# DEEP VISION CROWD MONITOR: AI FOR DENSITY ESTIMATION AND OVERCROWDING DETECTION

SARJANA NAGAMANIKANDAN

**Project Description:**

The objective of Deep Vision Crowd Monitor is to develop a realtime 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.

**DATASET USED:**

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

**Environment Setup, Data Exploration, and Data Preprocessing:**

The ShanghaiTech Crowd Counting Dataset is a widely used benchmark dataset designed for crowd counting and density estimation tasks. It consists of two parts: Part A, containing images of highly congested urban scenes, and Part B, consisting of relatively sparse crowd scenes from public places. Each image in the dataset is accompanied by ground-truth annotations provided as dot annotations stored inside MATLAB .mat files. These annotations indicate the locations of individuals in the image and are used to generate density maps for training deep learning models. The dataset structure typically includes separate folders for training and testing images along with corresponding ground-truth files. This dataset's diversity and accurate labeling make it ideal for developing robust crowdcounting models.

## 1. Environment Setup

For working with the ShanghaiTech Crowd Counting Dataset downloaded from Kaggle, a proper environment setup is essential to ensure that the dataset can be accessed, visualized, processed, and prepared correctly for building a crowd-counting model. The environment setup includes installing the required libraries, configuring the file paths, and preparing necessary tools to handle image data and MATLAB .mat annotation files.

**Key Components of the Environment Setup Required Python Modules**

The following libraries are essential for handling images, generating density maps, visualization, model training, and deployment:

- torch – for deep learning model development
- torchvision – for pretrained models and image transformations
- opencv-python – for image processing and resizing
- numpy – for numerical computations
- pandas – for handling metadata and tabular data
- matplotlib – for plotting and data visualization
- pillow (PIL) – for loading and manipulating images
- scipy – for reading and processing .mat ground-truth files
- scikit-learn – for splitting dataset and evaluation utilities
- plotly – for interactive visualizations
- tqdm – for progress bars during preprocessing and training
- flask – for deploying the model as a web API
- streamlit – for building interactive UI applications

- twilio – for SMS/notification integration (optional)
- requests – for external API calls
- pyyaml – for configuration file handling

1.Python Installation:

A stable version of Python (such as 3.8–3.12) is required to run the scripts. This allows compatibility with scientific libraries used for data processing.

2. Library Installation:

Kaggle's ShanghaiTech dataset requires specific libraries such as:

- numpy for numerical operations ○ pandas for metadata handling ○ matplotlib for visualization
- Pillow (PIL) for loading and manipulating images ○ scipy for reading .mat ground-truth files ○ opencv-python for image preprocessing and resizing

These dependencies must be installed using pip install -r requirements.txt or individually.

3. Folder Structure Setup:

Once the dataset is downloaded from Kaggle, it must be extracted so that the directory includes folders such as:

- part_A/train_data/images, part_A/train_data/ground_truth
- part_B/test_data/images, etc.
  This ensures that scripts can correctly load images and annotations.

4.IDE and Environment Configuration:

Tools such as VS Code or Jupyter Notebook help in running code efficiently. A virtual environment (e.g., python -m venv venv) ensures clean dependency management.

Overall, this setup ensures that all tools, libraries, and dataset files are ready for exploration and preprocessing.

## 2. Data Exploration

Data exploration is the next crucial step after setting up the environment. According to the Kaggle dataset documentation, the ShanghaiTech dataset consists of crowd images along with .mat files containing dot-annotations marking each individual in the scene. Data exploration helps understand dataset size, annotation style, and image characteristics.

**Key Objectives of Data Exploration**

1. Understanding Dataset Parts:
    - Part A: High-density crowd images collected from the internet with heavy congestion.
    - Part B: Sparse crowd scenes captured from streets and campuses.
2. Inspecting Image Properties:
    - Resolution differences between Part A and Part B ○ Image clarity, noise levels, and brightness variations

3.Analyzing Ground-Truth Annotations:

- Each .mat file contains point-level annotations ○ These points represent individual head positions ○ Understanding this helps generate density maps later

4. Dataset Size Overview:
    - Part A: ~482 images (train + test) ○ Part B: ~716 images (train + test)
5. Visualization:
    - Displaying random images
    - Overlaying dot-annotations to understand crowd distribution
    - Checking variations across scenes
6. Identifying Issues:
    - Inconsistent lighting
    - Highly congested scenes causing annotation overlap ○ Large differences in scale between individuals

Data exploration helps determine preprocessing requirements such as resizing, normalization, and density map creation.

## 3. Data Preprocessing

Data preprocessing prepares the downloaded Kaggle dataset for training crowd-counting models. Since the ShanghaiTech dataset includes raw images and .mat files containing human head annotations, several preprocessing steps are needed to convert them into model-readable formats.

**Key Preprocessing Steps**

1. Loading and Validating Dataset Files:
    - Verifying that each image has a corresponding groundtruth .mat file
    - Checking image integrity and removing corrupted samples
2. Annotation Extraction:
    - .mat files contain dot annotations stored under nested structures
    - Extracting these points provides the coordinates of each person in the image
    - These coordinates are later used to create density maps
3. Density Map Generation Using Gaussian Filter:
    - A Gaussian filter is applied to each annotated point
    - Each point is represented by a Gaussian kernel, and summing all kernels forms the density map
    - This technique smooths out the point annotations and helps models learn spatial crowd distribution effectively
4. Handling Density Map Resizing:
    - When images are resized, density maps must also be resized
    - It is crucial to multiply the resized density map by the *scaling factor squared* to preserve the total person count
    - This ensures that resizing does not distort the total density value
5. Image Resizing:
    - Image resolutions vary widely in ShanghaiTech Part A and Part B
    - Standardizing image sizes, such as 512×512 or proportional resizing, helps stabilize training performance

6. Normalization Using ImageNet Statistics:

- o For deep CNN-based models, ImageNet normalization is applied:
  -  Mean = [0.485, 0.456, 0.406]
  -  Std = [0.229, 0.224, 0.225]
- o This aligns image distributions with pretrained backbone networks

7. Data Augmentation Techniques:
To improve generalization and avoid overfitting:

- o Random horizontal/vertical flips o Random cropping
- o Brightness and contrast adjustment o Random rotation o

Gaussian noise addition

These augmentations help the model learn robust representations under different environmental conditions.

8. Train-Test Split:

- o Kaggle version already contains separate folders for training and testing
- o Ensures fair and consistent evaluation

**Purpose of Preprocessing**

- Ensure dataset uniformity
- Preserve density information during resizing
- Use Gaussian smoothing for spatial awareness
- Normalize images for pretrained networks
- Enhance model generalization through augmentation
- Prepare images and density maps for efficient training

With these preprocessing enhancements, the dataset becomes fully ready for training advanced crowd-counting models.

## 4.Model Training Process (Part A and Part B)

The ShanghaiTech Crowd Counting Dataset is divided into Part A (high-density crowd scenes) and Part B (low-density crowd scenes). Although both parts are trained using similar deep-learning procedures, the characteristics of each subset influence the preprocessing steps, model behavior, and convergence patterns.

4.1 Training Process Overview

The model training pipeline for both parts typically includes:

1. Loading preprocessed images and corresponding density maps
2. Applying transformations and normalization
3. Forward pass through the CNN-based network (e.g., CSRNet, MCNN)
4. Calculating loss between predicted and ground-truth density maps
5. Backpropagation and weight updates
6. Monitoring MAE and MSE for performance evaluation

---

4.2 Training on Part A (High-Density Scenes)

Part A contains highly congested images, making the task more complex. Training on this part requires careful handling of density maps and more robust feature extraction.

Characteristics

- Large number of individuals in each image
- High variation in scale and density distribution
- Requires stronger Gaussian kernels to represent dense regions

Training Considerations

- Smaller batch size due to high computational load
- Stronger data augmentation to generalize over complex scenes
- Careful learning rate tuning because gradients can be large due to dense annotations
- More training epochs to help the network learn fine spatial features

Training Steps

1. Load images and density maps generated with Gaussian filters
2. Apply resizing and ImageNet normalization
3. Feed input into the model
4. Compute loss using
   - MSE Loss for density regression
5. Backpropagate and update the network parameters
6. Monitor validation MAE and RMSE for early stopping

---

4.3 Training on Part B (Low-Density Scenes)

Part B contains sparse crowds with fewer people per image, making the learning process easier and faster.

Characteristics

- Fewer head annotations per image
- Clearer and less congested scenes
- Lower kernel density during map generation

Training Considerations

- Larger batch sizes can be used compared to Part A

- Simpler augmentation since images are clearer
- Fewer epochs required for convergence
- Faster learning due to less complexity

Training Steps

1. Load resized and normalized images
2. Generate density maps using Gaussian filters with appropriate sigma values
3. Forward pass through

## 5. Model Testing and Real-Time Inference – TASK 1

The trained CSRNet-based crowd counting model is tested using a real-time inference pipeline integrated with live webcam input. This testing stage validates the model's practical performance in real-world scenarios beyond offline datasets such as ShanghaiTech Part A and Part B.

### 5.1 Model Loading for Testing

During the testing phase, the trained model is loaded from a saved file (model.keras). This ensures that the same weights learned during training on Parts A and B are reused without modification. The model is set to inference mode to disable training-specific operations.

Key aspects:

- Pretrained CSRNet model is loaded using TensorFlow/Keras
- Model parameters remain fixed during testing
- Ensures consistency between training and deployment

---

### 5.2 Real-Time Input Acquisition

Instead of static test images, the testing process uses a **live webcam feed**:

- Frames are continuously captured using OpenCV
- Frame rate is controlled to match a target FPS
- Frames are processed in a separate thread for efficiency

This approach simulates real-world crowd monitoring conditions.

---

### 5.3 Frame Preprocessing During Testing

Each captured frame undergoes preprocessing identical to the training pipeline:

1. **Resizing** to a fixed input size (256×256)
2. **Normalization** of pixel values
3. **Batch dimension addition** for model compatibility

Maintaining the same preprocessing steps ensures reliable predictions.

---

### 5.4 Crowd Count Prediction

For each preprocessed frame:

- The frame is passed through the CSRNet model
- The model outputs a predicted crowd count
- Negative predictions are clipped to zero
- Predictions are smoothed using a moving average window to reduce noise

This smoothing helps stabilize predictions in dynamic scenes.

---

### 5.5 Density Map Visualization

To improve interpretability, the predicted crowd count is converted into a **density heatmap**:

- Gaussian-based hotspots are generated across the frame
- Intensity of the heatmap reflects estimated crowd density
- OpenCV's JET colormap is applied for visual clarity

This visual feedback allows users to understand spatial crowd distribution.

---

### 5.6 Performance Monitoring

The testing pipeline continuously tracks:

- Frames per second (FPS)
- Processing latency per frame
- Crowd count statistics (min, max, average)

These metrics help evaluate real-time feasibility and system efficiency.

---

**5.7 Alert and Threshold Testing**

The model output is compared against a predefined alert threshold:

- If the predicted count exceeds the threshold, an alert is triggered
- Alert status is displayed in real time
- Threshold can be dynamically adjusted during testing

This feature validates the system's usefulness for safety and crowd control applications.

---

**5.8 User Interface for Testing**

The testing phase is supported by a Gradio-based graphical interface:

- Displays live video feed
- Shows density heatmap
- Presents real-time statistics
- Allows user interaction for threshold updates

The interface enables interactive and intuitive evaluation of model performance.

---

**5.9 Testing Outcome**

Through real-time testing:

- The model demonstrates generalization beyond static datasets
- Performance on both dense and sparse crowd scenes is validated
- System stability and responsiveness are assessed

This testing stage confirms that the trained model is suitable for deployment in real-time crowd monitoring environments.

## Real-Time Crowd Monitoring Pipeline with Video Input - TASK 2

This program implements a **real-time crowd monitoring system** using **video files and YouTube videos** as input. It uses a **CSRNet-based deep learning model** to estimate the number of people present in each video frame and visualize crowd density using heatmaps.

The system supports:

- **Local video files (MP4/MOV/AVI)**
- **YouTube video download and processing**
- **Live preview and statistics via Gradio UI**

Each video frame is **preprocessed**, passed to the trained model for **crowd count prediction**, and post-processed to generate **density heatmaps** and **alert signals** when crowd size exceeds a predefined threshold.

---

**Key Functional Modules**

**1. Model Loading**

- Loads a trained **CSRNet TensorFlow model**
- Used for real-time inference
- Falls back to **synthetic crowd estimation** if the model is unavailable

**2. Frame Preprocessing**

- Resizes frames to **256×256**
- Normalizes pixel values
- Converts frames into batch format for prediction

**3. Crowd Count Prediction**

- Predicts crowd count per frame
- Applies **temporal smoothing** using a weighted moving average
- Ensures stable and realistic predictions across frames

**4. Density Heatmap Generation**

- Generates **Gaussian-based density maps**
- Visualizes crowd concentration using **JET colormap**
- Helps identify high-density zones in the scene

**5. Alert Mechanism**

- Triggers alerts when crowd count exceeds a configurable threshold
- Displays visual warnings and colored borders on video frames

**6. Video Processing**

- Reads video frame-by-frame
- Annotates frames with count, density, alert status, and progress
- Saves processed output video automatically

**7. Gradio User Interface**

- Upload local videos or process YouTube links
- Live preview of processed frames and heatmaps
- Displays real-time statistics (FPS, counts, alerts)
- Allows dynamic adjustment of alert threshold

---

## Pre-Recorded Video Processing with Synthetic Crowd Detection – TASK 3

1. Overview of the Synthetic Crowd Processing Module

This module demonstrates a complete end-to-end crowd monitoring pipeline using pre-recorded video files, even in the absence of a trained deep learning model. Instead of relying on CSRNet predictions, synthetic crowd detection logic is employed to simulate realistic crowd behavior.

The purpose of this module is to:

- Validate the video processing workflow
- Demonstrate crowd density estimation, smoothing, alerting, and visualization
- Provide a fallback or demo mode when a trained model is unavailable

This approach is useful for:

- System testing
- Demonstrations
- Educational and prototype validation

---

2. Environment and Configuration Setup

The system initializes configuration parameters that control processing behavior:

- Frame size for internal processing
- Display size for visualization
- Alert threshold to detect overcrowding
- Temporal smoothing window to stabilize predictions
- Output FPS for saved video
- Output directories for videos and statistics

All output folders are created automatically to ensure smooth execution.

---

3. Synthetic Crowd Detection Logic

3.1 Motivation for Synthetic Detection

In real-world systems, deep learning models may:

- Be unavailable during early development
- Require extensive training time
- Fail due to hardware constraints

Synthetic crowd detection provides a model-independent mechanism to simulate crowd dynamics while preserving system logic.

---

3.2 Temporal Crowd Modeling

Crowd behavior is modeled to follow realistic temporal patterns:

- Crowd density peaks near the middle of the video
- Density gradually reduces toward the start and end

This is achieved using a temporal factor, ensuring natural rise and fall patterns in crowd size.

---

3.3 Motion-Based Crowd Estimation

To estimate activity within a frame:

- Frames are converted to grayscale
- Canny edge detection is applied

- Edge density is used as a proxy for motion and crowd activity

Higher edge density corresponds to increased movement and visual complexity, indicating larger crowds.

---

### 3.4 Synthetic Density Map Generation

For visualization and spatial analysis:

- A low-resolution density map is generated
- Gaussian blobs are placed at random locations
- Blob intensity corresponds to estimated crowd size

This simulates how real density maps behave in CSRNet-based systems.

---

## 4. Density Heatmap Visualization

The synthetic density map is:

- Upscaled to match video resolution
- Normalized to a 0–1 range
- Converted into a JET colormap heatmap

The heatmap highlights:

- High-density crowd regions (red/yellow)
- Low-density areas (blue/green)

This closely resembles density visualizations produced by deep learning crowd-counting models.

---

## 5. Temporal Smoothing of Crowd Counts

Raw frame-wise predictions may fluctuate due to noise. To ensure stability:

- A moving average smoothing technique is applied
- A deque buffer stores recent predictions
- Smoothed values improve alert reliability and visual consistency

---

## 6. Crowd Alert and Risk Level Classification

Based on the smoothed crowd count:

- LOW: Safe crowd level
- MEDIUM: Approaching threshold
- HIGH: Alert condition triggered

Each level is associated with:

- Color coding
- On-screen warning messages
- Statistical tracking of alert frequency

This mimics real-world crowd safety systems used in public surveillance.

---

## 7. Video Processing Pipeline

Step-by-Step Workflow:

1. Load pre-recorded video using OpenCV
2. Extract video metadata (FPS, resolution, frame count)
3. Process frames sequentially
4. Generate synthetic crowd count and density

5. Apply smoothing and alert logic
6. Overlay heatmaps and annotations
7. Save processed video to disk
8. Record frame-wise statistics

This pipeline ensures real-time-like processing for offline videos.

---

## 8. Frame Annotation and Visualization

Each processed frame displays:

- Estimated crowd count
- Crowd risk level
- Alert messages for high-density scenarios
- FPS performance
- Processing progress

This enhances interpretability and usability for operators.

---

## 9. Statistical Analysis and Reporting

During processing, the system records:

- Crowd count per frame
- Alert occurrences
- Average, maximum, and minimum crowd levels
- Total processing time and FPS

All statistics are saved as a JSON file, enabling:

- Post-analysis
- Performance evaluation
- Report generation

---

## 10. Output Generation

The system produces:

- Annotated output video
- JSON statistics file
- Console-based progress reporting

This ensures traceability and reproducibility of results.

---

## 11. Significance of the Synthetic Module

This module:

- Demonstrates full system functionality without ML dependency
- Serves as a robust testing and fallback mechanism
- Complements CSRNet-based real-time deployment
- Enhances system reliability and development flexibility

---

## 12. Integration with the Overall Project

This synthetic pipeline aligns seamlessly with:

- CSRNet-based training (Part A & Part B)

- Real-time webcam monitoring
- Gradio-based user interface
- Alert and safety management system

Together, they form a complete intelligent crowd monitoring framework.