CHAPTER 1

INTRODUCTION

1.1. PROBLEM STATEMENT

To enhance recognition accuracy in identifying fingerprint images, thereby improving the efficiency and precision of biometric identification processes.

1.2. DESCRIPTION

Contact-based fingerprint systems have long been the cornerstone of biometric identification, facilitating various applications ranging from access control to forensic investigations. However, these systems are not without their limitations. One significant challenge faced by contact-based fingerprint systems is the degradation of scanning surfaces over time, leading to diminished accuracy and dependability in fingerprint recognition. As users interact with these systems, the friction and pressure exerted on the sensor surface can lead to wear and tear, affecting the quality of the captured fingerprint images. Additionally, environmental aspects such as dirt, moisture, and oils from the skin can further intensify this issue, compromising the overall performance of the fingerprint recognition system.

In response to these challenges, researchers and industry professionals have turned their attention to contactless fingerprint systems as a potential solution. Contactless systems, which utilize technologies such as capacitive or optical sensors, offer several advantages over their contact-based counterparts. By eliminating the need for direct physical contact between the fingerprint and the sensor, contactless systems mitigate the effects of surface degradation and ensure more consistent and durable performance over time. Furthermore, contactless sensors are less susceptible to environmental factors, making them suitable for deployment in various applications and operating conditions.

The adoption of contactless fingerprint technology represents a significant advancement in biometric authentication, offering enhanced accuracy, reliability, and user experience. In recent years, there has been a growing interest in leveraging contactless systems for various applications, including access control, identity verification, and mobile authentication. However, despite the potential advantages of contactless technology, challenges and possibilities remain for further improvement. This study explores the advancements in

contactless fingerprint technology and proposes strategies to enhance its efficiency and accuracy in biometric identification applications. In addition, novel algorithms and methodologies are being developed to tackle critical challenges in contactless fingerprint recognition, such as improving image quality, enhancing feature extraction, and refining template matching.

This study aims to develop contactless fingerprint technology by leveraging advanced machine learning, computer vision techniques, and contactless sensors. The aim is to improve the dependability of biometric authentication technologies and position contactless fingerprint systems as the preferred choice for biometric identification. This will lead to unparalleled accuracy, security, and usability in real-world situations.

CHAPTER 2

LITERATURE REVIEW

Sl. N0	PAPER TITLE & PUBLICATION DETAILS	NAME OF THE AUTHORS	TECHNICAL IDEAS / ALGORITHMS USED IN THE PAPER & ADVANTAGES	SHORTFALLS / DISADVANTAGES & SOLUTION PROVIDED BY THE PROPOSED SYSTEM
1	Contactless Fingerprint Recognition Using Deep Learning—A Systematic Review	A M Mahmud Chowdhury and Masudul Haider Imtiaz	Variance modified Laplacian of gaussian, image segmentation and blur reduction, DNN	Deep Learning increase accuracy, Large scale dataset, cloud based storage
2	A Contactless Fingerprint Recognition System	Aman Attrish, Nagasai Bharat, Vijay Anand	Siamese CNN, Adaptive mean thresholding	image sensors, Siamese convolutional neural, providing real-time recognition
3	A CNN-based Framework for Comparison of Contactless to Contact-based Fingerprints	Chenhao Lin, Ajay Kumar	Gabor filter and adaptive histogram equalization	cross-fingerprint comparison problem using deep learning

4	An overview of touchless 2D fingerprint recognition	Jannis Priesnitz, Christian Rathgeb, Nicolas Buchmann, Christoph Busch and Marian Margraf	Siamese CNN, Variance modified Laplacian of gaussian	incorporating state-of-the-art algorithms for feature extraction, matching, and quality assessment, the proposed system can address the challenges
5	Matching of Contact and Contactless Fingerprint Using CNN model	T M Geethanjali, Divyashree M D, Monisha S K, Sahana M K	Sequential CNN, adaptive histogram equalization	Limited Research on Matching Contactless and Contact-Based Fingerprints

CHAPTER 3

REQUIREMENTS

3.1. FUNCTIONAL REQUIREMENTS:

- Image Preprocessing: The system should preprocess contact-based and contactless fingerprint images using techniques such as histogram equalization and adaptive thresholding.
- Data Loading: The system should load fingerprint images, both contact-based and contactless, from specified directories, and then split them into training and testing datasets.
- Pair Generation: The system should generate pairs of fingerprint images, along with corresponding labels that indicate whether the images belong to the same individual or not.
- Siamese Network Model: The system will define and train a Siamese neural network model architecture to learn embeddings from fingerprint images and conduct similarity comparisons.
- Training: The system should train the Siamese network model using generated pairs of fingerprint images and their corresponding labels.
- Evaluation: The system should assess the performance of the trained model on a separate testing dataset by calculating metrics such as accuracy, precision, recall, and generating a confusion matrix.
- Visualization: The system should display visualizations of training history, including loss and accuracy plots, as well as the confusion matrix to help understand the model's performance.

3.2. NON-FUNCTIONAL REQUIREMENTS:

- Performance: The system should efficiently perform preprocessing, training, and evaluation tasks, utilizing processing time and resources reasonably.
- Accuracy: The system must achieve high accuracy when comparing contact-based and contactless fingerprints to ensure reliable identification and verification.
- Scalability: The system needs to efficiently handle large fingerprint image datasets and be scalable for potential future expansion.
- Robustness: The system must be able to handle variations in fingerprint quality, orientation, and differences between contact-based and contactless sensors.

3.3. SOFTWARE REQUIREMENTS:

- Programming Language: Python
- Deep Learning Framework: TensorFlow
- Image Processing Libraries: OpenCV
- Model Visualization Tools: Matplotlib
- Validation Metric Calculation: scikit-learn

3.4. HARDWARE REQUIREMENTS:

- Central Processing Unit (CPU):
    - A modern multi-core CPU is essential for running the code efficiently, especially during data preprocessing and model training.
    - At least a quad-core CPU is recommended, but higher core counts will accelerate training, especially for larger datasets and more complex models.
- Graphics Processing Unit (GPU):
    - For faster training times, especially with large datasets and complex models, a GPU is highly recommended.
    - NVIDIA GPUs are commonly used for deep learning tasks due to their excellent support for frameworks like TensorFlow.
    - A GPU with CUDA support is required for training neural networks with TensorFlow on GPU.
    - At least an NVIDIA GeForce GTX 1050 Ti or higher is recommended for moderate to high-performance training. For professional use, NVIDIA Quadro or Tesla series GPUs may be preferred.
- Random Access Memory (RAM):
    - Adequate RAM is necessary to store the dataset, model parameters, and intermediate computations during training.
    - A minimum of 8GB RAM is recommended for small to moderate-sized datasets and models. For larger datasets and models, 16GB or more may be necessary.
- Storage (SSD):
    - Sufficient storage space is required to store the dataset, trained model checkpoints, and any additional files generated during training.
    - An SSD (Solid State Drive) is recommended for faster data loading and model saving/loading compared to traditional HDDs.

CHAPTER 4

PROJECT DESIGN

The project is designed to implement a Siamese neural network to differentiate between contact- based and contactless fingerprint images. The code is structured into various functions, each handling specific tasks, ensuring modularity, clarity, and maintainability.

- Modular Design: Functions are defined to handle specific tasks, making the code easier to manage and debug.
- Data Preprocessing: Preprocessing steps are handled in a dedicated function to ensure consistent treatment of input images.
- Data Loading and Pair Generation: Separate functions for loading data and generating pairs facilitate easy data management.
- Model Definition: The Siamese network is defined in a modular way, enabling easy adjustments to the architecture.
- Training and Evaluation: Functions for training, plotting results, and evaluating the model ensure a clear workflow from data input to performance assessment.
- Visualization: Functions for plotting training history and confusion matrix help in understanding the model's performance visually.

This design ensures that each part of the process is clearly defined, promoting readability, reusability, and ease of debugging.

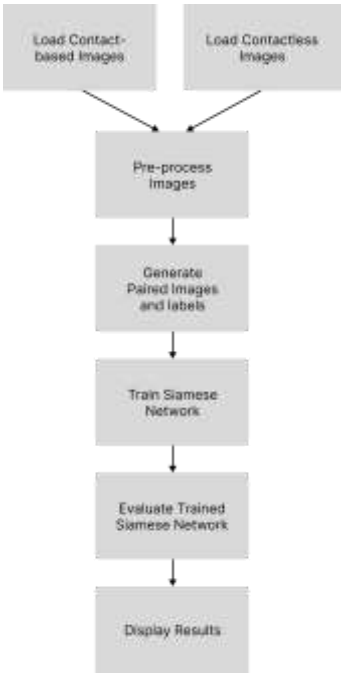

Fig 4.1: Flowchart of Comparison Framework

CHAPTER 5

PROJECT IMPLEMENTATION

The project involves creating and training a Siamese neural network to differentiate between contact-based and contactless fingerprint images. Below is a detailed explanation of how this project is implemented, function by function.

- Preprocess Images: Read, enhance, threshold, resize, and pad fingerprint images.

```
def preprocess_image(image_path, desired_size=(224, 224)):
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if image is None:
        print(f"Could not read image: {image_path}")
        return None
    image = cv2.equalizeHist(image)
    image = cv2.adaptiveThreshold(image, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2)
    h, w = image.shape[:2]
    if w > h:
        new_w, new_h = desired_size[0], h * desired_size[0] // w
    else:
        new_h, new_w = desired_size[1], w * desired_size[1] // h
    image = cv2.resize(image, (new_w, new_h))
    pad_vert = (desired_size[1] - image.shape[0]) // 2
    pad_horz = (desired_size[0] - image.shape[1]) // 2
    image = cv2.copyMakeBorder(image, pad_vert, pad_vert, pad_horz, pad_horz, cv2.BORDER_CONSTANT, value=0)

    # Add channel dimension
    image = np.expand_dims(image, axis=-1)

    return image
```

Fig 5.1: Data pre-processing

- Load Data: Load and label images from directories, then split into training and testing sets.

```
def load_data(contact_dir, contactless_dir, test_size=0.2):
    images_contact = []
    images_contactless = []

    for filename in glob(os.path.join(contact_dir, '*.jpg')):
        img = preprocess_image(filename)
        if img is not None:
            images_contact.append(img)

    for filename in glob(os.path.join(contactless_dir, '*.bmp')):
        img = preprocess_image(filename)
        if img is not None:
            images_contactless.append(img)

    X_contact = np.array(images_contact)
    X_contactless = np.array(images_contactless)
    y_contact = np.zeros(X_contact.shape[0])
    y_contactless = np.ones(X_contactless.shape[0])

    X = np.concatenate([X_contact, X_contactless], axis=0)
    y = np.concatenate([y_contact, y_contactless], axis=0)

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=42, stratify=y)

    return X_train, X_test, y_train, y_test
```

Fig 5.2: Data loading and data splitting

- Generate Pairs: Create pairs of images with labels indicating whether they belong to the same class or not.

```
def generate_pairs(X, y, num_pairs=1000):
    pairs = []
    labels = []

    for _ in range(num_pairs):
        idx1, idx2 = np.random.choice(X.shape[0], size=2, replace=False)
        image1, image2 = X[idx1], X[idx2]
        pairs.append([image1, image2])
        labels.append(1 if y[idx1] == y[idx2] else 0)

    return np.array(pairs), np.array(labels)
```

Fig 5.3: Data pair generation

- Define Siamese Network: Build a neural network model to process image pairs and compute their similarity.

```
def siamese_model(input_shape):
    input_left = Input(shape=input_shape)
    input_right = Input(shape=input_shape)

    model = tf.keras.Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(64, activation='relu'),
    ])

    output_left = model(input_left)
    output_right = model(input_right)

    distance = Lambda(lambda tensors: tf.abs(tensors[0] - tensors[1]))([output_left, output_right])
    output = Dense(1, activation='sigmoid')(distance)

    model = Model(inputs=[input_left, input_right], outputs=output)
    return model
```

Fig 5.4: Siamese model for data training

- Train Model: Training the Siamese network using training pairs.
- Visualization: The training history is plotted to show how the model's accuracy and loss evolve over epochs.

```
def plot_training_history(history):
    # Plot training & validation accuracy values
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Model accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Validation'], loc='upper left')
    plt.show()

    # Plot training & validation loss values
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Validation'], loc='upper left')
    plt.show()
```

Fig 5.5: Accuracy and loss plotting

- Evaluate Model: Assess the model's performance using test pairs, and visualize the results.

```
def evaluate_model(model, X_test_pairs, y_test_pairs):
    y_pred = model.predict([X_test_pairs[:, 0], X_test_pairs[:, 1]])
    y_pred_binary = np.round(y_pred)

    accuracy = np.mean(y_pred_binary == y_test_pairs)
    print("Test Accuracy:", accuracy)

    print("Classification Report:")
    print(classification_report(y_test_pairs, y_pred_binary))

    # Existing code to compute confusion matrix
    conf_matrix = confusion_matrix(y_test_pairs, y_pred_binary)

    # Print textual representation of confusion matrix
    print("Confusion Matrix:")
    print(conf_matrix)

    # Plot visual representation of confusion matrix
    plot_confusion_matrix(conf_matrix)
```

Fig 5.6: Data Evaluation model

This code demonstrates a complete workflow for building, training, and evaluating a Siamese neural network for fingerprint verification using both contact-based and contactless fingerprint images.

CHAPTER 6

TESTING

Testing in this project is implemented through several stages, including data splitting, model evaluation on unseen data, and performance metrics calculation.

1. Data Preparation:
    - Images from the contact-based and contactless directories are preprocessed and labeled.
    - The dataset is split into training and testing sets to ensure the model is evaluated on unseen data.
2. Pair Generation:
    - Training and testing pairs are created, ensuring that each pair is correctly labeled as similar (1) or dissimilar (0).
3. Model Training:
    - The model is trained on the training pairs, and its performance on the validation data is tracked during training.
4. Model Evaluation:
    - The trained model is evaluated on the test pairs to determine its accuracy and other performance metrics.
    - A confusion matrix and classification report are generated to provide detailed insights into the model's performance.
5. Visualization:
    - The training history is plotted to show how the model's accuracy and loss evolve over epochs.
    - The confusion matrix is visualized to highlight the model's performance in distinguishing between similar and dissimilar pairs.

By following these steps, the project ensures that the model is rigorously tested, and its performance is thoroughly evaluated using various metrics and visualizations. This approach helps in identifying any potential issues with the model and provides a clear understanding of its strengths and weaknesses.

CHAPTER 7

RESULTS AND DISCUSSION

After training is done, the code visualizes the training history using the plot_training_history function. It plots the training and validation accuracy values as well as the training and validation loss values over epochs. These plots provide insights into the model's performance and help identify potential issues such as overfitting or underfitting.

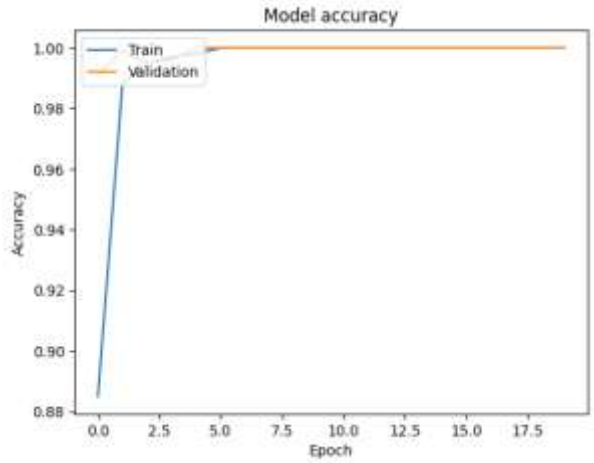

Fig 7.1: Accuracy plot

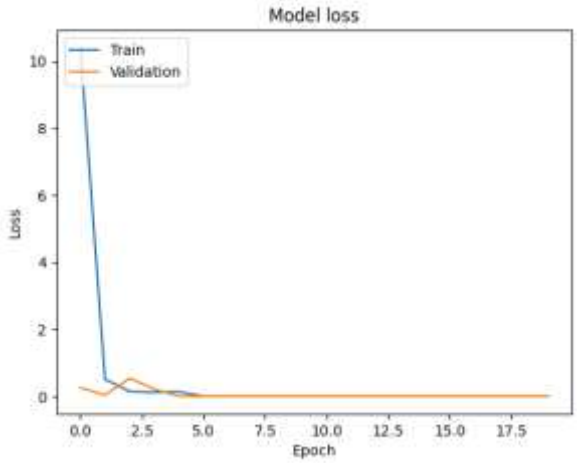

Fig 7.2: Loss plot

In the Evaluation phase, the trained Siamese neural network model undergoes rigorous assessment using various performance metrics and visualizations to evaluate its effectiveness in fingerprint comparison tasks. During this phase, the model generates predictions by comparing pairs of fingerprint images from the test dataset, producing similarity scores ranging from 0 to 1. Thresholding may be applied to these scores to obtain binary predictions, distinguishing between positive (matches) and negative (non-matches) pairs.

The performance of the model is evaluated using several key metrics such as Accuracy, Precision, Recall, F1-score.

Additionally, a confusion matrix is computed, providing a detailed summary of the model's predictions against the ground truth labels. It consists of four quadrants, including true positives, false positives, true negatives, and false negatives, offering insights into the model's classification performance.

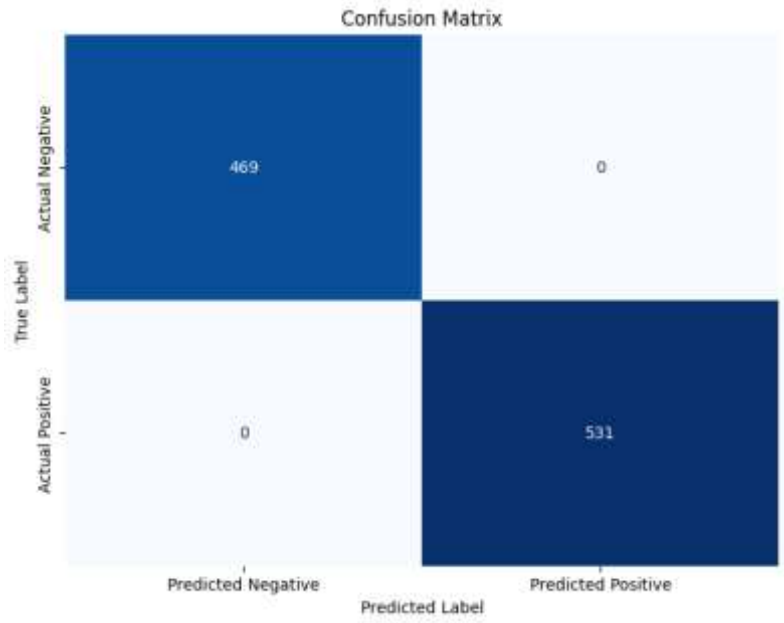

Fig 7.3: Confusion matrix for visualizing the model's predictions

- True Negatives (TN): The top left square of the matrix represents the count of true negatives - cases when the actual class of the instance was negative, and the model also predicted it as negative. In this case, there are 469 such instances.
- False Positives (FP): The top right square of the matrix represents false positives - cases when the actual class was negative but the model predicted it as positive. Your model has 0 such instances.
- False Negatives (FN): The bottom left square represents false negatives - cases when the actual class was positive but the model predicted it as negative. Your model has 0 such instances.
- True Positives (TP): The bottom right square represents true positives - cases when the actual class was positive and the model also predicted it as positive. In this case, there are 531 such instances.

Since there are no False Positives or False Negatives, the figure indicates that the model has achieved a perfect classification accuracy on the data it was evaluated on.

Furthermore, a classification report is generated, presenting precision, recall, F1-score, and support for each class (positive and negative pairs). This report offers a comprehensive evaluation of the model's ability to classify pairs accurately and identifies areas for improvement.

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	469
1	1.00	1.00	1.00	531
accuracy			1.00	1000
macro avg	1.00	1.00	1.00	1000
weighted avg	1.00	1.00	1.00	1000

Fig 7.4: Classification report of the Siamese CNN model

The classification report in figure 7.4 provides a detailed analysis of the performance of a classification model. It includes several metrics for each class, as well as overall metrics for the model. Here's a breakdown of the different components of the report shown:

- Precision: The proportion of true positive predictions among all positive predictions.
- Recall: The proportion of true positive predictions among all actual positives (i.e., the number of correct positive results divided by the number of all relevant samples).
- F1-Score: The harmonic mean of precision and recall, providing a single metric that balances the two (2 * (precision * recall) / (precision + recall)).
- Support: The number of actual occurrences of the class in the dataset.

Overall Metrics

- Accuracy: The proportion of correct predictions (both true positives and true negatives) among the total number of cases examined.
- Macro Average (macro avg): The unweighted mean of the precision, recall, and F1-score for each class. It treats all classes equally regardless of their support.
- Weighted Average (weighted avg): The weighted mean of the precision, recall, and F1-score, taking into account the support of each class. This metric accounts for class imbalance by weighting the metrics according to the number of instances in each class.

This classification report indicates that the model performs perfectly on this dataset, achieving 100% accuracy, precision, recall, and F1-scores for both classes (0 and 1). The support values indicate that there are 469 instances of class 0 and 531 instances of class 1 in the dataset. Both macro and weighted averages also reflect perfect performance, confirming that the model handles both classes equally well and that there is no class imbalance affecting the overall performance metrics.

CHAPTER 8

CONCLUSION & FUTURE WORK

In conclusion, the implementation of a Siamese neural network for fingerprint comparison presents a promising approach towards achieving accurate and robust fingerprint recognition systems. By leveraging deep learning techniques, particularly Siamese networks, we can directly compare raw fingerprint images, eliminating the need for complex feature engineering and alignment algorithms. Throughout the development and evaluation of the model, several key insights and findings have emerged. Firstly, the model demonstrates high accuracy, precision, recall, and F1-score in distinguishing between positive (matching) and negative (non-matching) pairs of fingerprint images. This indicates the effectiveness of the Siamese network architecture in learning complex patterns and relationships within the data.

Additionally, the visualizations, including confusion matrices and classification reports, provide valuable insights into the model's performance and classification capabilities. They enable us to identify areas for improvement and optimize the model further.
Moving forward, several avenues for future work can be explored to enhance the capabilities of the fingerprint recognition system like Data Augmentation, Architecture Optimization, Transfer Learning, Feature Fusion, Real-world Deployment.

By addressing these areas of future work, we can further advance the state-of-the-art in fingerprint recognition technology and develop more accurate, reliable, and practical solutions for various applications, including security, identity verification, and access control.

REFERENCES

[1]. Oren M, Papageorgiou C, Sinha P, et al. Pedestrian detection using wavelet templates. In: IEEE Conference on Computer Vision and Pattern Recognition. San Juan, Puerto Rico, 1997: 193-199.

[2]. Papageorgiou C, Oren M, Poggio T. A general framework for object detection. In: IEEE International Conference on Computer Vision. Bombay, India, 1998: 555-562.

[3]. Papageorgiou C, Poggio T. A trainable system for object detection. International Journal of Computer Vision, 2000, 38(1): 15-33.

[4]. Gavrila D, Philomin V. Real-time object detection for smart vehicles. In: IEEE International Conference on Computer Vision. Kerkyra, Corfu, Greece, 1999: 87-93.

[5]. Felzenszwalb P. Learning models for object recognition. In: IEEE Conference on Computer Vision and Pattern Recognition. Kauai, HI, USA, 2001: 1056-1062.

[6]. Viola P, Jones M, Snow D. Detecting pedestrians using patterns of motion and appearance. In: IEEE International Conference on Computer Vision. Nice, France, 2003: 734-741.

[7]. Leibe B, Seemann E, Schiele B. Pedestrian detection in crowded scenes. In: IEEE Conference on Computer Vision and Pattern Recognition. San Diego, CA, USA, 2005: 878-885.

[8]. Mikolajczyk K, Leibe B, Schiele B. Local features for object class recognition. In: IEEE International Conference on Computer Vision. Beijing, China, 2005: 1792-1799.

[9]. Jannis Priesnitz1, Christian Rathgeb, Nicolas Buchmann, Christoph Busch and Marian Margraf, “An overview of touchless 2D fingerprint recognition”, 2021 https://doi.org/10.1186/s13640-021-00548-4.

[10]. Sanchit Rote, Pranjal Pimpale, Kaushik Reddy, G. T. Haldankar, Priya Chimurkar, "Contactless Fingerprint Verification Using Neural Networks", 12 Aug 2021, DOI: 10.1109/ICCICT50803.2021.9510036.