DeepVision Crowd Monitor: AI for Density Estimation and Overcrowding Detection

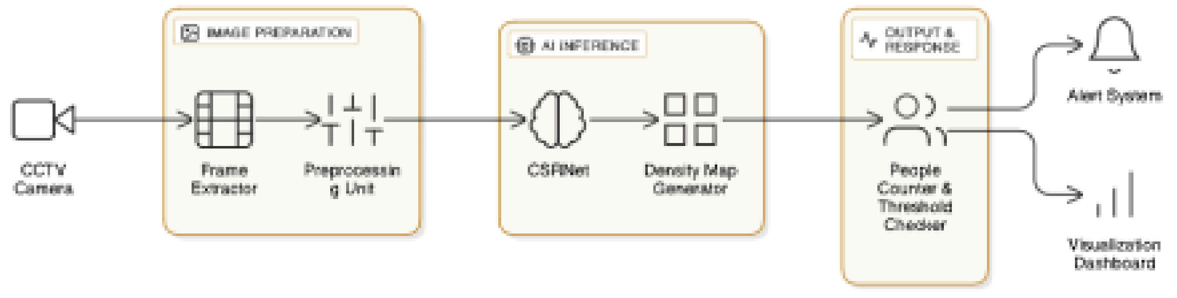
The objective of **DeepVision Crowd Monitor** is to develop a real-time deep learning-based system capable of accurately estimating crowd density and detecting overcrowded zones using surveillance video feeds. The system aims to enhance public safety, support emergency response, and optimize crowd flow management in high-footfall areas such as transit hubs, public events, religious gatherings, and smart city infrastructures.

By leveraging convolutional neural networks and crowd estimation algorithms, the project seeks to provide timely insights and automated alerts to authorities, enabling proactive interventions and efficient crowd control.

Project Workflow

1. **Video Input:** Live feed from CCTV or surveillance camera.
2. **Frame Extraction:** Sample video frames at fixed intervals.
3. **Preprocessing:** Resize, normalize, and clean image frames.
4. **Crowd Estimation:** Use deep learning (e.g., CSRNet) to generate density maps. 5. **Counting & Detection:** Estimate crowd count and flag overcrowded zones. 6. **Alert & Visual Output:** Display heatmaps and send real-time alerts if limits are crossed.

Architecture Diagram



Tech Stack

Deep Learning & Model

- CSRNet or MCNN: For crowd density estimation.

- PyTorch: For model development and inference.

Computer Vision & Processing

- OpenCV: For video capture, frame extraction, and visualization.
- NumPy, Pillow: For image manipulation.

Visualization & Alerts

- Matplotlib or Plotly: For heatmaps and overlays.
- Flask or Streamlit: For a web-based dashboard for real-time monitoring.
- SMTP / Twilio API: For sending alerts.

Deployment & Integration

- Docker: For containerization.
- Nginx: (Optional) reverse proxy for the dashboard.
- GPU Support: NVIDIA CUDA: For real-time performance.

Dataset

- ShanghaiTech Crowd Counting Dataset: Includes labeled images and density maps.

Milestone

Milestone 1: Setup and Data Preparation (Weeks 1-2)

- **Things:**
  - Development environment set up (Python, PyTorch, OpenCV, etc.).
  - ShanghaiTech Crowd Counting Dataset downloaded and preprocessed (image resizing, normalization).
  - Initial data loading and visualization scripts.
- **Evaluation:**
  ○ Successful installation and configuration of all required software and libraries. ○ Ability to load, preprocess, and display samples from the dataset without errors. ○ Documentation of the setup process and data preparation steps.

Milestone 2: Model Development and Training (Weeks 3-4)

- **Things:**
  - Implementation of CSRNet or MCNN architecture in PyTorch.
  - Training script developed and model trained on a subset of the dataset.
  - Initial density map generation and visualization from trained model.

- **Evaluation:**
  - Correct implementation of the chosen deep learning model architecture.
  - Successful training of the model with reasonable loss convergence.
  - Qualitative assessment of generated density maps (e.g., visual accuracy). ○ Initial performance metrics (e.g., Mean Absolute Error - MAE) on a validation set.

Milestone 3: Real-time Integration and Core Functionality (Weeks 5-6)

- **Things:**
  - Integration with OpenCV for live video feed processing and frame extraction.
  - Real-time crowd counting and detection of overcrowded zones.
  - Basic alert mechanism for exceeding predefined crowd limits.
- **Evaluation:**
  - Smooth and stable video input processing.
  - Accurate and consistent real-time crowd count estimations.
  - Reliable detection of overcrowded zones based on defined thresholds.
  - Functional alert trigger when limits are crossed.

Milestone 4: Dashboard, Alerts, and Deployment (Weeks 7-8)

- **Things:**
  - Web-based dashboard (Flask/Streamlit) displaying real-time density maps, crowd counts, and alert status.
  - Enhanced alert system via SMTP/Twilio API.
  - Docker containerization of the entire system for easy deployment.
  - Performance optimization with GPU support (NVIDIA CUDA).
- **Evaluation:**
  - Usability and responsiveness of the web dashboard.
  - Successful sending of alerts via chosen API.
  - Successful containerization of the application.
  - Demonstrated real-time performance and efficiency on target hardware (GPU).
  - Comprehensive documentation of the system and deployment guide.