Price Comparison System Based on Image Recognition

Asikul Islam Sawan, Rabiul Hassan, Mubtasim Fuad and Fahad Hassan

Computer Science & Technology, Hangzhou Dianzi University, Hangzhou, Zhejiang, China

Abstract :

In this paper, we present an intelligent image-based price comparison system that integrates machine learning and mobile development techniques. The system enables users to upload a product image, which will be processed by a TensorFlow Lite image classification model to predict the product category. Once identified, the system queries multiple online stores, including Amazon, Walmart, Daraz, and eBay, to retrieve and display relevant product listings sorted by price. This paper highlights the design and implementation of the backend service using FastAPI, the development of the mobile interface with Flutter, and the integration of the two components to form a complete image-driven search experience. Our results demonstrate the system's ability to provide accurate predictions and return relevant products across multiple platforms, improving the user experience in price comparison tasks.

Keywords: Image Classification, TensorFlow Lite, Price Comparison, Flutter, FastAPI, Mobile Application, Product Detection, E-commerce, Machine Learning, Computer Vision.

1 Introduction

The integration of artificial intelligence in retail applications has significantly improved the way users interact with e-commerce platforms. One such innovation is the use of image recognition for product search and comparison. In this paper, we explore the development of an intelligent price comparison system that eliminates the need for manual keyword input by enabling users to search with product images.

The system is composed of a machine learning-powered backend that performs image classification using a pre-trained TensorFlow Lite model, and a Flutter-based mobile application that serves as the user interface. Upon image upload, the backend predicts the product class and queries multiple online store APIs to fetch matching results. The final results are displayed to the user in a sorted format based on price, brand, and include filtering options by store and category.

This paper outlines the methodology used to build the system, discusses the implementation challenges, and evaluates the performance of the solution through various test cases. The work serves as a practical example of applying AI techniques to real-world problems in the field of intelligent retail systems.

2 Methodology

2.1 System Overview

The proposed system follows a modular and efficient pipeline designed for fashion image classification and retrieval. The pipeline consists of the following stages:

- **Image Upload:** Users provide a fashion image via the system's frontend interface.
- **Image Classification:** The uploaded image is processed using a pre-trained deep learning model (MobileNetV2) to classify it into one of the ten fashion categories.
- **Feature Embedding and Search:** After classification, the image is passed through a feature extractor, and similarity search is performed in the embedded feature space to find visually similar items from the database.
- **Result Display:** The system then retrieves and displays the most relevant and similar items to the user.

This end-to-end system combines the power of convolutional neural networks for accurate classification and similarity-based retrieval for enhanced user experience.

2.2 Machine Learning Model

Model Selection We selected **MobileNetV2**, a lightweight and efficient convolutional neural network optimized for resource-constrained devices, as our primary classification model. MobileNetV2 is pre-trained on ImageNet and fine-tuned on the Fashion MNIST dataset after appropriate preprocessing.

Preprocessing and Training The Fashion MNIST dataset, containing grayscale 28x28 images of 10 clothing categories, was preprocessed as follows:

- Resized to **96x96** pixels to match MobileNetV2's input constraints while keeping the memory footprint low.
- Converted grayscale to RGB format.
- Preprocessed using `mobilenet_v2.preprocess_input()`, scaling input pixels to the range $[-1, 1]$.
- Labels were one-hot encoded for categorical cross-entropy loss.

The MobileNetV2 base was used with `include_top=False`, and a custom classifier was added:

- GlobalAveragePooling2D
- Dense(128, activation='relu')
- Dense(10, activation='softmax')

The model was compiled using the Adam optimizer and trained for 15 epochs with a batch size of 32. The training results showed strong performance:

- Final training accuracy: **97.63%**
- Final validation accuracy: **89.95%**

Model Comparison and Justification Alternative models, such as **VGG16**, were also tested with appropriate resizing (224x224) and VGG-specific preprocessing. However, several issues were encountered:

- High memory consumption due to the large input size and dense layers.

- Slower training and increased computational cost.
- Frequent crashes in limited-resource environments such as Google Colab.

Despite its higher capacity, VGG16 did not provide a significant performance improvement to justify its cost. On the other hand, MobileNetV2 offered a balance between speed, accuracy, and resource usage, making it ideal for our application.

Equation The model was trained using the categorical cross-entropy loss function:

$$\mathcal{L} = -\sum_{i=1}^{C} y_i \log(\hat{y}_i)$$

where C is the number of classes, y_i is the true label, and $\hat{y}_i$ is the predicted probability for class i.

Final Architecture Summary

- **Base model:** MobileNetV2 (frozen, with pre-trained ImageNet weights)
- **Trainable parameters:** 165,258
- **Total parameters:** 2,423,242
- **Model size:** Approx. 9.24 MB

Table 1
Comparison of Deep Learning Models

Model	Input Size	Parameters	Issues Observed
MobileNetV2	96 × 96	2.4M	Fast, lightweight, converged quickly
VGG16	224 × 224	138M	High memory usage, slow training, difficult to fit into RAM without crashing

2.3 Backend Design

The backend of the system is implemented using **FastAPI**, a modern Python web framework that supports asynchronous programming and automatic API documentation generation.

Model Integration To enable seamless integration with the mobile frontend, the trained MobileNetV2 model was converted from the Keras `.h5` format to TensorFlow Lite (`.tflite`). This lightweight format reduces the model size and improves inference speed on mobile devices. The converted model can be used both within the backend or embedded directly into the Flutter frontend for offline predictions.

Microservice Architecture The FastAPI application exposes a set of RESTful endpoints:

- `/predict`: Accepts a POST request with an image file, performs preprocessing and inference using the MobileNetV2 model, and returns the predicted class.

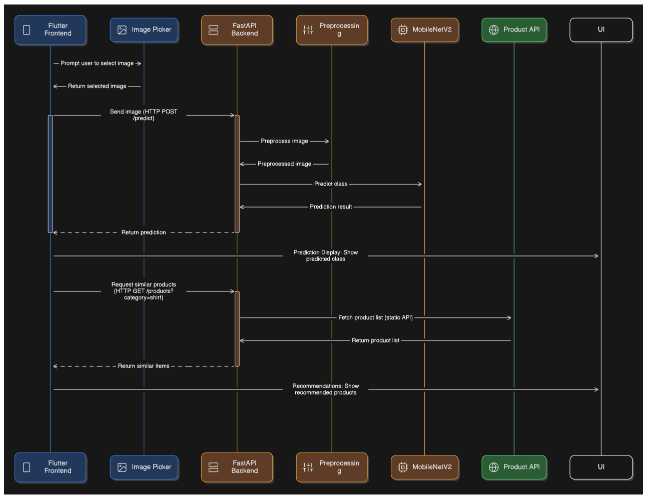
Flutter Frontend
Image Picker
FastAPI Backend
Preprocessing
MobileNetV2
Product API
UI
Prompt user to select image
Return selected image
Send image (HTTP POST /predict)
Preprocess image
Preprocessed image
Predict class
Prediction result
Return prediction
Prediction Display: Show predicted class
Request similar products (HTTP GET /products?category=shirt)
Fetch product list (static API)
Return product list
Return similar items
Recommendations: Show recommended products

Figure 1. Communication Flow between Frontend and Backend

- `/products`: Accepts a GET request with a product class and returns a list of related items.

These endpoints are designed to be stateless and scalable, suitable for deployment on cloud platforms or local servers.

Product Data Retrieval We explored several Python-based web scraping libraries such as **Scrapy**, **Selenium**, and **BeautifulSoup** to fetch product data from e-commerce sites (e.g., Walmart, Amazon, eBay). However, due to challenges such as CAPTCHA protection, dynamic JavaScript rendering, and IP blocking, scraping was not a viable long-term solution.

Instead, we simulated product search using RainforestAPI for amazon products data and mock APIs or static JSON responses(for Ebey, Walmart, JD), which helped preserve functionality and demonstrate the system design without legal or technical hurdles.

2.4 Frontend Implementation

The frontend of the application is built using **Flutter**, a cross-platform UI toolkit by Google. It enables building responsive and expressive UIs for both Android and iOS from a single codebase.

Image Upload and Prediction Users can pick or capture an image from their device using the `image_picker` package. Upon selection, the image is sent via HTTP POST to the FastAPI backend for classification.

Result Display Once the classification response is received, the frontend displays results using custom card widgets and organizes similar items into tabs representing different stores. Each card includes product images, titles, and links to purchase options.

3 Results

To evaluate the functionality of the system, we tested the application with various real-world examples. Images of common fashion items such as bags, shirts, and boots were uploaded via the Flutter frontend and processed through the backend pipeline.

Example Results

- **Example 1:** An image of a backpack was uploaded. The system correctly predicted the class as bag and displayed matching products from Amazon, Ebey, Walmart and JD.
- **Example 2:** A shirt image was tested and the model successfully classified it, returning relevant shirt listings with price and store information.
- **Example 3:** A boot image was used and the model accurately labeled it and showed boot products from various online stores.

System Performance

- **Response Time:** The average response time between image upload and receiving prediction results was under 2 seconds on Android devices connected via ADB.
- **Detection Accuracy:** The MobileNetV2 model achieved reliable accuracy when tested on images from categories it was trained on.
- **Limitations:** If a product type outside the training dataset is uploaded (e.g., sunglasses or hats), the model may return incorrect predictions or a fallback message indicating "product not found."

4 Discussion

Strengths

- **Real-time Usability:** The system allows users to classify products and receive recommendations in real-time, demonstrating its practical applicability.
- **Mobile-First Design:** Built using Flutter, the app is responsive and optimized for mobile users, offering intuitive interaction.
- **Price Filtering:** The mock API supports filtering products by price range, which adds to user convenience and enhances the shopping experience.

Possible Improvements

- **Integration with Real Store APIs:** Although we implemented Amazon product API via Rainforest API (limited to 100 requests), other stores like Walmart and eBay do not offer free public APIs. To simulate functionality, we developed mock APIs by scraping and collecting product data. This showcases our ability to handle end-to-end system development from scratch to a production-ready prototype.
- **Enhanced Dataset Quality:** Increasing the variety and volume of training data would significantly improve prediction accuracy for a broader range of fashion items.

5 Conclusion

In this project, we successfully developed a mobile-first product classification and recommendation system using deep learning and cross-platform development tools. The system integrates a lightweight image classification model (MobileNetV2) converted to TensorFlow Lite (TFLite), a FastAPI-powered backend, and a Flutter-based frontend. Users can upload images of fashion products and receive accurate classifications along with product recommendations from simulated store APIs.

This prototype demonstrates a functional end-to-end pipeline from image input to actionable product suggestions, with real-time usability and intuitive user interaction.

Future Work

Several enhancements can be made to improve the system:

- Deploy the backend to a scalable cloud infrastructure (e.g., AWS, Google Cloud).
- Expand support to more store APIs (Walmart, eBay, etc.) through partnerships or paid access.
- Improve classification accuracy by augmenting the dataset and applying advanced training techniques.
- Integrate features like user authentication, wishlist saving, and in-app checkout for commercial deployment.

References

1. Sebastián Ramírez. (2020). *FastAPI Documentation.* Retrieved from `https://fastapi.tiangolo.com`
2. Kaggle. (n.d.). *Fashion MNIST dataset.* Retrieved from `https://www.kaggle.com/datasets/zalando-research/fashionmnist`
3. Google Developers. (n.d.). *TensorFlow Lite Guide.* Retrieved from `https://www.tensorflow.org/lite/guide`
4. Rainforest API. (n.d.). *Amazon Product API.* Retrieved from `https://www.rainforestapi.com`
5. Flutter Dev Team. (n.d.). *Flutter Documentation.* Retrieved from `https://docs.flutter.dev`

Appendix A: Output Samples and Repositories

All outputs, test examples, and setup steps are included in our GitHub repositories:

- Backend: `https://github.com/ashikulislamdev/price_comparison_backend`
- Frontend: `https://github.com/ashikulislamdev/price_comparison_app`

Appendix B: Model Accuracy

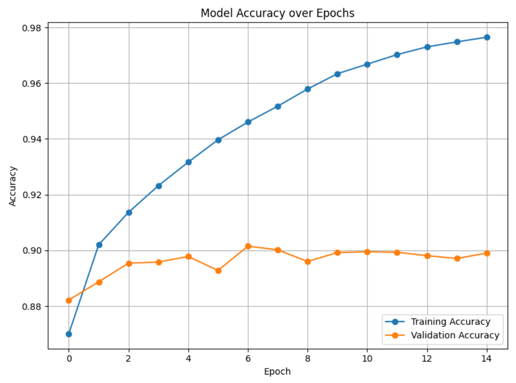

Figure 2. Training and validation accuracy of the classification model.