Image Object Detection Pipeline

Vikas Tarvecha (https://linkedin.com/in/vikas-tarvecha)

Objective

Build a light-weight pipeline to ingest dataset from specific source, process through object detection model to filter images based on specific criteria or labels, and enable downstream users to filter datasets and search for images that satisfy specific criteria.

Constraints

The model would have following constraints based on objective listed:

1. Image data will be sourced from https://huggingface.co/datasets/wikimedia/wit_base
2. Input will be assumed to be in a format as specified in step #1.
3. Model selection will be performed keeping in mind resource requirements of commodity computers with 12 GB RAM, NVIDIA GEFORCE GTX GPU and Intel Corei5 CPU.

Pre-requisites

1. PostgreSQL Database
2. Server with Compute and Storage
3. Python with all required libraries from requirement.txt
4. Pre-trained Haar cascade classifier for face detection haarcascade_frontalface_default.xml
5. Shape Predictor 68 Face Landmarks Data

Source Data Inspection

We are leveraging 2 out of 300 train splits from wikimedia/wit_base repository from Hugging Face for this project.

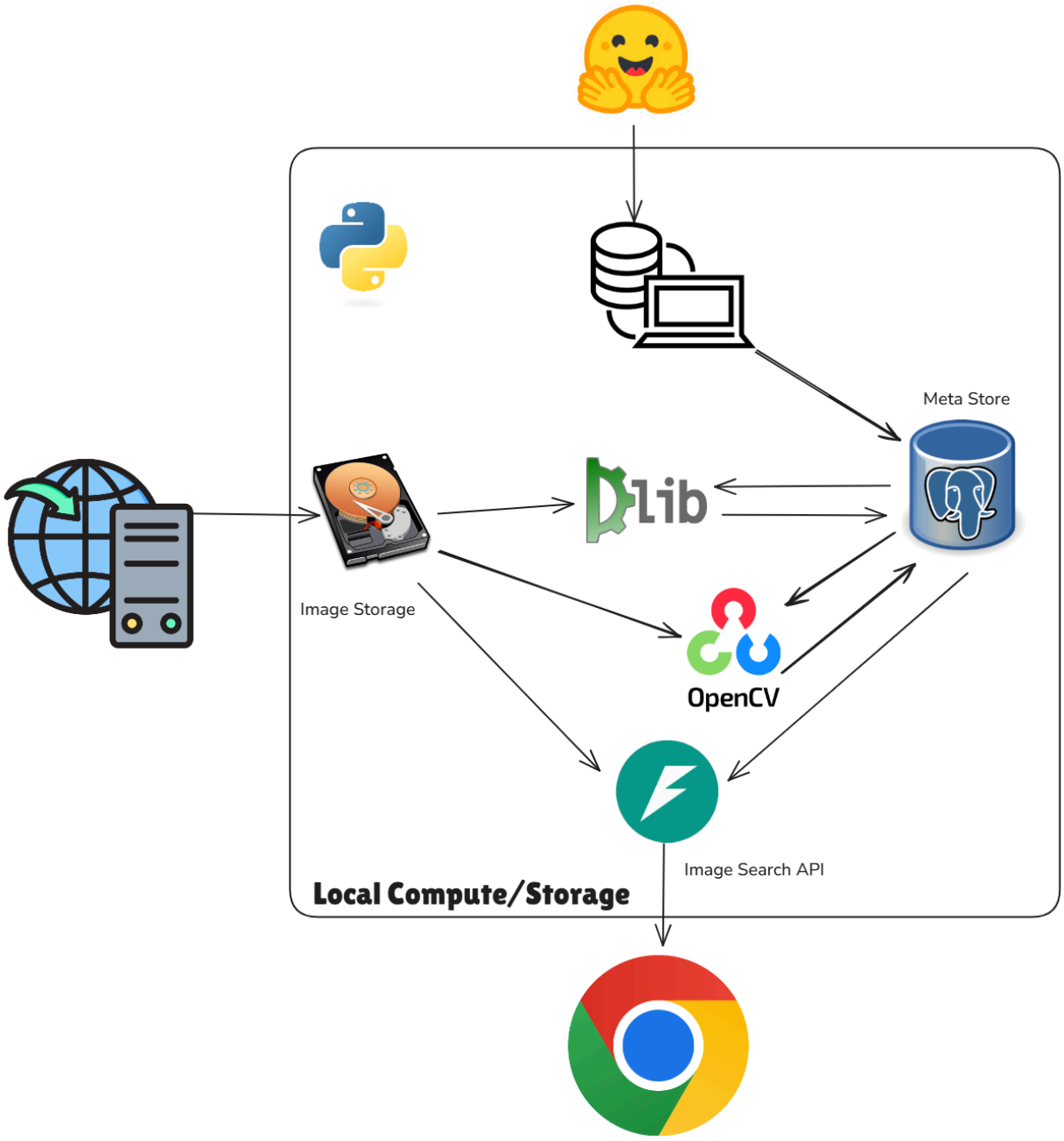
Here's the attribute list found in selected source data:

Field Name	Description	Type
image	Image resized to width of 300-px, preserving its aspect ratio.	PIL.Image.Image
image_url	URL to wikipedia image	URL
embedding	Precomputed image embedding with 2048-dimensional signature extracted from the second-to-last layer of a ResNet-50 neural network trained with Imagenet data	Vector
metadata_url	URL to wikimedia page containing the image and the metadata	URL
original _height	Original image height before resizing	Numeric
original_width	Original image width before resizing	Numeric
mime_type	Mime Type associated to the image	String
caption_attribution_description	This is the text found on the wikimedia page of the image.	String
wit_features	Sequences of captions for the image with language, page URL, information about the page, caption text, etc.	Dictionary

Checks

- There are 19629 records in each training dataset. In total there are 39258 records.
- The dataset is checked for existence of duplicate records by image_url and no duplicates are found.
- 669 Image URLs are not accessible. We will disregard these images.
- Image and Embeddings columns are large in size, which

System Design



Model Selection

There are two types of specific objects that we need to identify from images: Face and Eyesight Glasses

Following is the list of models that were considered for model selection.

Model	Face Detection	Eyesight Glass Detection	Comments
OpenCV cascade classifier	Low Accuracy	Low Accuracy	Low compute needs
RetinaFace	High Accuracy	Not readily available	Higher Accuracy, but slower processing
YOLOv8	High Accuracy	Not readability available	
Open AI CLIP	High Accuracy	High Accuracy	Wide Search Capabilities.

While transformer based models offer better accuracy, they need higher compute and additional storage for embeddings and running similarity search.

Due to compute limitations, the OpenCV model was leveraged to run the pipeline, which offered faster processing times with limited compute/memory available on personal computers.

MetaStore Schema

Field Name	Description	Purpose
image_uuid	Unique Identifier	Used as Identifier for object storage
image_url	URL to wikipedia image	Can be used to uniquely identify an image and download it.

embedding	Vector containing Image Embedding	Similarity Search
metadata_url	URL to wikimedia page containing the image and the metadata	Can be used to generate tags for images.
original_height	Original Height of Image	Analysis
original_width	Original Width of Image	Analysis
mime_type	Image Format	Analysis
has_face	If the image contains face	Status Tracking
has_glasses	If the image contains face with eyesight glasses	Status Tracking
image_download_status	If image has been downloaded to Local	Status Tracking
file_name	File Name from which record was inserted	Status Tracking

Implementation

Hugging Face Dataset Download

The source dataset is hosted on Hugging face hub. The source files are in parquet format. Hugging face provides hf_hub_download utility, to efficiently download the data from Hugging face hub.

download_files_to_local.py

- The script is hard coded to a specific repository and number of training splits for current requirements.
- The script iteratively downloads 2 out of 330 files from hugging face hub to data directory in current working directory.

Ingestion

In this step we generate a UUID for each image and load it to a postgres table, with metadata. Since the downloaded data was observed to be of larger size, the process is performed iteratively.

ingestion.py

- Creates a metastore table if it doesn't exist on postgres database.
- Check if the file is already loaded into the table before moving forward.
- Loads parquet file downloaded into local directory to postgres table iteratively.
- Checks count of dataset file and data inserted into postgres table.

Download Images

Since the dataset file downloaded from Hugging Face hub doesn't contain original images, we need to download original images.

Regardless of that, we still need to separate metadata, with object storage, to improve efficiency across the system, if we decide to scale this system multi-fold.

download_images.py

- Gets the list of image_urls from metastore table in postgres table, where current download status indicates that image is not already downloaded.
- Leverages Multiprocessing capabilities of spawn maximum number of processes to download images using URLs using Python Standard Libraries in batches.
- Tracks and updates the status of downloaded images in the metastore table.

Face Detection

After trying for a while to leverage models where I observed better accuracy, like RetinaFace and OpenAI CLIP, it was clear that it wouldn't be feasible to run those models for the amount of data and available compute capacity.

OpenCV offers efficient models based on Haar cascade to identify face landmarks.

detect_face_opencv.py

Glasses Detection

An attempt was made to run Glass detection with OpenAI based CLIP, but due to limited compute, it was not feasible to use it for the amount of data in consideration.

Alternatively, an approach based on detection of the bridge on the nose was discovered and due to its low compute requirement, I ended up using it. Since

detect_glasses_dlib.py

- Batches all the images that have a face, and doesn't have the existence of glasses set to any value.
- Spawns Multiple Processes to run the batch of images through Glass Detection Logic implemented with dlib library and nose bridge detection.
- Updates the status to metastore.

Search

As the final step, we want to search the images with required filtering conditions, in this case, specific to the presence of face and eyesight glasses.

I decided to use python based FastAPI to create a web app, which features filters, that let's users select "All", "With Faces" and "With Glasses" conditions and submit the request. Upon submission, the user is redirected to a separate page with an image grid of 300 images meeting search criteria.

image_search.py & image_search_filter.py

- image_search.py implements basic FastAPI with two pages, one that lets users select filtering conditions and submit the request, and another that shows the image grid of selected filters.
- image_search_filter.py is a helper module that queries metastore and generates 300 images for each filter condition.

Image Search Landing Page

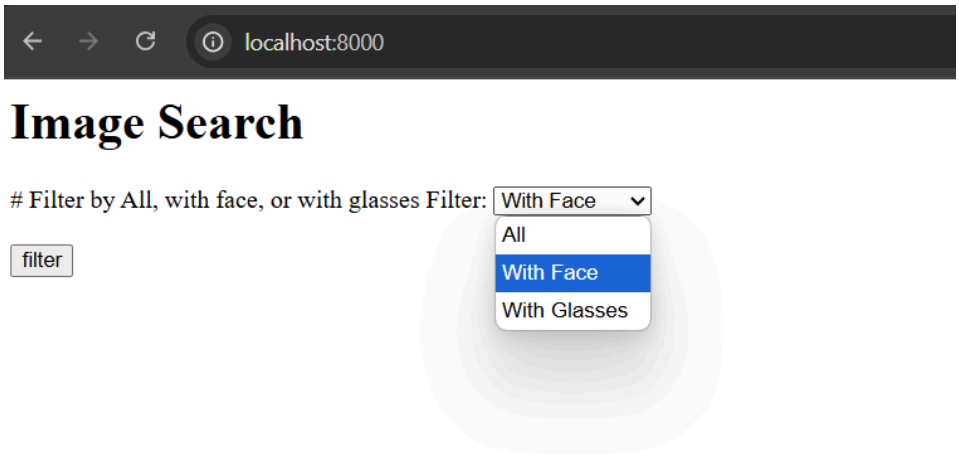
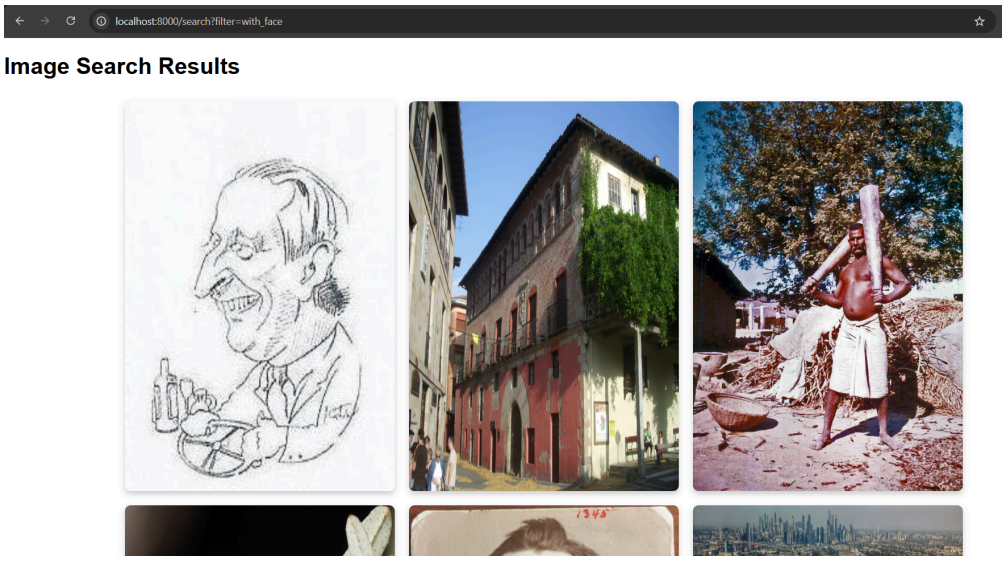


Image Search Results Page



Improvements

Model

The chosen model suffers from low accuracy to the point data is not considered reliable. If I had more compute at my disposal, I would choose Retina Face to detect faces, based on my limited

understanding of available models and their accuracies. I would collaborate with Research Scientists to gather more insights to tune and train custom models.

Apart from that, I would emphasize on choosing a transformer based model, which offers greater search flexibility instead of just focusing on specific labels.

System Design

While the proposed design is scalable to billions of images, individual components of systems are not. Current system leverages local storage, which needs to be replaced by Object Storage like S3 or Azure Blob. PostgreSQL may pose some horizontal scaling challenges as well. I would propose using a NoSQL database like CosmosDB or AWS DynamoDB.

Apart from components, the individual steps in the pipeline need to be distributed as well. The steps can be turned into stateless services, which would enable distributed execution of those steps, especially generating embeddings and running inference.

Search

Search capabilities are currently limited to filtering. These capabilities can easily be expanded into free form text search capabilities.

Repositories

Code

https://github.com/vtarvecha/image_classification

Hugging Face Hub

vtarvecha/detect_face_glasses

References

Image Search with Natural Language Queries | Google Cloud Blog

Glasses Detection - OpenCV, DLIB & Edge Detection | by Siddharth Mandgi | Medium

Build Facial recognition with Python, OpenCV, OpenAI CLIP and pgvector | by Krishabh | **AI monks.io** | Medium

https://www.kaggle.com/datasets/sergiovirahonda/shape-predictor-68-face-landmarksdat?resource=download