Visual Search Engine using CLIP and FAISS

Introduction

This project demonstrates how to build a Visual Search Engine using CLIP (Contrastive Language–Image Pretraining) from OpenAI and FAISS (Facebook AI Similarity Search) for fast similarity search. The system allows users to input either an image or a text query and retrieve the most semantically similar images from a given dataset.

By leveraging CLIP, which jointly embeds images and text into the same vector space, and FAISS, which enables efficient nearest neighbor search in large-scale vector spaces, this search engine achieves flexible and powerful multimodal retrieval.

Step-by-Step Implementation

☑ Step 1: Download the Dataset

A dataset of images is required for indexing and searching. The notebook starts by downloading or loading an existing dataset, such as a collection of bicycle images.

☑ Step 2: Extract Dataset (if needed)

If the dataset is provided in a compressed format (e.g., .zip), it is extracted into a directory where each image can be accessed and processed individually.

☑ Step 3: Load the CLIP Model

The CLIP model is loaded using the CLIP package. The model can be run on either CPU or GPU.

Model Device Handling:

```
if device == "cuda":
   model = model.eval().to(device).half()  # Use FP16 precision for GPU
else:
   model = model.eval().to(device)        # Use FP32 for CPU
```

This step ensures the model operates efficiently based on the hardware environment.

☑ Step 4: Load All Images from Dataset

All images from the dataset directory are loaded and preprocessed. Each image is resized, normalized, and converted into a format suitable for input to the CLIP model.

A list is maintained to store all image embeddings.

✅ Step 5: Process and Encode Images

Each image is passed through the CLIP model to extract its vector embedding.

- On CUDA devices, input images are cast to float16 for performance.
- On CPU, they remain in float32.
- The model handles type conversions internally.

After processing all images, the embeddings are stacked into a NumPy array.

✅ Step 6: Save Embeddings and Store in FAISS

The NumPy array of image embeddings is indexed using FAISS for similarity search.

FAISS supports different types of indexes; typically, an IndexFlatL2 is used for cosine similarity or Euclidean distance. The FAISS index is saved for future use.

```
import faiss
index = faiss.IndexFlatL2(embedding_dim)
index.add(image_embeddings)
```

✅ Function: Search Images with Text

A search function is implemented that:

1. Tokenizes and encodes a text query using CLIP.
2. Computes its embedding.
3. Uses FAISS to retrieve the top-N closest image vectors.
4. Returns the indices and distances of the closest matches.

⚠️ Note: The text_tokens must not be converted to float16; they are expected as Long tensors.

✅ Step 7: Example Query (Text-to-Image)

An example is demonstrated using the query "bicycle".

- The encoded text embedding is passed to FAISS to retrieve the top 3 closest images.
- The results are displayed using matplotlib.

Additional Notes

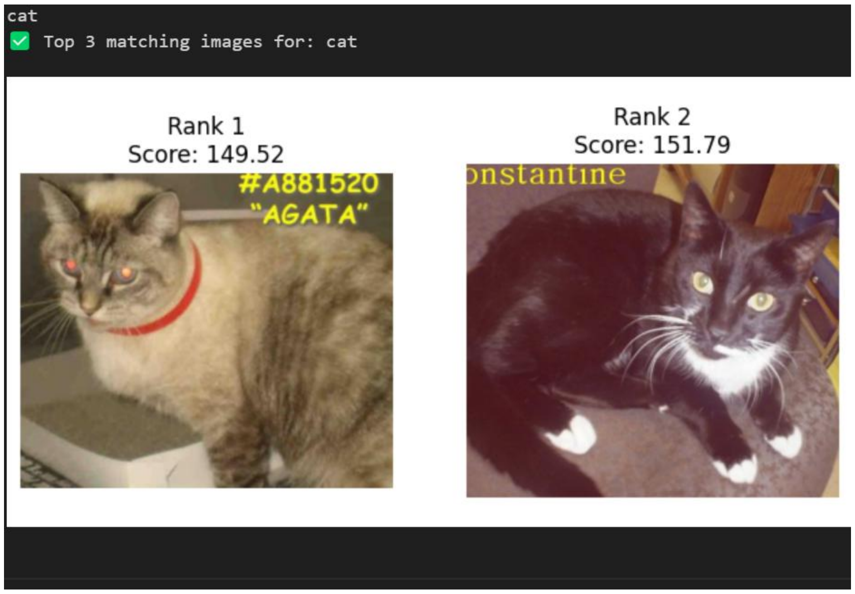
- Search Efficiency: FAISS provides fast approximate nearest neighbor search even for large datasets.
- CLIP's Power: CLIP enables zero-shot querying — the system works without training specifically on your dataset.
- Multimodal: The engine supports both image-to-image and text-to-image search.

Conclusion

This visual search engine showcases how pretrained models like CLIP can be used in conjunction with powerful tools like FAISS to build robust and efficient semantic search systems. The implementation allows for real-time image retrieval using either natural language descriptions or image queries.

This method can be expanded to support a variety of datasets and extended with UI elements or web integration for deployment in production systems.

Outputs



```
cat
✅ Top 3 matching images for: cat
```