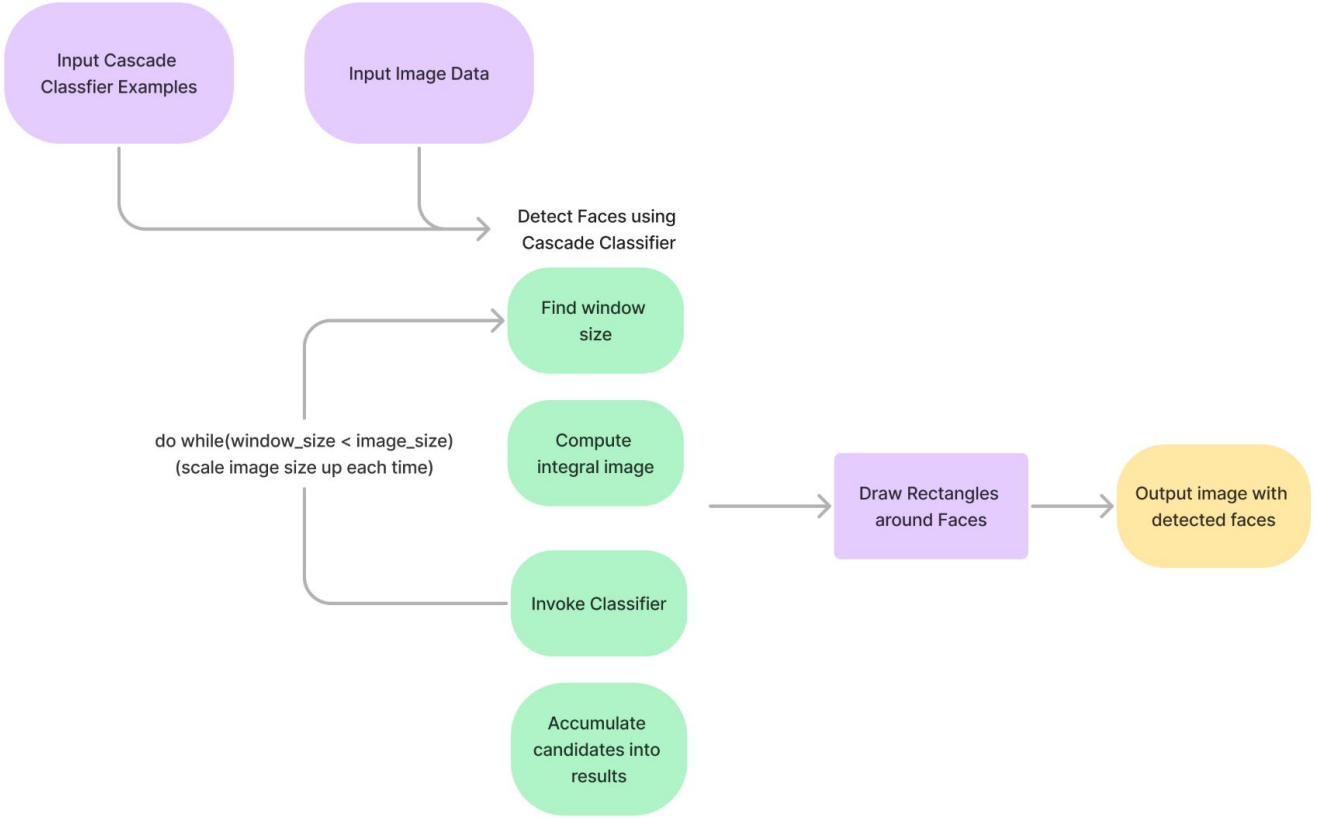
PARALLEL FACE DETECTION IN CUDA

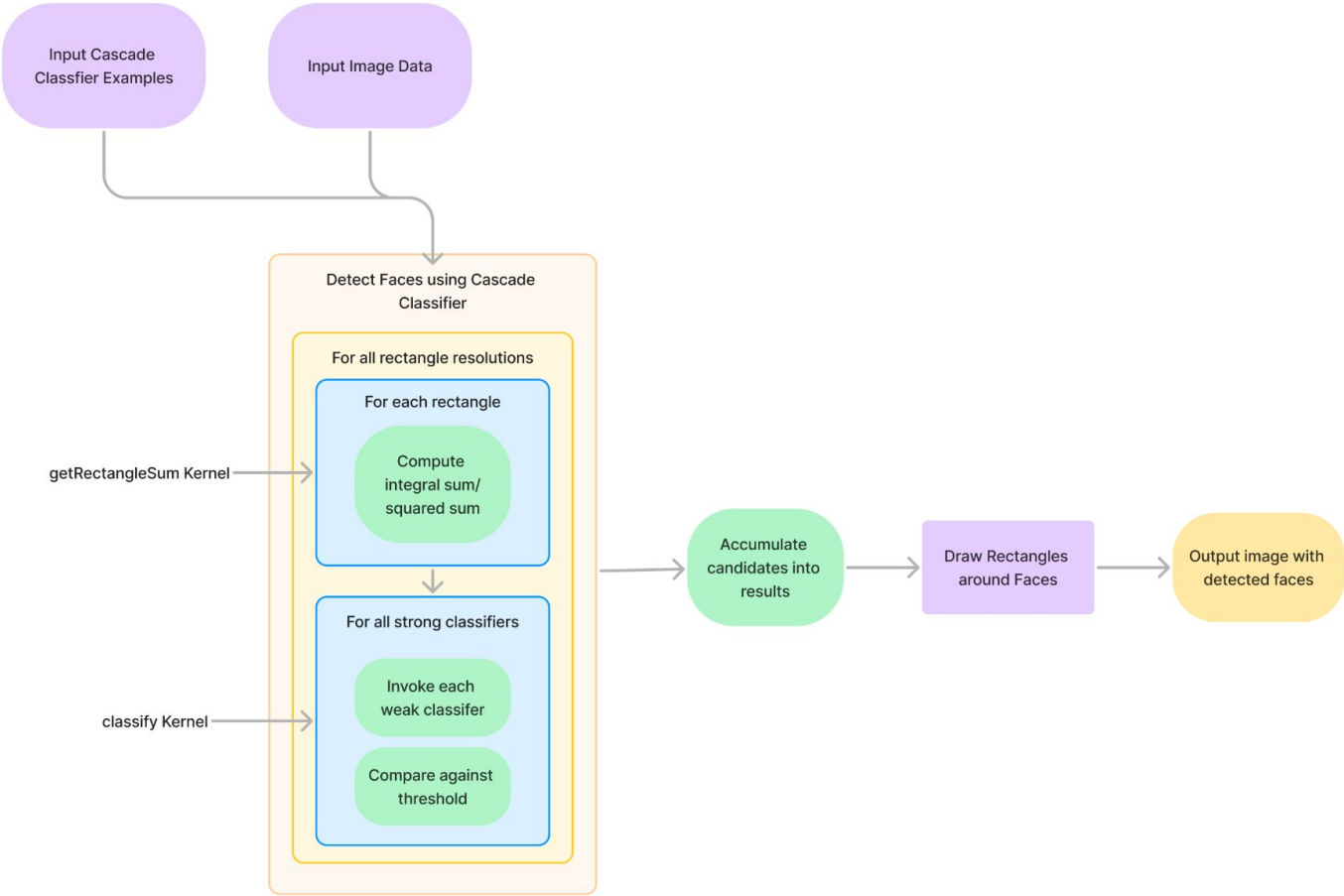
Ria Manathkar & Gaurika Sawhney

THE VIOLA JONES ALGORITHM

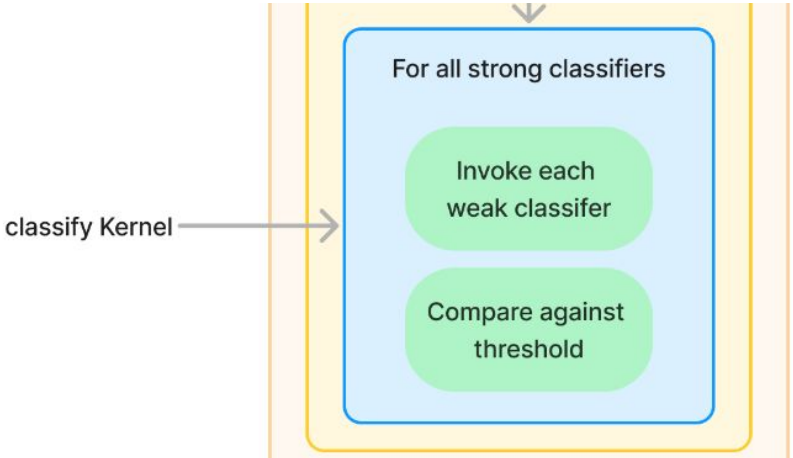
1. Uses Haar-like features to identify different parts of the face
2. Calculates the ”integral image” from the input image
3. The cascade classifier is designed to quickly reject non-face regions in the image.
4. After passing through all stages of the cascade, the remaining regions are considered potential faces and are passed through a threshold.



PARALLELIZATION APPROACH



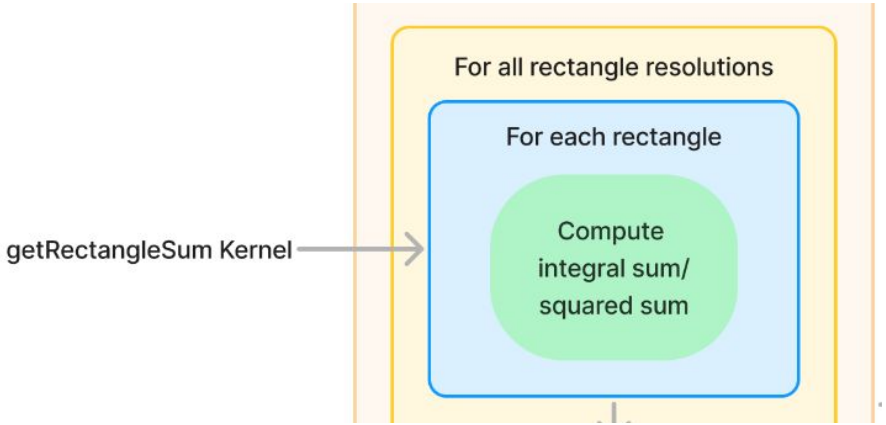
CASCADE CLASSIFICATION



- Inherently a sequential process
- Structured as a series of stages where each stage contains one or multiple weak classifiers
- If the stage classifies the region as positive, the region is passed on to the next stage in the cascade

1. Cascade itself executes sequentially since each stage must be passed before proceeding to the next.
2. Parallelized the evaluation within each stage.
3. Kernel evaluates a set of weak classifiers against a sub-region of an image to determine if the region meets certain criteria defined by the classifiers.

DETECT FUNCTION

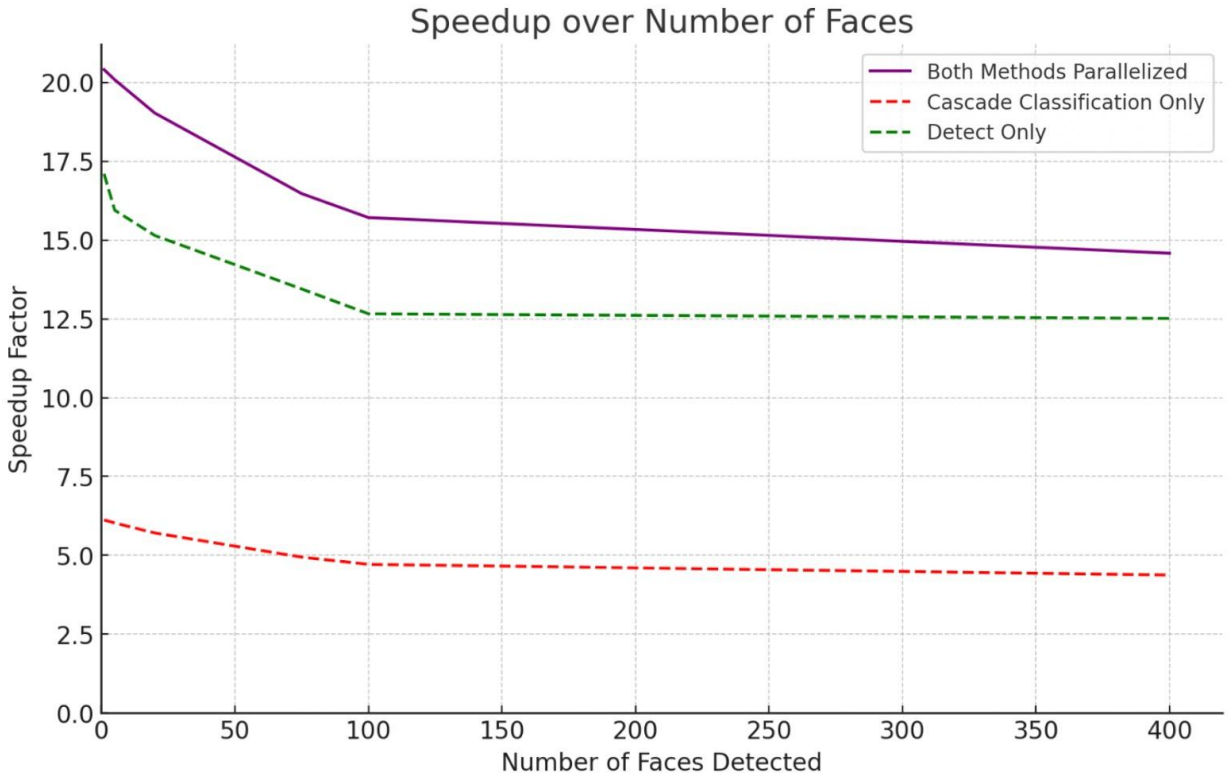


- Uses the cascade classifier to detect objects within the image.
- Sequential algorithm implemented a sliding window approach.
- Involves moving a window of a fixed size across the image and analyzing the content within that window at each position to determine if it contains a facial feature.

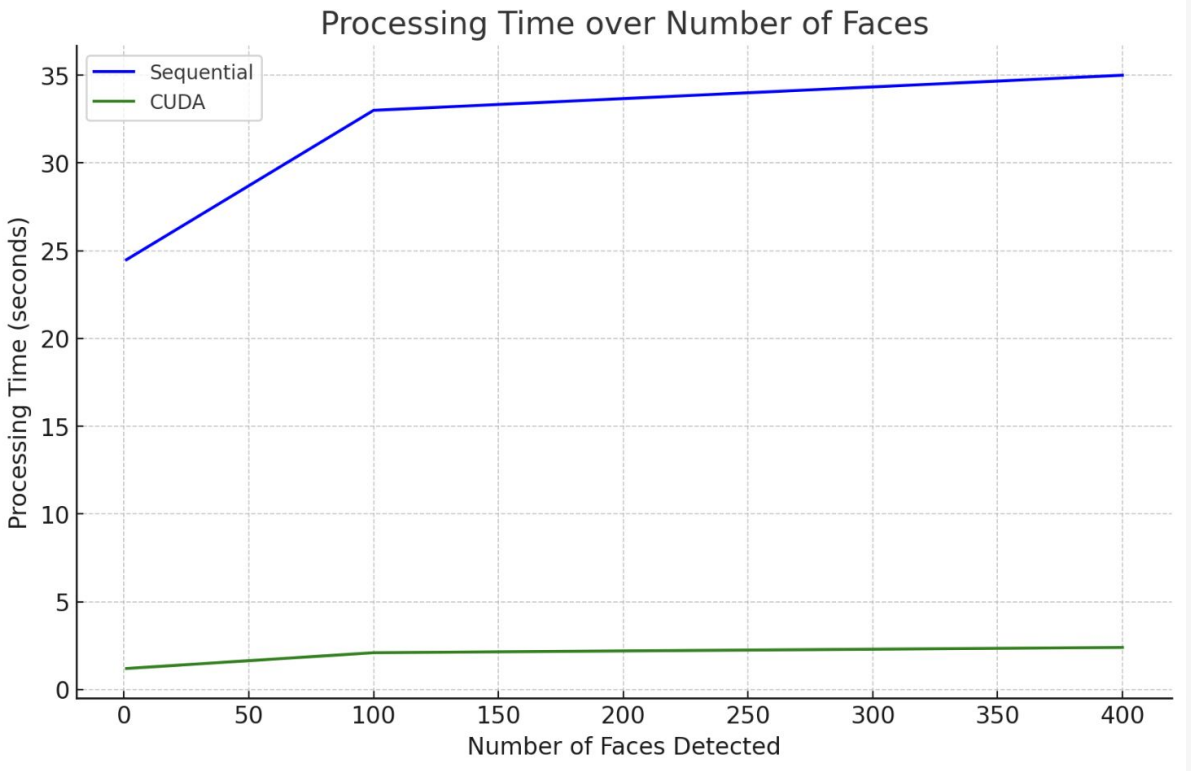
1. CUDA kernel is designed to process multiple windows simultaneously.
2. Each thread in the GPU can handle the classification of a different window beginning at its corresponding pixel.
3. For each rectangle, the kernel computes the sum of pixel values within the window based on the integral image and repeats over multiple rectangle sizes.

SPEEDUP RESULTS

- Both methods combined achieve highest speedup
- Initially high speedup → parallelizing both components is significantly more effective when number of detected faces is low
- Combined parallel implementation exploits partitioning of methods

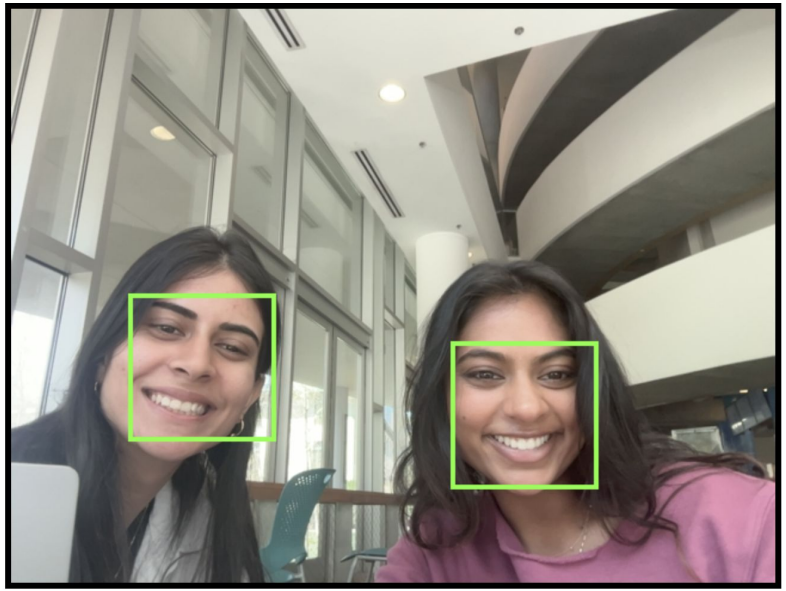


COMPUTATION TIME RESULTS



- The CUDA implementation not only drastically reduces the processing time
- Also shows stability against increasing workloads
- Both processing times increase with more faces
- With larger input sizes, increased computational complexity and overhead from managing more data

TAKEAWAYS



- Learned how to navigate dependencies
  - Cascade classifier + Integral image
  - Detect function synchronization
- Realized how to break down a very interconnected algorithm
  - Had to identify what parts were the most computationally expensive
  - Allowed us to find specific parts that benefited from parallelism and avoid unnecessary overhead
- Experimented with different tradeoffs
  - Navigating dependencies and modularity
- Further exploration
  - Cache coherency
  - Reusing previous frames in live-video