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. Conclusion

The preprocessing and data exploration stages of the DeepVision Crowd Density Estimation project have been successfully completed. All images and their corresponding ground-truth annotations were loaded, visualized, and thoroughly analyzed to understand the dataset structure. The images were then resized, downscaled, normalized, and converted into tensors, while accurate Gaussian-based density maps were generated for every image - following standard practices used in modern crowd-counting research.