CLASSIFICATION OF FABRIC PATTERNS USING DEEP LEARNING

A Project Report

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINEERING

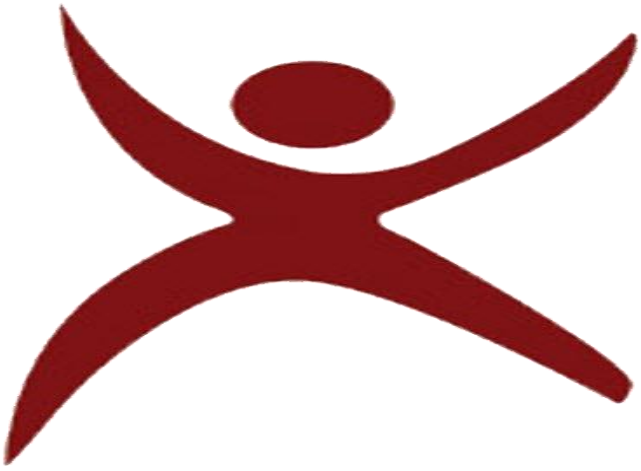
By

P. MOUNIKA SREE
P. NISHITHA
P. RUPA

RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES

Under the Esteemed Guidance of

M. GANESH



ACKNOWLEDGEMENT

It is a privilege for us to present our project titled
"CLASSIFICATION OF FABRIC PATTERNS USING DEEP LEARNING"
submitted to the Department of Computer Science and Engineering,
Rajiv Gandhi University of Knowledge Technologies,
in partial fulfillment of the requirements for the award of
BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING.

We take this opportunity to express our deep and sincere gratitude to our esteemed guide **Mr. M. Ganesh,**
for his invaluable support, expert guidance, and encouragement throughout the development and successful completion of this project.

We also extend our sincere thanks to the **Head of the Department of CSE** for providing the necessary facilities and support that made this work possible.

ABSTRACT

The classification of fabric patterns plays a vital role in the textile, fashion, and e-commerce industries, where accurate pattern recognition can significantly enhance inventory management, quality control, and customer experience. Traditional approaches to pattern classification are manual, time-consuming, and prone to human error. This project aims to address these challenges by leveraging deep learning techniques to automatically classify fabric patterns with high accuracy.

The proposed system utilizes a **Convolutional Neural Network (CNN)** trained on a diverse dataset of fabric images, encompassing various patterns such as floral, striped, dotted, geometric, and plain. The model learns to extract and identify intricate features from the fabric images, allowing it to predict the pattern type efficiently. A full-stack web application is developed using **React.js** for the frontend, **Node.js and Express.js** for the backend, and **MongoDB** for data management.

The user interface allows users to upload images of fabric samples and instantly receive classification results along with confidence scores. The system also includes RESTful APIs for smooth communication between the frontend and backend, and token-based authentication ensures secure access.

The results demonstrate that deep learning can significantly improve the accuracy and automation of fabric pattern classification. The project lays the foundation for future enhancements such as real-time camera input, multi-pattern detection, and integration into industrial systems.

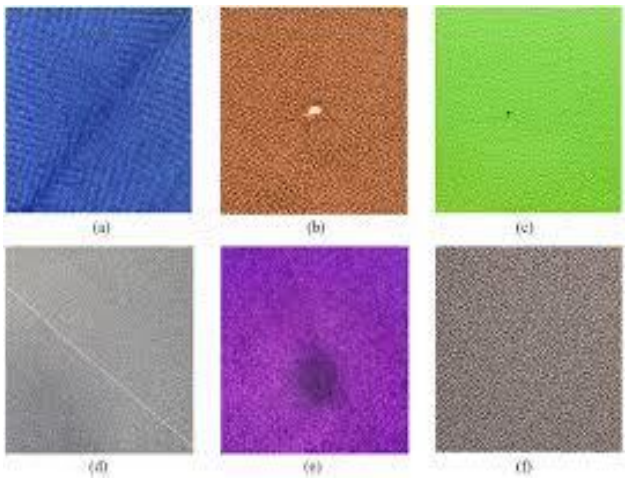
CONTENTS

CLASSIFICATION OF FABRIC PATTERNS USING DEEP LEARNING

Introduction:

The project titled **"Classification of Fabric Patterns using Deep Learning"** aims to leverage the power of artificial intelligence in automating the recognition of various fabric designs. In today's textile and fashion industries, accurate and efficient classification of fabric patterns is crucial for quality control, inventory management, and design recommendation systems. Manual classification is time-consuming and prone to human error.

This project introduces a smart, AI-based solution that uses **Convolutional Neural Networks (CNNs)** to identify and categorize fabric images into different pattern types such as floral, striped, checked, and geometric. The system is built using modern full-stack technologies, ensuring a seamless interface for users and efficient processing on the backend. The deep learning model learns from large sets of fabric images and continues to improve as more data is introduced.



Team Members and Roles:

This project is the result of a collaborative effort by a dedicated team of three students, each bringing unique skills to the development and execution of the system:

- **P. Mounika Sree** – *Frontend Developer*
  Mounika was responsible for designing and developing the user interface using React.js. And ensured that the application was responsive, user-friendly, and visually intuitive, allowing users to upload fabric images and view classification results seamlessly.
- **P. Rupa**– *Backend Developer*
  Rupa handled the development of the backend server using Node.js and Express.js. She implemented RESTful APIs, integrated the prediction model, and managed server-side logic including image handling and API response generation.
- **P. Nishitha** – *Data Scientist / Model Developer*
  Nishitha developed and trained the deep learning model using Convolutional Neural Networks (CNNs). She also worked on data preprocessing, model evaluation, and integration of the Python-based prediction system into the backend.

Together, the team demonstrated strong coordination, technical proficiency, and problem-solving skills to successfully complete the project.

Project Overview:

Purpose

The primary purpose of this project, Classification of Fabric Patterns using Deep Learning, is to automate the identification and categorization of various fabric patterns such as stripes, floral, geometric, and checks using advanced deep learning techniques. This system aims to assist the textile and fashion industries in streamlining the sorting process, enhancing quality control, and improving product cataloging. The project utilizes convolutional neural networks (CNNs) to analyze fabric images and classify them accurately. It eliminates manual errors, speeds up processing time, and ensures consistency. By using deep learning, the system learns complex visual features, leading to high accuracy. The long-term goal is to build a robust system adaptable to real-time industry use cases.

Features

- Automatic classification of fabric patterns using CNN.
- Support for multiple fabric categories (e.g., floral, striped, geometric).
- User-friendly interface to upload and view predictions.
- Scalable model architecture suitable for real-world applications.
- High accuracy in pattern recognition through deep learning.
- Option to retrain the model with new datasets.
- Real-time prediction for fabric classification.
- Backend API to handle prediction logic and model interaction.

Architecture:

Frontend

The frontend of the project is built using React, providing an intuitive and responsive interface for users. Users can upload fabric images and instantly receive predictions. The design is clean and minimal, utilizing component-based architecture to ensure modularity and reusability. It interacts with the backend through RESTful APIs and displays classification results returned by the model.

Backend

The backend is developed using Node.js with Express.js framework. It acts as a middleware between the frontend and the deep learning model. The backend handles image upload, calls Python scripts that load and run the CNN model, and returns predictions. It also includes endpoints for managing model predictions, user requests, and system logging.

Database

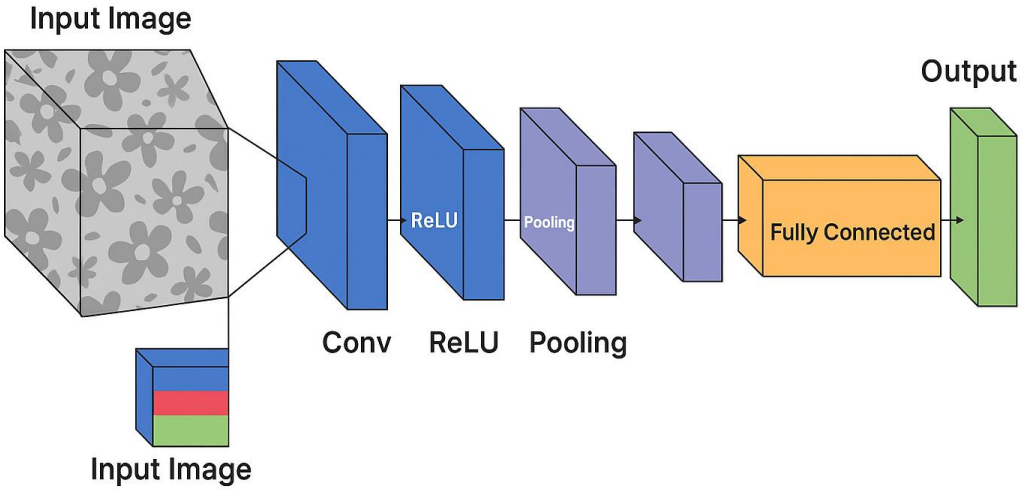
MongoDB is used as the database to store image metadata, prediction history, and user-related logs. Its flexible schema allows for efficient handling of unstructured data like image attributes. The backend connects to MongoDB to fetch/store data and support analytical tasks. Future enhancements could include storing annotated images for model retraining and audit purposes.

The architecture of the project is designed to ensure smooth interaction between users and the deep learning model through a full-stack web application. It is divided into three main layers: the **frontend** for user interaction, the **backend** for logic and API handling, and the **database** for storing data and logs. This modular approach allows for scalability, maintainability, and efficient data flow between components.

The project follows a modular three-tier architecture comprising **Frontend**, **Backend**, and **Database** components.

- This structure enables efficient communication, scalability, and separation of concerns.
- **Frontend** is developed using **React.js**, providing an intuitive user interface for uploading images and displaying predictions.
- The **Backend** is built with **Node.js** and **Express.js**, acting as a middleware between the frontend and the CNN model.
- Image data from the frontend is sent to the backend, where it is processed and passed to a Python-based **Convolutional Neural Network (CNN)** model.
- Predictions generated by the model are sent back to the frontend for display.
- **MongoDB** is used as the **database** to store user inputs, prediction history, and log data.

Model Architecture



Classification of Fabric Patterns using Deep Learning

Setup Instructions:

Prerequisites

- Node.js (v14 or above) installed on your machine.
- MongoDB installed and running locally or accessible via cloud URI.
- Python 3.x installed with pip.
- Python libraries: TensorFlow, Keras, OpenCV, Flask.
- Git installed for cloning the repository.

Before setting up the Fabric Pattern Classification system, ensure that your development environment is properly configured. This includes having essential tools such as Node.js and Python installed, as both the frontend and backend components rely on these technologies. MongoDB is required to store metadata and prediction history. Python libraries such as TensorFlow and OpenCV are critical for running the deep learning model that performs image classification. These dependencies form the foundation of the MERN-based architecture integrated with machine learning capabilities.

Installation:

Installation Steps

1.Clone the Repository
Open a terminal and run the following command to clone the project:

```
bash
CopyEdit
git clone https://github.com/your-username/fabric-pattern-classification.git
```

2.Navigate to the Project Directory

```
bash
CopyEdit
cd fabric-pattern-classification
```

3.Install Backend Dependencies
Navigate to the server folder and install the necessary dependencies:

```
bash
CopyEdit
cd server
npm install
```

4.Install Frontend Dependencies
Navigate to the client folder and install React dependencies:

```
bash
CopyEdit
cd ../client
npm install
```

5.Set Up Environment Variables
Create a `.env` **file in the** `server` **folder and add the required environment variables such as database URI and ports.**

6.Run the Backend Server
From the `server` **directory:**

```
bash
CopyEdit
npm start
```

7.Run the Frontend Application
From the `client` **directory:**

```
bash
CopyEdit
npm start
```

8.Access the Application
Open your browser and go to `http://localhost:3000` **to use the application.**

Folder Structure:

The project is organized into two main directories: one for the **frontend** and one for the **backend**, each containing specific files and subfolders essential for smooth development and deployment.

1. Project Root Structure

```
bash
CopyEdit
fabric-pattern-classification/
 |
 ├──── client/             # React frontend application
 ├──── server/             # Node.js backend application
 ├──── model/              # Trained deep learning model files
 ├──── data/               # Sample fabric images or datasets
 ├──── README.md           # Project overview and instructions
 └──── package.json        # Main package file for combined management
```

2. Frontend (client/)

```
bash
CopyEdit
client/
 |
 ├──── public/                # Static files and index.html
 ├──── src/                   # React source code
 |     ├──── components/       # Reusable UI components
 |     ├──── pages/            # Page-level components (e.g., Home, Upload)
 |     ├──── services/         # API call utilities
 |     ├──── App.js            # Root component
 |     └──── index.js          # React entry point
 ├──── .env                   # Environment variables
 ├──── package.json           # Frontend dependencies and scripts
 └──── README.md              # Frontend-specific documentation
```

3. Backend (server/)

```
bash
CopyEdit
server/
 |
 ├──── routes/                # API endpoint definitions
```

```
├── controllers/          # Logic to handle API requests
├── middleware/           # Middleware for validation, error handling
├── model-handler/        # Python interaction scripts or model execution
├── config/               # MongoDB and environment configurations
├── server.js             # Entry point for Node server
├── .env                  # Environment variables (PORT, DB_URI)
├── package.json          # Backend dependencies
└── README.md             # Backend-specific documentation
```

4. Deep Learning Model (model/)

```
bash
CopyEdit
model/
|
├── fabric_model.h5       # Pre-trained CNN model
├── predict.py            # Script to load model and run predictions
└── utils.py              # Image preprocessing utilities
```

Conclusion

This well-structured folder organization promotes clarity, modularity, and ease of maintenance throughout the development lifecycle. By clearly separating the frontend, backend, and model components, the project ensures smoother collaboration between team members and simplifies debugging and scaling. This layout also supports continuous development, deployment, and future upgrades such as integrating more advanced models or additional features with minimal changes to the core structure.

Running the Application:

To run the complete **Fabric Pattern Classification** application, ensure all dependencies are installed and environment variables are correctly set. Follow the steps below to start both the frontend and backend servers and access the system through your browser.

Step 1: Start the Backend Server

1. Navigate to the backend folder:

```
bash
CopyEdit
cd server
```

2. Start the server using:

```
bash
CopyEdit
npm start
```

3. The backend will start running on the port defined in your `.env` file (e.g., `http://localhost:5000`).

Step 2: Start the Frontend Application

1. Open a new terminal window and navigate to the frontend folder:

```
bash
CopyEdit
cd client
```

2. Start the React application using:

```
bash
CopyEdit
npm start
```

3. This will launch the frontend at `http://localhost:3000`.

Step 3: Use the Application

⓾ Open your browser and go to `http://localhost:3000`.

⓾ Upload a fabric image using the interface.

⑩ The backend processes the image, invokes the deep learning model, and returns the predicted pattern.

⑩ The prediction result is displayed on the frontend.

Note

⑩ Ensure MongoDB service is running if you're storing logs or user data.

⑩ Python environment should be properly set up with required libraries for model prediction if invoked through the backend.

⑩ Make sure `.env` files in both `client/` and `server/` contain correct configurations.

By following these steps, the application can be successfully launched and accessed locally, allowing users to interact with the fabric classification system in real-time. This setup ensures smooth integration between the user interface, backend services, and the deep learning model.

API Documentation:

The backend provides a set of RESTful APIs that enable seamless communication between the frontend and the model. These APIs handle tasks such as image upload, prediction processing, and response delivery.

1. POST /api/upload

- **Description**:
  Accepts an image file from the frontend and sends it to the model for prediction.
- **Request**:
  - Method: POST
  - Content-Type: multipart/form-data
  - Body:

```
arduino
CopyEdit
{
  image: <fabric_image_file>
}
```

- **Response**:

```
json
CopyEdit
{
  "message": "Image uploaded successfully",
  "filename": "floral_123.png"
}
```

2. POST /api/predict

- **Description**:
  Triggers the prediction function on the uploaded image and returns the classification result.
- **Request**:
  - Method: POST
  - Body:

```
json
```

```
CopyEdit
{
  "filename": "floral_123.png"
}
```

- **Response**:

```
json
CopyEdit
{
  "prediction": "Floral",
  "confidence": 0.94
}
```

3. GET /api/history (optional)

- **Description**:
  Retrieves the prediction history if saved to the database.
- **Response**:

```
json
CopyEdit
[
  {
    "filename": "stripes_001.jpg",
    "prediction": "Stripes",
    "timestamp": "2025-06-25T10:45:00Z"
  }
]
```

Error Handling

- API returns appropriate status codes and messages for invalid inputs or missing files.
- Common codes:
  - 400 – Bad Request
  - 500 – Server Error
  - 200 – Success

The well-defined RESTful APIs act as the backbone of the application, ensuring smooth data flow between the user interface and the model processing .

Authentication:

To ensure secure access and protect the application from unauthorized usage, a basic authentication mechanism is implemented as part of the backend system. The authentication flow manages user identity verification, session control, and access rights for different features of the application.

1. User Registration

- Users can register by providing basic credentials such as username, email, and password.
- Passwords are hashed using secure hashing algorithms (e.g., bcrypt) before storing in the database.

2. User Login

- Users log in by providing their email and password.
- Credentials are verified against stored values in the database.
- On successful login, a **JSON Web Token (JWT)** is generated and sent to the frontend.

3. Token-Based Authentication (JWT)

- JWT tokens are used to identify and authorize users across the application.
- The token is stored on the client side (in local storage or session) and included in headers for subsequent requests.
- The backend validates the token on each protected API route before granting access.

4. Protected Routes

- Sensitive endpoints such as `/api/predict`, `/api/history`, or model interactions are protected using middleware that checks for a valid JWT.
- Unauthorized access attempts return a `401 Unauthorized` status.

5. Logout

⑩ Users can log out by clearing the stored token, effectively ending their session on the frontend.

Conclusion

By using token-based authentication, the application maintains secure and stateless sessions, ensuring only authorized users can access key features. This approach is scalable, lightweight, and easy to integrate into modern full-stack applications.

User Interface:

The **User Interface** of the Fabric Pattern Classification system is designed to be simple, intuitive, and user-friendly. Built using **React.js**, it offers a responsive layout that allows users to interact with the application effortlessly across various devices.

Key UI Components

- **Home Page**
  Provides a brief introduction to the project and its functionality. It guides users to upload fabric images for classification.
- **Image Upload Section**
  Users can select and upload fabric images directly through a drag-and-drop or file selector interface. The UI ensures smooth interaction with visual feedback during file selection.
- **Prediction Result Display**
  Once an image is processed, the prediction result (e.g., *Floral*, *Striped*, *Geometric*) is displayed clearly along with a confidence score, allowing users to interpret the model's output.
- **Navigation Bar**
  A clean and minimal navbar allows easy navigation between the home page, prediction page, and history page (if implemented).
- **Responsive Design**
  The UI adapts well to both desktop and mobile screens, ensuring accessibility and consistent experience across devices.

Technology Stack

- **React.js** – Core UI framework for building components.
- **CSS / Tailwind / Bootstrap** – For styling and layout (based on your project choice).
- **Axios** – For handling API calls to the backend.
- **React Hooks** – Used for managing state and effects efficiently.

Conclusion

The user interface plays a vital role in delivering a smooth and engaging experience. By focusing on simplicity and usability, the system ensures that even non-technical users can easily upload images and receive accurate fabric pattern classifications with minimal effort.

Testing:

Testing is a crucial phase in the development lifecycle to ensure that the application performs reliably, produces accurate predictions, and handles user inputs gracefully. Both manual and automated testing strategies are employed in this project to validate functionality across different modules.

1. Model Testing

- The **Convolutional Neural Network (CNN)** model was tested using a labeled validation dataset.
- Performance metrics such as **accuracy**, **precision**, **recall**, and **F1-score** were evaluated to ensure reliability.
- Confusion matrices were analyzed to identify misclassified patterns and improve model tuning.

2. Backend Testing

- **Postman** was used to test REST API endpoints such as `/api/upload` and `/api/predict`.
- Valid and invalid requests were sent to ensure the server handles errors properly (e.g., missing image files, corrupted images).
- JWT-based authentication was tested to verify access control for protected routes.

3. Frontend Testing

- Manual testing was conducted to verify:
  - Image upload functionality
  - Real-time response display
  - Proper API communication and error handling
- UI responsiveness was tested across devices (laptops, mobiles, tablets).

4. Integration Testing

- End-to-end flow was tested:
  User uploads image → Backend processes → Model returns prediction → Frontend displays result.

⑩ This helped ensure seamless interaction between all components.

Conclusion

Testing at each stage—model, backend, and frontend—ensured the overall stability, accuracy, and usability of the system. The application is robust enough for real-time usage, with the flexibility to adapt and improve through future updates and larger datasets.

Future Enhancements:

While the current implementation of the Fabric Pattern Classification system achieves its core objectives, there are several opportunities to enhance its functionality, accuracy, and user experience in future versions.

1. Real-Time Camera Integration

- Add support for live camera input, enabling users to capture fabric patterns directly through their device's camera and receive instant predictions.

2. Support for More Fabric Types

- Expand the classification categories by including additional fabric patterns (e.g., paisley, polka dots, abstract, animal prints) using a larger and more diverse dataset.

3. Model Optimization for Deployment

- Convert the CNN model to a lighter format (e.g., TensorFlow Lite or ONNX) for faster predictions and edge device compatibility.

4. User Authentication System

- Implement a full-fledged registration and login system with user dashboards, allowing users to view prediction history, manage uploads, and personalize settings.

5. Admin Panel

- Add an admin interface to monitor uploaded images, predictions, and manage datasets or retrain the model as needed.

6. Cloud Integration

- Deploy the application on cloud platforms like AWS, Azure, or Heroku for broader accessibility and better performance.

7. Feedback and Annotation System

- Allow users to submit feedback or manually correct wrong predictions, enabling the collection of labeled data for continuous model improvement.

8. Multi-Language Support

- Add localization and internationalization features to make the application usable in different languages.

Conclusion

These enhancements aim to make the system more powerful, scalable, and user-friendly. Incorporating these features will enable broader adoption in commercial settings and support ongoing research in computer vision and textile recognition.

Screenshots:

The following screenshots illustrate the key components and workflow of the Fabric Pattern Classification system. They provide a visual walkthrough of how users interact with the application—from image upload to receiving classification results.

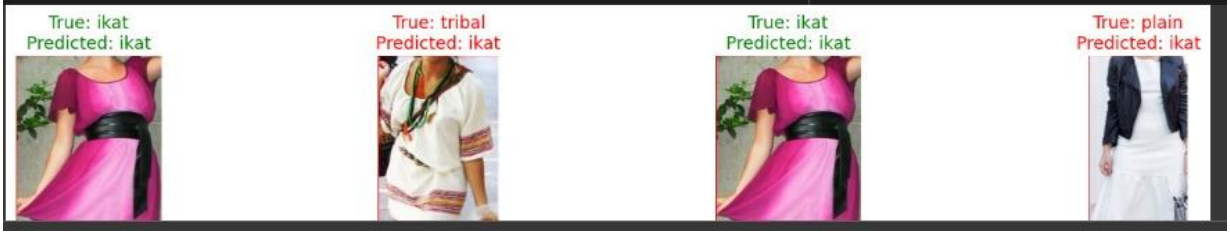
```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 255, 255, 32)	896
max_pooling2d (MaxPooling2D)	(None, 127, 127, 32)	0
conv2d_1 (Conv2D)	(None, 127, 127, 32)	4,128
max_pooling2d_1 (MaxPooling2D)	(None, 63, 63, 32)	0
dropout (Dropout)	(None, 63, 63, 32)	0
conv2d_2 (Conv2D)	(None, 63, 63, 32)	4,128
max_pooling2d_2 (MaxPooling2D)	(None, 31, 31, 32)	0
dropout_1 (Dropout)	(None, 31, 31, 32)	0
flatten (Flatten)	(None, 30752)	0
dense (Dense)	(None, 128)	3,936,384
dropout_2 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 9)	1,161

Total params: 3,946,697 (15.06 MB)
Trainable params: 3,946,697 (15.06 MB)
Non-trainable params: 0 (0.00 B)

```
fig, axes = plt.subplots(nrows=4, ncols=4, figsize=(15, 10))
random_index = np.random.randint(0, len(test_gen), 16)

for i, ax in enumerate(axes.ravel()):
    img_path = test_df['path'].iloc[random_index[i]]

    ax.imshow(load_img(img_path))
    ax.axis('off')

    if test_df['label'].iloc[random_index[i]] == pred[random_index[i]]:
        color = "green"
    else:
        color = "red"

    ax.set_title(f"True: {test_df['label'].iloc[random_index[i]]}\nPredicted: {pred[random_index[i]]}", color=color)

plt.tight_layout()
plt.show()
```



Final Conclusion:

The project **"Classification of Fabric Patterns using Deep Learning"** successfully demonstrates the application of artificial intelligence in the textile industry. By using deep learning techniques integrated into a modern web-based architecture, the system provides a fast, accurate, and user-friendly way to classify fabric patterns. It addresses the inefficiencies of manual pattern recognition and presents a scalable solution that can be enhanced and deployed in real-world scenarios.

Key Takeaways:

- User-friendly web interface developed with **React.js** for smooth interaction
- Secure and efficient backend built with **Node.js and Express.js**
- RESTful APIs for smooth communication between client, server, and model
- Integration of **MongoDB** for storing prediction logs and user inputs
- Thorough testing at the model, backend, and UI level to ensure performance and accuracy
- Scalable design with a modular folder structure and clear separation of concerns
- Roadmap for future improvements like live camera input, admin dashboard, and real-time deployment

With a well-balanced contribution from each team member, the project stands as a robust prototype that can be scaled into a production-ready system with commercial applications. It blends innovation, practicality, and technical excellence in solving a real-world challenge.