Visual Search for Marketplace

On-device visual retrieval for e-commerce clothing search

PROBLE

Problem

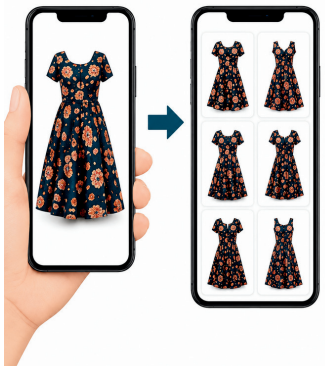
- Marketplaces usually support text search, not image search.
- Clothing is a high-value category where users want to search from a photo.
- Retrieval must distinguish not only object type, but also print, texture, and color.
- Catalogs are large and contain many categories and subcategories.

Goal

- Add image-based search for marketplace products.
- Return the most visually similar items, especially for clothing.
- Build a compact on-device demo for the course project.

Key challenge

- CLIP / SigLIP capture semantic similarity, but may miss fine-grained pattern differences.
- A real marketplace also needs category-aware retrieval and efficient search.



OFFLINE CATALOG PREPROCESSING

1. **Catalog product image** — e.g. 480x640 RGB
2. **Resize for segmentation** — 320x320 input
3. **Background removal** — SAM2 only
4. **Precise object mask** — prompt-based precise mask
5. **Cropped product image** — foreground only / RGBA PNG
6. **Semantic embedding** — Fashion SigLIP for dresses / t-shirts / jeans SigLIP2 for watches — 224×224 → 768-D vector
7. **Texture embedding** — DINOv2 pooled feature
8. **HSV color signature** — foreground-only hue histogram
9. **Store local files** — catalog.json, embeddings.bin, texture.bin, color_signatures.bin, category_prototypes.bin

ⓘ *SAM 2 is used offline for higher-quality catalog preprocessing.*

ON-DEVICE RUNTIME PIPELINE

1. **User photo**
2. **Prepare input** — resize 320x320
3. **U2Netp encoder-decoder** — fast saliency map
4. **Foreground mask** — threshold saliency map > 0.5
5. **Background removal**
6. **Category router: image tower** — RGB bitmap → RGB, resize, norm, transpose, batch dim. — query image embedding 224x224 → 768-D
7. **Text tower** — (runs once at app start) 4 categories x 4 prompts — process 16 phrases, save results
8. **12 self-attention layers** — Q K V — Transformer (12 layers) — process text embeddings
9. **CLS token / pooled output** — strongly colored by meaning of category — → 4 prototypes
10. **4 category prototypes** — L2-normalized category vectors — dress, t-shirt, jeans, watches
11. **Category prediction** — cosine similarity: query image vector vs saved prototypes → top categories — 1st, 2nd, 3rd, 4th
12. **Filter catalog** — search only relevant subset
13. **Category-specific semantic embedding** — choose model by category — dresses/jeans/T-shirts → Fashion SigLIP; watches → SigLIP2 — export to ONNX int8 ~370 MB → ~90 MB
14. **Semantic embedding output** — 768-D dense representation of entire image — each number encodes a combination of many learned features
15. **Cosine retrieval** — Compare query vector with catalog vectors — Higher cosine score = more similar — Query
16. **Top-50 candidates**
17. **Texture similarity** — DINOv2 (self-supervised) — capture textures, prints, fabric patterns
18. **Color similarity** — HSV histogram — H — dominant hue; histogram intersection jaccard
19. **Weighted reranking** — Final score = 0.45 semantic + 0.15 texture + 0.40 color
20. **Top-10 products** — Android UI product cards

Runtime uses precomputed local files from Section A (embeddings.bin, texture.bin, color_signatures.bin, category_prototypes.bin, catalog.json).

Why U2Netp on-device?

- small, fast, prompt-free
- good trade-off for mobile inference
- approx. 4-5 MB class model footprint

Why SAM 2 offline?

- higher-quality masks
- heavier model, better for preprocessing than mobile runtime

Category router details

- text tower runs once during app start
- processes 4x4 phrases
- stores 4 category prototypes
- image tower runs for each user photo
- category via cosine similarity to prototypes

Why DINOv2 for texture?

- self-supervised learning
- good for textures, prints, fabric patterns
- helps distinguish floral vs polka-dot vs solid

Why reranking?

- semantic embeddings find similar items in general
- texture and color recover details left by generic embeddings

RESULT

Output

- Top-10 visually similar products.
- Returned as product cards.
- Designed as an Android demo

Why on-device?

- Demonstrates a compact mobile computer vision pipeline.
- Useful for prototyping and showing end-to-end functionality.
- Highlights latency and resource constraints.

Production note

- Real marketplace deployment would likely use server-side retrieval.
- Large catalogs require scalable indexing, server-side vector retrieval, and stronger infrastructure.
- The on-device version is a research demo, not the final production architecture.

Student: Iuliia Medvednikova
Course: Computer Vision
Project: Visual Search for Marketplace
Supervisor: Prof. Francis Engelmann
Teaching Assistant: Nicolai Hermann
USI Master's in Informatics
May 2026
Contact: iuliia.medvednikova@usi.ch