Pattern Sense: Classifying Fabric Patterns Using Deep Learning

Team Members:

- Jalluri Chaitanya Naga Veera Siva Kumar
- Gummadi Nikhilesh
- Gudavalli Rohith Balaji
- Guraja Deepika

Phase-1: Brainstorming & Ideation

1. Problem Statement:

In the textile and fashion industries, identifying fabric patterns manually is a labor-intensive and subjective task. It often relies on expert judgment, leading to inconsistent classification and inefficiencies in production, inventory, and design processes. This limits the scalability of automated fashion solutions and creates challenges in textile digitization and e-commerce search systems.

2. Proposed Solution:

We propose **Pattern Sense**, a deep learning-powered solution to automate the classification of fabric patterns. Using a pre-trained MobileNetV2 model fine-tuned on a curated dataset of fabric images categorized into 10 classes (e.g., floral, stripes, geometric, polka dot), the system can predict the pattern category of an uploaded image with high accuracy. The backend is implemented in Flask, and the model is deployed via a user-friendly web interface.

3. Target Users:

- Fashion designers and manufacturers
- Textile inventory and quality control managers
- E-commerce platforms and recommender systems
- Researchers in computer vision and fashion technology

4. Expected Outcomes:

- An intelligent, web-based tool for classifying fabric patterns.
- Enhanced search and tagging in e-commerce systems.
- A deployable model for future mobile or embedded systems.

Phase-2: Requirement Analysis

Technical Stack:

- **Language:** Python
- **Frameworks/Libraries:** TensorFlow 2.15.0, Flask, NumPy, Pillow
- **Frontend:** HTML5, CSS
- **Model:** Transfer learning using **MobileNetV2**
- **Hosting Options:** Flask server locally or cloud-deployed (Colab/Heroku)

Functional Requirements:

- Upload image via web interface.
- Preprocess image and classify using the deep learning model.
- Display the predicted pattern type with image preview.
- Accept `.jpg`, `.jpeg`, `.png`, `.bmp` formats.

Model Specifics:

- Trained on a dataset of 10 fabric pattern categories.
- Saved model (`best_model.h5`) loaded during inference.
- Predictions handled server-side using Flask routes.

Phase-3: Project Design

System Architecture:

1. **Frontend (HTML/CSS)**:
    - `index.html`: Upload form
    - `page3.html`: Result display with predicted class and image
2. **Backend (Flask)**:
    - Handles image upload, saving, preprocessing, model inference, and response rendering.
3. **Model**:
    - MobileNetV2 base
    - Image input size: 224x224
    - Dense layers added for 10-class output.

User Flow:

1. User uploads an image.
2. Flask receives and stores the image.
3. Image is preprocessed (resized, normalized).

4. Model predicts the pattern class.
5. Result displayed to user with confidence.

Phase-4: Project Planning (Agile Methodology)

Sprint 1 – Dataset Preparation

- Organize dataset into class folders.
- Implement preprocessing and augmentation (if any).

Sprint 2 – Model Development

- Implement and train MobileNetV2.
- Apply early stopping, checkpointing.

Sprint 3 – Web Integration

- Build Flask backend and test API.
- Design frontend (`index.html`, `page3.html`).

Sprint 4 – Testing & Bug Fixing

- Validate predictions.
- Handle invalid uploads, fix UI bugs.

Phase-5: Project Development

Steps Taken:

1. Load and preprocess image using `tensorflow.keras.preprocessing`.
2. Build model using MobileNetV2 with top layers removed and custom layers added.
3. Train using categorical cross-entropy and Adam optimizer.
4. Save trained model as `best_model.h5`.
5. Deploy via Flask (`app.py`) with HTML interface.

Model Classes:

- `'animal'`, `'cartoon'`, `'floral'`, `'geometry'`, `'ikat'`, `'plain'`, `'polka dot'`, `'squares'`, `'stripes'`, `'tribal'`

Challenges:

- Small dataset may cause overfitting.
- Balancing class distribution.
- Ensuring smooth frontend-backend integration.

Phase-6: Functional & Performance Testing

Test Cases:

- Upload valid/invalid image formats.
- Verify correct class prediction.
- Model response time.
- UI responsiveness and error handling.

Validation Results:

- Accuracy: (based on your training logs – please insert actual number)
- Model loaded successfully without retraining.
- Correct predictions returned for test inputs.