**Pattern Sense: Classifying Fabric Patterns Using Deep Learning**

**Project Documentation format**

**1. Introduction**

**Project Title: [Pattern Sense: Classifying Fabric Patterns Using Deep Learning]**

* **Team Members:**

1. Somineni Jamuna[22HM1A05C7]
2. Pullimi Sruthi[22HM1A05A4]
3. Shaik Munni[22HM1A05B6]
4. Sowdigalla khasim Peera[22HM1A05C8]

**2. Project Overview**

* **Purpose:**
* The purpose of "PATTERN SENSE: CLASSIFYING FABRIC PATTERNS USING DEEP LEARNING" is to develop a system that automatically identifies and categorizes different fabric patterns using deep learning techniques. This aims to automate a task that is currently often done manually, improving efficiency and accuracy in the textile industry.
* To automatically recognize and classify different fabric patterns (e.g., plain, satin, twill, stripes, plaids, floral) using deep learning, replacing manual inspection and handcrafted feature extraction with an end-to-end, scalable image analysis approach
* **Goals:**
* The main goal of "Pattern Sense: Classifying Fabric Patterns Using Deep Learning" is to automate the process of classifying fabric patterns, specifically using deep learning techniques to improve accuracy and efficiency compared to traditional manual methods.
* This involves developing a system that can accurately identify and categorize different fabric patterns from images.
* The primary goal is to move away from manual, labor-intensive methods of classifying fabric patterns, which are prone to errors and time-consuming
* Automated classification can significantly speed up the process of identifying and categorizing fabric patterns, leading to increased efficiency in textile production and management.
* **Features:**

**Dataset & Preprocessing**

* High-quality fabric images captured under controlled illumination, using consistent focal length and ISO settings for clarity
* Data augmentation to create robust variance: flips, rotations (e.g., every 30°), zoom, shear, brightness changes — boosting generalization and avoiding overfitting

**CNN Architectures & Transfer Learning**

* Pre-trained models like ResNet‑50, VGG‑16/19, Google Net/Inception are fine-tuned for fabric textures — combining strong feature abstraction with task adaptation
* Architecture improvements include identity shortcuts (ResNet) to combat vanishing gradients, small-kernel stacks (VGG), and inception modules for multi-scale feature capture

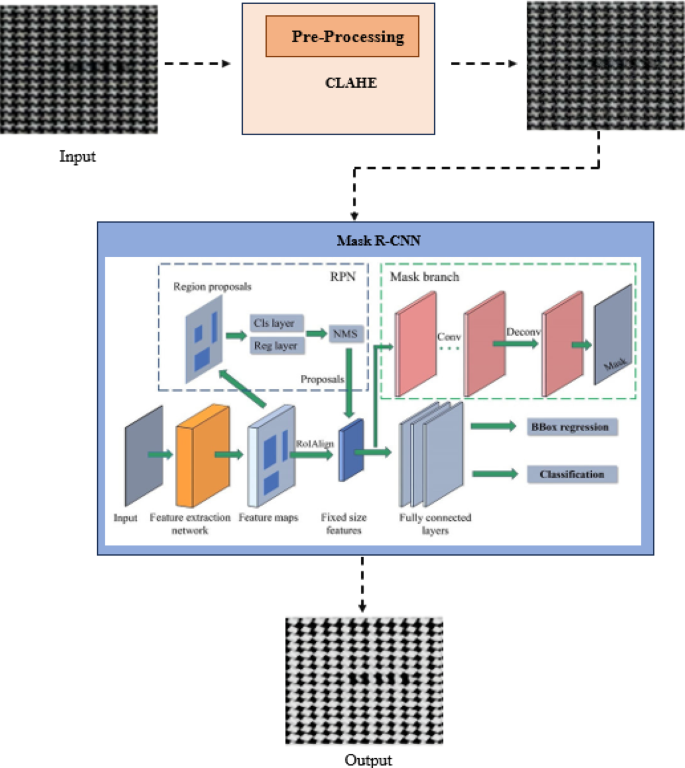
**Texture-Specific Feature Enhancements**

* Feature fusion: Combine CNN features with classical descriptors like HOG, HSV histograms, LBP, and GLCM to enrich shape and color cues
* Attention-enhanced networks such as DenseNet variants emphasize discriminative texture regions, boosting accuracy

**Scalability & Efficiency**

* Integration of depth wise-separable convolutions (e.g., Mobile Net-style) and channel pruning for lightweight and fast inference—vital for deployment on embedded devices
* Optional ensemble or segmentation heads for defect detection, allowing multi-task operation in production

**3. Architecture**



**4. Setup Instructions**

* **Prerequisites**:

To complete this project, you must require the following software and packages.

* Software Requirements:
  + Visual Studio Code (VS Code) or any Python-supported IDE
  + Python 3.10 for better suitable to all packages
* Python packages:
  + Open VS code terminal prompt etc.,
  + Type “pip install NumPy” and click enter.
  + Type “pip install pandas” and click enter.
  + Type “pip install scikit-learn” and click enter.
  + Type “pip install matplotlib” and click enter.
  + Type “pip install scipy” and click enter.
  + Type “pip install seaborn” and click enter.
  + Type “pip install tenser flow” and click enter.
  + Type “pip install Flask” and click enter
* **Installation:**
* Create a Virtual Environment (Optional but Recommended):

python -m venv .venv

source .venv/Scripts/activate For Windows

source .venv/bin/activate For Mac/Linux

* Install Required Packages:

Ensure you run:

pip install numpy

pip install pandas

pip install scikit-learn

pip install matplotlib

pip install scipy

pip install seaborn

pip install tensorflow

pip install Flask

pip install Pillow

* Download Dataset:

https://www.kaggle.com/datasets/nguyngiabol/dress-pattern-dataset

* Prepare the Dataset:

python data\_preparation.py

* Clean the Dataset (Optional for Transparency Issues):

python rgb\_cleaner.py

* Train the Model:

python model\_training.py

* Test the Model (Optional):

python model\_testing.py

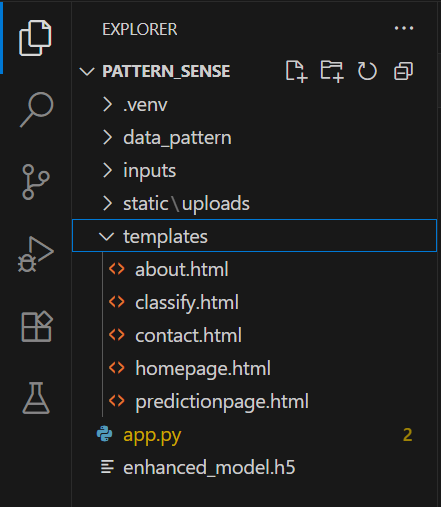
* Run the Flask Web Application:

python app.py

* Access the Application:

Open your browser and visit: [http://127.0.0.1:5050]

**5. Folder Structure**



**6. Running the Application**

1. **Backend (Model & API):**

**1.Deep Learning Model**

* Common architectures: ResNet‑50, VGG16/VGG19, DenseNet, Inception, Xception—typically pretrained on ImageNet and fine-tuned on specialized fabric datasets.
* Training pipeline includes:
  + Image preprocessing (resizing, normalization)
  + Data augmentation (rotations, flips, brightness, zoom)

**2. Model Serving API**

* Typically deployed with a lightweight web server (e.g., Flask/FastAPI).
  + Accepts image uploads (or base64 streams), preprocesses, runs inference through the model, returns JSON results (class label + confidence).
  + Seen in fabric defect detection literature using TensorFlow/Keras and Inception‑based systems

1. **Frontend (UI/UX & Deployment)**

**Streamlit Dashboard**

* Use [Streamlit](https://streamlit.io) to create an interactive UI where users can upload fabric images and view classification results instantly
* Error handling, result display, basic charts, and batch submission are built-in features.

**Full Stack Web (React/Next.js)**

* **Frontend**: React or Next.js web UI for image upload, progress indicators, and result display (class + probability + sample textures).
* **API Gateway**: A lightweight Node.js/Express layer can sit between frontend and Python model server. This proxy handles data formatting, CORS, and routing.

**7. API Documentation**

1. **Home Page**

* **URL:** /
* **Method:** GET
* **Description:** Displays the landing page of the application.
* **Response:** Returns the index.html template.

1. **About Page**

* **URL:** /about
* **Method:** GET
* **Description:** Displays information about the project.
* **Response:** Returns the about.html template.

1. **Inspect Page (Upload & Predict UI)**

* **URL:** /inspect
* **Method:** GET
* **Description:** Displays the image upload form for prediction.
* **Response:** Returns the inspect.html template.

1. **Image Prediction API**

* **URL:** /predict
* **Method:** POST
* **Description:** Accepts an image file, performs classification using the trained model, and returns the prediction.

**Request Parameters:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| image | File | The image file to be uploaded (JPG, PNG, etc.). |

**Example Request using HTML Form:**

<form method="POST" action="/predict" ectype="multipart/form-data">

<input type="file" name="image" required>

<button type="submit">Predict</button>

</form>

**Example Response (Rendered on Inspect Page):**

Upon successful prediction, the following details are displayed:

* Uploaded image preview
* Predicted class label (e.g., "tribal")
* Confidence score (percentage)

**Example Backend Response (if it were JSON API):**

(Note: Your current implementation renders a template, but if converted to pure JSON API, it would look like this)

{“predicted\_label": "cartoon",

"confidence": 97.35,

"Image\_path": "static/uploads/cartoon.jpg"

}

**8. Authentication**

**Current Status:**

Many textile manufacturers and retailers are exploring AI solutions to automate fabric inspection and pattern recognition. The application is designed as a publicly accessible image classification tool intended for demonstration purposes, allowing any user to:

✔ Access the website

✔ Upload images for fabric classification

✔ View results without requiring login or registration

**Future Scope for Authentication (Optional Enhancements):**

**Hybrid AI Models & Explainability**

* Physics-informed and hybrid models: Combining CNNs with physics-based models (e.g., fabric drape, fibre structure) can improve authentication robustness and interpretability
* Explainable AI (XAI): Especially for high-value fabrics (e.g., silk, Pashmina), systems that clearly show “why” a pattern is flagged as fake—based on thread density, weave irregularities—will foster trust.

**Enhanced Data & Real-Time Vision**

* High-resolution imaging improvements: Adoption of line-scan cameras, multi-spectral/hyperspectral imaging, and 3D capture can uncover authenticity features invisible to the naked eye.
* Real-time authentication during production: Inline visual inspection can identify anomalies on the fly—enhancing QC and reducing waste

**Generative AI & Pattern Design**

* GAN-based counterfeit detection: As counterfeiters use generative AI to create near-perfect fake patterns, authentication systems will need GAN-based “tampering detectors” to spot synthetic sequences
* Adaptive, co-created patterns: Counterparts could include AI-based modules that generate unique, traceable pattern IDs for each production batch, simplifying downstream verification.

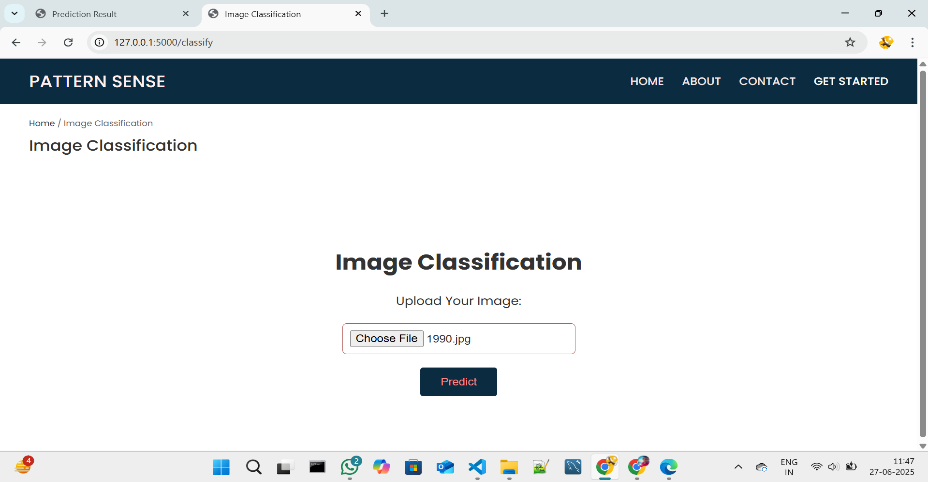
**Sustainability & Circular Fashion**

* Automated sorting for recycling: AI systems will classify fabric prints and materials for optimal recycling—for instance, segregating cotton vs polyester blends for better reuse streams
* Waste reduction & traceability: Authentication tied to lifecycle metadata (e.g., recycled content, eco-certifications) supports sustainable sourcing claims

**Recommended Future Features:**

* Admin Login: Only authorized personnel can retrain or upload new datasets.
* User Dashboard: Registered users can track prediction history.
* API Access Tokens: Protect REST APIs for mobile or external application integration.

**9. User Interface**



**10. Testing**

**Testing Strategy**

**Dataset Preparation & Splitting**

* High-quality, balanced dataset: Collect diverse, high-res fabric images across all pattern classes. Remove duplicates, fill missing labels, and balance classes via augmentation or sampling
* Split into train/validation/test:
  + Common split: 60–70% train, 15–20% validation, 15–20% test.
  + Use stratified sampling to preserve class distributions.

**Cross-Validation for Robustness**

* Stratified k‑fold (e.g. k = 5 or 10) during training/validation. Ensures each fold well represents pattern categories
* Evaluate model consistently across folds—use mean performance and standard deviation to detect variability

**Data Augmentation & Test-Time Augmentation**

* Train-time augmentation: Apply flips, rotations, scale, crop, brightness, shearing, translation, noise—key for texture generalization
* Test-time augmentation (TTA): Average predictions over multiple augmented crops/scales to improve stability

**Evaluation Metrics & Error Analysis**

* Use multiple metrics: accuracy, precision, recall, F1‑score, ROC–AUC to address class imbalance
* Employ confusion matrices to check misclassification patterns and identify confusing fabric classes.
* Conduct error-slice analysis: evaluate performance across image conditions like lighting, fabric type, camera resolution

**Preventing Overfitting**

* Use early stopping based on validation loss or accuracy
* Apply regularization: L2 weight decay, dropout layers.
* Monitor both train and validation performance to detect divergence or overfitting.

**Robustness & Bias Testing**

* Stress-test with perturbed inputs (e.g., noise, lighting variations) to gauge stability .
* Evaluate on different subgroups (e.g., handloom vs machine-made, varying capture devices) to identify biases
* Optional: adversarial attacks for edge-case analysis

**Final Testing & Generalization**

* After tuning, evaluate on the held-out test set one final time—this is the true measure of generalization
* Ensure test data is not used for hyperparameter tuning—it's only for final assessment

**Continuous Monitoring Post‑Deployment**

* Use a monitoring pipeline (e.g., MLflow, Tensor Board) to track model drift: monitor changes in accuracy, input data distribution, class representation.
* Implement proactive retraining triggers when performance drops below thresholds (e.g., F1 < 90%).

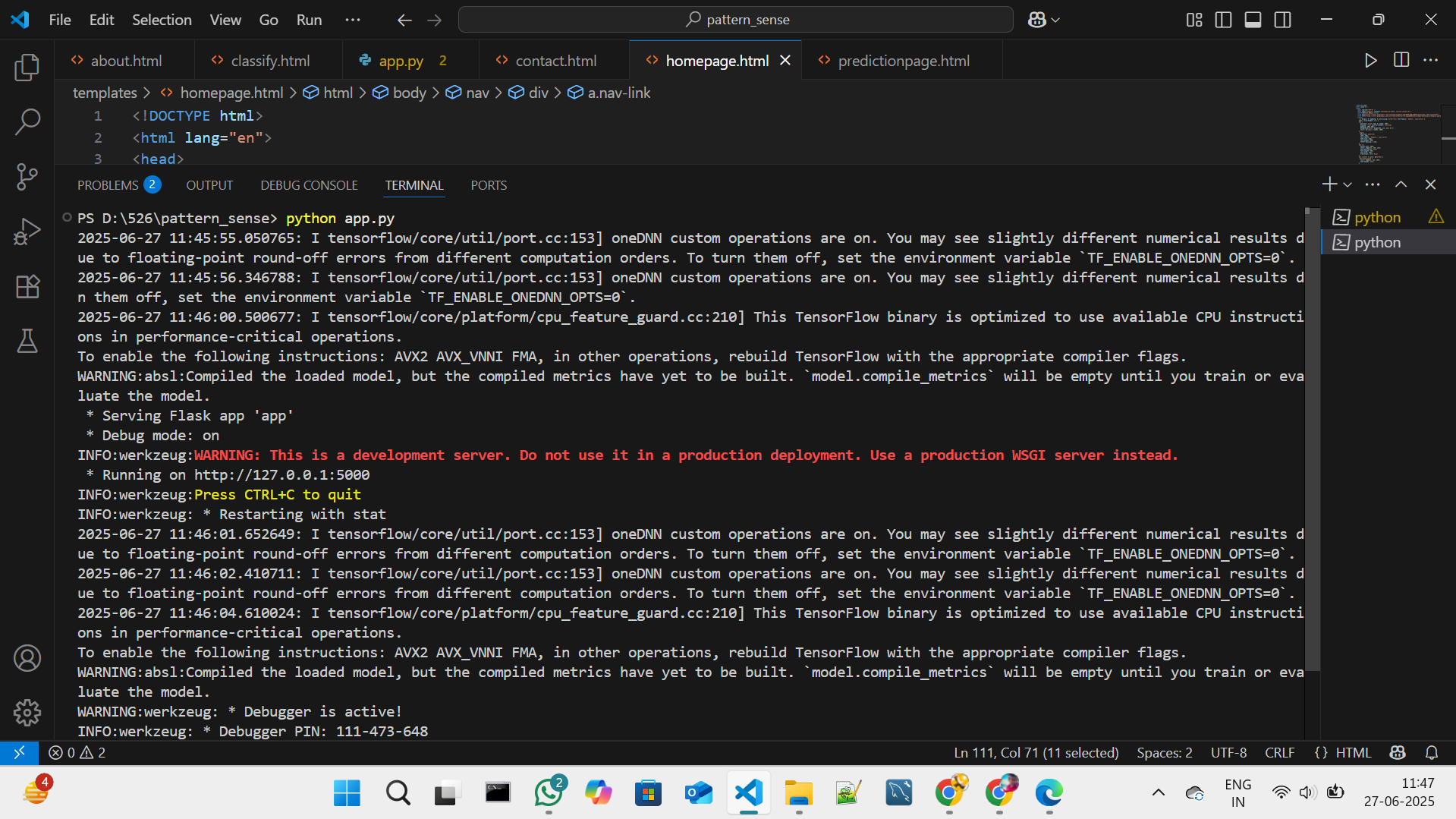
**Tools Used**

|  |  |
| --- | --- |
| **Tool/Library** | **Purpose** |
| **TensorFlow, Keras, PyTorch, Caffe** | Core deep learning frameworks. |
| **ResNet, DenseNet, VGG, Inception** | Pretrained models for feature extraction. |
| **Matplotlib/Seaborn** | Visualization of confusion matrix and performance metrics. |
| **GLCM, LBP via scikit-image** | Handcrafted texture descriptors |
| **Deep‑TEN, wavelet‑CNN, Texel‑Att** | Specialized texture representation. |

**11. Screenshots or Demo**

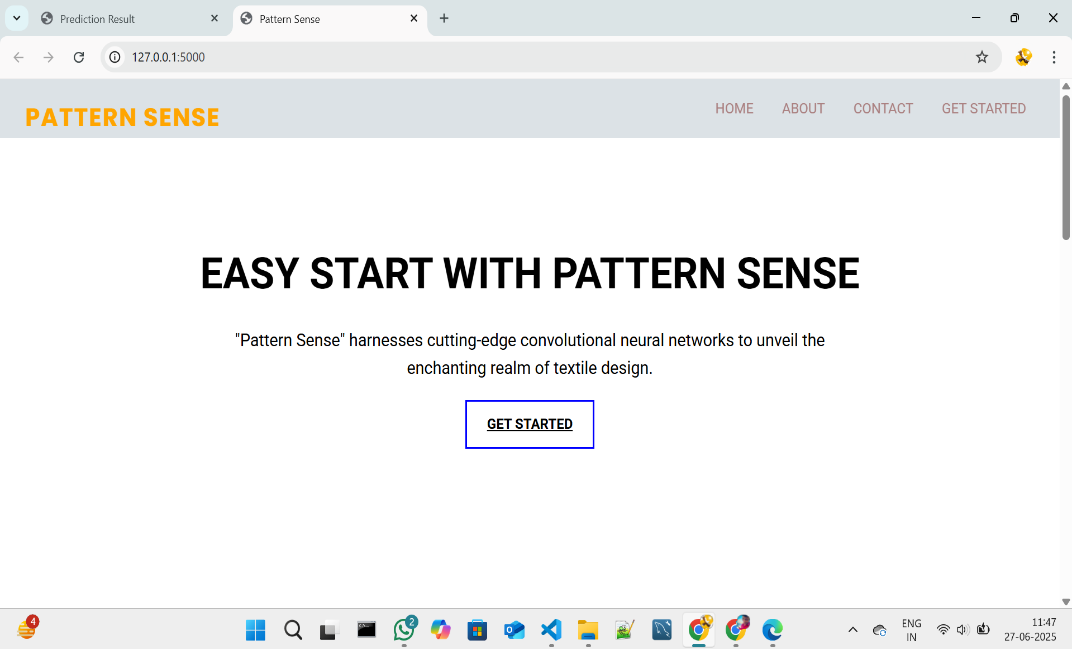
**Screenshots**

The complete execution of the Pattern sense application is shown in the images step by step as shown below.

**Step 1:** Run the app.py code and you will get a link in terminal as <https://127.0.0.1.5000> to access web page and to do the other process

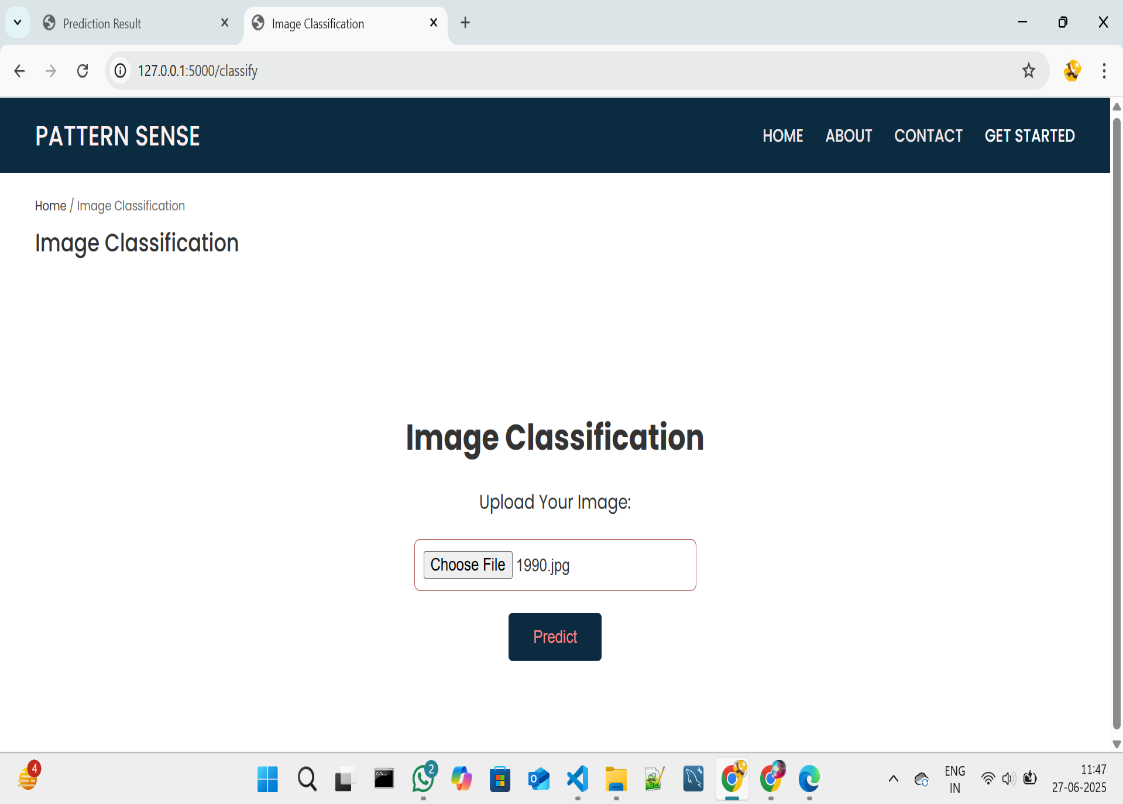
**Fig 7.1.1: Code running in Terminal**

**Step 2:** Click on that link a web page of fabric patterns will be open in the web browser.



**Fig 7.1.2: Fabric Pattern sense Home Page**

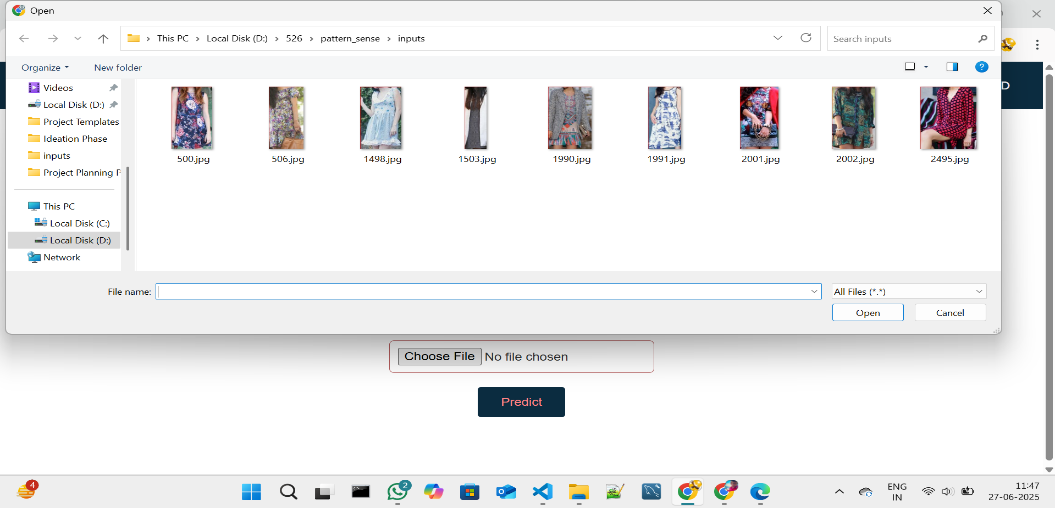
**Step 3:** Click on GET STARTED or PREDICT option to open the prediction page.



.

**Fig 7.1.3: Prediction page in pattern sense**

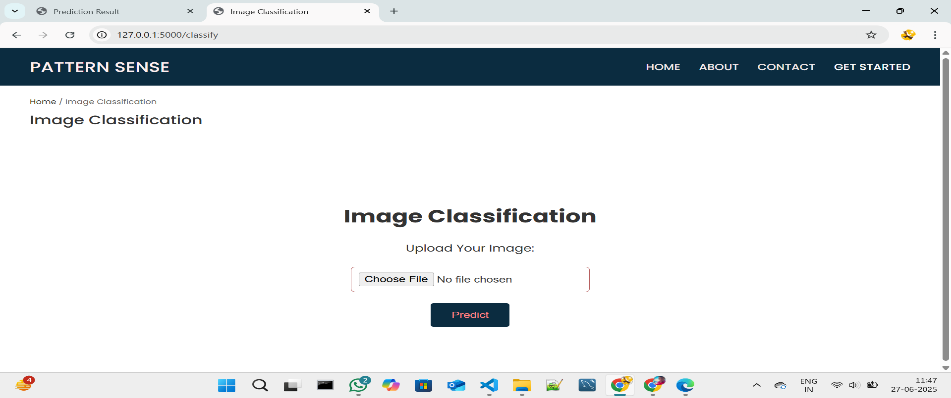
**Step 4:** Click on choose file option to choose the images that need to predict



**Fig 7.1.4: Window to choose image for prediction**

Select any image for prediction and click on Open.

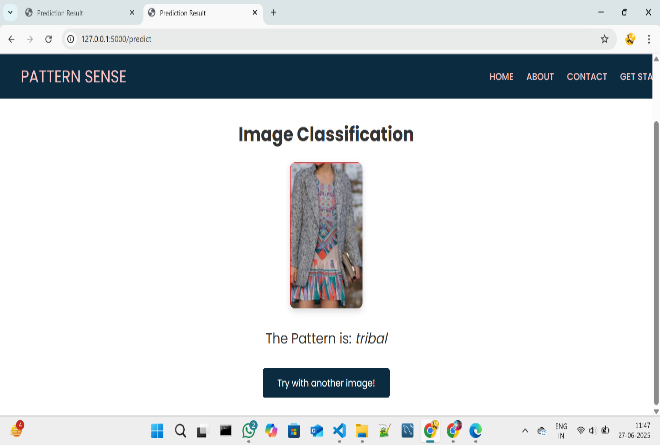
**Step 5:** Click on Predict to predict the quality of selected image.

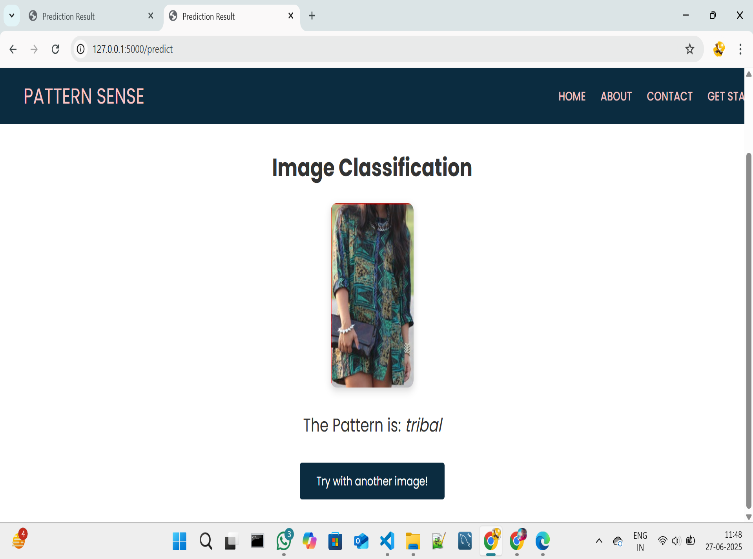
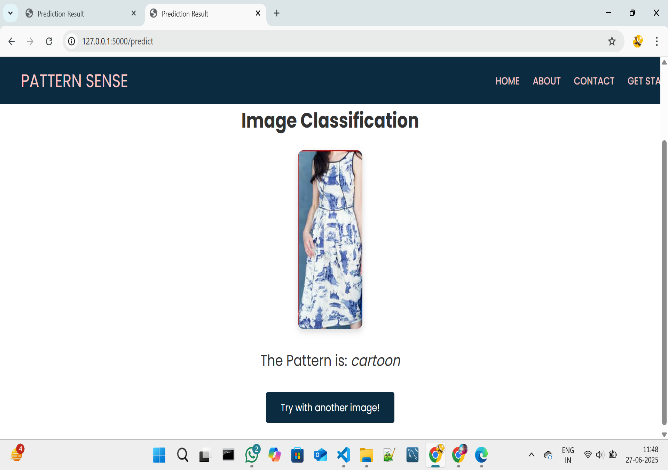
****

**Fig 7.1.5: Image selected in prediction page**

**Step 6:** After clicked on the predict button the model predicts the image quality and displays the quality of image.

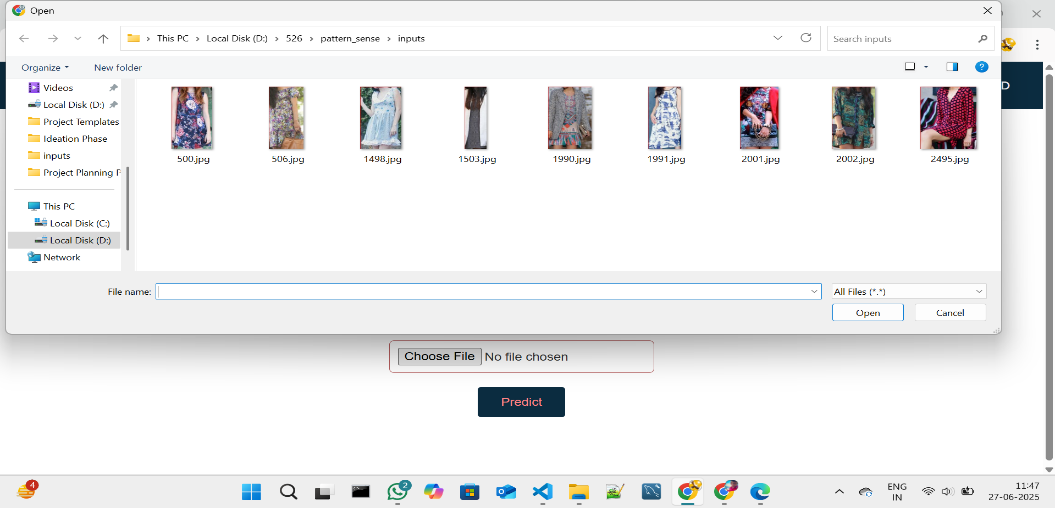
The below images are the some of samples tested for prediction.

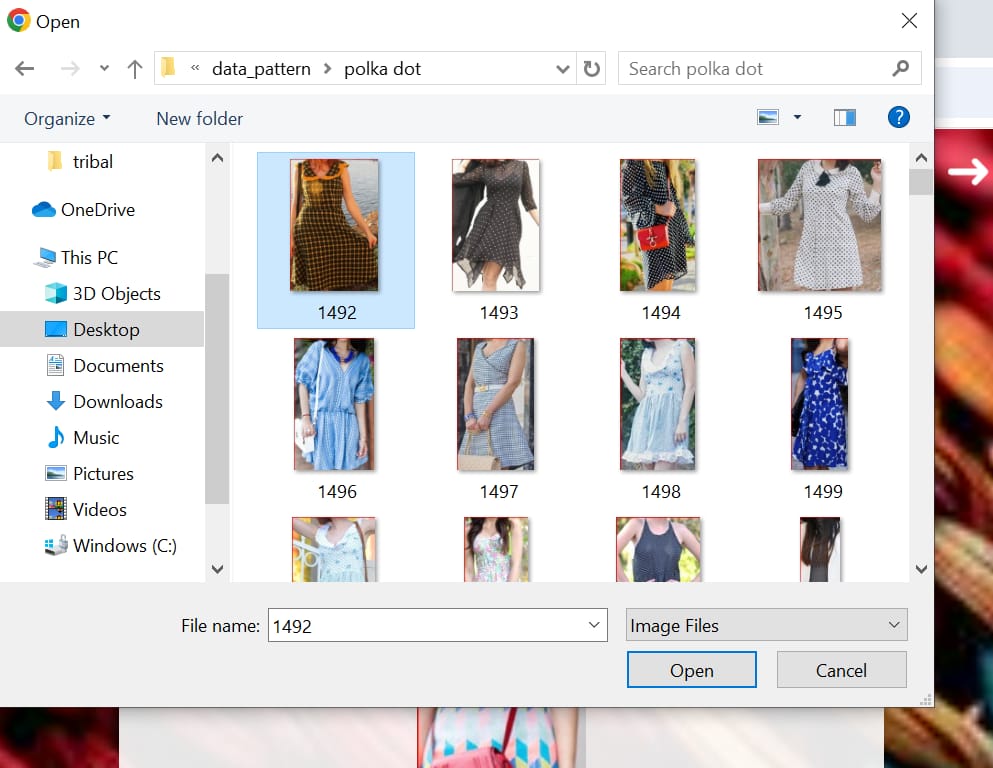


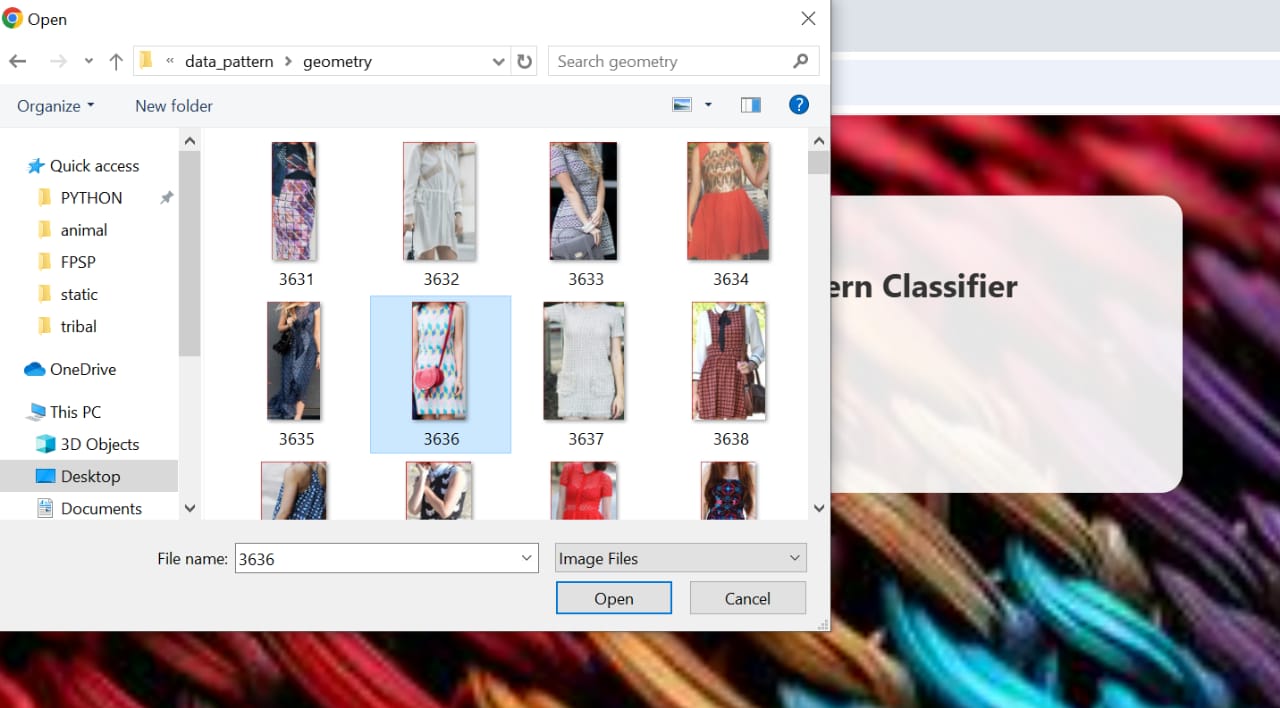
****

**Fig 7.1.6: Prediction output for the several inputs with accuracy**

After the Images Predicted the images will be stored in the uploads folder as fresh and rotten as shown in the figure given below.







**Fig 7.1.7: Folder Structure to store predicted images**

**Project Demo Link:**

[**https://drive.google.com/file/d/1UrcIGw5ZFwvoOvn5i39NIdt2OKl8JVVk/view?usp=sharing**](https://drive.google.com/file/d/1UrcIGw5ZFwvoOvn5i39NIdt2OKl8JVVk/view?usp=sharing)

**12. Known Issues**

**Limited & Biased Datasets**

* Small dataset sizes restrict coverage of pattern diversity. Fabric image datasets are often limited (e.g., 3K–10K images), hurting generalization and risking overfitting
* Sampling bias: Majority class images dominate, underrepresenting rare patterns, so models generalize poorly to unseen types

**CNN Bias Toward Texture Over Shape**

* Pretrained CNNs (e.g., ResNet‑50) tend to overly rely on texture, neglecting shape information. This bias can lead to misclassification under distortion or when fabrics vary substantially
* Mitigation: training with stylized-image augmentation or shape-texture debiasing methods can improve robustness

**Sensitivity to Rotation, Scale, & Lighting**

* Fabric textures change wildly with orientation, zoom, or lighting. Standard CNNs struggle without specific augmentation or encoding mechanisms.
* Wavelet CNNs or Deep‑TEN encoding layers help gain invariance to scale and viewpoint

**Insufficient Texture-Specific Feature Encoding**

* Typical fine-tuning can't fully capture micro-structures in patterns. Advanced modules (e.g., Deep‑TEN, bilinear pooling) improve representation but add complexity and training data requirements

**Computational Bottlenecks**

* High-capacity CNNs (e.g., DenseNet, ResNet) with encoding layers are expensive in memory/compute—problematic for edge devices
* Solutions include compact models, pruning, or knowledge distillation—but may reduce accuracy.

**13. Future Enhancements**

**Topological Deep Learning for Structural Awareness**

* Incorporate topological layers (e.g. persistence homology) to explicitly learn fabric’s multi-scale structure and weave topology—offering robustness to distortions and enhancing texture understanding beyond pixel-level features

**Multi-Modal & Depth-Enhanced Inputs**

* Add RGB-D or multi-view inputs (e.g., depth maps, multi-angle captures) to capture 3D surface features like fabric drape, thickness, and texture shadows—ideal for distinguishing similar weaves

**Advanced Texture Encoding Modules**

* Integrate state-of-the-art modules such as Deep-TEN, wavelet-based CNNs, or mixture-enhancement + attribute clustering to learn richer, more invariant texture representations

**Multi-Task Learning: Defect Detection + Classification**

* Implement unified pipelines combining classification + segmentation/detection heads (e.g., MobileNetV2-SSD-FPN, YOLOv5, U-Net) to detect defects alongside pattern types in industrial contexts

**Lightweight & Efficient Models**

* Apply model compression, pruning, quantization, or distillation to tailor models for edge devices—enabling real-time deployment in resource-constrained manufacturing workflows

**Unsupervised Anomaly Detection**

* Incorporate unsupervised or self-supervised techniques (e.g., motif-based CNNs trained on defect-free fabric) to detect rare or unseen defects with minimal labelling effort

**Domain Adaptation & Robustness Strategies**

* Deploy advanced augmentations (adversarial, style, lighting, geometric), as well as self-training / domain adaptation approaches, to ensure stability across new fabrics, lighting conditions, and production lines

**Explainability & Model Interpretability**

* Use Grad‑CAM, topological insights, or feature-importance mappings to highlight the fabric structures driving decisions—crucial for user trust and model validation in industrial settings.

**Automated Robotic Feedback Integration**

* Connect with robotic knitting/fabrication systems (e.g., reverse-engineering pipelines or CAM integrations) to adapt manufacturing based on detected pattern/defect insights.