


Figure 7.39: 376x390x3 Pixel Image of Right Loop Minutiae Feature Image

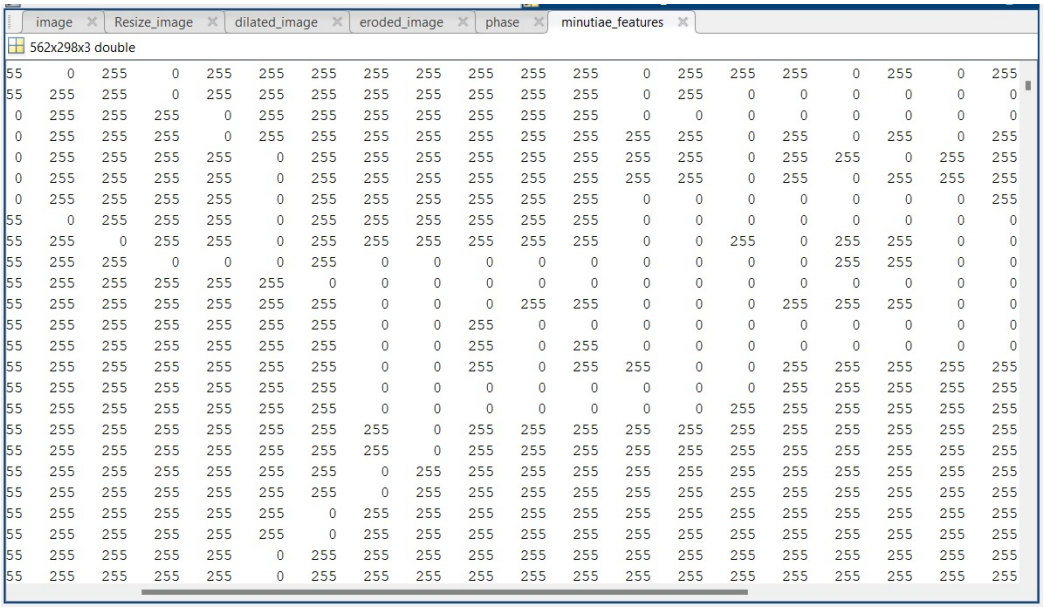


Figure 7.40: 562x298x3 Pixel Image of Tented Minutiae Feature Image

7.8 Kurtosis Values

In Principal Component Analysis (PCA), kurtosis helps us understand how the variables in our dataset are spread out. When kurtosis is high, it means the tails of the distribution are heavier, suggesting the data might not follow a normal pattern. This can affect how well PCA works. By spotting variables with high kurtosis, we can tell if we need to change the data or try different techniques to make PCA work better. Looking at kurtosis alongside other measures can help us pick the most important components more accurately, giving us a clearer picture of the data's variation. Each column in our dataset represents the kurtosis values for different types of images, like Arch, Tented, Left Loop,Right Loop and Whorl, with each type having 25 images as shown in Figure 7.41.

Types Of Fingerprints					
Images	Arch	Tented	Left Loop	Right Loop	Whorl
Image-1	1.6451	1.9274	2.2574	3.5861	1.7235
Image-2	1.6325	2.0282	1.8929	1.2128	2.1449
Image-3	1.9798	2.0282	2.0011	1.2069	2.1449
Image-4	2.0335	1.8177	1.9852	1.5739	2.0802
Image-5	1.4253	1.8177	1.8048	2.1405	2.0802
Image-6	1.6969	1.3158	2.0144	1.2923	2.1703
Image-7	1.6969	1.3158	1.9493	1.2798	2.1136
Image-8	1.3454	1.3274	2.5171	1.6674	2.08
Image-9	1.3331	1.3274	1.9798	1.434	1.1661
Image-10	1.17734	1.3274	1.9253	2.1641	2.1769
Image-11	1.8255	1.7237	1.8474	1.9679	2.3283
Image-12	1.6562	1.6181	2.0732	1.4948	1.7235
Image-13	1.6625	1.7237	1.9634	1.9372	1.6197
Image-14	1.5506	1.4916	2.3255	2.0409	1.6752
Image-15	1.6641	1.4916	1.5428	2.504	1.5669
Image-16	1.8469	1.728	1.6641	2.5776	1.5952
Image-17	1.9428	1.728	1.5329	1.5959	1.4054
Image-18	1.3305	1.9274	1.4325	1.6365	1.382
Image-19	1.6975	1.6181	1.7913	1.4905	2.1537
Image-20	1.5835	2.2652	1.5205	1.9439	2.1537
Image-21	1.4745	2.2652	1.5205	1.7744	2.0259
Image-22	1.5843	1.8822	2.0766	2.3547	2.0259
Image-23	1.5888	1.8822	1.8115	2.473	2.0455
Image-24	1.6617	2.2087	1.8115	1.2635	2.0455
Image-25	1.5063	2.2087	2.1115	1.2574	2.3918

Figure 7.41: 25X5 Kurtosis Values Of different types of Fingerprint Images

7.9 Performance Metrics

A confusion matrix is a table used to evaluate the performance of a machine learning model on a set of test data where the true values are known. It is a square matrix that shows the number of true positives, true negatives, false positives, and false negatives predicted by the model. A true negative (TN) is a case where the model correctly predicts the negative class. A false positive (FP) is a case where the model incorrectly predicts the positive class and a false negative (FN) is a case where the model incorrectly predicts the negative class. The rows represent the true class labels, while the columns represent the predicted class labels. The diagonal elements of the matrix (top left to bottom right) represent the number of cases where the model correctly predicted the class, while the off-diagonal elements represent the misclassifications. From the confusion matrix, performance metrics can be calculated, such as the accuracy for the evaluation of the model's performance [11].

1.Accuracy:

Accuracy can be defined as the percentage of correct predictions made by our classification model as shown in Equation 7.1.

$$Accuracy = \frac{TP + TN}{TP + TN + FN + FP} \tag{7.1}$$

2.False Acceptance Rate(F_{AR}):

This is the rate at which the system incorrectly identifies an imposter as a legitimate user . It is calculated as the ratio of false acceptances to the total number of impostor attempts as shown in Equation 7.2.

$$F_{AR} = \frac{F_P}{F_P + T_N} \tag{7.2}$$

3.False Rejection Rate(F_{RR}):

This is the rate at which the system incorrectly rejects a legitimate user . It is calculated as the ratio of false rejections to the total number of genuine attempts as shown in Equation

7.3.

$$F_{RR} = \frac{F_N}{F_N + T_P} \tag{7.3}$$

4.Precision:

It indicates out of all positive predictions ,how many are actually positive.It is defined as a ratio of correct positive predictions to overall positive predictions as shown in Equation 7.4.

$$F_{RR} = \frac{Predictionsactuallypositive}{Totalpredictedpositive} = \frac{T_P}{T_P + F_P} \tag{7.4}$$

5.Recall or Sensitivity:

It indicates out of all actually positive values,how many are predicted positive.It is defined as a ratio of correct positive predictions to overall number of positive instances in the dataset as shown in Equation 7.5.

$$Recall = \frac{Predictionsactuallypositive}{Actuallypositivevaluesinthedataset} = \frac{T_P}{T_P + F_N} \tag{7.5}$$

6.Specificity:

It tells us how good the model is at correctly identifying the negative cases out of all the actual negative cases as shown in Equation 7.6.

$$Specificity = \frac{T_N}{T_N + F_P} \tag{7.6}$$

7.F1 Score:

It is the harmonic mean of the precision and recall as shown in Equation 7.7.

$$F1Score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{7.7}$$

7.10 Confusion matrix for CNN

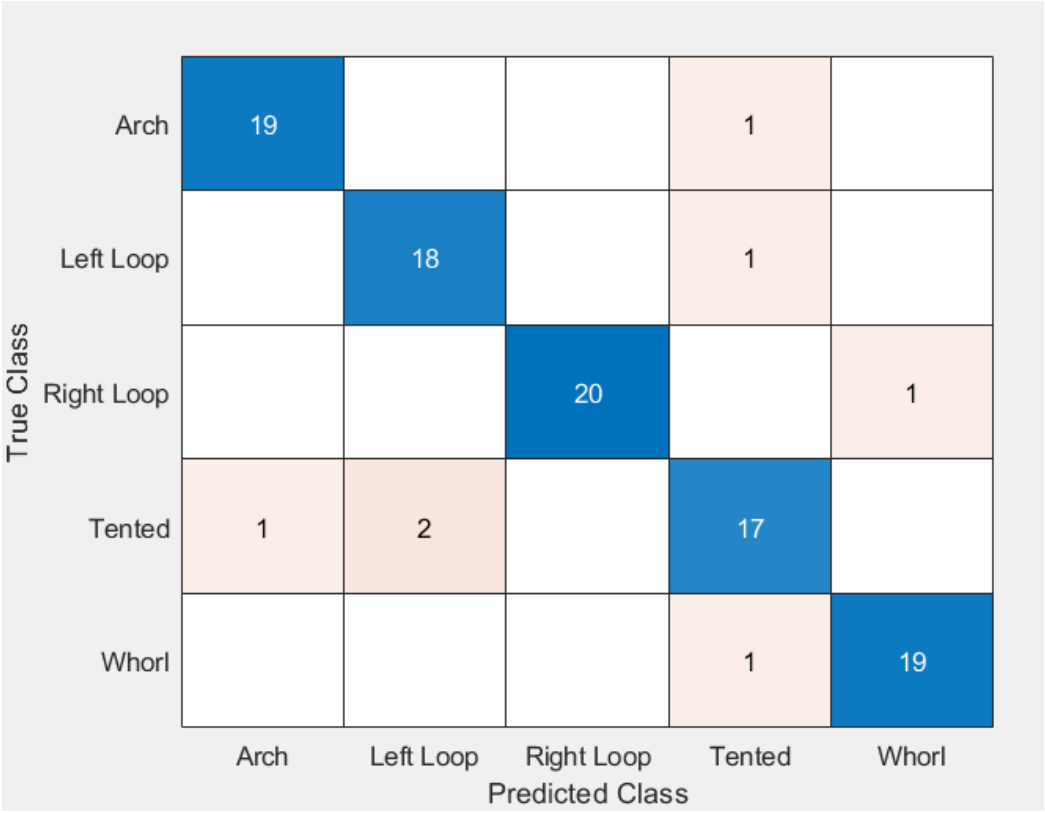


Figure 7.42: Confusion Matrix for CNN

Figure 7.42 presents the confusion matrix for the CNN model after applying PCA. The rows represent the true class labels of the samples, while the columns represent the predicted class labels by the CNN model as shown in Equation (7.8)(7.9)(7.10)(7.11)(7.12)(7.13)(7.14) [11].

Table 7.1: Confusion Matrix Parameters for CNN

Confusion Matrix Parameters	values
True Positives	26
True Negatives	79
False Positives	1
False Negatives	1

$$Accuracy = \frac{T_P + T_N}{T_P + T_N + F_P + F_N} = \frac{19 + 79}{19 + 1 + 1 + 79} = 98\% \tag{7.8}$$

$$Precision = \frac{T_P}{T_P + F_P} = \frac{19}{19+1} = 0.95 = 95\% \tag{7.9}$$

$$Recall = \frac{T_P}{T_P + F_N} = \frac{19}{19+1} = 0.95 = 95\% \tag{7.10}$$

$$Specificity = \frac{T_N}{T_N + F_P} = \frac{99}{99+1} = 0.99 = 99\% \tag{7.11}$$

$$F1Score = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * 0.95 * 0.95}{0.95 + 0.95} = 0.95 = 95\% \tag{7.12}$$

$$F_{AR} = \frac{F_P}{F_P + T_N} = \frac{1}{1+99} = 0.01 = 1\% \tag{7.13}$$

$$F_{RR} = \frac{F_N}{F_N + T_P} = \frac{1}{1+19} = 0.05 = 5\% \tag{7.14}$$

7.11 Confusion matrix for BPNN

Figure 7.43 presents the confusion matrix for the BPNN model after applying PCA. The rows represent the true class labels of the samples, while the columns represent the predicted class labels by the BPNN model. The values in the cells of the matrix represent the number of samples that were classified into each combination of true and predicted class labels as shown in Table 7.2 by using Equations (7.15)(7.16)(7.17)(7.18)(7.19)(7.20)(7.21).

Table 7.2: Confusion Matrix Parameters for BPNN

Confusion Matrix Parameters	values
True Positives	24
True Negatives	99
False Positives	2
False Negatives	0

$$Accuracy = \frac{T_P + T_N}{T_P + T_N + F_P + F_N} = \frac{24 + 99}{24 + 2 + 0 + 99} = 98.6\% \tag{7.15}$$

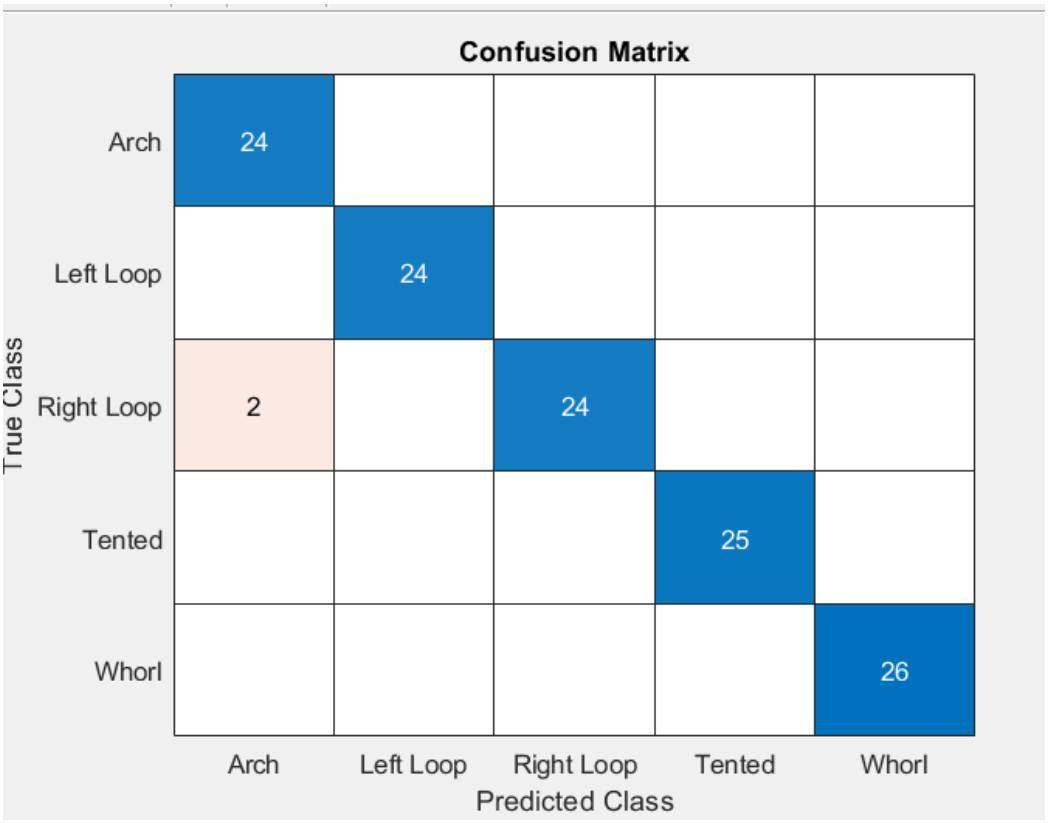


Figure 7.43: Confusion Matrix for BPNN

$$Precision = \frac{T_P}{T_P + F_P} = \frac{24}{24 + 0} = 1.0 = 100\% \tag{7.16}$$

$$Recall = \frac{T_P}{T_P + F_N} = \frac{24}{24 + 2} = 0.923 = 92.3\% \tag{7.17}$$

$$Specificity = \frac{T_N}{T_N + F_P} = \frac{99}{99 + 0} = 1.0 = 100\% \tag{7.18}$$

$$F1Score = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * 1.0 * 0.923}{1.0 + 0.923} = 0.95 = 95\% \tag{7.19}$$

$$F_{AR} = \frac{F_P}{F_P + T_N} = \frac{0}{0 + 99} = 0 = 0.00\% \tag{7.20}$$

$$F_{RR} = \frac{F_N}{F_N + T_P} = \frac{2}{2 + 24} = 0.0769 = 7.69\% \tag{7.21}$$

7.12 Comparison for different models

Table 7.3: Comaprision of Accuracy for Different Models

Neural Network Model	Accuracy of Different Models
CNN(Gabor and Minutiae)	98
BPNN(Gabor Features)	90
BPNN(Minutiae Features)	89
BPNN(Gabor and Minutiae)	98.6

Table 7.3 represents the accuracy of the Backpropagation Neural Network (BPNN with Gabor and Minutiae Features), Back Propagation Neural Network(Gabor Features), Back Propagation Neural Network(Minutiae Features) and Convolutional Neural Network (CNN with Gabor and Minutiae Features) were evaluated after applying PCA.

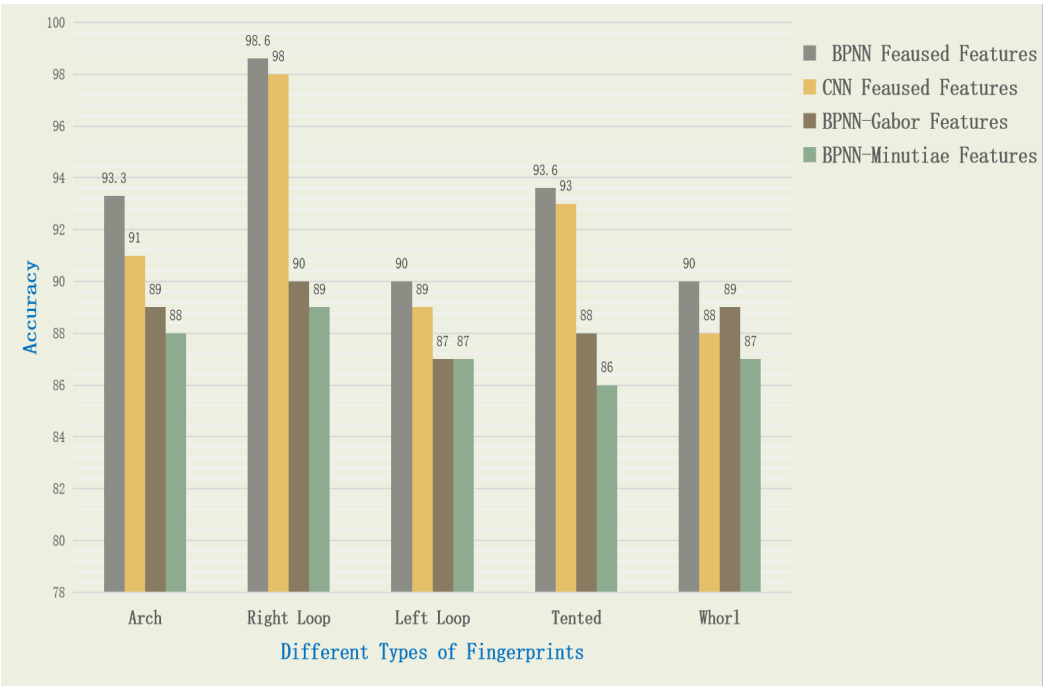


Figure 7.44: Comparison of Accuracy

Figure 7.44 represents the graphical representation of the accuracy and execution time of both the Convolutional Neural Network (BPNN) and Back Propagation Neural Network (RBFNN) were evaluated after applying PCA. After applying PCA, the accuracy of CNN was found to be 78%, while the accuracy of BPNN was 93.6%.

The results indicate that both PCA can contribute to enhancing the per formance of RBFNN, but BPNN remains relatively stable in accuracy with the additional step of PCA.

This suggests that BPNN is inherently more robust to feature reduction and selection, and the combination of PCA and CNN may provide significant additional benefits as BPNN-PCA takes comparatively less execution time.

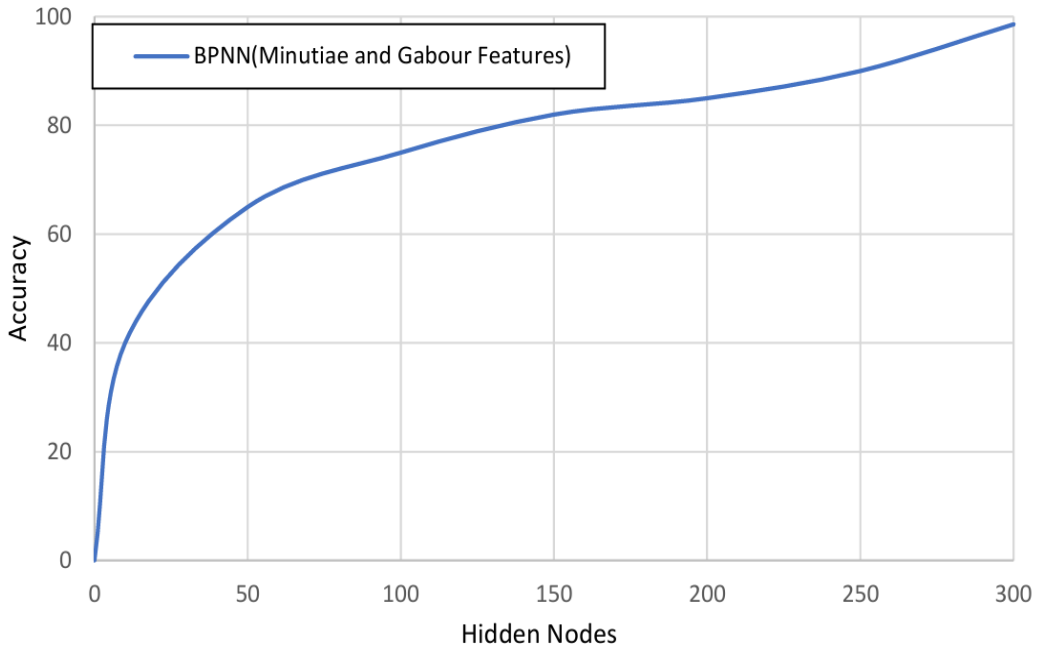


Figure 7.45: Comparison of accuracy relative to variations in the number of hidden nodes.

The accuracy of the BPNN network changes with the alteration of the number of hidden nodes in its hidden layer as shown in Figure 7.45.

Overall, the results highlight the potential of PCA in improving the performance of BPNN in the context of Fingerprint Identification [4].

8

CONCLUSION

In this project, the Fingerprint Identification system is designed for 5 different types of classes and each class contains 25 images are identified using Back Propagation Neural Networkthe fusion of Gabor and Minutiae features coupled with a BPNN classifier presents a robust approach to fingerprint identification. Through the systematic process involving preprocessing, feature extraction, fusion, dimensionality reduction, and classification, the system achieves accurate recognition of various fingerprint patterns including arches, loops, tented arches, and whorls. The application of morphological operations, PCA and kurtosis for dimensionality reduction, and the utilization of a BPNN classifier contribute to the system's reliability and efficiency. This methodology results in improving the accuracy of biometric authentication systems, thereby ensuring secure access control. The proposed method is the combination of Back Propagation Neural Network and Levenberg-Marquardt Optimization Algorithm showed better results compared with Convolutional Neural Network .

The accuracy of propsed Back Propagation Neural Network has accuracy of 98.6% , precision of 100%,Recall of 92.3% and Specificity of 100%. Compared with Convolutional Neural Network Fingerprint identification system has accuracy of 98%, precision of 95%, Recall of 95% and Specificity of 99%.

The accuracy of the proposed BPNN network of fused features of gabour and minutiae features of finger print identification system is 0.61% higher than CNN. Moving forward, further research could ex- plore optimizations to streamline the process and potentially integrate additional features or algorithms to enhance the system's performance in real-world applications.

9

REFERENCES

Bibliography

[1] Hasan H, Abdul-Kareem S. Fingerprint image enhancement and recognition algorithms: a survey. Neural Comput Appl 2013;(23):1605–10.

[2] Win KN, Li K, Chen J, Viger PF, Li K. Fingerprint classification and identification algorithms for criminal investigation: a survey. Fut Gener Comput Syst 2020; 110:758–71.

[3] Nazarkevych M, Riznyk O, Samotyy V, Dzelendzyak U. Detection of regularities in the parameters of the Ateb-Gabor method for biometric image filtration. East Eur J Enterp Technol 2019;1(2–97):57–65.

[4] Shemmary ENAAl. Classification of fingerprint images using neural networks technique. J Eng (JOE) 2012;1(3):40–8.

[5] Leung KC, Leung CH. Improvement of fingerprint retrieval by a statistical classifier. IEEE Trans Inform Forens Secur 2011;6(1):59–69.

[6] Wan GC, Li MM, Xu H, Kang WH, Rui JW, Tong MS. XFinger-net: pixel-wise segmentation method for partially defective fingerprint based on attention gates and U-net. Sensors 2020;20:4473.

[7] AlShehri H, Hussain M, AboAlSamh H, AlZuair M. A large-scale study of fingerprint matching systems for sensor interoperability problem. Sensors 2018;18: 1008.

[8] Wu F, Zhu J, Guo X. Fingerprint pattern identification and classification approach based on convolutional neural networks. Neural Comput Appl 2020;32: 5725–34.

[9] Takahashi, A., Koda, Y., Ito, K., Aoki, T. (2020). Fingerprint feature extraction by combining texture, minutiae, and frequency spectrum using multi-task CNN, 2020, arXiv preprint arXiv:2008.11917.

[10] Borra SR, Reddy GJ, Reddy ES. Classification of fingerprint images with the aid of morphological operation and AGNN classifier. Appl Comput Inform 2018;14: 166–76.

[11] Peralta D, Triguero I, García S, Saeys Y, Benitez JM, Herrera F. On the use of convolutional neural networks for robust classification of multiple fingerprint captures. Int J Intell Syst 2018;33:213–30.

[12] Peralta D, Galar M, Triguero I, Paternain D, García S, Barrenechea E, Beníteza JM, Bustince H, Herrera F. A survey on fingerprint minutiae-based local matching for verification and identification: taxonomy and experimental evaluation. Expert Syst Appl 2017;81:251–67.

[13] Jain AK, Feng J, Nandakumar K. Fingerprint matching. IEEE Comput 2010;43:36–44.

[14] Ge W, Sun M, Wang X. An incremental two-dimensional principal component analysis for object recognition. Math Probl Eng 2018:1–13. 2018, 2018.

10

PUBLICATIONS

List of Publications

- “J.Lakshmi Bhavani(Y20EC075), K.Kalyan(Y20EC079), L.Jagadeeswara rao(Y20EC109)”, “FINGERPRINT IDENTIFICATION WITH FUSION OF GABOR AND MINUTIAE FEATURES USING BPNN CLASSIFIER” , INTERNATIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING AND MANAGEMENT (IJSREM), ISSN: 2582-3930 , VOLUME: 08 ISSUE: 04, pp:58-62, APRIL - 2024.

INTERNATIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING AND MANAGEMENT (IJSREM)
VOLUME: 08 ISSUE: 04 | APRIL - 2024 SJIF RATING: 8.448 ISSN: 2582-3930

Fingerprint Identification with Fusion of Gabor and Minutiae Features Using BPNN Classifier

Janjanam Lakshmi Bhavani (Y20EC075)
dept of Electronics and communication engineering
RVR & JC College Of Engineering

Lanka Jagadeeswara Rao (Y20EC109)
dept of Electronics and communication engineering
RVR & JC College Of Engineering

Kalyan Kuna (Y20EC079)
dept of Electronics and communication engineering
RVR & JC College Of Engineering

Abstract-This study presents an innovative approach to fingerprint identification by leveraging the fusion of Gabor and Minutiae features, employing a Backpropagation Neural Network (BPNN) classifier for accurate categorization. The process involves initial handling of a fingerprint image dataset, including crucial pre-processing steps such as resizing images and implementing morphological operations like dilation, erosion, and opening. Subsequently, Gabor features and minutiae extraction are performed, followed by the fusion of these features to create a comprehensive representation. To address dimensionality concerns, Principal Component Analysis (PCA) is applied. The dataset, comprising the fused features and corresponding labels, is then loaded for the final step - classification using a BPNN. The network is configured for feed-forward backpropagation, distinguishing fingerprint patterns into categories such as arch, left loop, right loop, tented, and whorl. The evaluation metric used to measure the success of the classification process is accuracy. This approach aims to enhance fingerprint recognition by combining distinctive Gabor and minutiae features, ultimately achieving a more robust and precise identification system through the utilization of neural network-based classification.

Keywords—BPNN, PCA, Finger Print Images Dataset

I. INTRODUCTION

This study focuses on advancing fingerprint identification through the application of Back Propagation Neural Network (BPNN), a prominent deep learning technique.

Fingerprint identification plays a pivotal role in various domains such as law enforcement, access control, and immigration. Traditional methods of fingerprint analysis have relied on manual interpretation and feature extraction, which can be time-consuming and prone to error. By employing BPNN, which are capable of learning intricate patterns and features directly from raw data, this research seeks to enhance the accuracy and efficiency of fingerprint. The classification task involves categorizing fingerprints into distinct patterns, including Arch, Left Loop, Right Loop, Tented, and Whorl. BPNN offer significant advantages in handling complex data structures like fingerprint images, making them well-suited for this classification task. The utilization of BPNN in fingerprint identification holds promise for improving biometric security systems, streamlining identification processes, and enhancing overall security measures.

II. DATASET

The dataset consists of a total of 125 images, of which 25 are for whorl, 25 are for right loop, 25 are for left loop, 25 are for arch, and 25 are for tented, which are black and white images. Out of these, 70% is used for training the classifier, and 30% is used for testing.

III. METHODOLOGY

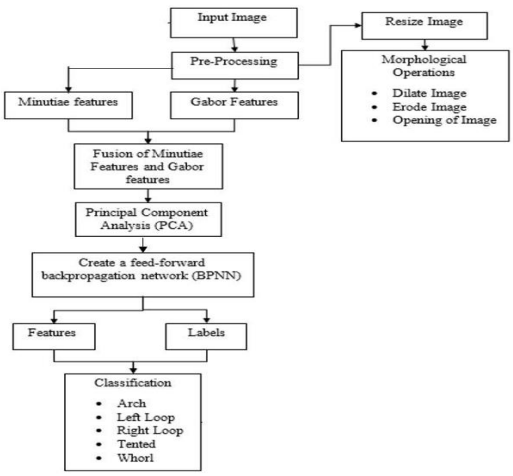


Fig 1: Block Diagram for Fingerprint dentification

Pre-Processing:

Preprocessing refers to the manipulation and transformation of raw data into a format that is more suitable for analysis, modelling, or other computational tasks.

1.Image Resizing

Fingerprint images often come in various sizes and resolutions. Resizing them to a standard size, such as 127x127 pixels, helps in achieving uniformity and reducing computational complexity. Resizing maintains the aspect ratio of the original image while fitting it into the specified dimensions (127x127 in this case). This step ensures that all fingerprint images have the same dimensions, facilitating further processing and analysis.

2. Erosion:

Erosion is a morphological operation that shrinks the boundaries of objects in a binary image. In the context of fingerprint image processing, erosion helps in removing small noise and thinning the ridges, making them more uniform and enhancing the quality of the fingerprint features This operation is typically applied using a structuring element (kernel) that moves through the image and removes pixels based on their local neighbourhood.

3.Dilation:

Dilation is the opposite of erosion; it expands the boundaries of objects in a binary image. In fingerprint preprocessing, dilation can help in filling gaps in the ridges and thickening them, which can improve feature extraction and matching accuracy. Like erosion, dilation is performed using a structuring element, which moves through the image and adds pixels to the objects based on their local neighbourhood.

4. Area of Opening:

After erosion and dilation operations, calculating the area of opening involves determining the size or extent of the changes made to the fingerprint image. This metric can provide insights into the effectiveness of erosion and dilation in enhancing the fingerprint features while minimizing noise and preserving essential details. The area of opening can be calculated by finding the difference between the area of the original fingerprint region and the area of the processed (eroded and dilated) region.

Formula:

Area of Opening=Area of Original Image−Area of Processed I mage

The area of the original image can be calculated by counting the number of foreground (non-zero) pixels in the binary image before preprocessing.

The area of the processed image can be calculated similarly after applying erosion and dilation operations.

IV.FEATURE EXTRACTION

1.Gabor Feature Extraction:

Gabor features are widely used in image processing and pattern recognition tasks, including fingerprint identification systems. Gabor filters are specifically designed to capture texture information from images.
Gabor features can be extracted by convolving the input image with a bank of Gabor filters at different scales and orientations. The response of each filter represents the local texture information captured by that filter. Features such as mean, variance, energy, and entropy can be computed from the filter responses to represent the texture characteristics of the fingerprint image.

1.1. Gabor filters:

Gabor filters are a class of linear filters used for texture analysis. They are defined by sinusoidal waveforms modulated by Gaussian envelopes. Gabor filters are localized in both spatial and frequency domains, making them suitable for capturing texture features at different scales and orientations.
In the context of fingerprint identification systems, Gabor features can be extracted from fingerprint images to represent the distinctive texture patterns of ridges and valleys. These features are robust to variations in illumination and noise, making them suitable for fingerprint recognition tasks.

Gabor Filter Equation:

$$g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \exp\left(i\left(2\pi\frac{x'}{\lambda} + \psi\right)\right)$$

$$x' = x\cos\theta + y\sin\theta \text{ and } y' = -x\sin\theta + y\cos\theta.$$

After extracting Gabor features from the fingerprint image, a feature vector is formed by concatenating the feature values obtained from different filters. This feature vector serves as a compact representation of the fingerprint image's texture information, which can be used for fingerprint identification and matching tasks.

2. Minutiae feature extraction:

Minutiae extraction is a fundamental step in fingerprint recognition systems, where the distinctive features of fingerprint patterns are identified and extracted. Minutiae points represent the ridge endings and bifurcations present in the fingerprint image. The most common types of minutiae are ridge endings and ridge bifurcations.

2.1. Ridge Ending: A ridge ending is a point where a ridge terminates. It appears as a single ridge that does not continue further.

2.2. Ridge Bifurcation: A ridge bifurcation is a point where a ridge divides into two branches. It appears as a Y-shaped structure.

Minutiae extraction is a critical component of fingerprint recognition systems and forms the basis for fingerprint matching algorithms. Accurate and reliable minutiae extraction is essential for achieving high-performance fingerprint recognition systems used in various applications, including biometric authentication, forensic analysis, and access control.

3. Feature Fusion:

Fused features combining Gabor and minutiae features can enhance the robustness and discriminative power of fingerprint recognition systems. You can concatenate the Gabor feature vector with the minutiae feature vector to create a single feature representation for the fingerprint.

V.FEATURE SELECTION

Principal Component Analysis (PCA) transforms high-dimensional data into a lower-dimensional space while preserving the most important information.

In the context of fingerprint recognition, PCA can be applied to reduce the dimensionality of the feature space while retaining the essential characteristics of the fingerprint patterns.

By reducing the dimensionality, PCA can help mitigate the curse of dimensionality, improve computational efficiency, and enhance the performance of fingerprint recognition algorithms.

In Principal Component Analysis (PCA), after obtaining the coefficients (eigenvectors) and scores (principal components) from the decomposition of the covariance matrix, you can use various methods for feature selection or further analysis. One such method is using kurtosis, which measures the "tailedness" or the degree of peakedness of a distribution.

kurtosis can be used to assess the shape of the distribution of principal components and prioritize the selection of those with higher kurtosis values. Here's the formula for calculating kurtosis:

The formula for sample kurtosis Kurt is given by:

$$\text{Kurt}(x) = E\left[\left(\frac{x-\mu}{\sigma}\right)^4\right]$$

VI.NETWORK TRAINING

In the Network Training we use classifiers. Classifiers are algorithms that are used to map input features to output categories. Neural networks are a powerful class of classifiers that can learn complex relationships between the input features and output categories. In this paper Back Propagation neural network are used and for weight Updating, LM Optimization are used.

6.1 Back Propagation Neural Network (BPNN)

Back propagation Neural Network (BPNN) is a type of artificial neural network that is trained using the backpropagation algorithm. It is a type of feedforward neural network, where the data flows in one direction from input to output layer through multiple hidden layers.

The backpropagation algorithm is a supervised learning algorithm, which means that it requires labelled data for training. The algorithm is used to adjust the weights and biases of the connections between the neurons in the network in order to minimize the error between the predicted output and the actual output. The activation function used in BPNN is Sigmoid activation function.

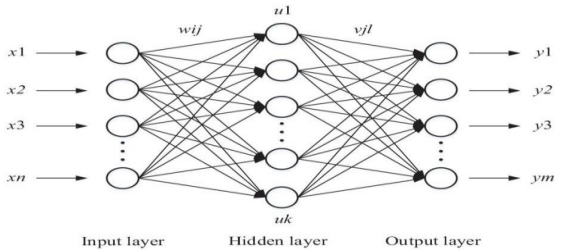


Fig.2: BPNN architecture

Input Layer: This layer consists of neurons that represent the input features. Each neuron corresponds to one feature of the input data.

Hidden Layers: Hidden layers serve as intermediary stages between the input and output layers within a neural network. Neurons within these hidden layers establish connections with neurons in both the preceding and succeeding layers. The number of hidden layers and neurons in each layer is configurable and depends on the complexity of the problem.

Activation function: An activation function is a key component of artificial neural networks, serving as a non-linear transformation applied to the output of a neuron or a layer of neurons.

Output Layer: The quantity of neurons within the output layer is contingent upon the nature of the problem under consideration. For example, in a binary classification problem, there would be one neuron for each class, while in a regression problem, there would be a single neuron.

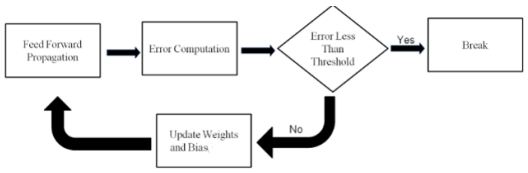


Fig.3: Proposed Model Algorithm

1.Initialization:
Randomly assign initial values to the weights and biases of the network. Let us denote the weights between input layer and hidden layer as *Win*, where *i* represents the input neuron and *j* represents the hidden neuron. Similarly, denote the weights between hidden layer and output layer as *Vjk* , where *j* represents the hidden neuron and *k* represents the output neuron. Also, initialize biases *bj* and *ck* for the hidden and output layers, respectively.

2.Forward Pass:
For each input sample *x*, compute the activations of the hidden layer neurons

$$h_j = \sigma(\sum_{i=1}^{n} W_{ij}x_i + b_j)$$

Where σ is the activation function for the hidden layer.
Similarly, compute the activations of the output layer neurons

$$y_k = \sigma(\sum_{j=1}^{m} V_{jk}h_j + c_k)$$

Where σ is the activation function for the output layer.

3.Calculate Loss:
Compute the loss between the predicted output *yk* and the actual output *tk* using a suitable loss function such as Mean Squared Error (MSE):

$$E = \frac{1}{2}\sum_{k=1}^{p}(t_k - y_k)^2$$

4.Backward Pass:
Compute the gradient of the loss function with respect to the output layer activations:

$$\frac{\partial E}{\partial y_k} = y_k - t_k$$

Update the weights and biases between the hidden layer and the output layer using the gradient descent algorithm:

$$V_{jk} \leftarrow V_{jk} - \eta\frac{\partial E}{\partial V_{jk}}$$

$$c_k \leftarrow c_k - \eta\frac{\partial E}{\partial c_k}$$

Where η is the learning rate.
Similarly, compute the gradient of the loss function with respect to the hidden layer activations:

$$\frac{\partial E}{\partial h_j} = \sum_{k=1}^{p}\frac{\partial E}{\partial y_k}\cdot V_{jk}\cdot\sigma'(h_j)$$

Where σ' is the derivative of the activation function for the hidden layer.
Update the weights and biases between the input layer and the hidden layer:

$$W_{ij} \leftarrow W_{ij} - \eta\frac{\partial E}{\partial W_{ij}}$$

$$b_j \leftarrow b_j - \eta\frac{\partial E}{\partial b_j}$$

5.Repeat:
Continue executing steps 2 through 4 either for a predetermined number of iterations or until convergence is achieved.

6.2 Levenberg-Marquardt optimization (LM)

Levenberg-Marquardt (LM) is a widely used optimization algorithm for solving nonlinear least squares problems, often employed in training neural networks. It's an iterative method that combines aspects of both steepest descent and Gauss-Newton methods. The main idea is to adjust the parameters (weights and biases in the context of neural networks) in a way that minimizes a specified error function.

In the realm of fitting a parameterized mathematical model to a dataset, least squares problems emerge as the objective involves minimizing the sum of squared differences of the errors between the model function and a set of data points. If a model is linear in its parameters, the least square objective is quadratic in the parameters. This objective may be minimized with respect to the parameters in one step via the solution to a linear matrix equation. When dealing with nonlinear fit functions, solving least squares problems requires iterative methods. These algorithms iteratively refine the model parameters to minimize the sum of squared errors between the model and data points. The Levenberg-Marquardt algorithm integrates two numerical minimization approaches: gradient descent and Gauss-Newton methods. In gradient descent, parameters are adjusted in the direction of steepest descent to reduce the sum of squared errors. Conversely, the Gauss-Newton method treats the least square function as locally quadratic in parameters, aiming to minimize this quadratic. The Levenberg-Marquardt algorithm behaves more like gradient descent when parameters are distant from their optimal value and shifts towards Gauss-Newton when parameters approach optimality.

LM Algorithm:

1.Initialization: Start with an initial guess for the parameters (weights and biases in the case of neural networks). Choose an initial value for the damping parameter (λ). Define the error function to be minimized (e.g. Mean Square Error).

2. Evaluation: Compute the error function (e.g., Mean Square Error) using the current parameter values. Calculate the Jacobian matrix, which contains the partial derivatives of the outputs with respect to the parameters. This matrix helps in approximating the local gradient of the error function.
The objective function represents the error we want to minimize. In the context of neural networks, it's often the mean square error (MSE) between the predicted output and the actual output.

Objective Function:

$$J(b_0, b_1) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2$$

3.Parameter Update Rule: Update the parameters using the LM update rule, which combines aspects of gradient descent and Gauss-Newton methods.
Adjust the damping parameter (λ) to control the step size. If the error decreases, decrease λ to allow larger steps; if the error increases, increase λ to take smaller steps.

The update rule is given by:

$$\left[\boldsymbol{J}^{\mathsf{T}}\boldsymbol{W}\boldsymbol{J} + \lambda \boldsymbol{I}\right] \boldsymbol{h}_{\mathrm{lm}} = \boldsymbol{J}^{\mathsf{T}}\boldsymbol{W}(\boldsymbol{y} - \hat{\boldsymbol{y}}) \ ,$$

where small values of the damping parameter λ result in a Gauss-Newton update and large values of λ result in a gradient descent update. The damping parameter λ is initialized to be large so that first updates are small steps in the steepest-descent direction. If any iteration happens to result in a worse approximation (χ 2 (p + h_{lm}) > χ2 (p)), then λ is increased. As the solution progresses and improves, the Levenberg-Marquardt method decreases the value of λ, gradually converging towards the behavior of the Gauss-Newton method. Consequently, the solution tends to accelerate towards the local minimum.

4. Convergence Check:
Check for convergence criteria, such as:
i. Whether the change in the error function is below a specified threshold.
ii. Whether the change in the parameters is below a specified threshold.
iii. Maximum number of iterations reached.

5.Iteration:
If convergence criteria are not met, repeat steps 2-4 until convergence is achieved. Otherwise, stop the algorithm and consider the current parameter values as the optimized solution.

Throughout these phases, the algorithm iteratively refines the parameter values to minimize the error function. By combining gradient descent with the Gauss-Newton method and adjusting the damping parameter, LM strikes a balance between convergence speed and stability, making it effective for optimizing nonlinear least squares problems, including neural network training.

VII. RESULTS AND DISCUSSIONS

The Following Figure is Confusion Matrix.
A confusion matrix is a table that is used to evaluate the performance of a classification model by comparing the predicted and actual class labels of a set of test data. The matrix shows the number of true positives, true negatives, false positives, and false negatives, arranged in a tabular form.
In a binary classification problem, a confusion matrix typically has two rows and two columns. The rows represent the predicted class labels (positive or negative), while the columns represent the actual class labels.

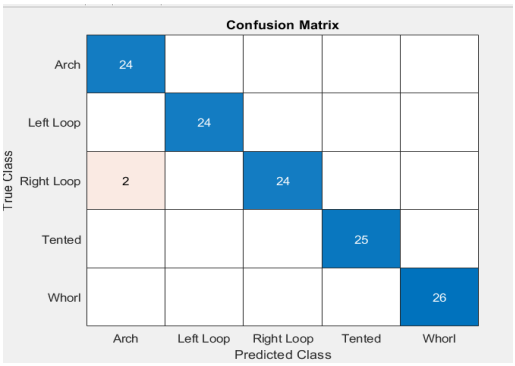


Fig.4: Confusion Matrix of BPNN(Fused Features Of both gabor and Minutiae)

These Fig-4 is useful for calculating the True Positives, True Negatives, False Positives, and False Negatives. These Parameters are used for calculating the accuracy, precision, recall, specification, sensitivity and F1 score of the network.

TP=24 FP=2 FN=0 TN=99

Performance Metrices :

Method	Formula	BPNN
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$	98.40%
Precision	$\frac{TP}{TP+FP}$	92.30%
Recall	$\frac{TP}{TP+FN}$	100.00%
Specificity	$\frac{TN}{TN+FP}$	98.00%
Sensitivity	$\frac{TP}{TP+FN}$	100.00%
F1 Score	$\frac{2*precision*recall}{precision+recall}$	95.99%

Table1: Results of BPNN

Method	Accuracy
BPNN (Gabor and Minutiae)	93.60
CNN (Gabor and Minutiae)	78.00
BPNN (Gabor Features)	84.00
BPNN (Minutiae Features)	84.00

Table 2: Comparison of Accuracy for different methods

Comparison of Accuracy:

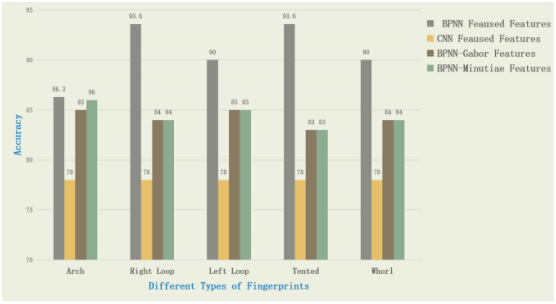


Fig.5: comparison of Accuracy Bar graph

VIII.CONCLUSION

In conclusion, the fusion of Gabor and Minutiae features coupled with a BPNN classifier presents a robust approach to fingerprint identification. Through the systematic process involving preprocessing, feature extraction, fusion, dimensionality reduction, and classification, the system achieves accurate recognition of various fingerprint patterns including arches, loops, tented arches, and whorls. The application of morphological operations, PCA for dimensionality reduction, and the utilization of a BPNN classifier contribute to the system's reliability and efficiency. This methodology demonstrates promising results in enhancing the accuracy of biometric authentication systems, thereby ensuring secure access control. Moving forward, further research could explore optimizations to streamline the process and potentially integrate additional features or algorithms to enhance the system's performance in real-world applications.

REFERENCES

[1] Hasan H, Abdul-Kareem S. Fingerprint image enhancement and recognition algorithms: a survey. Neural Compute Appl 2013;(23):1605–10.

[2] Win KN, Li K, Chen J, Viger PF, Li K. Fingerprint classification and identification algorithms for criminal investigation: a survey. Fut Gener Compute Syst 2020; 110:758–71.

[3] Nazarkevych M, Riznyk O, Samotyy V, Dzelendzyak U. Detection of regularities in the parameters of the Ateb-Gabor method for biometric image filtration. East Eur J Enterp Technol 2019;1(2–97):57–65.

[4] Shemmary ENAAl. Classification of fingerprint images using neural networks technique. J Eng (JOE) 2012;1(3):40–8.

[5] Leung KC, Leung CH. Improvement of fingerprint retrieval by a statistical classifier. IEEE Trans Inform Forens Secur 2011;6(1):59–69.

[6] Wan GC, Li MM, Xu H, Kang WH, Rui JW, Tong MS. Finger-net: pixel-wise segmentation method for partially defective fingerprint based on attention gates and U-net. Sensors 2020; 20:4473.

[7] AlShehri H, Hussain M, AboAlSamh H, AlZuair M. A large-scale study of fingerprint matching systems for sensor interoperability problem. Sensors 2018;18: 1008.