Supervised Deep Learning

PROJECT BASED LEARNING

Submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

in COMPUTER SCIENCE & ENGINEERING

By

Naman Arora
(04511502722)

on

Fake Currency Detection

Guided by

Dr. Preeti Nagrath

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

BHARATI VIDYAPEETH'S COLLEGE OF ENGINEERING

(AFFILIATED TO GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY, DELHI)

DELHI – 110063

Fake Currency Detection

1. Overview

The demand for robust and reliable security systems has driven the rapid adoption of **biometrics**, which utilizes unique physiological or behavioral traits for automatic recognition. Among the various biometric modalities, **fake currency** remains the most widely accepted and deployed technology globally. Fingerprints—the patterns of ridges and valleys on the surface of the fingertip—are considered the **gold standard** due to their two fundamental properties:

1. **Uniqueness:** No two individuals, even identical twins, have the same fingerprint pattern.
2. **Permanence:** The patterns are fully formed during fetal development and remain unchanged throughout a person's life, barring severe scarring or damage.

2. Problem Statement: The Need for Automation

Traditional methods of fingerprint analysis were manual, time-consuming, and prone to human error, often relying on trained experts to categorize and match prints. With the massive growth of large-scale databases (like those used in law enforcement and national IDs), efficient automation became critical. This project addresses the challenge of creating an automated system capable of both:

- **Classification:** Grouping fingerprints into primary categories (e.g., Arch, Loop, Whorl) to reduce the search space for efficient indexing.
- **Identification:** Pinpointing a specific individual from a database by accurately matching their unique fingerprint pattern.

3. Project Objective and Methodology

This project aims to leverage advanced computational techniques to create a high-accuracy, end-to-end fingerprint recognition system.

Objective	Methodology
Feature Extraction	Employing **Deep Learning**, specifically a **Convolutional Neural Network (CNN)**. Unlike traditional approaches that manually extract features like minutiae (ridge endings and bifurcations), the CNN automatically learns the most descriptive features directly from the raw image data.

Objective	Methodology
Classification	Implementing a **multi-class classification model** to categorize fingerprints into the five common classes of the Henry system, thus improving database search efficiency.
Working Model	Developing a robust, working Python application using **TensorFlow/Keras** for model building and **OpenCV** for image preprocessing, demonstrating a modern approach to biometric security.

4. Machine Learning & Deep Learning Context

The inherent complexity, noise, and intra-class variation (e.g., rotation, skin distortion, pressure differences) in fingerprint images make them challenging for traditional algorithms. **Deep Learning (DL)** offers a powerful solution:

- **Machine Learning (ML) Foundation:** Historically, ML models like Support Vector Machines (SVMs) and Random Forests were used, but they required labor-intensive preprocessing and feature engineering.
- **Deep Learning Advantage:** CNNs overcome this limitation by utilizing multiple processing layers to learn **hierarchical representations**—from basic edges and textures in early layers to complex global and local ridge structures in deeper layers. This ability to learn optimal features makes the CNN the most effective tool for pattern recognition tasks like fingerprint analysis.

Model Specification: CNN Architecture

The model uses a sequential CNN architecture, a standard and effective design for image classification tasks. The goal is to build a hierarchy of feature extractors that can capture both local patterns (minutiae) and global patterns (ridge flow).

1. Model Summary

Layer Type	Output Shape	Parameters	Purpose
Input Layer	(96, 96, 1)	0	Defines the input size: 96x96 pixels, 1 channel (grayscale).
Conv2D (32 filters)	(94, 94, 32)	320	Learns basic features (edges, lines).

Layer Type	Output Shape	Parameters	Purpose
MaxPooling2D	(47, 47, 32)	0	Reduces dimensionality, retains most important features.
Conv2D (64 filters)	(45, 45, 64)	18,496	Learns more complex, localized features (e.g., minutiae).
MaxPooling2D	(22, 22, 64)	0	Further reduction of feature map size.
Conv2D (128 filters)	(20, 20, 128)	73,856	Learns highly abstract features (e.g., core and delta structure).
MaxPooling2D	(10, 10, 128)	0	Final pooling before flattening.
Flatten	(12,800)	0	Converts 3D feature maps into a 1D vector for the Dense layers.
Dense (512 units)	(512)	6,554,112	The main feature vector; acts as the **fingerprint embedding**.
Dropout (0.5)	(512)	0	Prevents overfitting by randomly setting 50% of inputs to zero.
Dense (5 units)	(5)	2,565	Output layer for the 5 classes (Arch, Loop, Whorl, etc.).

2. Key Components

- **Activation Function: ReLU (Rectified Linear Unit)** is used in all hidden layers to introduce non-linearity, which allows the model to learn complex relationships.
- **Loss Function: Categorical Cross-Entropy** is used because this is a multi-class classification problem with 5 mutually exclusive classes.
- **Optimizer: Adam Optimizer** is selected for its efficiency and good performance across various deep learning tasks.
- **Metrics: Accuracy** is the primary metric used to evaluate the model's performance.

Training Dataset Description

The accuracy and robustness of any fingerprint recognition system heavily depend on the quality and size of the training data.

1. Typical Datasets (e.g., NIST SD4)

Real-world academic projects rely on publicly available standard benchmarks:

Dataset	Type	Purpose in Project
NIST Special Database 4 (NIST SD4)	Public Domain	Commonly used for fingerprint classification research. Contains roughly 2,000 pairs of fingerprints.
FVC (Fingerprint Verification Competition)	Competition Data	Used for testing verification and identification algorithms. Features multiple sensors and noise variations.

2. Data Characteristics

Characteristic	Description	Importance for CNN
Image Type	Grayscale, usually 8-bit.	Reduces computational complexity compared to color images. The relevant information (ridges/valleys) is intensity-based.
Resolution	Varies (often 500 DPI). The images are resized to a uniform **96x96 pixels** for model input.	A consistent input size is mandatory for the CNN architecture.
Class Distribution	Imbalanced. Typically, **Loop (Left/Right)** patterns are the most common, while **Arch** and **Tented Arch** are the least common.	**Requires careful handling** (e.g., weighted loss functions or oversampling) to prevent the model from becoming biased toward the more frequent classes.
Variations (Noise)	Contains noise from real-world capture, including: rotational shift, translation, pressure distortion, cuts, and dryness/wetness of the finger.	The CNN's convolutional layers and the use of data augmentation help the model become **robust** to these variations.

3. Data Preprocessing and Augmentation

To ensure the CNN learns generalized features rather than memorizing training examples, the following steps are performed:

- **Rescaling (Normalization):** All pixel values are scaled from the original range (0-255) to a floating-point range of **0.0 to 1.0**. This is a standard practice to improve gradient descent convergence during training.

- **Data Augmentation:** The ImageDataGenerator applies geometric transformations to the training images on-the-fly, creating synthetic variations. This includes:
    - Rotation
    - Zooming
    - Shifting (width/height)
    - Horizontal Flips

This augmentation is crucial for building a **robust identification system** that can handle differences in how a user places their finger on a scanner.

For Classification (Grouping by Pattern Type)

The primary standard for the **classification** task (Arch, Loop, Whorl) is the **NIST Special Database 4 (NIST SD4)**.

- **Key Features:**
    - **Purpose:** Specifically designed for research on automated fingerprint **classification**.
    - **Labels:** Contains approximately 2,000 pairs of fingerprints (4,000 images total) that are pre-labeled according to the **Henry classification system** (Arch, Tented Arch, Right Loop, Left Loop, Whorl).
    - **Utility:** Its clear, predefined classes make it the ideal benchmark for training a CNN to categorize the global ridge structure of a fingerprint.

For Identification (Matching to an Individual)

For the **identification** and **verification** tasks, datasets that provide multiple, varied impressions of the *same* finger are crucial. The **Fingerprint Verification Competition (FVC) Series** is the industry standard.

- **Key Features:**
    - **Purpose:** Designed for testing the robustness of matching and identification algorithms.
    - **Multiple Impressions:** Provides **multiple impressions (usually 8-12)** for each unique finger, captured under varying conditions (rotation, pressure, noise, skin distortion, different time gaps).
    - **Sub-Databases:** The FVC series (FVC2000, FVC2002, FVC2004, etc.) are separated into categories based on the **sensor type** used (e.g., optical sensor, capacitive sensor, synthetic generator), allowing you to test generalization across hardware.
    - **Utility:** Training on FVC data forces the deep learning model to learn a highly stable feature representation (embedding) that is invariant to common real-world noise and distortion, which is essential for accurate individual identification.

Code:

```
def identify_individual(new_image_path, base_cnn_model, training_image_paths, training_embeddings):
    if not training_image_paths or training_embeddings is None or len(training_image_paths) !=
training_embeddings.shape[0]:
        print("Error: Training data is not properly loaded or provided.") return None, None
    try:
        new_image = load_and_preprocess_image(new_image_path, target_size=(IMG_SIZE, IMG_SIZE))
        new_image = np.expand_dims(new_image, axis=0) except Exception as e:
        print(f"Error preprocessing new image: {e}") return None, None
     try:
        new_embedding = base_cnn_model.predict(new_image) except Exception
    as e:
        print(f"Error getting embedding for new image: {e}") return None, None
    distances = np.sum(np.abs(new_embedding - training_embeddings), axis=1)

    min_distance_index = np.argmin(distances)
    min_distance = distances[min_distance_index] closest_training_image_path =
    training_image_paths[min_distance_index]
    identified_individual = os.path.basename(closest_training_image_path).split('_')[0]
    return identified_individual, min_distance

base_cnn_model_for_identification = siamese_model.get_layer('sequential_1') #
data_dir = os.path.join(base_path, 'dataset_FVC2000_DB4_B', 'dataset', 'train_data')
all_image_paths_for_embedding = [os.path.join(data_dir, f) for f in os.listdir(data_dir) if f.endswith('.bmp')]
training_image_paths_for_embedding = train_paths # Use the paths from the train_test_split
training_embeddings = []
print("Generating embeddings for training images...") for img_path in
training_image_paths_for_embedding:
    try:
        img = load_and_preprocess_image(img_path, target_size=(IMG_SIZE, IMG_SIZE))
        img = np.expand_dims(img, axis=0) embedding =
        base_cnn_model_for_identification.predict(img)
        training_embeddings.append(embedding[0])
    except Exception as e:
        print(f"Error generating embedding for {img_path}: {e}")



training_embeddings = np.array(training_embeddings)
print(f"Generated embeddings for {training_embeddings.shape[0]} training images.")
sample_new_image_path = test_paths[0]
sample_true_individual = os.path.basename(sample_new_image_path).split('_')[0] print(f"\nIdentifying individual for

sample image: {sample_new_image_path}")

identified_individual, min_distance = identify_individual( sample_new_image_path,
    base_cnn_model_for_identification, training_image_paths_for_embedding,
    training_embeddings
)
if identified_individual is not None:
    print(f"Identified  Individual:  {identified_individual}") print(f"Minimum Distance to closest training
    image: {min_distance:.4f}") print(f"True Individual (for this sample): {sample_true_individual}")
```

```
similarity_threshold = 0.5
if min_distance < similarity_threshold:
    print("Match is considered positive (below similarity threshold).") else:
    print("Match is considered negative (above similarity threshold).")
```

Output:

```
Generated embeddings for 560 training images.

Identifying individual for sample image: /root/.cache/kaggl
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
Identified Individual: 00005
Minimum Distance to closest training image: 31.4703
True Individual (for this sample): 00005
Match is considered negative (above similarity threshold).
```

Result:

NIST DB4 (3-class classification):

- **Dataset Size:** 1679 training images, 479 validation images, and 243 testing images across 3 classes.
- **Model:** A standard CNN model.
- **Training Performance:** The model trained for 10 epochs showed increasing accuracy on the training data.
- **Validation Performance:** The validation accuracy also improved over epochs, reaching approximately 86.01%.
- **Test Performance:** The model achieved a test loss of approximately 0.4624 and a test accuracy of approximately 0.8272 on the unseen test set.

FVC2000 DB4-B (Individual Identification - Siamese Network):

- **Dataset Size:** The original dataset was split into 560 training images, 120 validation images, and 120 testing images for creating pairs.
- **Model:** A Siamese network with a shared base CNN.
- **Training Performance:** The Siamese model trained for 10 epochs showed increasing accuracy on the training pairs.
- **Validation Performance:** The validation accuracy fluctuated and reached approximately 73.21%.
- **Test Performance:** The model achieved a test loss of approximately 1.0447 and a test accuracy of approximately 0.6518 on the unseen test pairs.

Summary Comparison:

The CNN model trained for 3-class classification on the NIST DB4 dataset achieved a higher test accuracy (around 83%) compared to the Siamese network trained for individual identification on the FVC2000 DB4-B dataset (around 65%). This suggests that classifying the general pattern of a fingerprint might be an easier task than identifying a specific individual from their fingerprint with the current Siamese network architecture and training on this specific dataset split.