**Visual Product Matcher - Project Report**

**Author:** Sanket Kumar Das **Date:** October 12, 2025 **Project:** Technical Assessment - Software Engineer Position

**1. Introduction**

This document provides a detailed overview of the "Visual Product Matcher" project, a full-stack web application built as a technical assessment. The primary goal of the project is to create a platform where users can upload an image of a product and receive a list of visually similar items from a predefined catalog.

This report outlines the project's architecture, the technical decisions made, a deep dive into the visual search algorithm, and instructions for setup and deployment. It serves as comprehensive documentation of the solution, demonstrating a professional approach to software development, from initial design to final deployment.

**2. Project Requirements**

The application was built to satisfy the following key requirements as outlined in the project brief:

* **Frontend:** A responsive, mobile-friendly web interface.
* **Image Input:** Support for user-uploaded image files.
* **Search Interface:** The ability to display the user's query image alongside a grid of similar product results.
* **Filtering:** A mechanism for users to filter the results based on their similarity score.
* **Backend:** A server capable of handling image processing and similarity matching.
* **Database:** A catalog of at least 1000 products, each with a name, category, and image.
* **Deployment:** The final application must be hosted on a publicly accessible URL.
* **Documentation:** Clean, production-quality code accompanied by a README.md and a brief write-up.

**3. Architecture and Technology Stack**

A decoupled, full-stack architecture was chosen for this project, separating the frontend client from the backend server. This is a modern, scalable approach that allows each part of the application to be developed and deployed independently.

**3.1. Frontend**

The frontend is a single-page application (SPA) responsible for all user interactions.

* **Framework:** **React (with Vite)** was chosen for its component-based architecture, which promotes reusable code and a structured development process. Vite provides an extremely fast development server and an optimized build process.
* **Styling:** **Plain CSS3** was used to create a clean, modern, and fully responsive user interface. This approach avoids reliance on external frameworks and demonstrates strong foundational styling skills.
* **API Communication:** **Axios** was used as the HTTP client to handle communication with the backend API, providing a simple and robust way to manage asynchronous requests.

**3.2. Backend**

The backend is a Python-based API server designed to handle the heavy computational work of the visual search.

* **Framework:** **FastAPI** was selected for its high performance, asynchronous capabilities, and automatic generation of API documentation. It is a modern framework ideal for building efficient, production-ready APIs.
* **Server:** **Uvicorn** serves as the ASGI server, which is the standard for running asynchronous Python web applications like FastAPI.

**3.3. AI / Machine Learning (The Visual Search Core)**

The heart of the project is its visual search algorithm, which leverages deep learning to understand and compare images.

* **Core Library:** **PyTorch** was chosen as the deep learning framework.
* **Model:** A pre-trained **ResNet50** model was used for feature extraction. This model has been trained on the massive ImageNet dataset and can recognize a vast array of complex visual features (shapes, textures, patterns) in images.
* **Similarity Calculation:** **Scikit-learn** was used for its highly optimized cosine\_similarity function, which provides a fast and accurate way to measure the similarity between two feature vectors.
* **Image Processing:** **Pillow (PIL)** and **NumPy** were used for handling image data, including opening, resizing, and converting images into the numerical format required by the AI model.

**4. The Visual Search Algorithm Explained**

The application's visual search is not based on simple metadata or color matching. It uses a sophisticated technique known as **Content-Based Image Retrieval (CBIR)**, implemented via deep learning feature extraction.

**4.1. What is a Feature Vector?**

A feature vector is a numerical "fingerprint" of an image. The ResNet50 model takes an image as input and, instead of classifying it, outputs a list of 1000 numbers (a vector). This vector represents the high-level concepts the model has detected in the image, such as the presence of certain shapes, textures (like denim or wool), and object parts. Two visually similar images will have very similar feature vectors.

**4.2. The Two-Phase Process**

To ensure the application is fast, the search is split into two phases:

**Phase 1: Offline Preprocessing (The preprocess.py script)**

This script is run once before the server is deployed. Its job is to prepare the data for fast searching.

1. **Load Products:** It reads the products.json file.
2. **Download Images:** It downloads all product images to a local folder.
3. **Generate Vectors:** For each of the 1000+ product images, it uses the ResNet50 model to generate its unique feature vector.
4. **Save Data:** It saves all these feature vectors, along with their corresponding product IDs, into a single, optimized file: features.pkl.

This process is computationally intensive and takes several minutes, but because it's done offline, it doesn't affect the user's experience.

**Phase 2: Real-Time Search (The FastAPI Server)**

This is what happens when a user uploads an image to the live application.

1. **Load Model:** The server loads the trained ResNet50 model and the features.pkl file into memory when it starts.
2. **Receive Image:** The /api/search endpoint receives the user's uploaded image.
3. **Generate Query Vector:** It instantly generates a feature vector for the user's image.
4. **Calculate Similarity:** It uses the cosine\_similarity function to mathematically compare the user's vector against all 1000+ vectors from the database. This calculation is extremely fast.
5. **Rank and Return:** The products are ranked by their similarity score (from highest to lowest), and the top 20 results are sent back to the React frontend as a JSON response.

This architecture ensures that the user gets a near-instant response, even though a complex AI-powered search is happening in the background.

**5. Deployment Strategy**

A professional, decoupled deployment strategy was used, hosting each part of the application on a service specialized for its technology.

* **Frontend (Vercel):** The React application is deployed on Vercel, a platform optimized for hosting static and serverless frontends. It connects directly to the project's GitHub repository for continuous deployment.
* **Backend (Render):** The Python FastAPI server is deployed on Render, a cloud platform well-suited for hosting web services and running build scripts. Render's build command was configured to run the entire preprocessing pipeline (pip install, download\_images.py, preprocess.py) automatically on every deployment, ensuring the server always has the necessary data.

**6. Conclusion**

The Visual Product Matcher project successfully meets all the requirements of the technical assessment. It demonstrates the ability to design, build, and deploy a modern, full-stack web application. The implementation of a deep learning-based visual search algorithm showcases a strong understanding of AI/ML principles and their practical application in a real-world product. The final result is a functional, responsive, and performant application that provides a powerful and intuitive user experience.