DeepVision Crowd Monitor: AI for Density Estimation and Overcrowding Detection

Sneha Walilkar

Table of Contents

1. Project Description

The **DeepVision Crowd Density Estimation System** is a deep-learning based computer vision project designed to accurately estimate the number of people in highly crowded scenes. The primary goal of this project is to preprocess crowd images and their corresponding ground truth density maps in a structured way so that they can be used for training a neural network.

Crowd counting is an important application in surveillance, event management, public safety, smart cities, and congestion monitoring. This project implements the full workflow from dataset loading to density map creation, making the data ready for training deep learning models such as CSRNet, MCNN, CANNet, or custom CNN architectures.

2. Dataset Used

The project uses the **ShanghaiTech Crowd Counting Dataset**, which contains two parts:

- **Part A:** Highly dense crowds (collected from the internet)
- **Part B:** Medium-density crowds (street-level images)

For each image in the dataset:

- A .jpg image file is provided.
- A .mat ground truth file contains the coordinates of each person's head.

This dataset is widely used for benchmarking modern crowd-counting models because of its high-quality annotations and diversity of crowd densities.

3. Environment Setup

The project is implemented in **Python** using**VS Code**. The key libraries required include:

- **PyTorch (torch, torchvision) –** Used to build and train the deep learning model.
- **OpenCV (opencv-python) and Pillow (pillow) –** For loading, resizing, converting, and manipulating images.
- **NumPy and Pandas –** For efficient numerical operations and tabular data handling.
- **SciPy and Scikit-learn –** For mathematical processing such as loading .mat files and performing basic analysis.
- **Matplotlib and Plotly –** Used for visualizing images, density maps, and data insights during exploration.
- **TQDM –** Provides progress bars while processing large datasets.
- **Flask and Streamlit –** Used for API creation and building the user interface for displaying predictions.
- **Requests and Twilio –** Used for integrating external services such as notifications.
- **PyYAML –** For reading structured configuration files.

A local virtual environment was created to maintain package consistency and avoid conflicts. GPU acceleration is optional but recommended for training stage.

4. Data Exploration

Before preprocessing, the dataset directory structure was explored to ensure correct configuration. The exploration involved:

4.1. Verifying dataset folder paths

- Checking whether the dataset exists in the file system
- Viewing sample image paths
- Confirming ground truth file availability

4.2. Displaying a sample image

A sample image was loaded to visually confirm its shape and quality.

4.3. Visualizing ground truth annotations

The .mat file was inspected to extract head point coordinates. The coordinates were plotted on the image to verify that annotations were correctly aligned. This ensures the dataset is correctly structured before proceeding with preprocessing

5. Data Preprocessing

This stage involves converting raw dataset images and .mat annotations into deep-learning-ready inputs. The preprocessing pipeline includes the following key steps:

5.1. Image Loading

Images are loaded using OpenCV, which reads them in **BGR format**. The image is then converted to **RGB** for visualization consistency.

Purpose:

- Ensures the image is loaded in the correct color space
- Provides a consistent input format for further operations

5.2. Ground Truth Extraction

Each .mat file contains head annotations in the form of coordinate points.

Steps:

- Load .mat file using SciPy
- Access the annotation structure (usually under image_info → annPoints)
- Extract all (x, y) coordinate pairs representing head positions

Purpose:

- Converts raw MATLAB annotation data into usable Python arrays
- Provides exact locations for density map creation

5.3. Image Resizing & Downscaling

Original images are often large (e.g., 1024×768, 1200×900). For training efficiency and consistent model input, images are resized.

Two key resizing operations are used:

(1) Upscale to 1024×1024

Ensures all images have identical resolution.

(2) Downscale by factor of 2 → 512×512

This is the **final default resolution** used for training.

Purpose:

- Reduces memory usage
- Speeds up training
- Maintains uniform input dimensions for neural networks

5.4. Density Map Generation

This is the most important preprocessing stage.

A **density map** represents the distribution of people in the image. Each annotated point is replaced by a **Gaussian kernel** (based on a sigma value). The final output is a heatmap-like representation.

Steps:

1. Create an empty matrix of the same shape as the image
2. For each head annotation point, place a value at that location
3. Apply a Gaussian filter to spread that value over a small region

Purpose:

- Allows the model to learn *spatial presence* of people
- Enables the network to estimate crowd count by summing density map values

Density maps are essential for crowd-counting architectures like CSRNet.

5.5. Tensor Conversion

Deep learning models require tensor inputs.

Two tensors are created:

- **Image Tensor (C × H × W)** → normalized image
- **Density Map Tensor (1 × H × W)** → target output

Purpose:

- Converts NumPy arrays into PyTorch tensors
- Prepares input-output pairs for training

Both tensors are stored or passed directly to the training loop.

6. Proposed System

1. Raw High-Resolution Image

The system starts with the original crowd image captured from real scenes.
These images contain varying densities, lighting conditions, and resolutions that require standardization.

2. Preprocessing (Resize & Normalize)

The image is resized to a fixed dimension and normalized using ImageNet statistics.
This ensures consistent input quality, stabilizes training, and helps CSRNet learn effectively.

3. CSRNet (Model Training / Inference)

The preprocessed image is then passed to the CSRNet architecture, which consists of:

- VGG-16 front-end for feature extraction
- Dilated convolution back-end for enhancing receptive fields
  The model learns dense spatial features required to identify crowded regions and estimate the number of people.

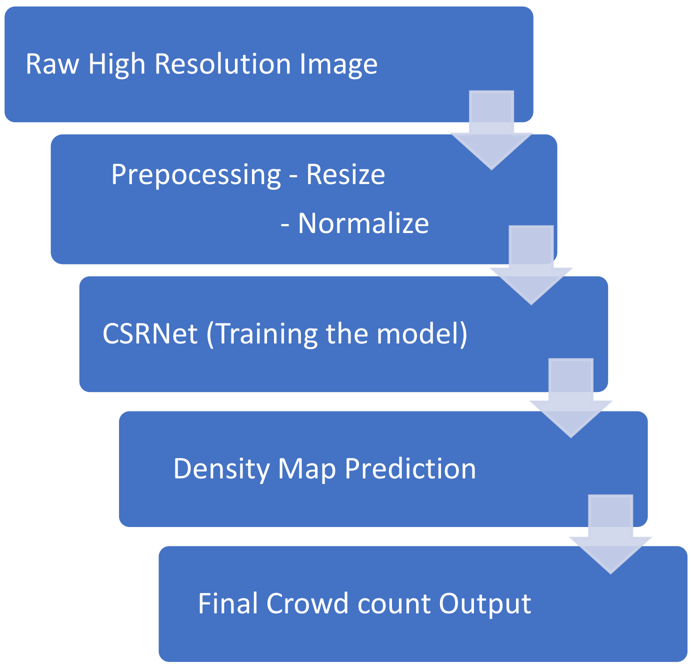

Figure. System Architecture

4. **Density Map Prediction**

The model outputs a density map where each pixel represents local crowd density.
Brighter regions indicate higher people concentration.

5. **Final Crowd Count Output**

The density map values are summed (integrated) to estimate the total number of people in the image.
This final numerical output is used for evaluation and comparison with the ground truth.

7. Training Pipeline

Training script includes:

Features:

- Checkpoint saving after each epoch
- Auto-resume from last checkpoint
- Patch merging for inference
- Evaluation metrics calculation

Why patch-based training?

- CSRNet is sensitive to resolution. Patches preserve high detail.

Why VGG-16 frontend?

- CSRNet expects VGG-16 backbone to extract low-level crowd features.

Results for Part A:

- **MAE = 59.26**
- **MSE = 8408.63**
- **RMSE = 91.70**

Results for Part B:

- **MAE = 49.39**
- **MSE = 4093.74**
- **RMSE = 63.98**

8. Conclusion

The preprocessing stage of our CSRNet-based Crowd Density Estimation project was completed successfully. All high-resolution images and their ground-truth point annotations were loaded, visualized, and analyzed to understand the variations in scene density. Each image was then resized, normalized using ImageNet statistics, and converted into tensors suitable for CSRNet. For every image, high-quality Gaussian density maps were generated from the annotation points, ensuring accurate spatial representation of crowd distribution. These preprocessed image–density map pairs formed the input for both training and evaluation of our model on the ShanghaiTech Part A and Part B datasets.