Deep Learning Project: AI-Powered Image Similarity Search and Recommendation System

INTEL UNNATI Internship Report

Submitted by

TEAM NAME: INNOVATORS

Name of Student(s)

Ritam Sarkar

Saptarshi Dey

Arnab Samanta

STREAM: CSE BTECH (4^{TH} YEAR}

Conducted at
INTEL®

Department of Computer Science & Engineering,
MCKV Institute of Engineering
243, G.T. Road(N)

Liluah, Howrah - 711204

ACKNOWLEDGEMENT

We would like to express our heartfelt gratitude to **Intel** for providing us with the opportunity to undertake this internship project titled **“Deep Learning Project: AI-Powered Image Similarity Search and Recommendation System.”** This project has been an enriching learning experience that allowed us to explore cutting-edge technologies in computer vision, deep learning, and recommendation engines.

We convey our sincere thanks to our project mentor(s) at Intel for their continuous guidance, along with the constant cooperation of our college’s mentor **Dr. Deep Suman Dev** sir for valuable suggestions, and constructive feedback throughout the course of this work. Their expertise and encouragement greatly enhanced our understanding and helped us apply theoretical knowledge to practical, real-world problem solving.

We are also thankful to our faculty coordinators and institution for their constant support and for enabling us to participate in this internship program. Their encouragement motivated us to explore innovative ideas and maintain high academic and professional standards.

Finally, we extend our appreciation to our teammates, and everyone who directly or indirectly contributed to the successful completion of this project. The collaborative environment and shared enthusiasm made this journey both enjoyable and educationally significant.

TABLE OF CONTENTS

ABSTRACT

This internship project focuses on the development and implementation of a sophisticated gender-aware, multi-modal product recommendation system specifically designed for e-commerce applications. The system combines computer vision and natural language processing to bridge the semantic gap between visual product images and textual queries, enabling intelligent product discovery based on user preferences including gender- specific filtering.

The core architecture leverages MobileNetV2 for visual feature extraction, BERT-based embeddings for semantic text understanding, and triplet loss optimization to create a unified embedding space where semantically similar products are positioned closer together. The system was trained and evaluated on the Fashion-MNIST dataset augmented with e-commerce metadata (pricing, descriptions, gender tags), achieving impressive performance metrics with Top-1 accuracy of 97.00% and Mean Reciprocal Rank of 0.9850 in text-to-image retrieval tasks.

The project demonstrates the practical applicability of deep learning techniques in real world e-commerce scenarios, addressing the need for personalized, intuitive product search that understands both visual similarity and user demographic preferences.

INTRODUCTION

The rapid growth of e-commerce has fundamentally transformed how consumers discover and purchase products online. Traditional search systems rely heavily on keyword matching or basic categorical filtering, which often fails to capture the nuanced relationship between visual similarity and user intent. When a customer encounters a product they like and wants to find similar items—or when they search using textual descriptions like "comfortable blue casual shirt for men"—they expect the system to understand both visual characteristics and contextual preferences.

This challenge becomes even more complex in the fashion and apparel industry, where visual aesthetics, materials, fit, and sizing considerations play crucial roles. Additionally, gender-based product categorization is significant in e-commerce, as clothing, shoes, and accessories are often designed differently for different demographics. A system that can intelligently bridge the gap between these modalities while respecting demographic preferences provides substantial value to both customers and retailers.

The primary challenge addressed by this project is: How can we build a recommendation system that effectively combines visual product features with textual queries while incorporating gender-aware filtering to provide personalized product suggestions.

OBJECTIVES

The primary objectives of this internship project are:

- To design and implement a triplet-loss-based deep neural network for learning discriminative visual embeddings from product images.
- To integrate transformer-based text encoders (BERT) to extract semantic representations from product descriptions and user queries.
- To develop a mapping layer that translates textual embeddings into visual embedding space, enabling cross-modal retrieval.
- To implement gender-aware filtering mechanisms that personalize recommendations based on demographic preferences.
- To evaluate the system across multiple metrics including retrieval accuracy, ranking quality, and user preference alignment.
- To demonstrate practical end-to-end functionality through interactive image-to-product recommendation features.

STUDY FOR THE PROJECT

1.Fashion-MNIST Dataset (Zalando Research)

This paper/dataset introduction explains the reason for the **Fashion-MNIST** was created.

Dataset Used: Fashion-MNIST

- **Details of the dataset:**
  - Keeps the exact same structure as MNIST (grayscale images of size 28×28, total $70{,}000$ samples)
  - Replaces digits with fashion products, making the recognition task more representative of real objects.
  - Organized into 10 balanced classes.
  - Contains $60{,}000$ training images and $10{,}000$ test images..

Accuracy: Models usually achieve around $90\% - 95\%$ accuracy rather than "almost 100%" as on MNIST because of fashion categories have higher intra-class variation and some inter-class similarity (e.g., "shirt" vs "T-shirt/top").

Use in our project: The dataset is useful as a controlled starting point to test classification and similarity learning pipelines quickly before moving to larger, more realistic fashion datasets.

Limitations:

- The images are low-resolution and grayscale.
- The images are relatively clean and standardized.

2. Triplet Loss for Metric Learning (Concept + Mining)

This article focuses on triplet loss as a core method for **metric learning**, where the goal is not to classify an image into a label but to learn an embedding space where similar items are close and dissimilar items are far.

Special Feature: The paper uses the triplet structure: Anchor A, Positive P (same class as anchor), and Negative N (different class). The training objective is to ensure the anchor is closer to the positive than to the negative by at least a margin.

The loss is typically expressed as:

$$L = \max\left(d(A,P) - d(A,N) + m, 0\right)$$

where $d(\cdot)$ is a distance metric (often Euclidean or cosine distance) and m is the margin.

Main focus of the paper: Practical hyperparameters like the embedding size (commonly 128–512), the choice of distance metric and margin impacts convergence and final embedding quality.

Implementations in this paper:

- Hard negatives: negatives closest to the anchor (strong learning signal but can be unstable)
- Semi-hard negatives: negatives farther than positives but still within the margin (often more stable)
- Hard positives: positives that are far from the anchor (helps tighten clusters)

Use in our project: For our fashion similarity project, this paper provides the theoretical base: instead of only predicting "this is a sneaker," the model learns "these two sneakers are close, but sneaker vs bag should be far," which is exactly what a retrieval/recommendation embedding needs.

3. Siamese Network with Triplet Loss

This article is more implementation-oriented and explains how to build a Siamese-style pipeline for triplet loss in practice. The main architecture is a three-branch network (Anchor/Positive/Negative), where all branches share weights. Weight sharing matters because it forces the network to encode all images into the same feature space, making the embeddings directly comparable.

A common design in the paper is to use a pretrained backbone (like ResNet50) and then add a custom embedding head. The backbone acts as a feature extractor, and the embedding head reduces high-dimensional features (e.g., 2048) into a smaller embedding (like 128 or 256). The paper highlights training stability practices such as:

- adding Batch Normalization in the embedding head
- applying L_2 normalization so embeddings have consistent scale
- freezing early layers of the pretrained model initially, then fine-tuning later layers

Evaluation Metrics: The paper emphasizes tracking whether the positive distances decrease and negative distances increase over time. The most meaningful evaluation is a retrieval test: embed a query image and rank database images by distance, then compute metrics like Top-K retrieval accuracy or mean average precision (mAP).

Use in our paper: In our project, this paper is important because it provides a direct blueprint for converting “triplet loss theory” into a working fashion similarity model that can later be plugged into recommendation.

4. Deep Learning Recommender with BPR Loss

This article shifts from pure similarity learning to building a recommendation system optimized for ranking. The central idea is to represent both users and items as dense embeddings and train them so that preferred (interacted) items score higher than non-preferred items.

Main feature of this paper: The key contribution is the use of **Bayesian Personalized Ranking (BPR) loss**, which directly optimizes pairwise ranking. Let a user u , a positive item i (liked/clicked/purchased), and a negative item j (not interacted), the model maximizes the probability that the score of i is higher than j. The paper expresses this as:

$$L = -\log\left(\sigma(s(u,i) - s(u,j))\right)$$

where $\sigma(\cdot)$ is the sigmoid and $s(u,i)$ is commonly a dot product between user and item embeddings.

Evaluation Metrics: It differs from similarity learning. Instead of Top-K nearest neighbors in embedding space only, the paper uses ranking metrics such as Precision@K, Recall@K, and NDCG

Use in our project: In our project, this paper is useful as the “end-to-end recommendation step”: after learning strong visual item embeddings, BPR provides a standard way to train personalized ranking using implicit feedback and produce user-specific fashion recommendations.

DATASET PREPARATION

For our Intel Unnati project, we had to create an e-commerce fashion recommendation system supporting both image queries and text searches. To implement this, we began with the Fashion MNIST dataset-a very common dataset in machine learning. It consists of 70,000 grayscale images of fashion items, such as t-shirts, shoes, bags, and many others.

However, the original dataset consisted of only images with numerical labels, which was far from being sufficient for a real e-commerce platform. It has been turned into a labelled datset with added descriptions and categories.

Fashion MNIST Dataset

The Fashion MNIST dataset is downloaded as a CSV file. Each image is 28x28 pixels-that's 784 pixel values-and each row in the CSV contains these pixel values along with a label from 0 to 9. These numbers represent 10 different categories: T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, and Ankle boot.

I) Extraction and Initial Loading

The CSV file is obtained from Kaggle source and has been analysed, reviewed and updated with labellings for smoother functioning and desirable results.

II) Readable Labels

Numeric labels are mapped 0 through 9, to actual product names. and a simple lookup has been made in which label 0 maps to "T-shirt/top", label 1 maps to "Trouser", and so on. This already made the data much easier to understand and work with.

III)Creation of a Balanced Smaller Dataset

Working with 70,000 images would require an inordinate quantity of computational time and resources, so the dataset has been made smaller yet balanced. It is straight forward sampled to select exactly 500 images from each of the 10 categories, giving a total of 5,000 images. This is done to ensure our model does not get biased toward any particular type of clothing.

IV) Adding E-Commerce Capabilities

To make this dataset perform like a real e-commerce platform, labels and descriptions are added like-size and prices for each products, gender, colour, material, style, occasion ,brand and description of product generation along with short descriptions of the products. It made the dataset much more realistic, distinct and suitable for embedding in commercial purposes.

V) Efficient and reliable use

Important for the text-to-image search feature, a searchable keyword string has been created for each product by combining all its attributes. For example: "sneaker men black canvas lace-up puma casual". Whenever a user searches for "black canvas shoes," the system can match these keywords to show relevant products.

Faced Challenges

- Ensuring attribute combinations made sense- we had to be careful with the random generation not to get unrealistic combinations, like "winter sandals" or "formal sneakers for sports".

  Solution: we made separate attribute lists for each category of products.

- Reducing the size of the dataset while maintaining balance we couldn't simply randomly select 5,000 images because we might end up with 1000 shoes and only 200 bags.
  Solution: Stratified sampling was used to ensure equal representation

- Writing descriptions to match how people really search People do not always use complete sentences or perfect grammar when searching.
  Solution: Created multiple description formats and keyword fields in order to catch different search patterns

MODEL ARCHITECTURE

The Image Similarity search and recommendation system is based on the training of our dataset according to the algorithms involving MobileNetV2, Faiss Index, Triplet Network technique.

Embedding model network trained with MobileNetV2

It is built on an embedding network, which is often used in triplet loss, or similarity learning tasks i.e. mapping inputs (here images) into a compact vector space where similar items are close together and dissimilar ones are made to be far apart for clear discrimination. The embedding dimension value is taken to be 64 i.e. the final output will be a 64-dimensional vector representing the image's embedding. The input shape has been reduced to 96*96 for smooth computation and reducing complexity. In our model, the feature extractor used is a fixed feature extractor, MobileNetV2,a pre-trained CNN model. The basic parameters tuned in this model includes-

- **POOLING TECHNIQUES->**Average Global Pooling to flatten the spatial features into a 1D vector.
- **WEIGHTS->**ImageNet has been used that loads pretrained weights from ImageNet.
- **WIDTH MULTIPLIER->**The parameter alpha is known as the width multiplier that controls how wide the network is i.e how many filters (channels) each convolutional layer has.

If the original MobileNet layer has **N** filters, and if alpha is considered to be as less than 1.0,each layer will use alpha×N filters.

The inputs are defined to be created as the KERAS input layer and is passed through a base network that feeds the input image through MobileNetV2 and outputs a feature vector representing the image content.

Regularisation used here is DropOut having a value of 0.3.

Normalisation used here is Batch Normalisation and the activation function used here is ReLu activation function.

At the final output layer, the 256-dimensional vector is projected down to defined embedding dimension i.e 64. The normalisation used here is L2 Normalisation that scales each embedding vector to have unit length. Finally 64 Dimension embedding network is obtained.

BATCH NORMALISATION: It is a technique in Convolutional Neural Networks (CNNs) that normalizes the activations of a layer by standardizing the inputs to each layer within a mini-batch. This process stabilizes training, allows for higher learning rates.

ReLu ACTIVATION: It is a non-linear function used in neural networks that outputs the input directly if it's positive, and zero otherwise. The expression can be expressed as f(x)=max(0,x).

TRIPLET LOSS NETWORK

It is a discriminative embedding space where images from the same class are close together, and images from different classes are far apart as earlier mentioned but is achieve using the **Triplet Loss** function, trained on **triplets of images**.

The triplet include -> **Anchor Image**: This is a reference image.

Positive Image: This is another image considered from the same class as the anchor image.

Negative Image: It is an image from a different class.

The y_pred used as the parameter that contains concatenated embeddings of these three images. The distances used here is positive_distance i.e squared distance between anchor and positive and the negative_distance i.e squared distance between anchor and negative. The triplet loss formula expression is-

$$L=\max(\|A-P\|^2-\|A-N\|^2+margin,0).$$

The mean of all triplet losses in the batch is taken for optimization.

TRIPLET GENERATOR

The generator dynamically creates triplets for each batch.

Working of Triplet Generator:
It randomly selects a class and randomly pick two images from that class → ***Anchor* and *Positive*.**

It randomly pick a different class and one image from it → ***Negative*.**

It returns three NumPy arrays (anchor, positive, negative) and dummy labels (zeros), because the loss does not use y_true.
This helps the model learn on varied triplets in each epoch using efficient memory utilisation by not storing in the memory and dynamically generates batches of (anchor, positive, negative) samples.

TRIPLET NETWORK

This ensures the use of embedding network(MobileNetV2 along with the dense layers) to encode images into embeddings. The same **embedding network** (shared weights) processes all these three inputs(anchor,positive_image,negative_image).These embeddings are then concatenated into a single tensor. The returned model outputs all three embeddings.

MODEL TRAINING

The model has been trained for 10 epochs along with batch_size 16. **Adam optimizer** is used for efficient gradient updates along with the **custom triplet loss** (with margin 0.5) that drives the embeddings apart or together. Then the Triplet Generator generates the batches of triplet samples.

FAISS INDEX

The FAISS index serves as a database of known embeddings that can be queried to find the most similar images for a given input. Facebook AI Similarity Search is an open-source library that efficiently indexes and searches large collections of high-dimensional vectors for similarity tasks, commonly used in deep learning and AI applications. The working of this FAISS INDEX depends on Initialisation, Training, Adding Data ,Querying and Tuning. In our model, After training, the embedding model can transform each image into a feature vector (embedding).These embeddings are stored in a **FAISS index**, which allows a faster similarity search using distance metrics -Euclidean distance.

Text-to-Image Model Architecture

The main aim of architecture is to map text embeddings along with image embeddings to find similar text-image pair such that it will receive the query input as text and classify the images accordingly from our trained dataset. The networks used here in this part for text to image generation are-

Image Embedding Model

The base model used here is MobileNetV2 with alpha=0.75 to extract compact, discriminative feature vectors from product images. The additional layers used here is Dense of 256 dimensions with ReLU activation function along with Batch Normalisation and dropout regularisation of 0.3.The L2 normalisation layer is also used for training purpose. Then this model is trained with Triplet Loss ensuring that Distance(anchor, positive) < Distance(anchor, negative).

Text Embedding Model

The model used here is sentence Transformer mainly a transformer based encoder with a dimension of 384 to encode and embed the text for future mapping purpose. It captures contextual meaning and linguistic semantics along with normalized embeddings to ensure stable similarity metrics.

Text-to-Image Network

Once both modalities are represented numerically (text: 384-dim, image: 64-dim), a mapping network is trained to project text embeddings into the image embedding space.

Loss Function used: Mean Squared Error(MSE) to minimize the Euclidean distance between the predicted image embedding (from text) and the true image embedding (from the paired product image)

Optimizer Used: Adam

Regularizer used: Early Stopping to prevent overfitting with a patience value of 3.

WORKING PRINCIPLE OF OUR TEXT-TO-IMAGE MODEL

- A given text query is encoded using the Sentence Transformer that produces a text embedding.
- This embedding is passed through the mapping layer, generating a predicted image embedding.
- The mapped embedding is used to query a FAISS index containing precomputed embeddings of all training images.
- FAISS performs nearest-neighbour search (L2 distance) to find the most visually and semantically similar products.

Additional Smoothing: The recommendation logic applies strict gender and category filters to prevent irrelevant matches.

IMAGE-TO-IMAGE SIMILIARITY AND RECOMMENDATION SYSTEM

The final deployment stage of our problem statement is based on image to image similarity search and recommendation system to perform visual similarity-based product recommendations such that when a user uploads an query image ,the system identifies visually similar products from the training dataset using the learned image embeddings and refer for user recommendations.

WORKING PRINCIPLE OF OUR IMAGE-TO-IMAGE MODELS

- The Embedding model used on the basis of a Triplet Network that learns to map visually similar items closer in embedding space and outputs a compact vector representation (e.g., 64-dimensional) for each image.

- A Classifier model has been set up in which the embedding model has been extended by tuning certain parameters like- a Dense layer (128 neurons, ReLU activation),Dropout (0.3) for regularization, Final Dense layer with softmax activation to classify product categories (e.g., T-shirt, Dress, Bag). **Sparse categorical cross entropy** loss is used as a loss function.

- The FAISS (Facebook AI Similarity Search) index stores all image embeddings from the training set and enables fast nearest neighbour search using L2 distance to retrieve top-k most similar embeddings.

- **Results filtering and extraction:** Here two modes of representing the image to image search has been done one based on simple similarity search and the other with gender filtering mode awareness in which the received are adjusted based on gender filter to give exact images for accurate recommendation.

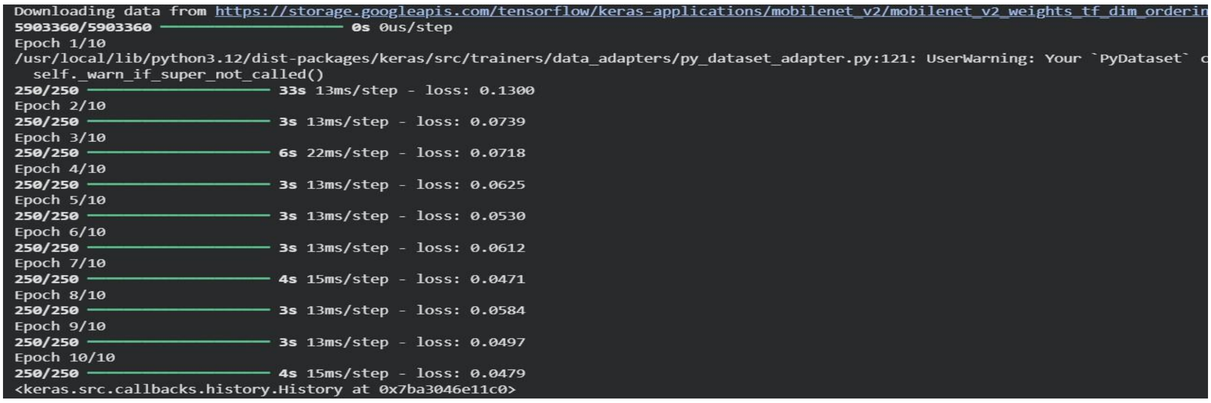
The top-k recommended images are displayed side by side,

showing: Category, Price, Gender, Product Description, Similarity Score using the labelled dataset for detailed product information printed for all recommendations.

RESULTS AND DISCUSSIONS

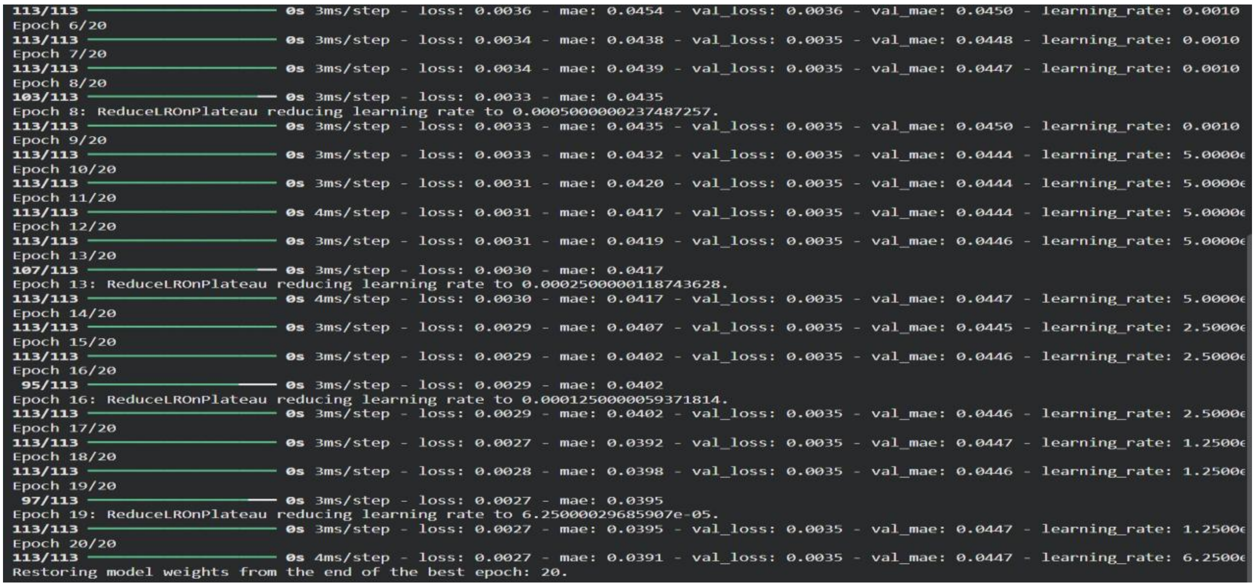
As we have already mentioned we have taken fashion-mnist as our dataset but as the whole dataset is very large (almost 70000 images from 10 classes) and we doesn't have so much resource support for the code execution so we have chosen total 5000 images from 10 classes (500 images each) and also added some extra columns to improve our model and make the dataset convenient for the model.

1.Epoch training on training set: To train the CNN model with our images, we have to train our architecture with all the training images (4000 images). To get a decent model we have to check for the loss values, if they are less then the training is absolutely fine. In our case the loss is not too much less but it can work as decent model as you can see.



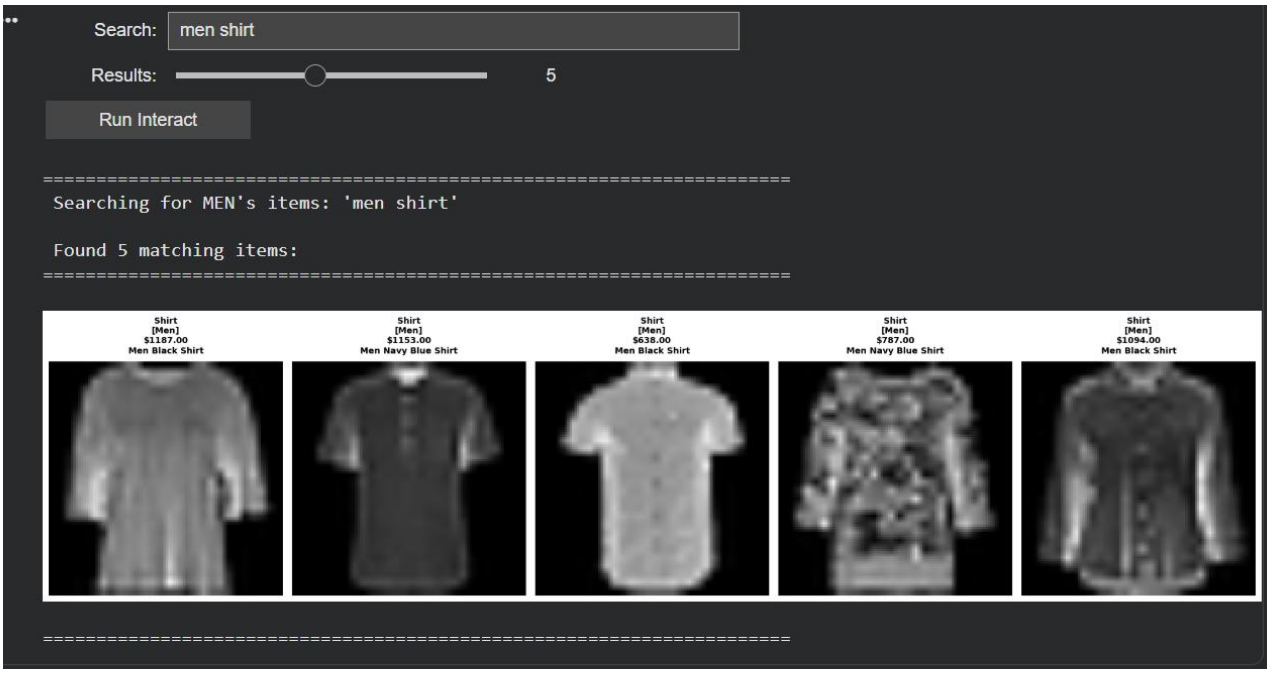
```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_orderin
5903360/5903360 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Epoch 1/10
/usr/local/lib/python3.12/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` c
  self._warn_if_super_not_called()
250/250 ━━━━━━━━━━━━━━━━━━━━ 33s 13ms/step - loss: 0.1300
Epoch 2/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 0.0739
Epoch 3/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 6s 22ms/step - loss: 0.0718
Epoch 4/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 0.0625
Epoch 5/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 0.0530
Epoch 6/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 0.0612
Epoch 7/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 4s 15ms/step - loss: 0.0471
Epoch 8/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 0.0584
Epoch 9/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 0.0497
Epoch 10/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 4s 15ms/step - loss: 0.0479
<keras.src.callbacks.history.History at 0x7ba3046e11c0>
```

2.Making the text model and training for the text-to-image model: As we have added some extra columns for the text-to-image recommendation system, so we have to make a text model (vocabulary) to store all the relevant search keywords that someone can enter to search some product. So for that, we have to make an embedding model itself and train them with proper parameters and vocabular



```
113/113 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0036 - mae: 0.0454 - val_loss: 0.0036 - val_mae: 0.0450 - learning_rate: 0.0010
Epoch 6/20
113/113 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0034 - mae: 0.0438 - val_loss: 0.0035 - val_mae: 0.0448 - learning_rate: 0.0010
Epoch 7/20
113/113 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0034 - mae: 0.0439 - val_loss: 0.0035 - val_mae: 0.0447 - learning_rate: 0.0010
Epoch 8/20
103/113 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0033 - mae: 0.0435
Epoch 8: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
113/113 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0033 - mae: 0.0435 - val_loss: 0.0035 - val_mae: 0.0450 - learning_rate: 0.0010
Epoch 9/20
113/113 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0033 - mae: 0.0432 - val_loss: 0.0035 - val_mae: 0.0444 - learning_rate: 5.0000e
Epoch 10/20
113/113 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0031 - mae: 0.0420 - val_loss: 0.0035 - val_mae: 0.0444 - learning_rate: 5.0000e
Epoch 11/20
113/113 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0031 - mae: 0.0417 - val_loss: 0.0035 - val_mae: 0.0444 - learning_rate: 5.0000e
Epoch 12/20
113/113 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0031 - mae: 0.0419 - val_loss: 0.0035 - val_mae: 0.0446 - learning_rate: 5.0000e
Epoch 13/20
107/113 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0030 - mae: 0.0417
Epoch 13: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
113/113 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0030 - mae: 0.0417 - val_loss: 0.0035 - val_mae: 0.0447 - learning_rate: 5.0000e
Epoch 14/20
113/113 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0029 - mae: 0.0407 - val_loss: 0.0035 - val_mae: 0.0445 - learning_rate: 2.5000e
Epoch 15/20
113/113 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0029 - mae: 0.0402 - val_loss: 0.0035 - val_mae: 0.0446 - learning_rate: 2.5000e
Epoch 16/20
 95/113 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0029 - mae: 0.0402
Epoch 16: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
113/113 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0029 - mae: 0.0402 - val_loss: 0.0035 - val_mae: 0.0446 - learning_rate: 2.5000e
Epoch 17/20
113/113 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0027 - mae: 0.0392 - val_loss: 0.0035 - val_mae: 0.0447 - learning_rate: 1.2500e
Epoch 18/20
113/113 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0028 - mae: 0.0398 - val_loss: 0.0035 - val_mae: 0.0446 - learning_rate: 1.2500e
Epoch 19/20
 97/113 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0027 - mae: 0.0395
Epoch 19: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-05.
113/113 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0027 - mae: 0.0395 - val_loss: 0.0035 - val_mae: 0.0447 - learning_rate: 1.2500e
Epoch 20/20
113/113 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0027 - mae: 0.0391 - val_loss: 0.0035 - val_mae: 0.0447 - learning_rate: 6.2500e
Restoring model weights from the end of the best epoch: 20.
```

3.Run the text-to-image model: As in here, you just enter a search keyword to search specific search result based on the search keywords added to the dataset. Here string matching is also applied to recommend the products by matching it with its search keyword. Also you can choose how many results you can see in recommendations and then it will show that many results along with the details (like price, name, category etc.).


Search: men shirt

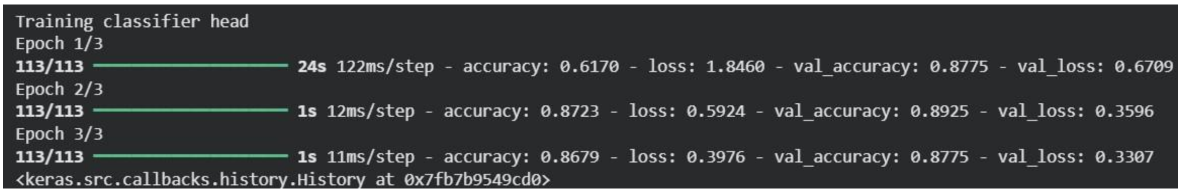
Results: 5

```
==================================================================
 Searching for MEN's items: 'men shirt'

 Found 5 matching items:
==================================================================
```

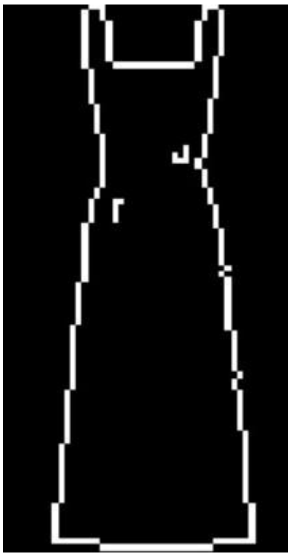
```
==================================================================
```

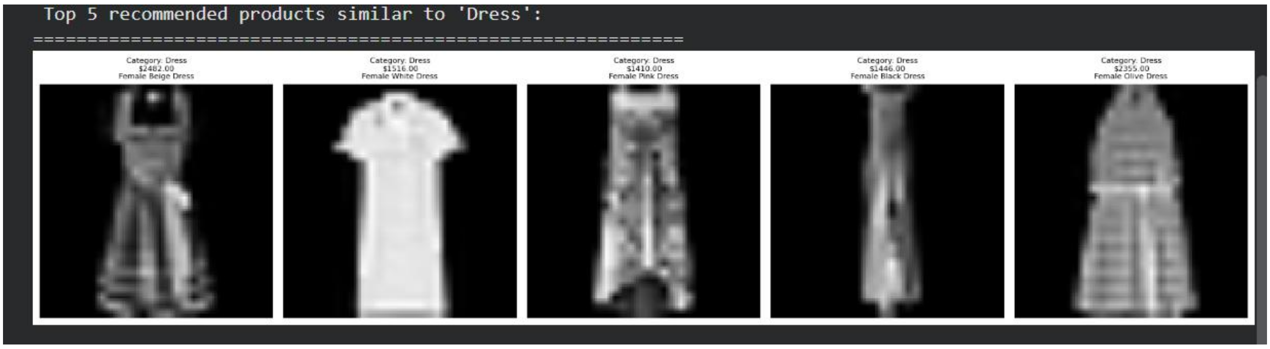
4.Train the classifier head: Now the custom classifier head that is solely made for this model according to the dataset, is then trained on the training set. If the accuracy is fine, then the model should recommend the images decently.



```
Training classifier head
Epoch 1/3
113/113 ━━━━━━━━━━━━━━━━━━━━ 24s 122ms/step - accuracy: 0.6170 - loss: 1.8460 - val_accuracy: 0.8775 - val_loss: 0.6709
Epoch 2/3
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.8723 - loss: 0.5924 - val_accuracy: 0.8925 - val_loss: 0.3596
Epoch 3/3
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.8679 - loss: 0.3976 - val_accuracy: 0.8775 - val_loss: 0.3307
<keras.src.callbacks.history.History at 0x7fb7b9549cd0>
```

5.Image-to-Image recommendation (generalized): Now after all this things the model should good to go to recommend some similar images. So an query image is uploaded through google colab's file uploading feature and then the edge detection, class detection is done, after which top 5 images with similar edge and shape is recommended by the model as seen in the image below.

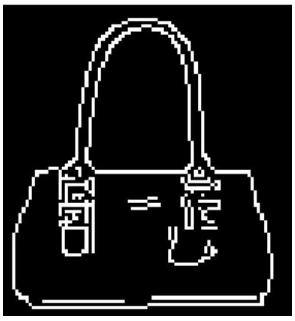


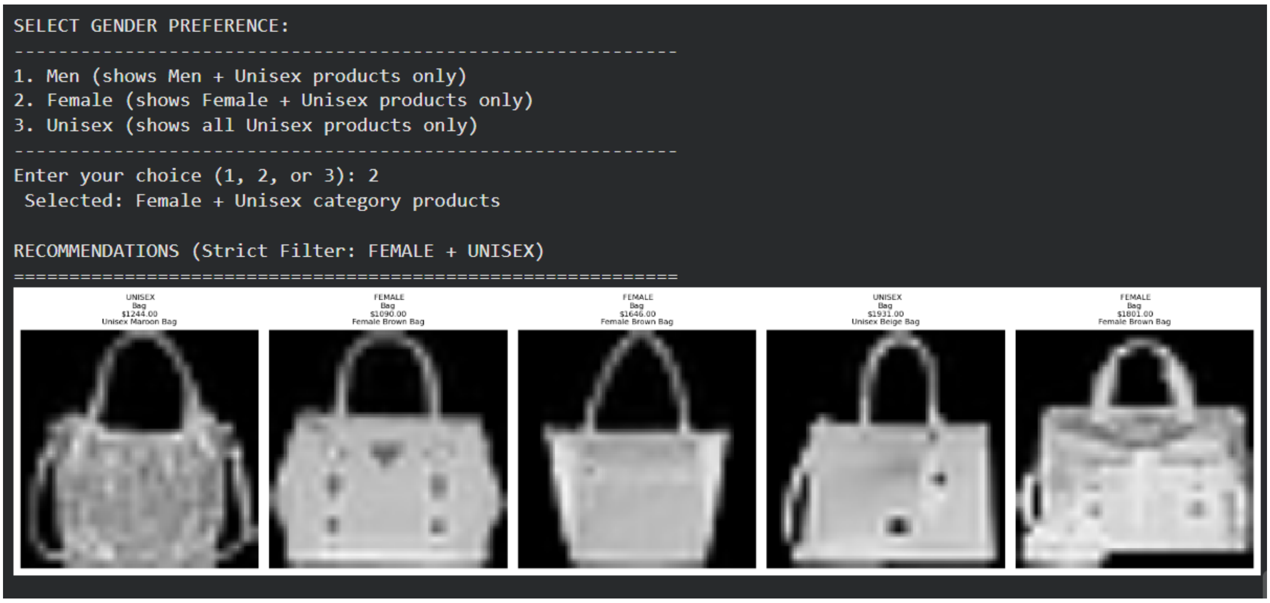


```
Top 5 recommended products similar to 'Dress':
============================================================
Category: Dress      Category: Dress      Category: Dress      Category: Dress      Category: Dress
$2482.00             $1516.00             $1410.00             $1446.00             $2355.00
Female Beige Dress   Female White Dress   Female Pink Dress    Female Black Dress   Female Olive Dress
```

6.Image-to-Image recommendation (gender specific): Along with the generalized image-to-image recommendation, we have added one extra feature here, that is the gender specific recommendation system, where after uploading the image user can easily choose between general or gender specific recommendation, and furthermore also choose what he/she wants to see, male, female or unisex products. According to the user input, products will be shown.

For Female:

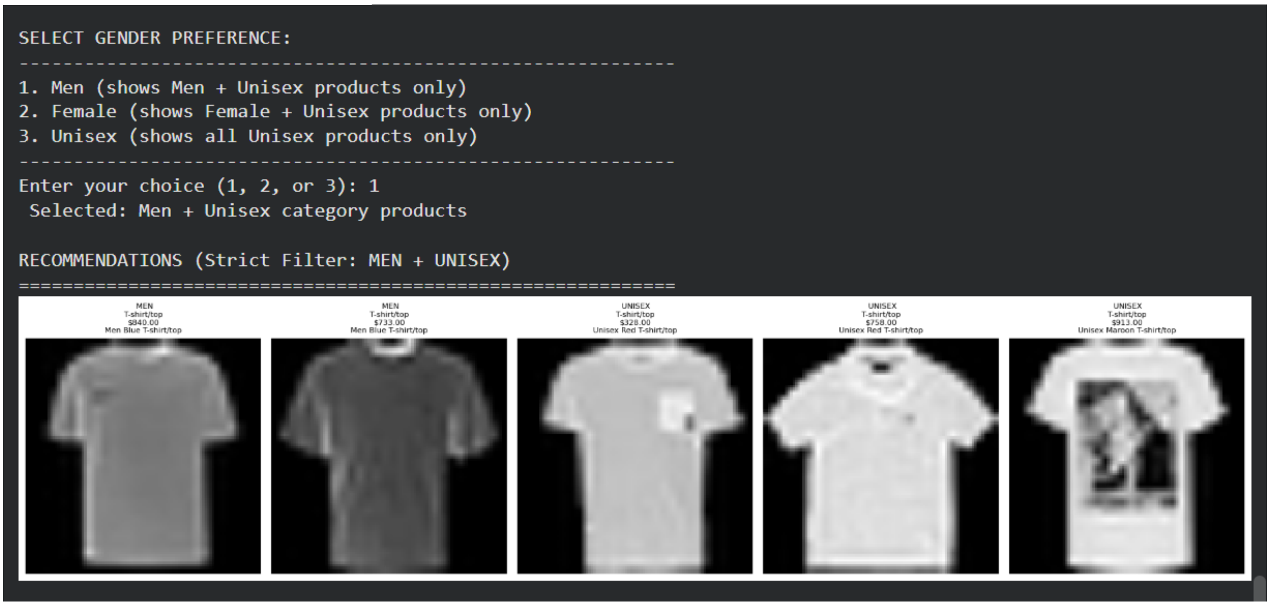




```
SELECT GENDER PREFERENCE:
------------------------------------------------------------
1. Men (shows Men + Unisex products only)
2. Female (shows Female + Unisex products only)
3. Unisex (shows all Unisex products only)
------------------------------------------------------------
Enter your choice (1, 2, or 3): 2
 Selected: Female + Unisex category products

RECOMMENDATIONS (Strict Filter: FEMALE + UNISEX)
============================================================
```

For Male:

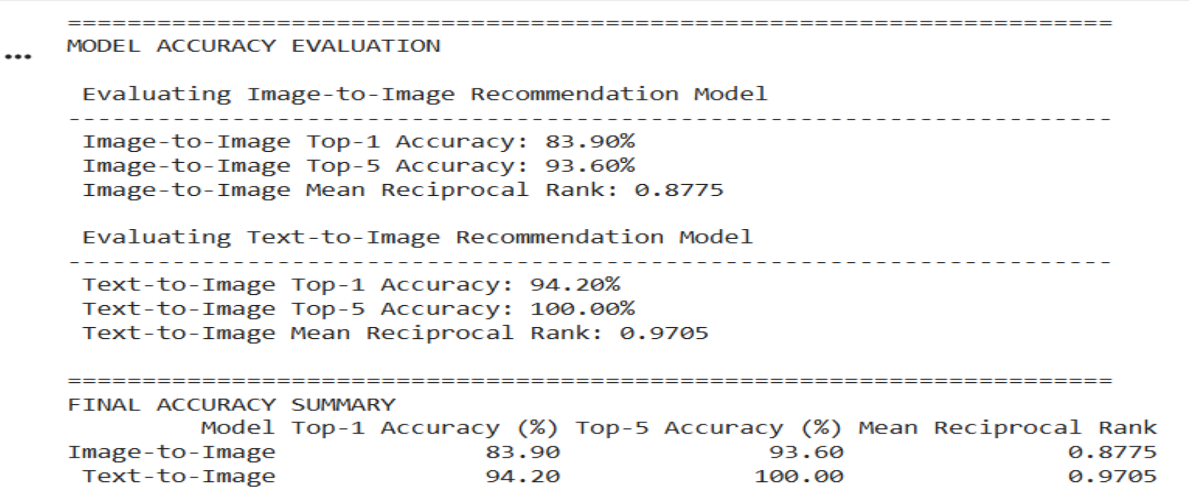


```
SELECT GENDER PREFERENCE:
------------------------------------------------------------
1. Men (shows Men + Unisex products only)
2. Female (shows Female + Unisex products only)
3. Unisex (shows all Unisex products only)
------------------------------------------------------------
Enter your choice (1, 2, or 3): 1
 Selected: Men + Unisex category products

RECOMMENDATIONS (Strict Filter: MEN + UNISEX)
============================================================
```

7. Final Model Accuracy comparison and Visualization of text-to-image and image-to-image recommendation:

So after all this, we just finally visualize a graphical structure of the accuracy for both the model and also the Mean Reciprocal Rank (MRR) comparison. So MRR is actually a metric which is actually used to determine whether a model can rank the best search results at the top of recommendation. Higher MRR means better search results. So here is the visualization of the model.

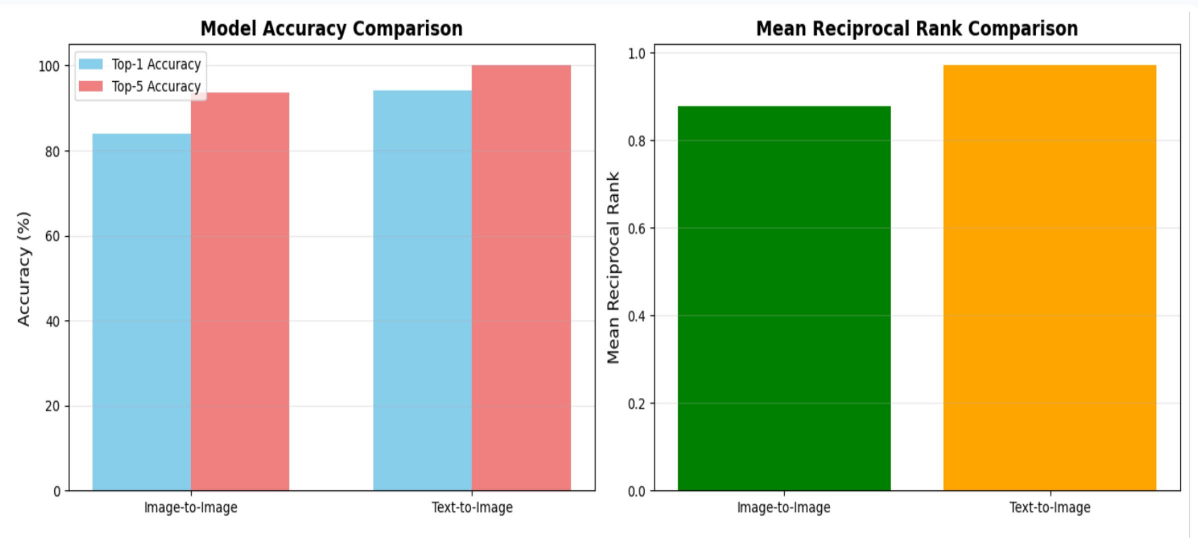


```
    ======================================================================
... MODEL ACCURACY EVALUATION

     Evaluating Image-to-Image Recommendation Model
    ----------------------------------------------------------------------
     Image-to-Image Top-1 Accuracy: 83.90%
     Image-to-Image Top-5 Accuracy: 93.60%
     Image-to-Image Mean Reciprocal Rank: 0.8775

     Evaluating Text-to-Image Recommendation Model
    ----------------------------------------------------------------------
     Text-to-Image Top-1 Accuracy: 94.20%
     Text-to-Image Top-5 Accuracy: 100.00%
     Text-to-Image Mean Reciprocal Rank: 0.9705

    ======================================================================
    FINAL ACCURACY SUMMARY
             Model  Top-1 Accuracy (%)  Top-5 Accuracy (%)  Mean Reciprocal Rank
    Image-to-Image               83.90               93.60                0.8775
     Text-to-Image               94.20              100.00                0.9705
```



FUTURE SCOPE

As this is a very unique model, and trained with decent amount of images, so it is working fine, but still have some flaws for which we can’t call it as a fully complete model, which can be improved in future.

1. **Dataset with limited images:** for this project as a dataset we have chose the fashion-mnist dataset which consist of greyscale and very low quality images as we have limited resource, but to further improve the model we can use a dataset with millions of RGB and high quality images to make the model more robust.
2. **Custom CNN model:** If there are a large amount of images present in the dataset, then we don’t need any transfer learning concept of the pretrained model to work properly and we can also train our own custom CNN model with that dataset.
3. **Test case failures:** Though In most of the cases, the recommend images are pretty much correct, but for some cases where some lighting factor or some other edges detected inside the picture, the model started to provide wrong output. This is due to the simplicity of the classifier head which is trained on the simple dataset. Training on complex dataset can solve this problem.

REFERENCES

1. https://www.kaggle.com/datasets/zalando-research/fashionmnist
2. **https://medium.com/@jyotidabass/triplet-loss-demystified-a-beginnersguide-to-image-matching-with-batch-tricks-ff0aa9ee43d3**
3. **https://elcaiseri.medium.com/image-similarity-estimation-using-asiamese-network-with-triplet-loss-a-practical-guide-124938e24b3a**
4. **https://medium.com/deep-recommender-system/a-deep-recommendersystem-e2b765d27350**