

# Trademark Analysis & Identification Tool (TRAIT)

Team 11 Sprints 11 + 12 Lane Keck, Logan Taggart, Caleb Stewart

# FAISS

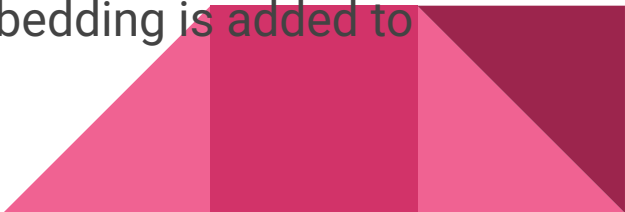
- **FAISS (Facebook AI Similarity Search)** is a library developed by Meta that allows for efficient similarity search and clustering of dense vectors.
- It's designed to handle large-scale vector data and provides fast nearest neighbor search, even over high-dimensional embeddings (like those produced by CLIP, ResNet, or BEiT).



# How We Are Using FAISS for General Logo Search in Videos

Each detected logo is converted into a feature embedding vector using our fine tuned YOLO model

When a new frame is processed:

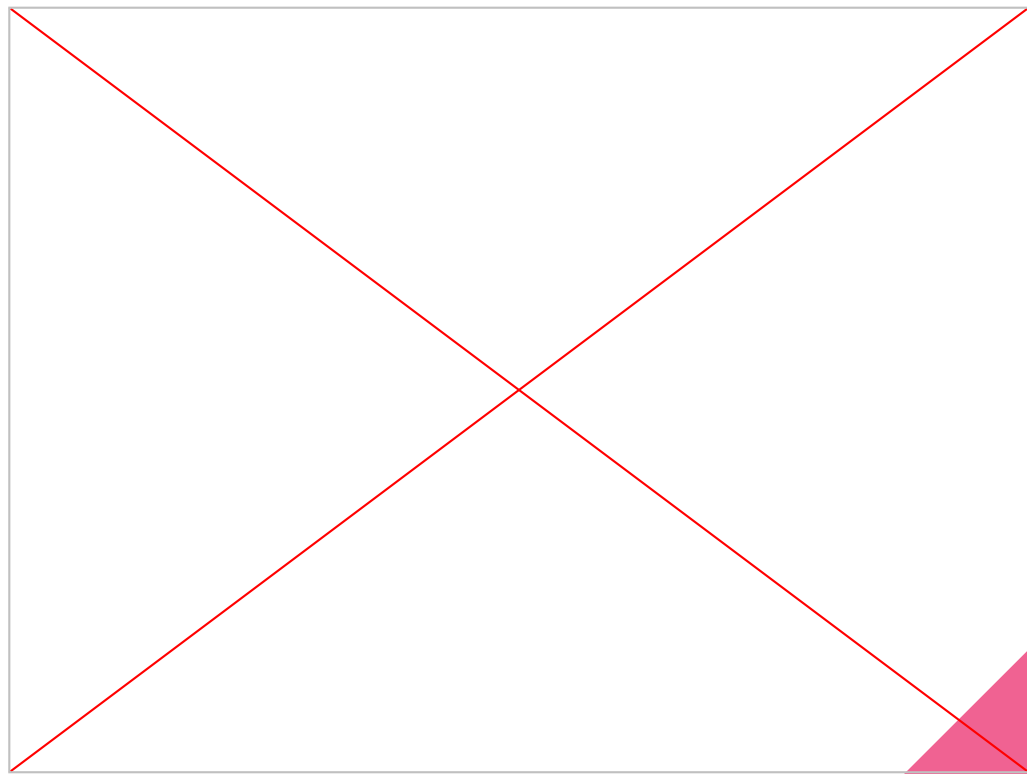
- Extract the logos embedding
  - Query the FAISS index to check for similar logos already seen
  - If the similarity score of the FAISS index is below a certain threshold, we treat it as the same logo
  - If not, we consider it a new logo instance, and its embedding is added to FAISS
- 

# Video General Logo Search

- We have finished implementing general search for videos!
- This was done by first testing our idea in an interactive python notebook, where we were able to process the uploaded video, extract and embed frames, and match logos across frames using FAISS
- Once our logic was solid in the interactive python notebook, we were able to integrate it into our main application



# Demonstration - General Video Search



# Video Specific Search Logo Search

- We have successfully prototyped logo verification in an interactive Python notebook
- This feature builds on the general search functionality but adds an extra step:
  - After detecting logos in video frames, we check if we have this logo stored in our FAISS vector database
    - If it exists, we've already verified that this logo is the same of what we are looking for
    - If not, then run a verification vote to determine if the detected logo and the reference logos are the same
- This prototype works, but not yet integrated in the main application


# Video sending

We ran into OpenCV having some copyright issues with the H264 Codec, so we needed to compress a video in a different way

- Previously, it was making all videos 900 mb in size → Now an appropriate size for the length of the video

Now we are able to compress the video and choose the amount of compression based on what we find to be most appropriate

To simplify the sending process, we decided to split it to be two different routes one for returning the video file, and one for returning the metrics



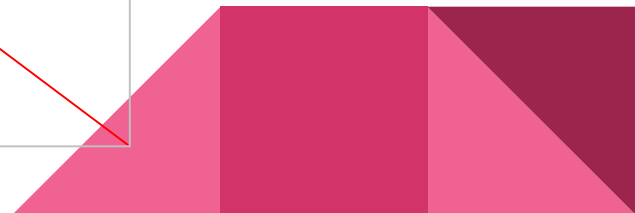
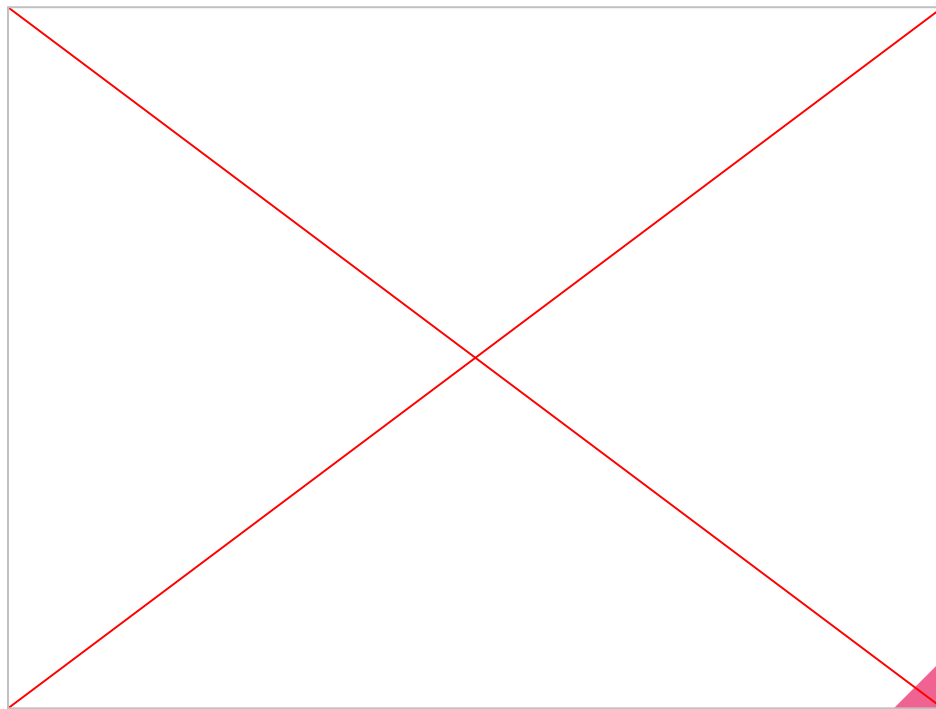
# CSRT Tracker

- We are trying to draw boundary box in every frame
- But we are only processing every 5th frame
- To solve this problem we are going to use a CSRT Tracker in OpenCV
- This will take the space within the original boundary boxes and track the position in the video without us having to perform another YOLO detection
- Currently this is working in our testing environment, but we still need to implement it into our application





# CSRT Tracker Demonstration



# Challenges

- Processed video size was very bad → Needed to find a way to manually compress the video ourselves
- CSRT tracker was very difficult to setup and configure
- Tracker sometimes fails and loses track of the area it is tracking
- Model is detecting some false positives in videos → Maybe we can find a way to be able to reduce these without compromising too much processing speed?



# What's Next?

- Implement specific video search
- Clean up code → We reused a lot of code just to get our application working
- Give user ability to be able to X out displayed logo metrics in the event of false positive
- Find more quick/efficient way to package backend into executable
- Optimize code so that detections/processing times are as short as possible

We've done most of the heavy lifting. We feel that all that's left is the nitty gritty details of our application.

