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

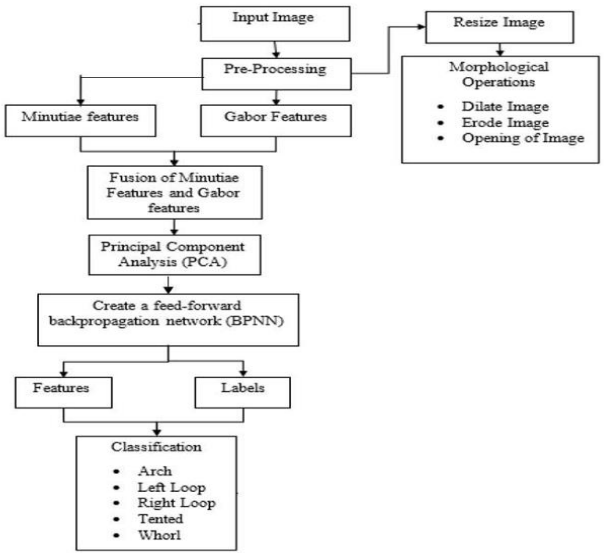


Fig 1: Block Diagram for Fingerprint dentification

**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.

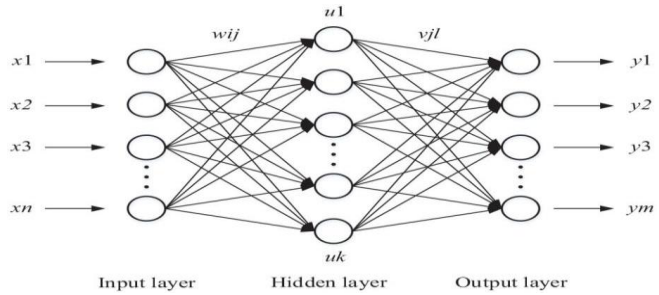


Fig.2: BPNN architecture

**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.

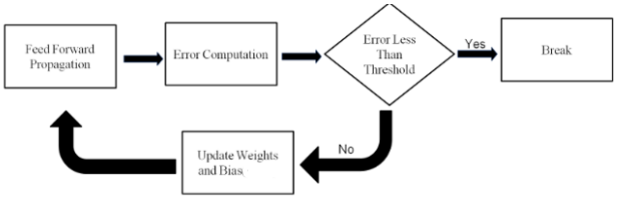


Fig.3: Proposed Model Algorithm

**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 $\sigma$ 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 $\sigma$ 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 $\eta$ 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 $\sigma'$ 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:

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

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 ($\chi$ 2 (p + $h_{lm}$) > $\chi$2 (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.

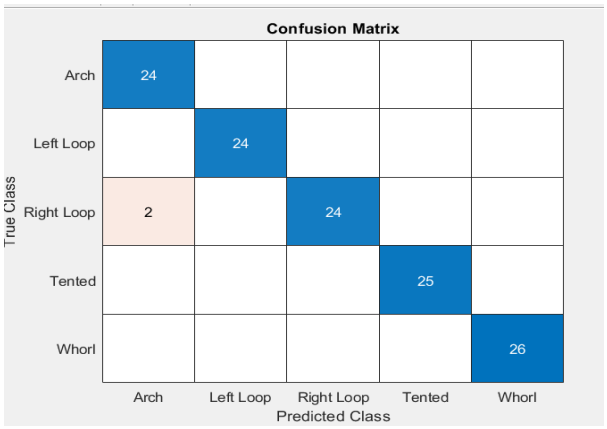


Fig.4: Confusion Matrix of BPNN(Fused Features Of both gabor and Minutiae)

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:**

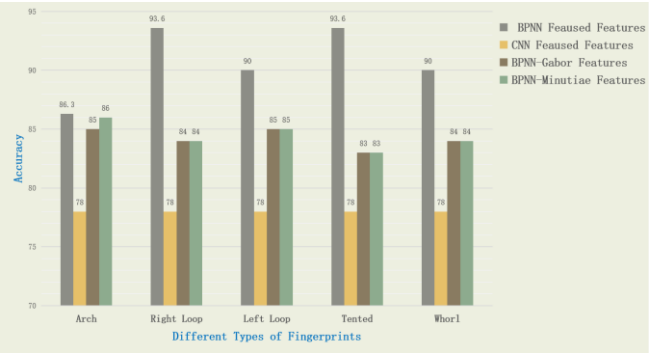


Fig.5: comparison of Accuracy Bar graph

## 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.