1.INTRODUCTION

1.1 About the project

A fashion product recommendation system is intended to improve the shopping experience by offering customers tailored fashion recommendations based on their choices, actions, and other information. Typically, it consists of a user profile that records preferences, a product database with comprehensive details about fashion items, and a recommendation engine that matches people with products using content-based, collaborative, or hybrid filtering strategies.

The system strives to strike a balance between relevance and diversity in its recommendations, despite potential obstacles like data sparsity and the cold start issue. By utilising cutting-edge user interfaces, cloud computing, machine learning, and other technologies, the system is able to adapt and change over time. In the future, it may incorporate social media data, fashion trends, and AI-powered tools to provide a more engaging and dynamic shopping experience.

Typically, this system is made up of multiple core components: a user profile that records implicit data (like browsing history and time spent on particular items) as well as explicit data (like user-specified preferences and past purchases); a comprehensive product database that lists all the fashion items that are available; and a recommendation engine that uses advanced algorithms. To provide accurate suggestions, strategies like content-based filtering, which matches products to a user's known preferences, and collaborative filtering, which makes product recommendations based on the behaviour of similar users, are frequently employed, sometimes in combination.

The method aims to balance providing relevant ideas with avoiding data sparsity and cold start issues. It uses cloud services, machine learning frameworks, and contemporary web building tools to scale to meet expanding user bases and product inventories, ensuring a wide range of ideas is provided without compromising on data quality.

1.2 Objective

The objective of this project is to develop an AI-powered fashion recommendation system using deep learning and computer vision techniques to enhance the online shopping experience. The system aims to provide personalized, visually similar product suggestions based on images uploaded by users, replicating the natural human behavior of searching for appealing alternatives.

To achieve this, the project employs a pre-trained ResNet50 model, a deep convolutional neural network, to extract high-dimensional feature vectors from images. These feature vectors represent each fashion item in a multi-dimensional space, capturing intricate visual patterns and details. The extracted features are then used to calculate the Euclidean distance between the user-uploaded image and the existing inventory, enabling the nearest neighbors algorithm to identify and recommend the most visually similar items.

In addition to visual similarity, the recommendation system integrates metadata from a cleaned dataset, including price, rating, usage, and subcategory. This contextual information helps tailor recommendations to individual preferences, ensuring that users receive suggestions aligned with their specific needs, such as budget or intended use.

The system is implemented using Streamlit, a Python-based framework, to create a user-friendly interface that allows for easy image uploads and real-time recommendations. Users are presented with detailed information about each recommended item, including descriptions, prices, ratings, and usage categories, offering a comprehensive and engaging shopping experience.

This project aims to showcase the potential of deep learning and computer vision in e-commerce by providing an intelligent, scalable, and responsive recommendation system. By integrating state-of-the-art technology with practical application considerations, the system seeks to enhance user engagement and satisfaction, setting a new standard for online fashion retail.

2.SYSTEM ANALYSIS

2.1 Existing System

In the current landscape of online fashion retail, recommendation systems typically rely on collaborative filtering, content-based filtering, or hybrid models. These systems suggest products based on user preferences, past purchase history, and browsing behavior, or by analyzing product descriptions, categories, and user reviews.

Collaborative Filtering This approach recommends products based on similarities between users or items. If two users have a similar purchase history, the system might recommend products favored by one user to the other. Collaborative filtering relies heavily on user data, making it effective for users with extensive purchase histories.

Content-Based Filtering This method recommends products based on the attributes of items the user has previously interacted with. The system analyzes product descriptions, tags, and categories to find similar items. Content-based filtering works well for users with specific preferences.

Limitations of Existing System

Cold Start Problem Existing systems often struggle to provide accurate recommendations for new users or products due to insufficient data. This limits their ability to cater effectively to first-time visitors or newly added items.

Limited to Textual Data Many recommendation systems rely heavily on textual data like product descriptions and tags, ignoring visual aspects. This limitation prevents the system from recognizing and recommending items based on visual appeal.

Over-Specialization and Lack of Diversity Content-based filtering can lead to recommendations that are too similar to previous choices, reducing variety. This narrow focus prevents users from discovering new or diverse fashion items. It can make the shopping experience monotonous and less engaging.

2.2 Proposed System

The proposed system is an advanced AI-driven fashion recommendation engine designed to provide highly personalized product suggestions by combining deep learning and computer vision techniques. At the core of this system is the ResNet50 model, a powerful convolutional neural network (CNN) pre-trained on the ImageNet dataset. ResNet50 is known for its ability to learn intricate patterns and details in images through its deep residual learning framework, which includes skip connections that help the network retain information across layers. This model is particularly effective at extracting high-dimensional feature vectors from fashion product images, capturing fine-grained visual details such as texture, color, pattern, and shape. These features are then normalized to ensure that the system can efficiently compute visual similarities between different items, allowing it to identify products that are visually similar to those selected or uploaded by the user.

Once the features are extracted using ResNet50, the system employs the K-Nearest Neighbors (KNN) algorithm to find and recommend items that are closest in visual appearance to the input image. KNN is a simple yet powerful algorithm that operates on the principle of similarity; it identifies the 'k' closest points in the feature space to the input image based on a defined distance metric, such as Euclidean distance. In this context, KNN searches through a precomputed database of image embeddings to identify products that share similar visual characteristics with the uploaded image, providing users with relevant recommendations. This approach leverages the high-dimensional features extracted by ResNet50 to create a robust recommendation mechanism that is both precise and adaptable.

In addition to visual similarity, the system enhances its recommendations by integrating additional metadata such as price, rating, usage, and subcategory. This contextual information is used to filter and prioritize recommendations, ensuring they are not only visually similar but also relevant in terms of practical attributes. The user interface, built with Streamlit, provides a seamless experience, allowing users to interact with the recommendation engine in real-time by uploading images or selecting from predefined options. The system is designed to be scalable and continuously learn from user interactions,

updating its suggestions to reflect evolving preferences and new product additions. By focusing on both the visual appeal and practical attributes of fashion items, the proposed system aims to overcome common limitations of traditional recommendation systems, such as the cold start problem and over-specialization, ultimately delivering a more engaging and personalized shopping experience.

2.3 Feasibility study

2.3.1 Details

Technical Feasibility This includes evaluating the capabilities of the pre-trained ResNet50 model for feature extraction and its compatibility with the Nearest Neighbors algorithm for recommendations. The study confirms that the necessary deep learning frameworks (e.g., TensorFlow) and libraries (e.g., scikit-learn) are available and compatible with the system requirements.

Operational Feasibility The study examines how well the system can be integrated into existing e-commerce platforms. It includes assessing user interface design using Streamlit, the ease of image upload and processing, and how the system handles real-time recommendations. Additionally, it considers the operational aspects of maintaining the system and ensuring scalability as the user base and product catalog grow.

Economic Feasibility This involves analyzing the cost of development, including hardware, software, and development team expenses. It also considers ongoing costs for maintenance, updates, and data storage. The potential economic benefits are assessed by estimating increased user engagement, higher sales conversion rates, and the return on investment from improved recommendation accuracy and customer satisfaction.

2.3.2 Impact on Environment

A fashion product recommendation system's environmental impact includes both direct and indirect consequences. The method directly increases energy usage by using data centres to run the recommendation algorithms and keep track of user information. These data centres may use a lot of non-renewable energy, which increases their carbon impact. In addition, while not as much as with data centres, using user devices to access the system increases energy usage. The system may have a significant indirect impact on the fashion industry. Improved suggestions might encourage consumers to spend more money, which could result in more output and a bigger environmental effect from waste, manufacturing, and resource extraction. But the system also has the ability to encourage environmentally beneficial behaviour by promoting eco-friendly goods.

2.3.3 Safety

For a fashion product recommendation system to be secure, user privacy, data security, and system dependability must all be taken into consideration. The system uses strong encryption protocols to secure data, preventing unwanted access to user information while the data is in transit and at rest. To limit access to sensitive data, secure access controls are put in place. These measures include multi-factor authentication and roles with defined authorisation. Frequent vulnerability assessments and security audits assist in locating and resolving such vulnerabilities, guaranteeing continued protection. By following data privacy laws like the CCPA and GDPR, getting express consent before collecting any data, and using data minimisation techniques to reduce the amount of personal information collected, user privacy is protected.

2.3.4 Ethics

In order to prevent perpetuating stereotypes or unfairly disadvantageous discriminating against particular groups, one important component is algorithmic bias mitigation, which entails identifying and correcting any biases in the recommendation algorithms. By doing this, recommendations are guaranteed to be fair and inclusive of a range of requirements and preferences. Another important ethical consideration is transparency; consumers should know how their data is used and how recommendations are made so they can comprehend how the system functions. Giving users the power to manage their data, including the ability to change choices and refuse data collection, respects their privacy and sense of autonomy. Furthermore, by emphasising environmentally and socially conscious fashion options, the system should encourage consumers to make thoughtful and informed selections about products

2.3.5 Cost

The cost analysis for the movie recommendation system includes the expenses related to developing, launching, and maintaining the platform. The study evaluates both initial and ongoing costs, ensuring that the system remains financially sustainable while providing value to users. The analysis also considers potential revenue models that could support the platform's operations, such as premium features or partnerships with streaming services.

Development Costs

Software Development This includes the cost of developing the content-based filtering algorithms, vectorization techniques, and cosine similarity calculations. It also covers the design and development of the user interface using Streamlit.

Data Acquisition and Management Expenses related to acquiring and managing the movie database, including licensing fees for accessing movie metadata and any required third-party APIs.

Testing and Quality Assurance Costs associated with testing the system for accuracy, performance, and user experience to ensure a high-quality product.

Operational Costs

Server Hosting and Maintenance Ongoing expenses for hosting the platform, including the costs of cloud services, server maintenance, and data storage to ensure the platform remains scalable and responsive.

Cybersecurity Investments in cybersecurity measures, such as encryption, secure authentication protocols, and regular security audits, to protect user data and maintain platform integrity.

User Support Costs associated with providing customer support services, including handling user inquiries, troubleshooting issues, and maintaining a helpdesk.

Revenue Model

Affiliate Marketing Partner with fashion retailers to include their product links in your recommendations. Earn a commission on sales made through these links. This model aligns with user interests and turns engagement into revenue.

Freemium Model Offer a basic version of your platform for free while charging for premium features like advanced personalization or an ad-free experience. This attracts a broad user base and converts engaged users into paying subscribers.

Ad-Based Revenue Display ads from fashion brands or sponsored recommendations within your platform. Generate revenue through ad placements, leveraging user engagement for consistent income. This model integrates naturally with your platform's content.

2.3.6 Type

The type of a project can be categorized based on its purpose and the platform it targets. For a fashion recommendation model, the type could fall under various categories:

Application This could be a mobile app, web application, or a standalone software that provides fashion recommendations to users. A mobile app would be designed for iOS or Android devices, while a web application would be accessible via a browser. Standalone software might be a desktop application.

Product If the fashion recommendation model is being developed as a commercial product, it would be aimed at end-users or businesses to enhance their fashion choices or inventory management.

Research If the focus is on exploring new algorithms, improving accuracy, or understanding user behavior in fashion recommendations, it could be categorized as research. This involves experimenting with various approaches to advance knowledge in the field.

Review If the project involves analyzing existing recommendation systems or evaluating their effectiveness, it might be considered a review. This would entail assessing current technologies, methods, and their application in the fashion industry.

2.3.7 Standards

Combining Agile with CMMI and Six Sigma allows for a robust development approach that balances flexibility with quality management. Agile supports iterative development and quick adaptations, while CMMI and Six Sigma ensure that processes are standardized, quality-driven, and continuously improved. This integrated approach helps in efficiently developing a high-quality fashion recommendation system that can adapt to changing user needs and trends while maintaining rigorous standards of performance and reliability.

Flexibility and Iterative Development Agile is well-suited for projects involving evolving requirements and iterative development. Since fashion trends and user preferences can

change rapidly, Agile allows for incremental improvements and adaptations based on user feedback, ensuring the recommendation system remains relevant and effective.

Continuous Testing and Integration Agile promotes continuous testing and integration, which is crucial for a system that depends on deep learning models and real-time processing. Frequent iterations help identify and resolve issues early, improving the overall quality of the system.

Collaboration and User Feedback Agile emphasizes collaboration among cross-functional teams and incorporates user feedback regularly. This approach ensures that the system aligns with user needs and preferences, enhancing its effectiveness and user satisfaction.

2.4 Scope of the Project

Development of a Deep Learning Model Implement a fashion recommendation engine using a pre-trained ResNet50 model for feature extraction from product images. This involves fine-tuning the model to accurately capture visual features and similarities between different fashion items.

Integration with Existing Data Incorporate a cleaned dataset containing product metadata, including price, rating, usage, and subcategory. The system will use this information to enhance the relevance and contextuality of recommendations, ensuring a holistic approach to product suggestions.

User Interface Design Develop a user-friendly interface using Streamlit to facilitate image uploads and display recommendations. The interface will allow users to interact with the system, upload images, and view recommended products in real-time.

Recommendation Algorithm Implementation Utilize a Nearest Neighbors algorithm to identify and recommend similar items based on visual features extracted from user-uploaded images. This will involve optimizing the algorithm for accurate and efficient similarity searches.

Performance Evaluation Test and evaluate the system's performance in terms of accuracy, user satisfaction, and system responsiveness. This includes assessing the quality of recommendations and the system's ability to handle various types of user inputs and product images.

Scalability and Maintenance Ensure the system is scalable to accommodate growing numbers of users and products. Develop maintenance strategies to keep the system updated with new products and evolving fashion trends.

Privacy and Data Security Implement measures to protect user data and ensure privacy. This involves secure handling of user-uploaded images and compliance with data protection regulations.

Deployment and User Training Deploy the system to a production environment and provide training for users to maximize the benefits of the recommendation engine. This includes creating documentation and support resources.

2.5 System Configuration

Hardware Requirements

Processor Advanced modern processor with multi-core capabilities .

Memory (RAM) Minimum 16 GB of RAM to ensure smooth performance, especially when handling large datasets.

Storage Adequate storage space for dataset management and system files .

Network High-speed internet connection for accessing movie metadata, updates, and user interactions.

Software Requirements

Operating System Windows 8 or later

Front-End Technologies

Streamlit For building the user interface.

Pre-Model and Python Libraries

ResNet50 Pre-trained deep learning model for feature extraction

OpenCV or Pillow Libraries for image processing and handling

Development Environment

IDE/Code Editor

Visual Studio Code or PyCharm for code development.

Jupyter Notebook for data analysis and experimentation .

Version Control Git for source code management, with repositories hosted on platforms like GitHub.

3.LITERATURE OVERVIEW

3.1 Literature Review

Collaborative Filtering Recommendation Algorithms based on Items.

Sarwar, B. M., Konstan, J. A., Karypis, G., and Riedl, J. "Collaborative Filtering Recommendation Algorithms based on Items." Information Systems Transactions ACM (TOIS).

This important work presents collaborative filtering techniques based on items. It illustrates how suggestions can be made using item-item similarity, which addresses scalability concerns and improves recommendation accuracy over user-based methods.

Recommendation Systems Based on Content

Billsus, D., and M. J. Pazzani (2007). "Recommendation Systems Based on Content." In The Adaptive Web: Web Personalisation Techniques and Strategies.

Pazzani and Billsus talk about user profiles and item features as the foundation for content-based recommendation systems. The study examines several methods for content-based filtering and how they are used to customise recommendations according to user preferences.

Collaborative neural filtering

He (2017), Liao (2017), Zhang (2017), Nie (2017), and Hu (2017). "Collaborative neural filtering." Proceedings of the 26th World Wide Web (WWW) International Conference.

This work presents Neural Collaborative Filtering (NCF), a recommendation system approach based on deep learning. The authors present how recommendation performance can be enhanced by modelling intricate user-item interactions using neural networks.

Zhang, Y., Sun, X., and Yu, M. (2019). "A Survey of Context-Aware Recommendation Systems." ACM Surveys on Computing (CSUR).

Yu and colleagues examine the creation and utilisation of recommendation systems that are aware of context, taking into account variables including time, place, and patterns. The study investigates several models and methods for incorporating contextual data into suggestions.

Recommender Systems: Difficulties and Prospects for Research

Shapira, B., Rokach, L., and Ricci, F. (2011). "Recommender Systems: Difficulties and Prospects for Research." Springer.

This book offers a thorough introduction to recommender systems, covering both the difficulties and advantages of customising recommendations. It addresses methods for raising user engagement as well as user profiling and adaptive learning.

Jannach, D., and M. Zanker (2009). "Appraising Suggestions: A Comparative Analysis of Different Methodologies." User-Adapted Interaction and User Modelling.

This study investigates different methods of evaluating recommendation systems, emphasising the role that user feedback and interaction play in enhancing and optimising suggestion accuracy.

Attacks Using Membership Inference on Machine Learning Models

Shokri, R., Shmatikov, V., and Stronati, M. (2017). "Attacks Using Membership Inference on Machine Learning Models." 2017 IEEE Security and Privacy Symposium (SP).

This work addresses machine learning privacy issues, particularly membership inference attacks. It emphasises how important privacy-preserving methods are for safeguarding user data in recommendation systems.

A Big Data Processing Survey

In 2017, Huang, J., and Yao, L. "A Big Data Processing Survey." IEEE Access.

Yao and Huang examine large-scale data processing methods, which are pertinent to recommendation systems that must manage massive volumes of user interactions

4.SYSTEM DESIGN

4.1 System architecture

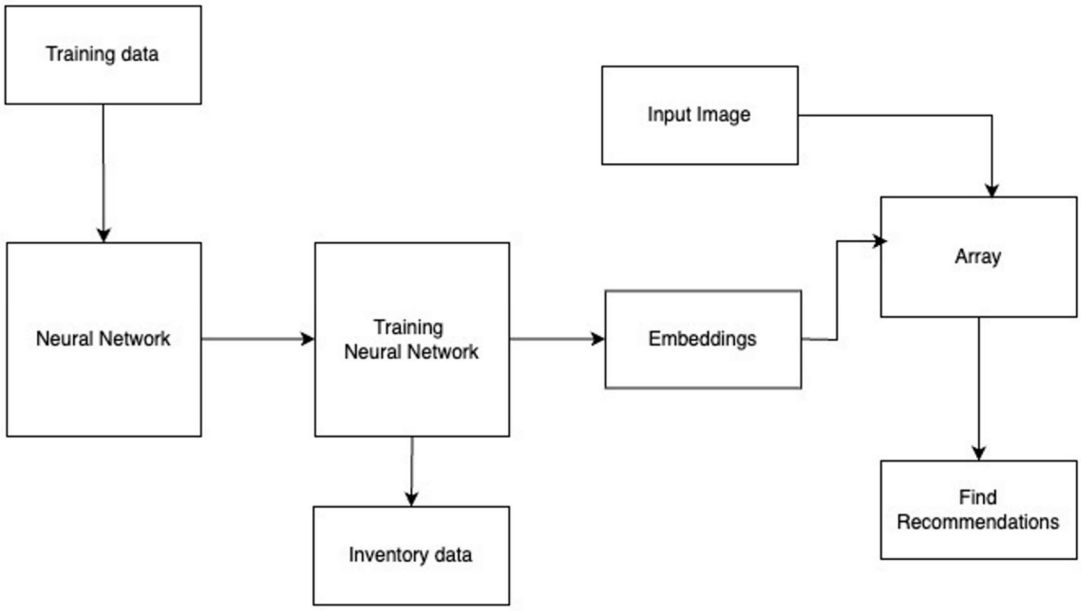

Fig 4.1 System architecture

Training Data

The system starts with a dataset of 44,000 images, which includes various fashion items. This dataset is essential for training the ResNet50 model to recognize and understand different fashion elements. The images are preprocessed to enhance their quality and consistency, ensuring the model can learn effectively. This step involves normalizing the images, augmenting them to increase diversity, and labeling them with relevant metadata such as categories, styles, and colors. The quality and diversity of this training data are crucial for the model's performance.

Neural Networks (ResNet50)

ResNet50, short for Residual Networks with 50 layers, is a powerful deep convolutional neural network widely used for complex image recognition and classification tasks. It is particularly renowned for its capability to overcome the vanishing gradient

problem, a common issue in deep neural networks where gradients used to update weights diminish as they are propagated backward through the layers. This problem often hinders the training of very deep networks. ResNet50 addresses this challenge using a unique architecture known as "residual learning."

During the training process, ResNet50 is exposed to a large number of fashion images from the dataset, each labeled with specific categories or attributes (e.g., shirts, dresses, accessories). The network adjusts its weights and biases iteratively through backpropagation to minimize the prediction error, typically measured by a loss function like categorical cross-entropy in classification tasks. With each pass, the network fine-tunes its internal parameters to better capture the subtle details and distinctions between different fashion items, such as patterns, textures, colors, and shapes.

Once the ResNet50 network is trained, it can be used as a feature extractor. This means that for any new input image, the network can generate a fixed-length feature vector, also known as an embedding, from one of its final layers before the classification output. These embeddings encapsulate the essential visual characteristics of the input image, such as edges, textures, and more abstract features that represent the high-level attributes of fashion items. This process of feature extraction is crucial because it transforms raw image data into a format that can be efficiently used for various downstream tasks, such as image retrieval, clustering, or in this case, recommendation.

By leveraging the ResNet50 model, the recommendation system benefits from a state-of-the-art deep learning framework capable of handling the complexities of visual data. The model's ability to learn and represent subtle variations in fashion items makes it exceptionally suited for the fashion domain, where visual nuance is critical. Additionally, its use of residual connections allows for deeper network architectures without the risk of degraded performance due to vanishing gradients, making it possible to capture more complex patterns and relationships in the data.

Trained Neural Networks

After the training phase, the ResNet50 model is considered “trained” and ready to make predictions on new data. These trained networks can now recognize and extract relevant features from new fashion images, such as patterns, colors, and styles. They serve as the backbone of the recommendation system, processing input images and generating embeddings. The trained networks are continuously updated with new data to improve their accuracy and relevance over time, ensuring the system provides personalized and up-to-date fashion recommendations.

Input Image

Users can upload images of fashion items they are interested in, such as clothing, accessories, or shoes. The system processes these input images to extract features and generate embeddings that represent the visual characteristics of the items. This step allows the recommendation system to understand the user’s preferences based on the visual content of the images. The input images are compared with the inventory data to find similar items that match the user’s style. This feature makes the system interactive and user-friendly, allowing for personalized recommendations based on visual inputs.

Embeddings

Embeddings are numerical representations of the input images, capturing their essential features in a lower-dimensional space. Generated by the trained ResNet50 model, they serve as a compact and efficient way to represent the visual characteristics of fashion items. Embeddings enable the system to perform similarity searches, finding items in the inventory that are visually similar to the input image. This step is crucial for the recommendation process, as it allows the system to match user preferences with available inventory. Embeddings also facilitate efficient storage and retrieval of fashion items, making the system scalable.

K-Nearest Neighbors (KNN)

The KNN algorithm is used to find the closest items in the embedding space to the input image. It works by calculating the distance between the embeddings of the input image and those of the items in the inventory. The KNN algorithm then selects the ‘k’ nearest neighbors, which are the most similar items. This method is simple yet effective for recommendation tasks, as it directly compares the visual features of fashion items. KNN ensures that the recommendations are relevant and closely match the user’s preferences.

Final Recommendations:

The final step involves generating fashion recommendations based on the input image and the inventory data processed by the trained neural networks and KNN algorithm. The system combines the embeddings and similarity searches to provide users with personalized fashion suggestions. This ensures that the recommendations are both relevant and available in the inventory, enhancing the user experience.

4.2 UML Diagrams

4.2.1 USE CASE DIAGRAM

A use case diagram for a fashion product recommendation system visualizes the interactions between users and the system, highlighting the main functionalities and user roles.use case diagram provides a high-level overview of how users, administrators, and external data providers interact with the fashion product recommendation system. It helps in understanding the system's functionality and the roles of different actors.

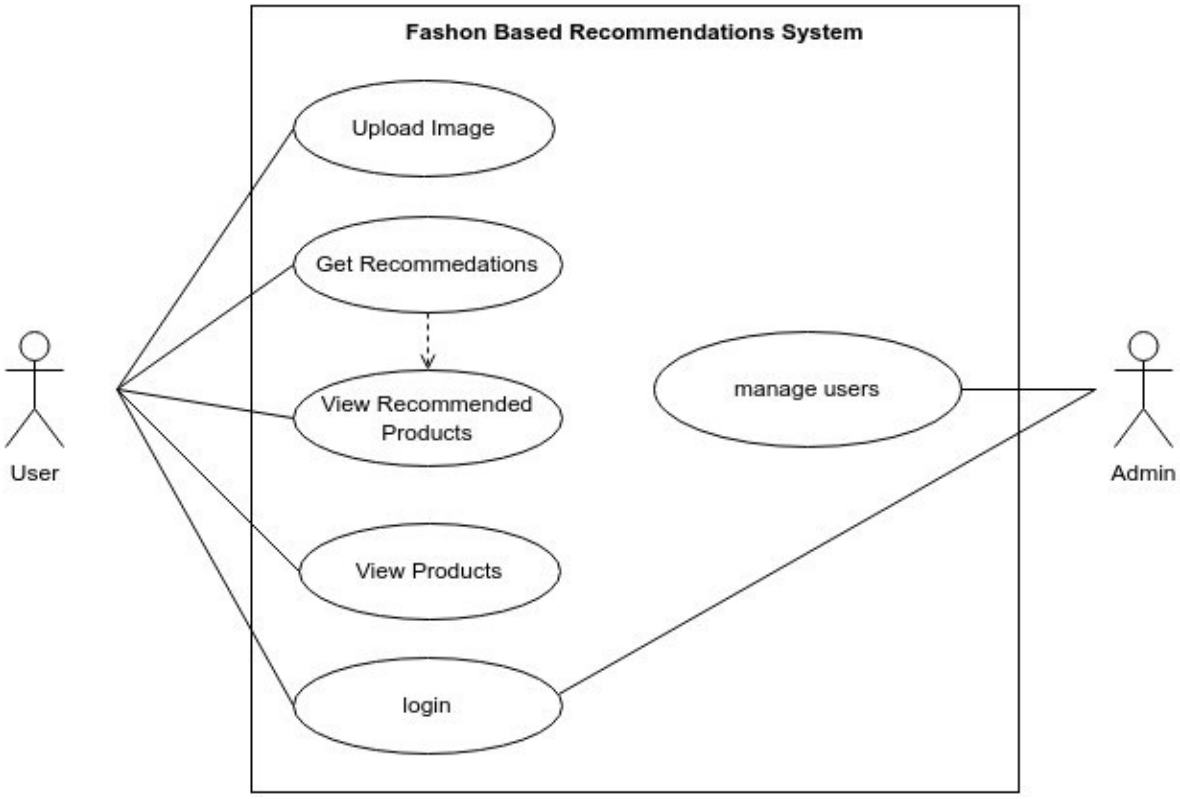

Fig 4.2 Use case Diagram

4.2.2 CLASS DIAGRAM

The class diagram for the fashion product recommendation system provides a structured view of the system's components and their interactions. At the heart of the system is the **User** class, which manages user-related attributes such as userID, username, and preferences, and includes methods for logging in, managing profiles, and viewing recommendations. The **Product** class represents individual products with attributes like productID, name, and price, and methods for retrieving details and updating stock levels.

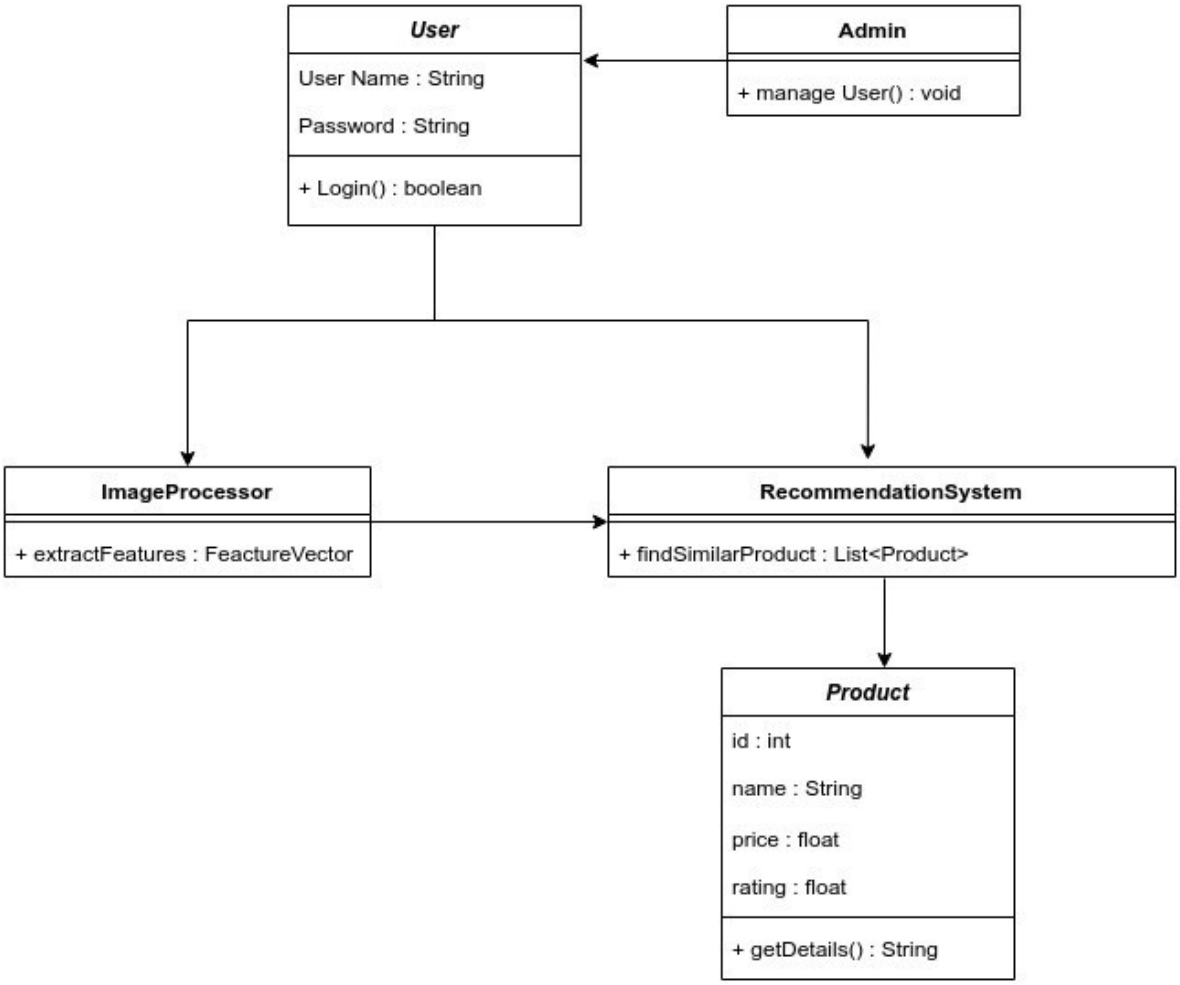

Fig 4.3 Class Diagram

4.2.3 ACTIVITY DIAGRAM

The flow of procedures involved in creating and providing customised recommendations is depicted in the activity diagram for the fashion product recommendation system. The user logs in or authenticates himself into the system to start it. The consumer can peruse different product categories after completing the authentication process successfully. After processing user input, the recommendation engine is used by the system to provide customised recommendations. The user can then comment on the suggestions or products they have viewed after seeing these recommendations

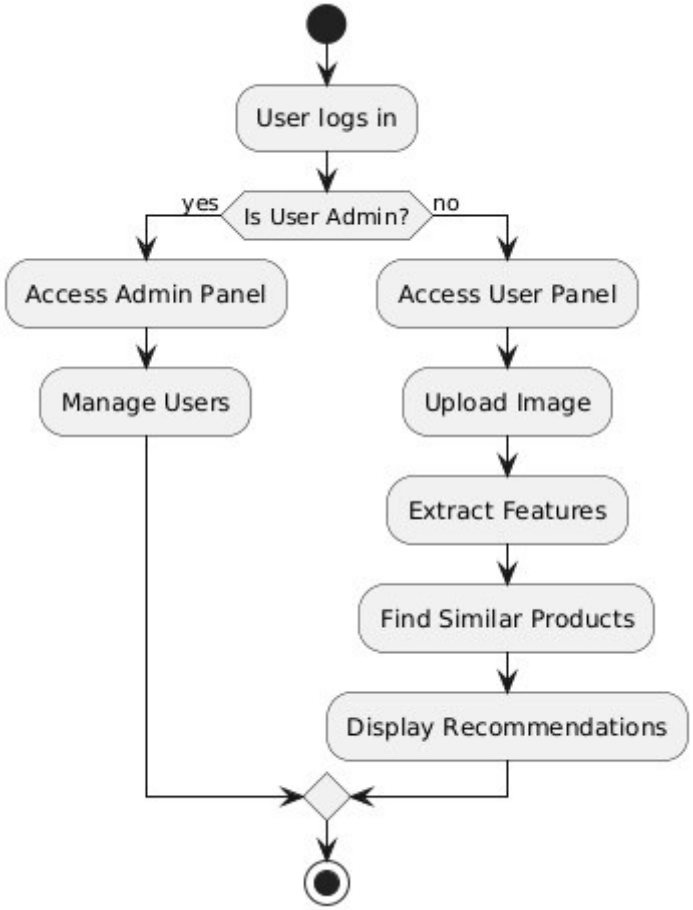

Fig 4.4 Activity Diagram

4.2.4 SEQUENCE DIAGRAM

The relationship that develops over time between users, the system, and its components is depicted in the sequence diagram for the fashion product suggestion system. When a user logs into the system to start a session, the procedure begins. Following successful authentication, the user explores products by interacting with the system. Relevant product listings are retrieved for display by the system through queries of the product database.

The recommendation engine is triggered by the system as the user examines various items, and it then generates personalised product recommendations based on the user's past interactions and interests. After processing this data, the recommendation engine provides a list of suggested products. The user is then shown these recommendations on the interface.

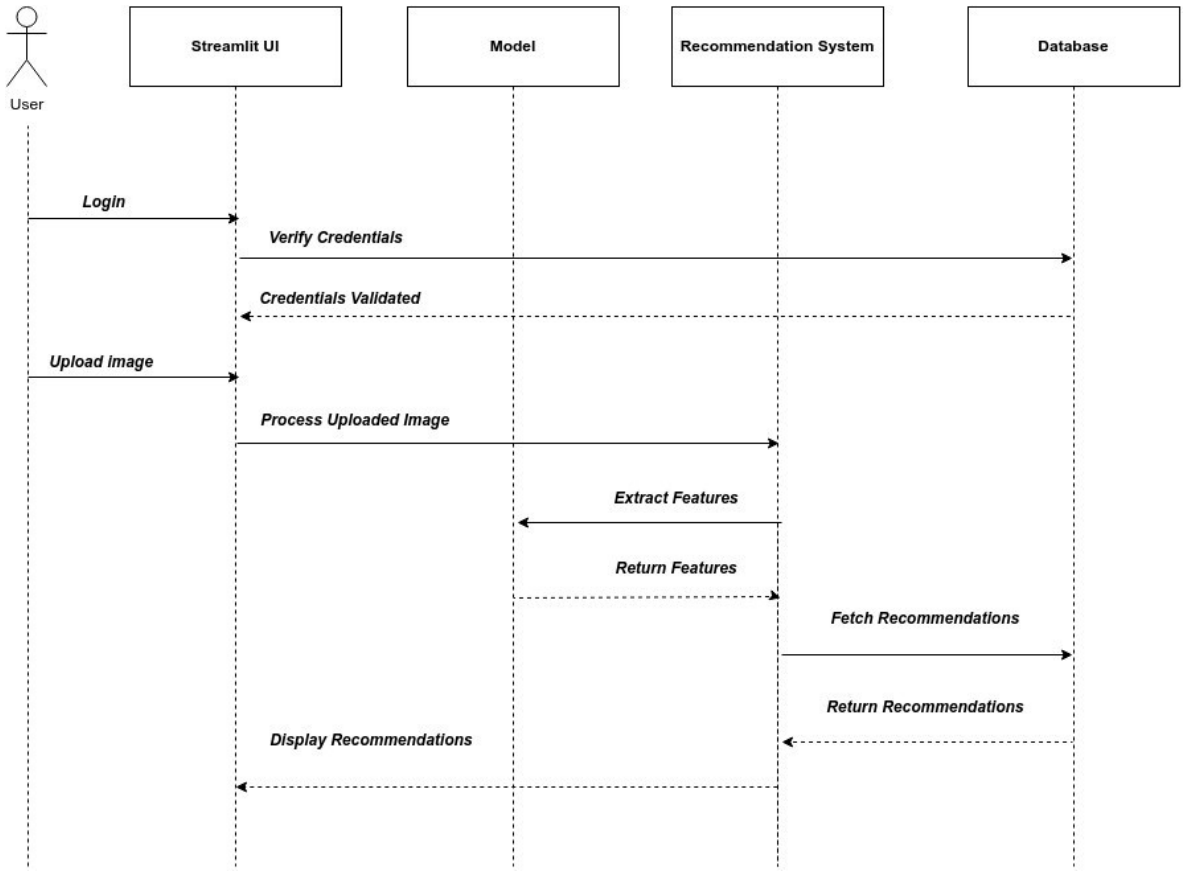

Fig 4.5 Sequence Diagram

5.IMPLEMENTATION

5.1 Implementation

Dataset

Using a dataset of 44,000 fashion product images from Myntra, sourced from Kaggle, the system focuses on capturing both the visual appeal and practical attributes of fashion items. The dataset contains a wide variety of clothing, accessories, and footwear, each labeled with metadata including category, subcategory, color, season, usage, price, and rating, which are crucial for making contextually relevant recommendations.Each image is around 4-20 KB and dimensions are 60x80 ie width is 60 pixels and Height is 80 pixels.

Data Preprocessing

The first step in implementing the system involves preprocessing the dataset. All images are resized to a uniform dimension of 224x224 pixels, a standard input size for the ResNet50 model. This resizing ensures consistency and compatibility with the model's input layer, which expects images in this format. The images are also normalized to match the distribution that the ResNet50 model was originally trained on, enhancing the accuracy of feature extraction. Any accompanying metadata is cleaned and structured to ensure it can be seamlessly integrated into the recommendation process.

Feature Extraction Using ResNet50

The ResNet50 model, a 50-layer deep convolutional neural network pre-trained on the ImageNet dataset, is employed to extract high-dimensional feature vectors from each image. ResNet50 is chosen for its superior ability to learn complex visual patterns and its efficient architecture, which mitigates the vanishing gradient problem common in deep networks. By leveraging transfer learning, we use the model's convolutional base to compute feature vectors without modifying the pre-trained weights, capturing essential visual features such as color, texture, and shape. These features are represented as a fixed-length vector for each image, obtained by passing the images through the network up to the

GlobalMaxPooling2D layer, which condenses the feature maps into a single vector that encapsulates the most salient visual details.

Building the Feature Database

After feature extraction, the resulting vectors are stored in a database or a NumPy array, depending on the desired scalability and speed of the system. This feature database forms the backbone of the recommendation engine, allowing for efficient similarity searches. Each vector is paired with its corresponding image metadata, enabling the system to make recommendations based not only on visual similarity but also on additional criteria such as price, usage, and subcategory.

Implementing K-Nearest Neighbors (KNN) for Recommendations

The K-Nearest Neighbors (KNN) algorithm is then applied to the feature vectors to find the 'k' nearest neighbors for any given query image. KNN is a non-parametric algorithm that identifies the closest training examples in the feature space, making it well-suited for this task. When a user uploads an image or selects an item, the system extracts its feature vector using the same ResNet50 model and compares it against the precomputed feature database using a distance metric, typically Euclidean distance. The 'k' closest items are identified as the most visually similar to the input image.

Step 1 Select the number K of the neighbors

Step 2 Calculate the Euclidean distance of K number of neighbors

Step 3 Take the K nearest neighbors as per the calculated Euclidean distance.

Step 4 Among these k neighbors, count the number of the data points in each category.

Step 5 Assign new data points to the category with the maximum number of neighbors.

Real-Time Recommendations with Streamlit Interface

To provide a user-friendly experience, the system integrates with Streamlit, a popular web framework for building interactive applications. Users can upload an image or select an item of interest directly through the interface, prompting the system to compute real-time recommendations. Upon receiving a query image, the system extracts its features and retrieves the 'k' most similar items from the database using KNN. These recommendations are then displayed along with their metadata—price, rating, usage, and subcategory—offering users a comprehensive view of their options.

Algorithm

Step 1 Collect Fashion Images

Step 2 Preprocess the Images(Resize,Normalize,Data Augmentation)

Step 3 Load a Pre-trained ResNet Model

Step 4 Train the Model

Step 5 Extract Features using ResNet

Step 6 Using KNN Algorithm for Recommendations

Step 7 Generate Recommendations

Step 8 Build a Simple UI(Streamlit)

5.2 Sample Code

Model.py

```
filenames = []

for file in os.listdir('images'):

    filenames.append(os.path.join('images',file))

len(filenames)

model = ResNet50(weights='imagenet', include_top=False, input_shape=(224,224,3))

model.trainable = False

model = tf.keras.models.Sequential([model,GlobalMaxPool2D()])

model.summary()

img = image.load_img('C:/Users/Lenovo/OneDrive/Desktop/mini proj/images/1163.jpg', target_size=(224,224))

img_array = image.img_to_array(img)

img_expand_dim = np.expand_dims(img_array, axis=0)

img_preprocess = preprocess_input(img_expand_dim)

result = model.predict(img_preprocess).flatten()

norm_result = result/norm(result)

norm_result

def extract_features_from_images(image_path, model):

    img = image.load_img(image_path, target_size=(224,224))
```

```
    img_array = image.img_to_array(img)
    img_expand_dim = np.expand_dims(img_array, axis=0)
    img_preprocess = preprocess_input(img_expand_dim)
    result = model.predict(img_preprocess).flatten()
    norm_result = result/norm(result)
    return norm_result
extract_features_from_images(filenames[0], model)
image_features = []
for file in filenames:
    image_features.append(extract_features_from_images(file, model))
image_features
Image_features = pkl.dump(image_features, open('embeddings.pkl','wb'))
filenames = pkl.dump(filenames, open('filenames.pkl','wb'))
```

Login.py

```
import streamlit as st
if 'authentication_status' not in st.session_state:
    st.session_state['authentication_status'] = None
def main():
    if st.session_state['authentication_status'] is None:
        st.title("Login Page")
```

```
username = st.text_input("Username")
password = st.text_input("Password", type="password")
if st.button("Login"):
    if username == "admin" and password == "admin":
        st.session_state['authentication_status'] = 'admin'
        st.success("Welcome Admin!")
    elif username == "user" and password == "user":
        st.session_state['authentication_status'] = 'user'
        st.success("Welcome User!")
    else:
        st.error("Invalid username or password")
st.markdown("---")
st.subheader("Developed by:")
col1, col2, col3 = st.columns([1, 1, 1], gap="small")
with col1:
    st.image("pho.jpg", caption="Harshit", width=80)
with col2:
    st.image("sarath.jpg", caption="Sharath", width=80)
with col3:
    st.image("chakri.jpg", caption="Chakradhar", width=80)
```

```
    elif st.session_state['authentication_status'] == 'admin':
        st.title("Admin Page")
        import app  # Import and run app.py content directly here
        app.run()  # This assumes you have a function `run()` in app.py
    elif st.session_state['authentication_status'] == 'user':
        st.title("User Page")
        import rec
        rec.run()
if __name__ == "__main__":
    main()
```

rec.py(User)

```
def run():
    st.header('Fashion Recommendation System')
        def extract_features_from_images(image_path, model):
            img = image.load_img(image_path, target_size=(224, 224))
            img_array = image.img_to_array(img)
            img_expand_dim = np.expand_dims(img_array, axis=0)
            img_preprocess = preprocess_input(img_expand_dim)
            result = model.predict(img_preprocess).flatten()
            norm_result = result / norm(result)
```

```
    return norm_result

base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

base_model.trainable = False

model = tf.keras.models.Sequential([base_model, GlobalMaxPool2D()])

neighbors = NearestNeighbors(n_neighbors=6, algorithm='brute', metric='euclidean')

neighbors.fit(image_features)

num_columns = 4

rows = len(image_data) // num_columns + (len(image_data) % num_columns > 0)

recommended_images = []

for row in range(rows):

    cols = st.columns(num_columns)

    for col_index in range(num_columns):

        img_index = row * num_columns + col_index

        if img_index < len(image_data):

            img_data = image_data[img_index]

            with cols[col_index]:

                st.write(f"**{img_data['item']}**")

                img = Image.open(img_data['path'])

                filename = os.path.basename(img_data['path'])
```

```
                description_data = descriptions.get(filename, {"productDisplayName": "No description available"})
                st.image(img, caption=description_data['productDisplayName'], use_column_width=True)
                if st.button(f"Recommend {img_data['item']}"):
                    input_img_features = extract_features_from_images(img_data['path'], model)
                    distance, indices = neighbors.kneighbors([input_img_features])
                    recommended_images = [filenames[indices[0][j+1]] for j in range(5)] # Skip the first one because it's the same image
    upload_file = st.file_uploader("Upload Image")
    if upload_file is not None:
        upload_path = os.path.join('C:/Users/Lenovo/OneDrive/Desktop/mini proj/upload', upload_file.name)
        with open(upload_path, 'wb') as f:
            f.write(upload_file.getbuffer())
        st.subheader('Uploaded Image')
        st.image(upload_file)
        input_img_features = extract_features_from_images(upload_path, model)
        distance, indices = neighbors.kneighbors([input_img_features])
        recommended_images = [filenames[indices[0][j+1]] for j in range(5)] # Skip the first one because it's the same image
```

```
    if recommended_images:
        st.subheader('Recommended Products')
        rec_cols = st.columns(5)
        for j, rec_col in enumerate(rec_cols):
            with rec_col:
                if j < len(recommended_images):
                    rec_img_path = recommended_images[j]
                    rec_img = Image.open(rec_img_path)
                    rec_description_data = descriptions.get(
                        os.path.basename(rec_img_path),
                        {"productDisplayName": "No description available", "Price": "N/A", "Rating": "N/A", "usage": "N/A", "subCategory": "N/A"}
    main()
```

app.py(Admin)

```
def run():
    st.header('Fashion Recommendation System')
    Image_features = pkl.load(open("C:/Users/Lenovo/OneDrive/Desktop/mini proj/embeddings.pkl",'rb'))
    filenames = pkl.load(open("C:/Users/Lenovo/OneDrive/Desktop/mini proj/filenames.pkl",'rb'))
    def extract_features_from_images(image_path, model):
```

```
        img = image.load_img(image_path, target_size=(224,224))
        img_array = image.img_to_array(img)
        img_expand_dim = np.expand_dims(img_array, axis=0)
        img_preprocess = preprocess_input(img_expand_dim)
        result = model.predict(img_preprocess).flatten()
        norm_result = result/norm(result)
        return norm_result
    model = ResNet50(weights='imagenet', include_top=False, input_shape=(224,224,3))
    model.trainable = False
    model = tf.keras.models.Sequential([model,GlobalMaxPool2D()])
    neighbors = NearestNeighbors(n_neighbors=6, algorithm='brute', metric='euclidean')
    neighbors.fit(Image_features)
    upload_file = st.file_uploader("Upload Image")
    if upload_file is not None:
        with open(os.path.join('C:/Users/Lenovo/OneDrive/Desktop/mini proj/upload',
upload_file.name), 'wb') as f:
            f.write(upload_file.getbuffer())
        st.subheader('Uploaded Image')
        st.subheader('Train the model')
if __name__ == "__main__":
    run()
```

6.TESTING

6.1. Testing

Unit Testing

In this project, unit testing ensures that individual functions like image uploading, feature extraction, and data loading work correctly. It verifies that each component behaves as expected in isolation.

Integration Testing

Integration testing checks how different parts of the system, such as the user interface, image processing, and recommendation engine, work together. It ensures that the components interact seamlessly and that data flows correctly through the system.

System Testing

This testing validates the entire recommendation process from image upload to displaying recommendations, ensuring that the full system meets the functional requirements and performs as expected in a simulated real-world environment.

User Acceptance Testing (UAT)

UAT involves end-users testing the system to ensure it meets their expectations and needs. In this project, it checks if users find the recommendations accurate and the interface intuitive.

Performance Testing

Performance testing assesses how the system handles multiple users and large datasets, ensuring it remains responsive and efficient. It tests the system's capability to provide quick recommendations without lag.

Security Testing

Security testing identifies vulnerabilities in the system to protect against unauthorized access and data breaches, ensuring user data is securely handled throughout the recommendation process.

6.2. Test cases

Test Case 1 Image Upload and Display

Objective Ensure the system correctly handles image uploads and displays them in the UI.

Precondition User is on the Streamlit application interface.

Steps

User uploads an image using the file uploader.

Verify that the image appears on the screen with the correct formatting.

Expected Output The uploaded image is displayed in the designated area with no errors or distortions.

Test Case 2 Feature Extraction from Uploaded Image

Objective Verify that the system correctly preprocesses and extracts features from an uploaded image.

Precondition User has uploaded an image.

Steps

Trigger feature extraction after the image is uploaded.

Check if the ResNet50 model processes the image and generates a feature vector.

Expected Output The system successfully generates a normalized feature vector without any errors.

Test Case 3 Generate and Display Recommendations

Objective Test the recommendation engine to ensure it generates appropriate recommendations based on the uploaded image.

Precondition Image feature extraction is completed.

Steps

Click the "Recommend" button.

The system retrieves the nearest neighbors using the k-NN algorithm.

Verify that recommended images are displayed with relevant metadata.

Expected Output The system displays a set of recommended images with correct descriptions, prices, ratings, usage, and subcategories.

Test Case 4 Metadata Retrieval and Integration

Objective Ensure metadata such as price, rating, usage, and subcategory is correctly retrieved and displayed with recommended items.

Precondition Recommendations have been generated.

Steps

Check the display of each recommended product.

Verify that metadata is accurate and matches the products.

Expected Output All recommended products are displayed with the correct price, rating, usage, and subcategory information.

Test Case 5 Handling Unsupported File Formats

Objective Verify that the system gracefully handles unsupported file formats.

Precondition User is on the Streamlit application interface.

Steps

Upload a non-image file (e.g., a text or PDF file).

Check the system's response.

Expected Output The system displays an error message indicating the unsupported file format without crashing.

Test Case 6 Real-Time Performance and Response

Objective Ensure the system performs efficiently and provides recommendations in real-time.

Precondition User uploads an image.

Steps

Measure the time taken from image upload to displaying recommendations.

Expected Output The entire process from upload to recommendation should be completed within a few seconds, providing a smooth user experience.

Test Case 7 No Recommendations Found

Objective Test the system's behavior when no similar items are found.

Precondition User uploads an image of a product not in the database.

Steps

Upload an uncommon or unique image.

Trigger the recommendation engine.

Expected Output The system should display a message indicating that no similar items were found, rather than returning an error.

7.OUTPUT SCREENS

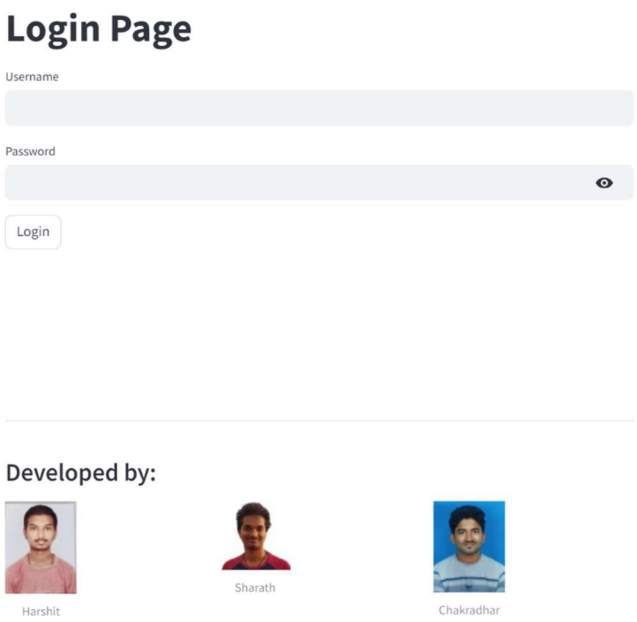

Fig .7.1.Login Page

Login screen with fields username and password, developer names and images.

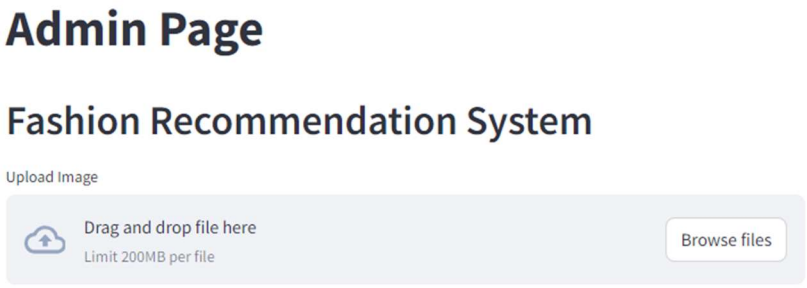

Fig .7.2.Admin Page

This image depicts the "Admin Page" for a "Fashion Recommendation System." It has an area where users may upload photographs by dragging and dropping them or clicking the "Browse files" button.

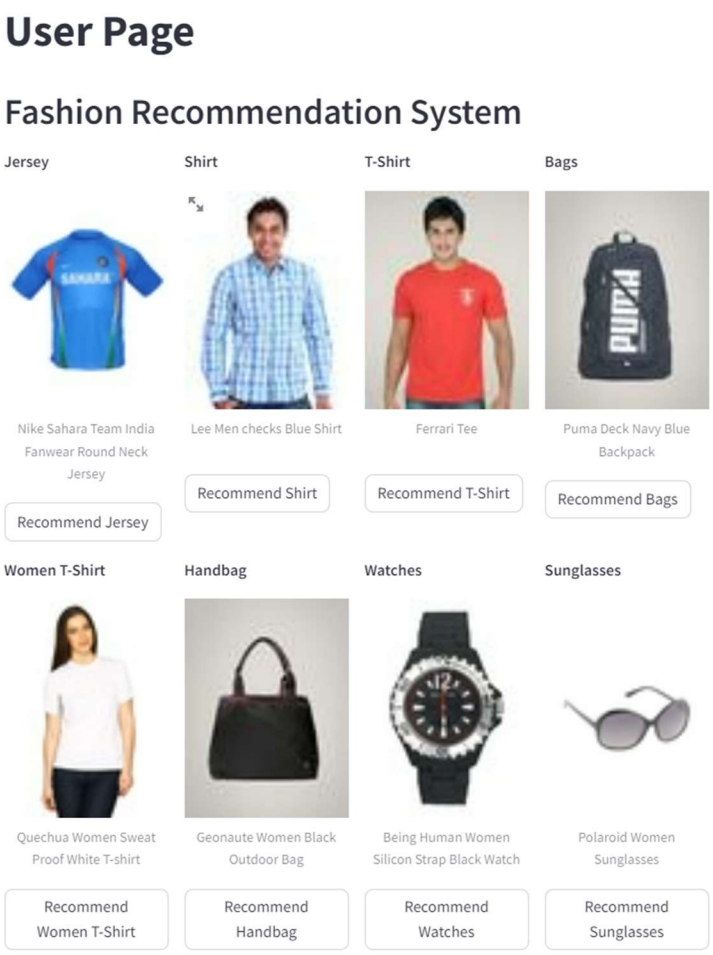

Fig .7.3.User Page1

This is a user page with some predefined recommend categories and a button to get recommended items.

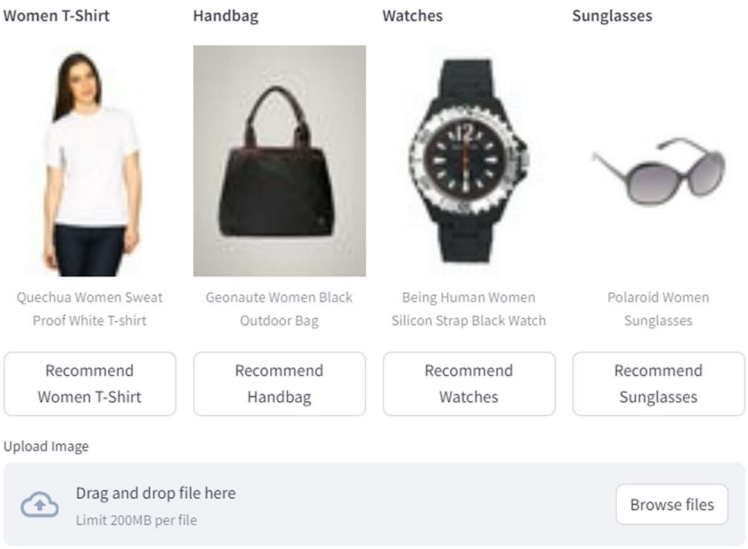

Fig .7.4. User Page 2

There is another option to upload image and get the recommended products to it.

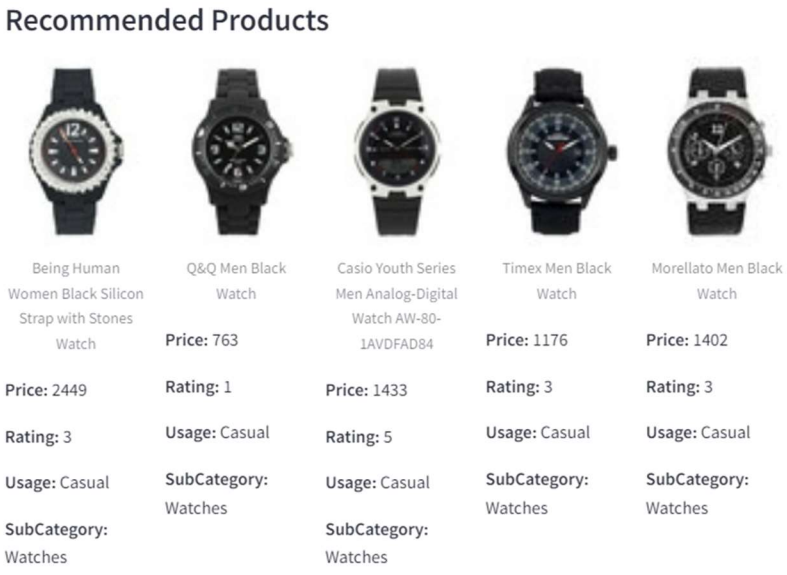

Fig .7.5.Recommended products

This is the recommended product section where display the recommended products for given input

8. CONCLUSION

8.1 Conclusion

The fashion recommendation system we developed leverages advanced deep learning techniques and machine learning algorithms to provide personalized fashion suggestions based on uploaded images. Utilizing a ResNet50 model pre-trained on the ImageNet dataset, the system efficiently extracts meaningful features from clothing and accessory images, transforming them into a numerical representation that captures their visual characteristics. This feature extraction is coupled with a nearest neighbors algorithm that searches through a precomputed database of image embeddings to identify the most visually similar items, offering users recommendations that align with their preferences.

This project demonstrates the effectiveness of using convolutional neural networks (CNNs) like ResNet50 for feature extraction in fashion applications, which is crucial for creating a scalable and robust recommendation engine. By integrating a user-friendly interface built with Streamlit, the system provides a seamless user experience, allowing users to interact with the recommendation engine easily. The inclusion of metadata such as product descriptions, prices, ratings, subcategories, and usage information enhances the recommendations, providing a more comprehensive shopping experience that helps users make informed decisions.

While the system currently provides high-quality recommendations based on visual similarity, there are some areas for improvement. The recommendation engine can be further optimized by incorporating additional data types, such as user preferences, past purchase history, and contextual data like current fashion trends and seasonal changes. Also, the model could be trained with more diverse datasets to improve its understanding of a wider range of fashion styles and cultural contexts, making the recommendations even more relevant and personalized.

Overall, the project lays a solid foundation for building sophisticated fashion recommendation systems that combine deep learning with real-time user interaction.

8.2 Further Enhancements

Expand to Multi-Modal Recommendations

Currently, the system focuses on visual similarity for recommendations. Future versions could include multi-modal data inputs, such as text descriptions, user reviews, and ratings, to provide a richer context for each item. Natural Language Processing (NLP) techniques could analyze textual data, while sentiment analysis could gauge user sentiment from reviews, adding another layer of refinement to the recommendation engine.

Enhance Feature-Extraction with TransferLearning

While ResNet50 is effective for general image recognition tasks, further fine-tuning the model on a specific fashion dataset could improve its feature extraction capabilities for this domain. Transfer learning techniques could be employed to adapt the model to recognize finer details in fashion items, such as fabric types, patterns, or brand-specific styles, enhancing the system's ability to provide more nuanced recommendations.

Optimize Performance for Scalability

As the system scales to include a larger dataset of images and an increasing number of users, optimizing the performance of both the image processing pipeline and the recommendation algorithm becomes crucial. Techniques such as distributed computing, GPU acceleration, and efficient indexing methods like Approximate Nearest Neighbors (ANN) could be employed to ensure the system maintains quick response times and can handle a growing user base.

Enhance Security and Privacy Measures

As the system collects more user data to provide personalized recommendations, ensuring robust security and privacy measures is essential. Future enhancements could include implementing secure user authentication, data encryption, and privacy-preserving machine learning techniques to safeguard user information and maintain trust.

9.BIBLIOGRAPHY

9.1 Books References

[1] **Deep Learning with Python** by François Chollet. (2nd Edition). Manning Publications.

This book provides a comprehensive guide on deep learning concepts and their applications using Keras and TensorFlow, which are relevant to the fashion recommendation system.

[2] **Pattern Recognition and Machine Learning** by Christopher M. Bishop. Springer.

This book offers detailed insights into machine learning models and algorithms, including those used for recommendation systems.

9.2 Websites References

[1] TensorFlow Official Documentation

URL: https://www.tensorflow.org/

Referenced for implementing and fine-tuning the ResNet50 model used in the project.

[2] Streamlit Documentation

URL: https://docs.streamlit.io/

Referenced for building the interactive web application to display the recommendation results.

9.3 Technical Publication References

[1] ImageNet Classification with Deep Convolutional Neural Networks" by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton.This publication was crucial in understanding convolutional neural networks, which are the backbone of feature extraction in this system.

10. APPENDICES

Software Used

TensorFlow Used for developing and running the deep learning models, specifically the ResNet50 for feature extraction.

Streamlit Used for creating the web interface of the fashion recommendation system.

Scikit-learn Used for implementing the Nearest Neighbors algorithm to find similar products based on extracted features.

Python Main programming language used throughout the project.

Pandas Used for handling and processing the CSV files containing product descriptions and metadata.

PIL (Python Imaging Library) Utilized for image handling and processing tasks.

Methodologies Used

Feature Extraction Utilized a pre-trained ResNet50 model to extract features from product images.

Similarity Search Employed the Nearest Neighbors algorithm with Euclidean distance to find and recommend similar products.

Web Application Development Streamlit was used to build the front-end user interface for interaction with the recommendation system.

Model Evaluation Used validation datasets to evaluate the effectiveness of the recommendation system based on accuracy metrics and user feedback.

11. PLAGIARISM REPORT

Fig 11.1 Plagiarism Report