CrowdFlowAI: A Deep Learning-Based Framework for Pedestrian Flow Analysis and Stampede Risk Prediction

CS-671

GROUP 28 :

Ajay- B22257, Dipan-B22262, Saksham-S24114, Umang-B22279, Amuel-B22192, Sayan-B22238

Abstract

CrowdFlowAI is an integrated deep learning and computer vision framework designed for the surveillance and proactive safety management of large-scale human gatherings. It enables real-time pedestrian detection, trajectory tracking, panic simulation, and anomaly-based risk prediction using a curated dataset of surveillance footage from the Kumbh Mela. The system combines state-of-the-art object detection, multi-object tracking (MOT), and unsupervised learning models to generate interpretable visual analytics and early warning indicators of potential crowd disasters.

Contents:

1. Dataset and Preprocessing

Dataset Description

The dataset consists of 20 surveillance video clips from the Kumbh Mela, each approximately 20 seconds in duration. The dataset encompasses a diverse range of crowd densities, movement patterns, and environmental conditions.

Preprocessing Pipeline

Objective: Enhancing contrast and reducing camera shake in a folder of crowd-scene videos.

Core Components:

1. **Contrast Boost (CLAHE):**
   • Convert each frame to LAB space, apply CLAHE on the L-channel, revert to BGR.

2. **Stabilization:**
   • Compute sparse optical flow between consecutive frames (max 200 features).
   • Estimate a robust affine transform (RANSAC) and warp the current frame.

Configuration:

- **Input:** `Crowd_Videos_Dataset/*.mp4`
- **Outputs:**
  - Enhanced videos → `Processed_Videos/`
  - Comparison images → `Screenshots_Comparisons/`

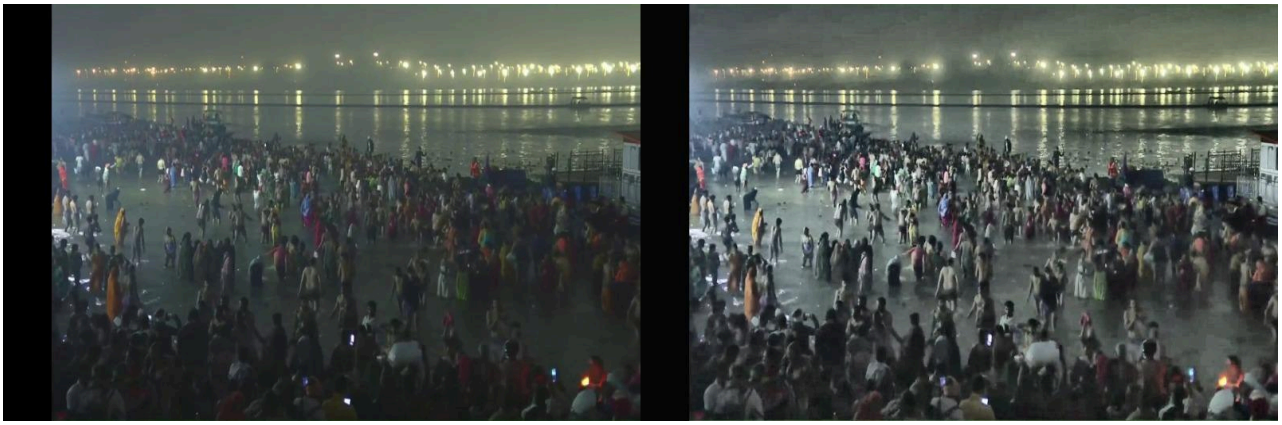
- **Adjustable:** number of screenshots (`frames_to_capture`), CLAHE parameters, frame sampling range.

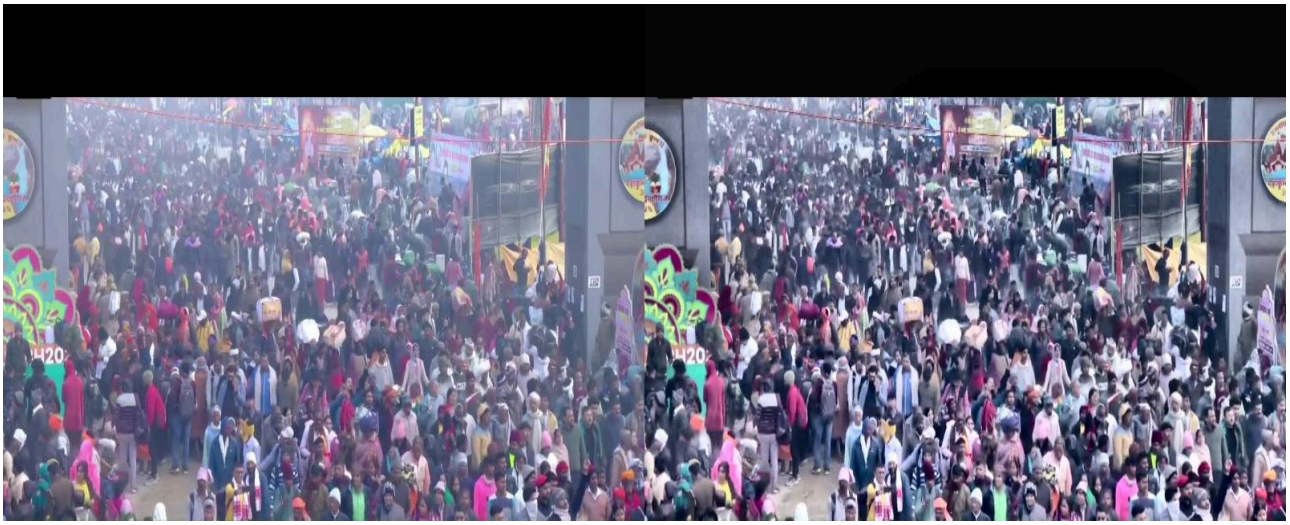
Results:

- Processed videos maintain original resolution & FPS with visibly improved contrast and smoother motion.
- Screenshots provide quick verification of enhancement & stabilization effects.

Before

After





End-to-End Video Frame Sampling and Model Refinement:

Our workflow automates data preparation (frame sampling + resizing), bootstraps annotations via YOLOv5, and then refines a custom detector through self-supervised fine-tuning :

Frame Extraction & Resizing

- Read each video from `Crowd_Videos_Dataset` at its native FPS.
- Sample frames at 5 FPS (one frame every `fps/5` frames).
- Resize each extracted frame to 640x360 and save under `Processed_Frames/<video_name>/`.

Pseudo-Label Generation

- Run a pre-trained YOLOv5 head-detector on these frames to produce “pseudo” bounding-box labels for visible heads (not all heads are detected).

Model Fine-Tuning

- Combine the frames and their YOLO-generated labels into a training set.
- Fine-tune our detection model on this pseudo-labeled data to improve head-detection performance.

Link to the fine tuning Jupyter notebook -

Fine_tuning.ipynb

Fine-tuning with pseudo-labels is a self-supervised strategy where a pretrained model first generates fake annotations on unlabeled data, and these predictions are then treated as ground truth to enlarge the training set. By selecting only high-confidence detections (e.g. above a chosen threshold) and optionally down-weighting them in the loss, we can mitigate noise while exposing the network to more varied examples. Retraining on this combined set—original labels plus pseudo-labels—helps the model adapt to domain-specific patterns it initially missed, boosting recall on hard-to-detect instances. Iterating this process—re-predicting labels with the refined model and fine-tuning again—gradually strengthens performance, provided we monitor a held-out validation set to prevent drift from accumulating errors

2. Pedestrian Detection :

These steps yield a robust, real-time capable detection pipeline that adapts to scale variations in crowd footage and produces stable, easy-to-interpret crowd-count estimates.We implement a multi-scale, confidence-thresholded person detector on video, smoothing its counts over time and saving a zoom-aware annotated output.

Setup & Configuration

- **Model Initialization:** We load a fine-tuned YOLOv5 network to leverage its powerful, anchor-based detection head for real-time person spotting. Using pretrained weights jump-starts learning of general visual features, while our custom `.pt` file adapts it to the crowd-scene domain.

- **Threshold Strategy:** Two confidence cut-offs balance recall vs. precision: a **lower threshold** (`conf_top = 0.20`) on zoomed-in regions catches faint, small-scale detections, while a **higher threshold** (`conf_bottom = 0.40`) on the full frame avoids spurious boxes on larger targets.

- **Temporal Smoothing:** A sliding window of recent counts (`N=10`) implements a simple moving average, reducing frame-to-frame jitter in detections and giving a more stable “People Count” display.

Multi-Scale Frame Splitting & Detection

- **Rationale for Splitting:** Crowds contain both near (large) and far (tiny) people. By partitioning the top half into two quadrants and upscaling them (`zoom_factor=2.5`), we effectively enlarge distant heads so they exceed the detector’s minimum object size. The bottom

half—where people tend to appear larger—can be processed at full resolution.

- **Inference Passes:** We run three separate inferences per frame—two on zoomed quadrants with a lenient threshold to maximize recall on small objects, and one on the bottom half with a stricter threshold to maintain precision on larger detections. This “divide-and-detect” approach exploits YOLO’s speed while mitigating scale-related blind spots.

Coordinate Mapping, Annotation & Count Smoothing

- **Back-Projection of Boxes:** Detections from the zoomed images arrive in scaled coordinates. We invert the zoom (`/ zoom_factor`) and add regional offsets to reproject each box correctly onto the original frame. This preserves spatial accuracy for tracking or further analysis.

- **Drawing & Counting:** Each mapped box is drawn in green to visually confirmed detections. We count total boxes per frame, append that to our deque, and compute the **mean** over the last 10 frames. This moving average suppresses transient false positives/negatives and yields a smooth, human-readable crowd estimate.

Output Generation & Aggregate Reporting

- **Annotated Video Output:** Frames augmented with detection overlays and the smoothed count (rendered in red text) are written out at the original video’s FPS and resolution, ensuring seamless playback and easy integration into presentation or monitoring dashboards.

- **Final Metrics:** After processing, we compute the overall average of the per-frame smoothed counts as a single scalar Estimated Average Crowd Count. This summary statistic captures the typical density across the entire clip, useful for capacity planning, safety evaluations, or comparative studies.

- **Logging & Monitoring:** Periodic console updates (every 30 frames) help track progress and diagnose pipeline throughput, ensuring the system meets real-time or batch processing requirements.

3.Multi-Object Tracking (MOT)

Real time Detection using Norfair

Tracking Methodology

We implement multi-object tracking using the **Norfair** tracking library, which associates pedestrian detections across frames using **Euclidean distance-based matching**. Each detection from the YOLOv5 model is converted into a `Detection` object, and Norfair assigns consistent track IDs by minimizing spatial displacement frame-to-frame. This approach is resilient to brief occlusions and handles dense scenes effectively when combined with multi-scale detection.

Trajectory Representation

Each pedestrian is modeled as a time-evolving spatial point defined by:

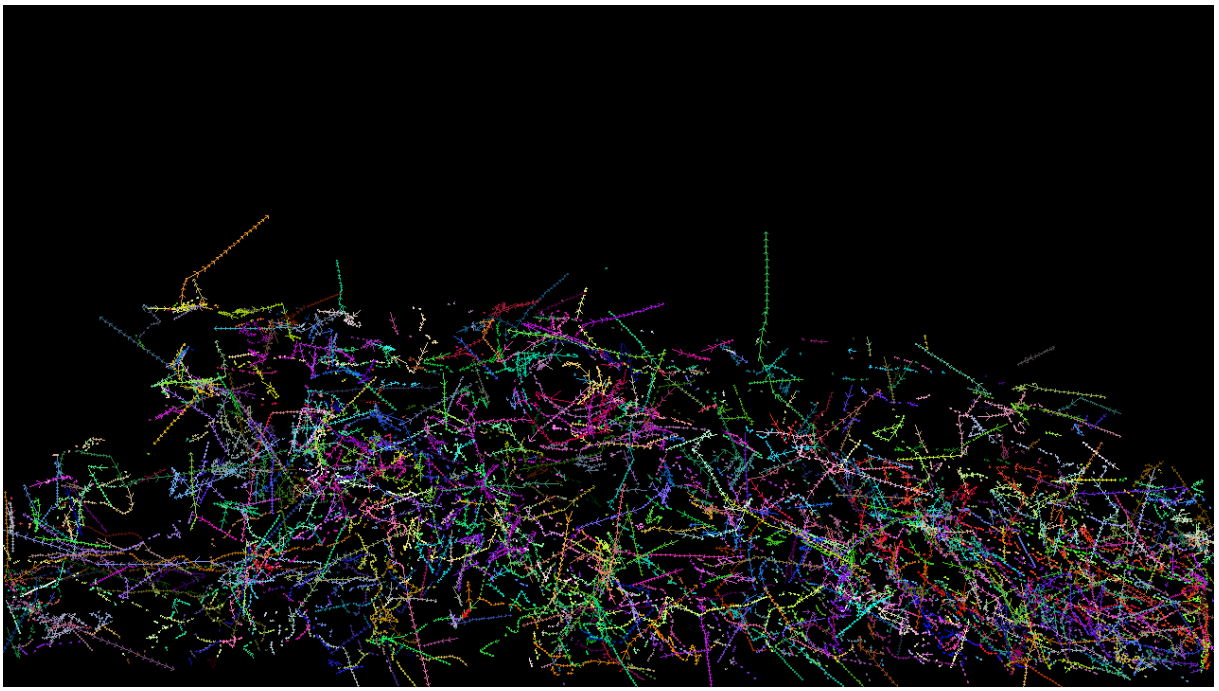
$$T_i = \{(x_t, y_t) \mid t = t_0 \cdots t_n\}$$

where (x_t, y_t) denotes the centroid of the ith bounding box at frame t.
From these trajectories, we compute key motion descriptors including:

- **Velocity vectors**: Derived from successive displacements over time.
- **Motion direction**: Calculated as the angle of travel relative to the image plane.
- **Dwell time**: Total number of frames the object remains within the frame, indicating residence time in a given zone.

These trajectories are saved as structured CSV logs and used to generate downstream visualizations such as density maps and flow fields.

4. Flow Analytics & Visual Interpretation



Final Flow Vector Mapping

Metrics

- **Trajectory Arrows:** Rendered via OpenCV primitives to indicate directionality of motion.
- **Flow Field Estimation:** Gaussian Kernel Density Estimation (KDE) is applied to pedestrian centroids to visualize spatial intensity.
- **Zone-wise Density:**

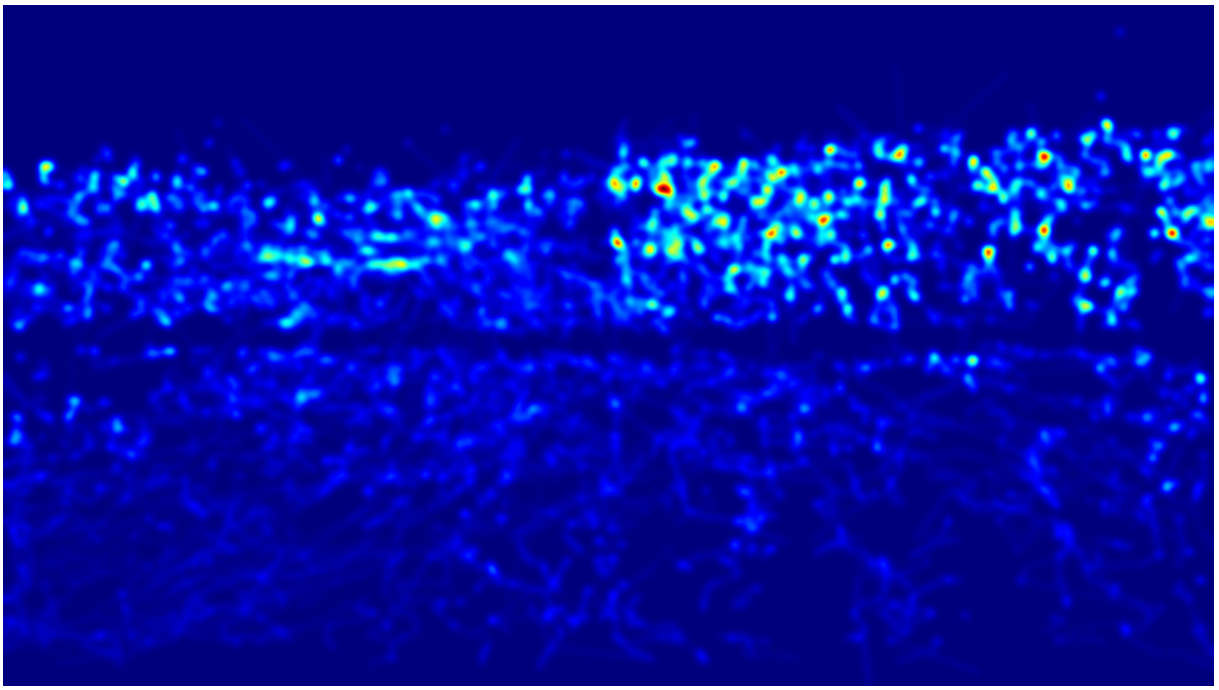
$$\rho_z(t) = \frac{N(z)/}{A_z}$$

where $N_z(t)$ is the count of tracked individuals in zone z at time t and A_z is the area.

Dashboard Integration

A Streamlit-based interface displays:

- Annotated live frames
- Density heatmaps



Density Map generated for a video

5. Panic Simulation

Simulation Framework

Our panic simulation is built on a simplified **force-based model** inspired by Social Force Model (SFM) principles, where agents respond to repulsive forces originating from a localized panic source.

- **Panic Trigger**: Panic is synthetically injected at **frame 100**, centered at a specific spatial coordinate `(600, 400)`.
- **Repulsive Influence**: Pedestrians within a defined **panic radius (150 pixels)** experience directional repulsion away from the panic origin. The intensity of this force decreases with distance, mimicking crowd dispersal under stress.
- **Trajectory Distortion**: Displacements are scaled based on inverse proximity to the panic origin, resulting in sudden directional changes for affected individuals.

📈 **Analytical Outputs**

Following panic injection, the system exhibits measurable deviations in pedestrian dynamics:

- **Increased flow divergence**: Visualized through longer and more dispersed trajectory vectors post-panic.
- **Density redistribution**: Local densities decrease near the panic origin and cluster elsewhere.
- **Motion amplification**: Stronger velocity vectors appear around the disturbance zone, reflecting abrupt pedestrian acceleration.

- **Thickened trajectory lines**: Used to visually distinguish post-panic movement from normal flow.

This module demonstrates how real-time visual analytics can reflect behavioral shifts in response to emergent events in dense crowds.

6. Stampede Detection

In our project, we designed a practical and interpretable pipeline for **Anomaly detection** using computer vision and trajectory-based analysis. We used a custom-trained YOLOv5 model for detecting heads in crowd scenes and applied Norfair for real-time multi-object tracking.

To detect risk zones and assess stampede likelihood, we extracted the following **spatiotemporal features** from the tracked pedestrian movements:

- **Instantaneous and accumulated crowd density**: computed using a Gaussian-smoothed spatial heatmap across all frames.

- **Trajectory-based motion patterns**: captured through flow vectors using displacements between sequential positions of tracked individuals.

- **Congestion hotspots**: localized by aggregating heatmap densities in fixed-size grid cells.

- **Reverse movement and irregular motion**: inferred from abrupt changes in direction or speed, particularly in the simulated panic scenarios.

- **Panic simulation modeling**: implemented using a repulsion-based force near a panic

origin, distorting trajectories realistically post-event.

High-density zones and abnormal flow behaviors were flagged as potential risk areas, and annotated on video outputs for interpretability. This framework enables both **preventive surveillance** and **post-incident analysis**.