GPU-Accelerated Human Differentiation Network

Pryce Matsudaira & Benjamin Kim

Project Overview

This project implements a real-time detection and identification system for specific individuals. ASTRA (our senior design drone project) detects all people in a video feed using a pre-trained YOLOv8 model. The core objective of this project is to extend ASTRA's scope to accurately detect specific people in a video stream and differentiate them by comparing visual features to a gallery of precomputed identity embeddings. To achieve this, we utilized a fine-tuned ResNet18 system for generating person-specific vector embeddings. Cosine similarity (via a custom CUDA kernel) is then used to compare those embeddings with a vector that represents the individual (e.x, differentiating between Ben and Pryce).

How the GPU is Used to Accelerate the Application

The GPU is used to accelerate the cosine similarity computation. When an individual is detected by a web camera, their ResNet18-generated feature vector is compared with a gallery of that individual's stored embeddings. To make this comparison process efficient, a custom CUDA kernel is used to parallelize the cosine similarity calculations.

Our parallel algorithm is structured so that the query embedding is compared against all gallery embeddings at the same time. Each GPU thread computes the cosine similarity between those two vectors. Additionally, each thread also computes the similarity score for one gallery embedding, with 256 threads in one thread block. To ensure all gallery embeddings are executed, the grid size is set as the ceiling of the number of gallery embeddings divided by the thread block size. This allows for the similarity scores of N gallery embeddings to be computed in parallel within one kernel launch. The person identification stage of the software pipeline is parallelized.

Implementation Details

Each team member recorded two videos of themselves: one focused on their face (close-up) and another focused on their entire body. OpenCV was then used to extract frames from the videos at 5-frame intervals. Of those frames, the 10 best lighting and resolution images were manually selected and processed through a ResNet18 embedding model to extract a 512-dimensional feature vector. The vectors for each individual were averaged to create a gallery embedding for later use.

The implementation process of h*uman_differentiation.py* begins with real-time human detection via a YOLOv8 object detection algorithm, which had to be re-implemented for a desktop or laptop webcam through OpenCV. This was because the previous implementation for YOLOv8 on ASTRA relied on the compatibility with the drone's depth camera drivers through Linux. Frames from a web camera will pass through YOLOv8 and output bounding boxes around detected humans. YOLOv8 will use such bounding boxes to crop regions of the detected individual and pass them through a pretrained ResNet18 model. The model will output a 512-dimensional feature vector of the individual's visual features.

The script then ideally loads a compiled Parallel Thread Execution file with a CUDA kernel for efficient and fast cosine similarity computation. However, due to complications discussed later, the cosine similarity kernel had to be created and directly implemented in the *human_differentiation.py* script using Numba. A ResNet18 model is initialized as somewhat of an embedding extractor and deployed on the GPU. As mentioned earlier, Pryce and Ben each have stored image embeddings, which are loaded and organized into a NumPy array.

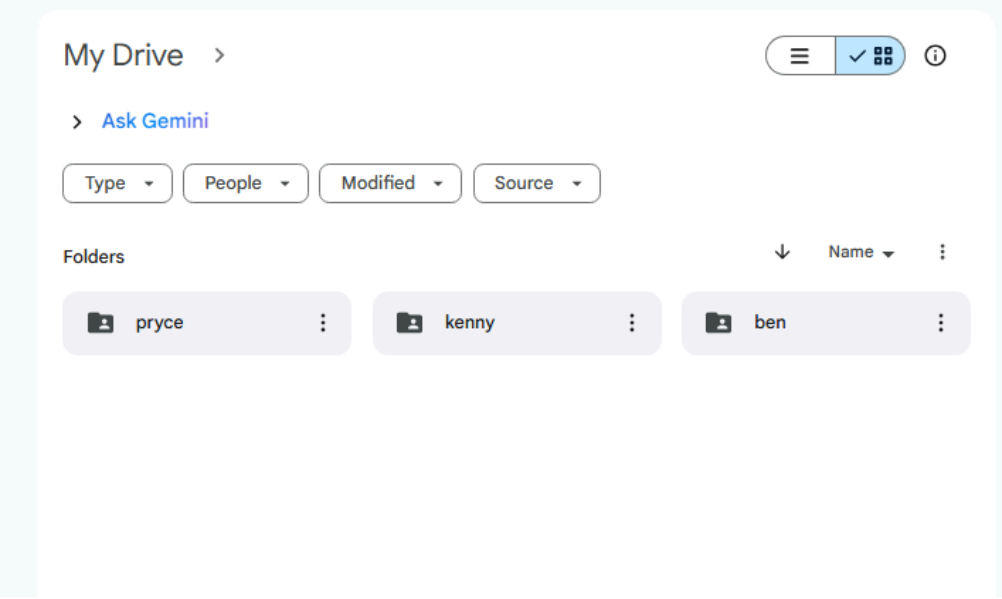
When a human is detected from the web camera, a ResNet18 model analyzes a cropped region of the detected individual and converts it into a 512-dimensional embedding vector. This vector analyzes the visual features of the individual and compares them against precomputed averaged embeddings using cosine similarity. Each thread calculates similarities between the ResNet18 embedding and the gallery embedding, allowing one-vs-many comparisons in a kernel launch.

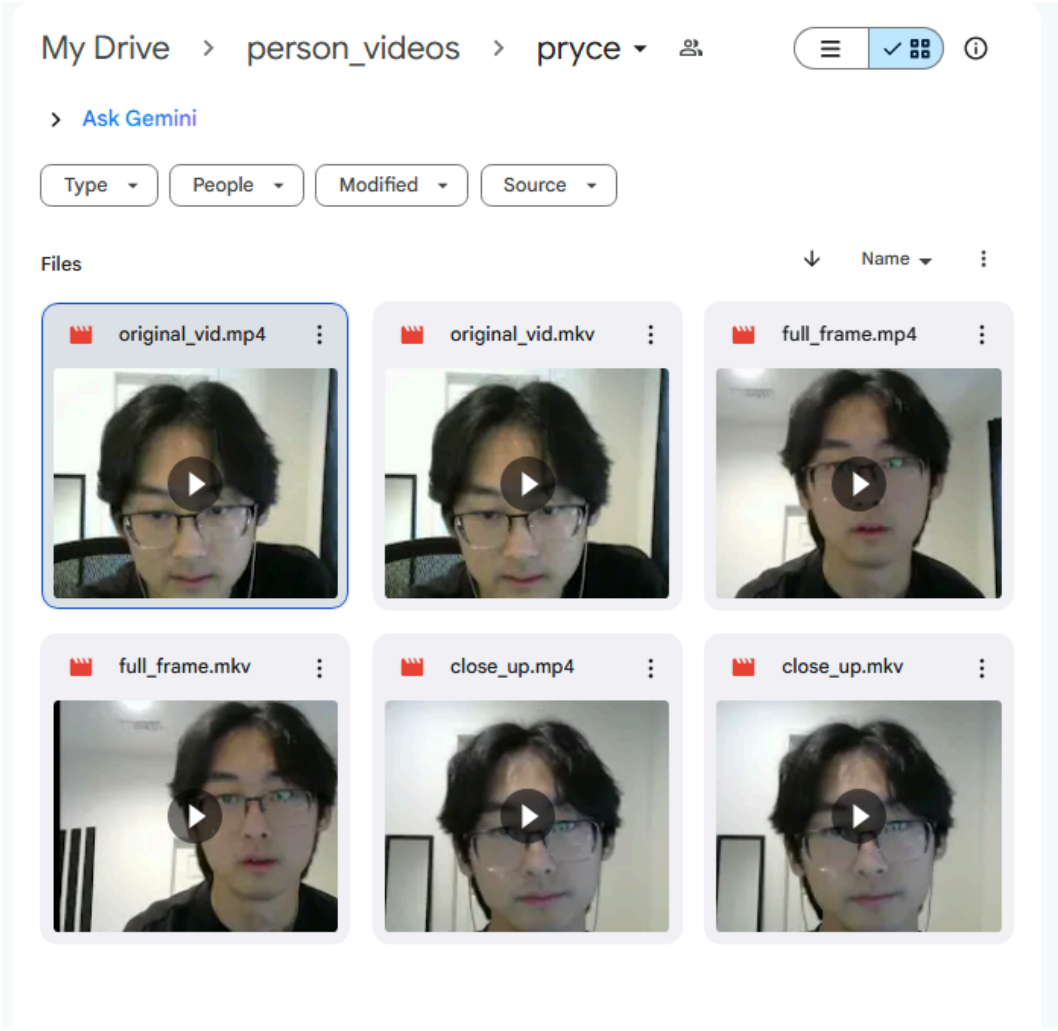
If the calculated similarity score is higher than the defined threshold of 0.55 or 55%, the system will label the predicted human accordingly above their bounding box. If no match was detected, the system will label "Unknown Person". This human differentiating system utilizes deep learning models, computer vision algorithms, and GPU acceleration to detect and label individuals.

Installation/Setup Process

1) Install Python (3.12 or higher) and required libraries, including opencv-python, numpy, numba, torch, cuda-python, torch, pillow, and ultralytics. Additionally, make sure the NVIDIA CUDA Toolkit (12.9) is installed and that NVIDIA GPU drivers are up to date.

2) First, upload video files of yourself and the people you want to differentiate to Google Drive. (Ex. My Drive → person_videos → pryce, ben)
    a) Google Drive was mainly used to store datasets and videos for this project due to compatibility issues with each script on respective laptops/desktops. Not all laptops/desktops have the same file paths C:\User\etc\etc, which would require us to change the code every time to create dataset updates. Google Drive also allows you to share these videos and datasets easily
    b) Example Image(s):

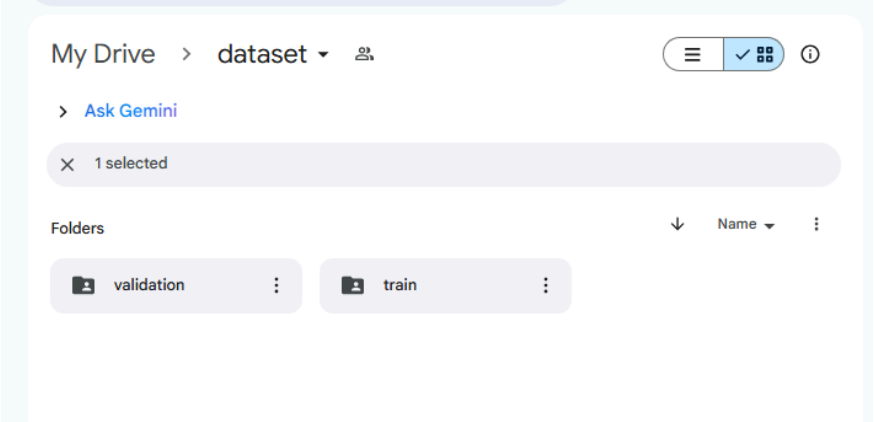


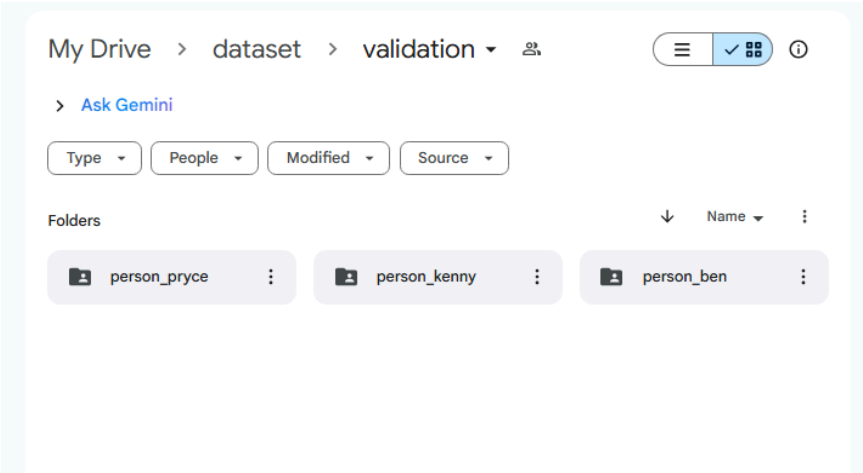


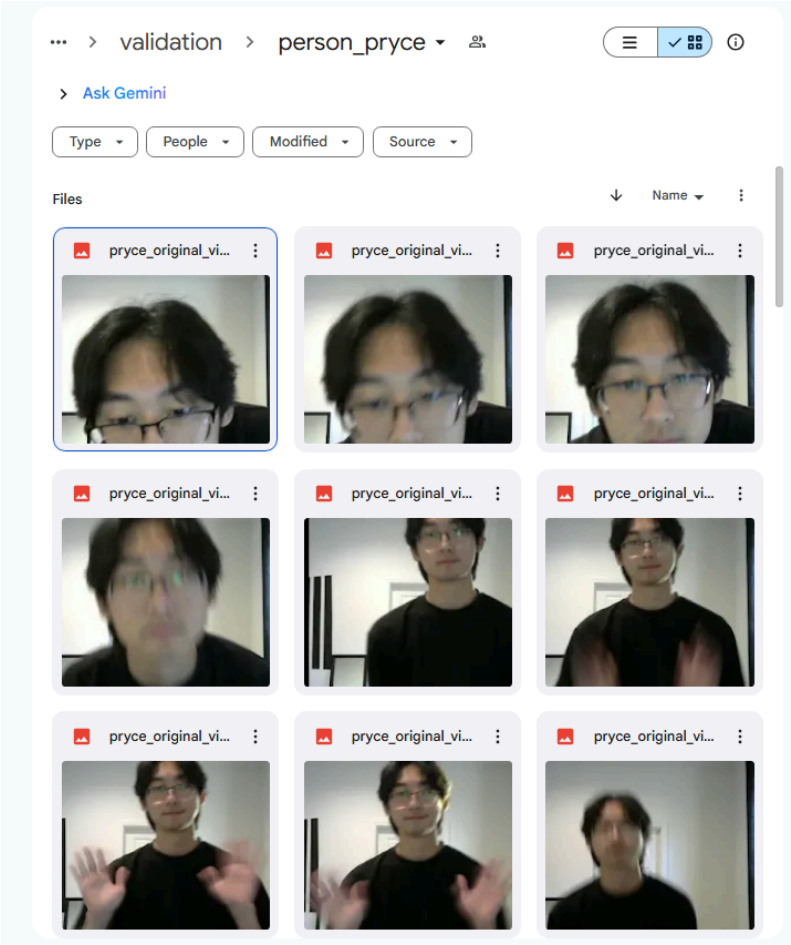
c)

3) After you have uploaded the required videos, run the *datasets_personID.ipynb* file.

   a) This will mount to the Google Drive and cut up the uploaded videos into images every 5 frames. Then it will randomly distribute image frames into validation and test folders for each person.







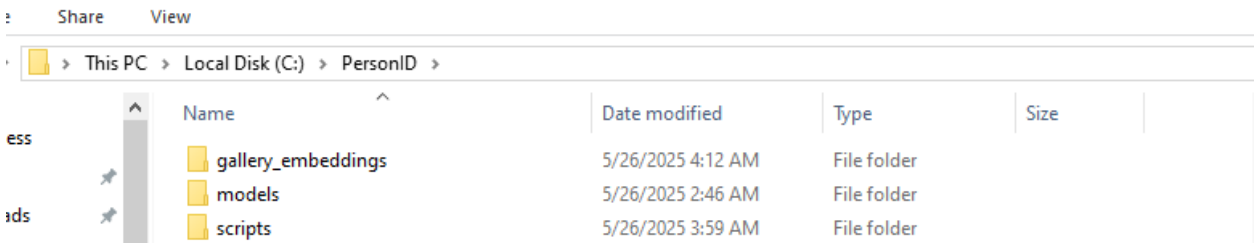
4) Run each cell of *train_classifications.ipynb* one at a time. This will create the ResNet18 model path and class labels (pryce, ben) as separate files in Google Drive. The files are now created in Google Drive: *person_classifier_model.pth*, *person_reid_classes.json*

a) Each cell will set up the classifier, loss function, and optimizer for classification, training loop, then classification model training (Epochs train loss/accuracy and validation loss)

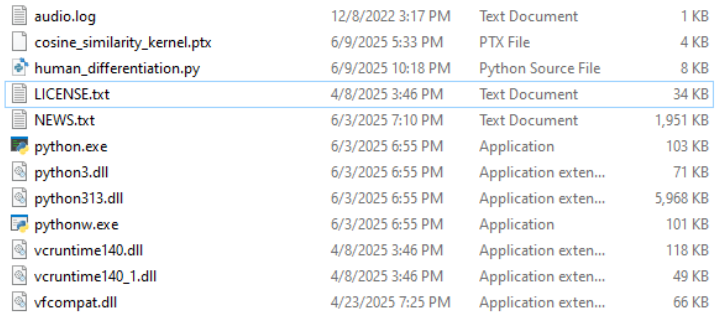
5) Run each cell one at a time for *embeddings_personID.ipynb*. This file will begin building the ResNet18 model and initializing previously created weights. It will then extract features into a vector embedding. You will then test the shape and viability of the extracted embedding.
    a) You will then manually input each image name from the validation folder "ben_IMG_7093_frame_00370.jpg" 10 times for each person you want to create an average embedding vector of.
6) You will then need to take **person_classifier_model.pth** and **person_reid_classes.json** files from Google Drive and import them to your specific computer.
7) Ensure folders and files are organized for proper file paths

- models/ *(ResNet18 Models)*
    - person_classifier_model.pth
    - person_reid_classes.json
- gallery_embeddings/ *(Averaged embeddings of specific individuals from embeddings_personID.ipynb)*
    - ben_gallery_embedding.npy
    - pryce_gallery_embedding.npy
- scripts/
    - yolov8n.pt

Example Image:



8) Make sure to have your Python (3.12+) installation in the same directory as human_differentiation.py



audio.log	12/8/2022 3:17 PM	Text Document	1 KB
cosine_similarity_kernel.ptx	6/9/2025 5:33 PM	PTX File	4 KB
human_differentiation.py	6/9/2025 10:18 PM	Python Source File	8 KB
LICENSE.txt	4/8/2025 3:46 PM	Text Document	34 KB
NEWS.txt	6/3/2025 7:10 PM	Text Document	1,951 KB
python.exe	6/3/2025 6:55 PM	Application	103 KB
python3.dll	6/3/2025 6:55 PM	Application exten...	71 KB
python313.dll	6/3/2025 6:55 PM	Application exten...	5,968 KB
pythonw.exe	6/3/2025 6:55 PM	Application	101 KB
vcruntime140.dll	4/8/2025 3:46 PM	Application exten...	118 KB
vcruntime140_1.dll	4/8/2025 3:46 PM	Application exten...	49 KB
vfcompat.dll	4/23/2025 7:25 PM	Application exten...	66 KB

9) Make sure *cosine_similarity_kernel.cu* is in the PersonID folder or in a file that has administrator access. Now convert *cosine_similarity_kernel.cu* to a PTX file.
    a) Open Developer Command Prompt for VS 2022 then input cd C:\YourPath\PersonID —> nvcc -ptx -ccbin "C:\Your\Full\Path\To\MSVC\bin\Hostx64\x64" cosine_similarity_kernel.cu -o cosine_similarity_kernel.ptx
    b) A *cosine_similarity_kernel.ptx* file should now appear in the PersonID folder. Drag that to the same directory as *human_differentiation.py*
10) Run “python human_differentiation.py” in the command prompt of Visual Studio Code
11) If the cosine similarity kernel refuses to work with *human_differentiation.py* (like us). Error Statement: **‘cuda.cudart’ has no attribute ‘cuInit’**
    a) You will have to run the newly modified *human_differentiation.py* using Numba that directly implements the cosine_similarity_kernel and fulfills the project goal.
12) Run “python human_differentiation.py” in the command prompt of Visual Studio Code
13) Press “q” to quit

Demo Video

Link: https://youtu.be/pVjVxKaa4ik