

# Problem Statement – 6

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

### Problem Description:

In today's digital world, we are surrounded by millions of images across various platforms - social media, e-commerce websites, photo galleries, and digital libraries. Users often struggle to find visually similar images or get relevant recommendations based on image content. Traditional text-based search methods fail to capture the visual semantics and similarity between images.

**Challenge:** How can we automatically identify and recommend visually similar images from a large database without relying on manual tags or text descriptions?

### **Real-world Examples:**

- E-commerce: "Find similar products" feature on shopping websites
- Social media: Instagram's "Related Posts" recommendations
- Stock Photography: Getty Images' visual search functionality
- Fashion: "Shop the Look" features on fashion websites

### **Project Objective:**

Develop an intelligent image similarity search system that can:

1. Learn visual patterns and semantic relationships between images
2. Recommend similar images based on visual content alone
3. Handle large-scale databases efficiently (1000+ images)
4. Provide real-time responses for user queries

### **Technical Challenge:**

Build a deep learning model using Triplet Networks that learns to map visually similar images closer together in an embedding space, enabling fast and accurate similarity search.

# 1. Introduction:

In the modern digital ecosystem, users interact with massive collections of images across e-commerce platforms, social media, fashion catalogs, and digital libraries. Traditional text-based search systems are insufficient for retrieving visually similar images because they rely heavily on manually assigned tags and metadata, which are often incomplete or inconsistent.

This project focuses on building an AI-powered image similarity search and recommendation system that retrieves visually similar images based solely on image content. The system leverages deep learning, metric learning using Triplet Networks, and FAISS (Facebook AI Similarity Search) to achieve efficient and accurate similarity retrieval.

# 2. Dataset Description:

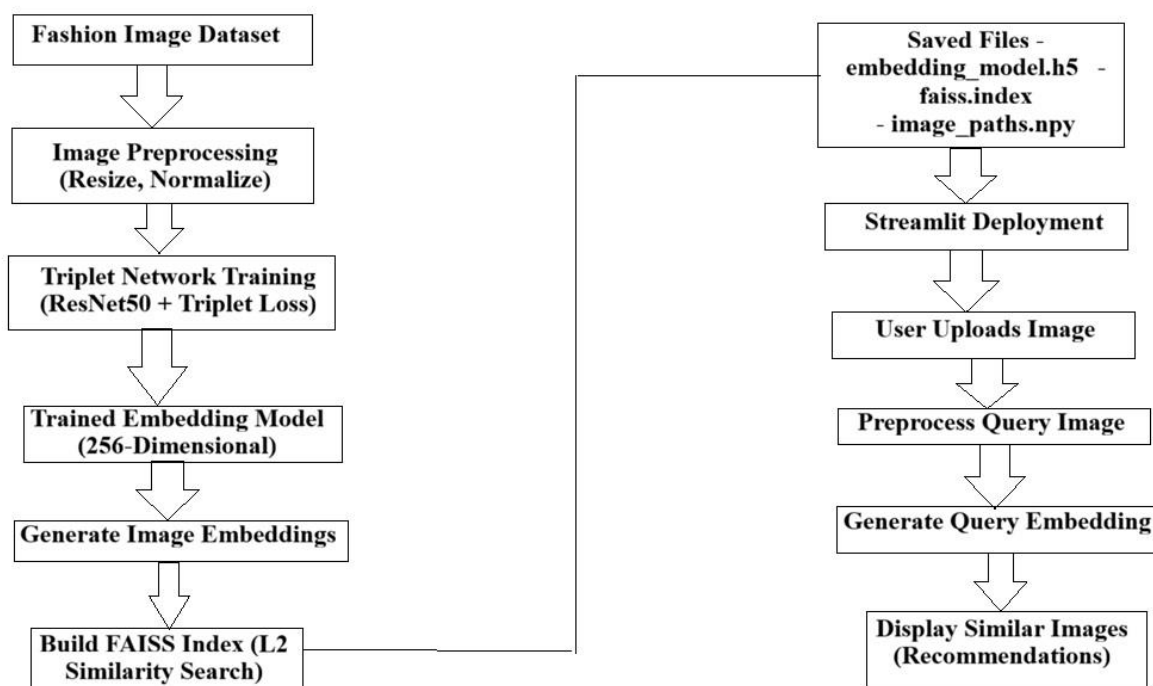
The project uses a fashion product image dataset, containing thousands of clothing and accessory images. Each image belongs to a specific category (article type), which is used during training to generate meaningful triplets.

Dataset Characteristics:

- RGB images
- Diverse fashion categories
- Large-scale dataset (44,000+ images available)

For deployment efficiency, a representative subset of 5,000 images was used for embedding generation and indexing.

# 3. Flowchart of work:



## 4. Methodology:

### 4.1 Image Preprocessing

Each image undergoes the following preprocessing steps:

- Resize to  $224 \times 224$  pixels
- Convert BGR to RGB format
- Normalize pixel values to  $[0, 1]$

This ensures compatibility with the deep learning model and consistency between training and inference.

### 4.2 Deep Learning Model Architecture

A Triplet Network is used to learn an embedding space where visually similar images are close and dissimilar images are far apart.

Base Model:

- ResNet50 (pretrained on ImageNet)
- Top layers removed

Embedding Head:

- Global Average Pooling
- Fully Connected Layer (512 units, ReLU)
- Fully Connected Layer (256 units)
- L2 Normalization

The final output is a 256-dimensional embedding vector.

---

### 4.3 Triplet Loss

The Triplet Loss function ensures:

- $\text{Distance}(\text{anchor}, \text{positive}) < \text{Distance}(\text{anchor}, \text{negative})$

Mathematically:

$$\text{Loss} = \max (||A - P||^2 - ||A - N||^2 + \text{margin}, 0)$$

This encourages better separation between similar and dissimilar images in the embedding space.

---

## 5. Algorithm Used:

### Algorithm 1: Triplet Network Training

#### Objective

To learn a deep embedding function that places visually similar images closer together and dissimilar images farther apart in the embedding space.

#### Triplet Network with Triplet Loss

#### Pseudo-Code

Input: Image dataset D with labels

Output: Trained embedding model  $f(x)$

1. Initialize CNN backbone (ResNet50) with ImageNet weights

2. Add:

- Global Average Pooling
- Dense (512, ReLU)
- Dense (256)

3. Apply L2 normalization to output embeddings

4. For each training epoch:

- a. Randomly select anchor image A
- b. Select positive image P with same class as A
- c. Select negative image N with different class
- d. Compute embeddings:

$$e_a = f(A)$$

$$e_p = f(P)$$

$$e_n = f(N)$$

e. Compute triplet loss:

$$L = \max (\|e_a - e_p\|^2 - \|e_a - e_n\|^2 + \text{margin}, 0)$$

f. Backpropagate loss and update weights

5. Save trained embedding model

### Algorithm 2: Embedding Generation

#### Objective

Convert all dataset images into fixed-length numerical vectors for fast similarity search.

## Pseudo-Code

Input: Trained embedding model  $f(x)$ , dataset images  $I$

Output: Embedding matrix  $E$ , image path list  $P$

1. Initialize empty lists  $E$  and  $P$
2. For each image  $img$  in dataset:
  - a. Resize image to  $224 \times 224$
  - b. Normalize pixel values
  - c. Compute embedding  $e = f(img)$
  - d. Append  $e$  to  $E$
  - e. Append image path to  $P$
3. Save  $E$  as `embeddings.npy`
4. Save  $P$  as `image_paths.npy`

## Algorithm 3: FAISS Index Construction

### Objective

Enable efficient and scalable nearest-neighbour search over large embedding datasets.

## Pseudo-Code

Input: Embedding matrix  $E$  ( $N \times 256$ )

Output: FAISS index

1. Initialize FAISS index using L2 distance
2. Train FAISS index on embeddings  $E$
3. Add embeddings  $E$  to the index
4. Save index to disk as `faiss.index`

## Algorithm 4: Image Similarity Search

### Objective

Retrieve the top-K visually similar images for a given query image.

## Pseudo-Code

Input: Query image  $Q$ , embedding model  $f(x)$ , FAISS index

Output: Top-K similar images

1. Preprocess query image  $Q$
2. Compute query embedding  $e_q = f(Q)$
3. Perform FAISS search:

*$distances, indices = index.search(e_q, K)$*

4. Retrieve image paths using indices
5. Display query image and top-K similar images

### **Algorithm 5: Streamlit Deployment Workflow**

#### **Objective**

Provide a real-time interactive interface for image similarity search.

#### **Pseudo-Code**

1. Load trained embedding model (.h5)
2. Load FAISS index (faiss.index)
3. Load image\_paths.npy
4. Launch Streamlit UI
5. Accept user-uploaded image
6. Generate embedding for uploaded image
7. Perform FAISS similarity search
8. Display query image and recommended images

### **6. Embedding Generation**

After training, the embedding model is used to generate embeddings for all selected images.

Steps:

1. Load trained embedding model weights (embedding\_model.h5).
2. Preprocess each image.
3. Generate a 256-dimensional embedding.
4. Apply L2 normalization.
5. Store embeddings and corresponding image paths.

The embeddings are saved as:

- embeddings.npy
- image\_paths.npy

## **7. Similarity Search using FAISS:**

To enable fast similarity search, FAISS (Facebook AI Similarity Search) is used.

### **7.1 FAISS Index**

- Index type: IndexFlatL2
- Distance metric: Euclidean (L2)
- Embedding dimension: 256

Each embedding is added to the FAISS index in the same order as `image_paths.npy`, ensuring a one-to-one mapping.

The index is saved as:

- `Faiss.index`

## 8. Evaluation Metrics:

The system was evaluated using standard information retrieval metrics:

- Precision@5: 0.6080
- Recall@5: 0.0039
- Mean Average Precision (mAP): 0.6929
- Top-5 Accuracy: 1.0000

These results indicate that the system retrieves highly relevant images within the top recommendations.

---

## 9. Deployment using Streamlit:

An interactive web application was built using Streamlit to demonstrate real-time image similarity search.

### Features:

- Upload an image
- Generate query embedding
- Perform FAISS similarity search
- Display top-5 visually similar images

### Advantages:

- Simple and interactive UI
  - Fast response time
  - Easy local deployment
- 

## 10. System Workflow:

1. User uploads an image via Streamlit UI.
2. Image is preprocessed and passed to the embedding model.
3. A normalized embedding vector is generated.
4. FAISS retrieves the top-K nearest neighbors.
5. Corresponding images are displayed as recommendations.

---

## 11. Performance Optimization:

To reduce computation time and ensure smooth deployment:

- A subset of 5,000 images was used.
- Embeddings were precomputed offline.
- FAISS enables constant-time similarity search.

This approach balances accuracy, speed, and resource usage.

---

## 12. Tools and Technologies Used:

- Programming Language: Python
  - Deep Learning Framework: TensorFlow / Keras
  - CNN Architecture: ResNet50
  - Similarity Search: FAISS
  - Web Framework: Streamlit
  - Libraries: NumPy, OpenCV, PIL
- 

## 13. Conclusion:

This project successfully demonstrates an AI-powered image similarity search system capable of retrieving visually similar images without textual metadata. By combining deep metric learning with FAISS and Streamlit, the system achieves both accuracy and real-time performance.

The approach is scalable, efficient, and directly applicable to real-world use cases such as e-commerce recommendations and visual search engines.

---

## 14. Future Scope:

- Scale to millions of images using FAISS IVF or HNSW indexes
- Deploy on cloud platforms (Streamlit Cloud / AWS / GCP)
- Use lightweight models (MobileNet) for faster inference
- Incorporate multimodal (image + text) search