Deep Vision Crowd Monitor: AI for Density Estimation and Overcrowding Detection

MahaLakshmi Tetala

Table of Contents

1. Project Overview

The DeepVision Crowd Monitor project focuses on creating a preprocessing pipeline that prepares crowd images for deep learning–based density estimation. Instead of directly counting people in an image, the system transforms visual inputs into numerical representations that a model can learn from.

The goal is not to train the model, but to prepare accurate inputs—images, ground-truth points, density maps, and tensors. These processed inputs will later be used by architectures like CSRNet or MCNN.

This preprocessing pipeline is crucial in real-world applications such as security monitoring, event management, and congestion analysis, where accurate density information is valuable.

2. Dataset Information

The dataset used is the **Shanghai Tech Crowd Counting Dataset** consisting of real images paired with annotations (.mat files).

Part A – Highly congested scenes

Part B – Medium-density outdoor images

Each .mat file contains pixel coordinates for each person's head with helps in counting and generating density maps

3. Environment Setup

The project uses Python in VS Code or Jupyter Notebook with a virtual environment. Main tools and libraries:

- **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

GPU acceleration is optional but preprocessing is fast even on CPU.

4. Data Exploration

Before converting images into training-ready data, the dataset must be explored to ensure everything is present and correctly arranged.

Exploration steps performed:

1. **Folder Inspection**

Verified the path structure, file counts, and that each image has a corresponding. mat annotation.

2. **Sample Image Viewing**

A few images were displayed to confirm resolution, clarity, and the general spread of people.

3. **Annotation Inspection**

Using the RobustMatReader, annotation coordinates were extracted and plotted over sample images.

This helps confirm that the points align perfectly with visible heads, ensuring the dataset is correctly annotated.

5. Data Preprocessing

The preprocessing workflow converts raw dataset inputs into model-ready formats. Steps include image loading, annotation extraction, resizing, density map generation, and tensor conversion.

5.1 Image Loading

Images are loaded using cv2.imread (BGR format), then:

- converted to RGB for consistency
- optionally transformed to floating-point values
- resized to a uniform working size

This ensures consistency and prepares images for normalization.

5.2 Annotation Extraction

Using RobustMatReader, .mat annotation files are opened safely. Head coordinates are extracted from the 'image_info' structure and converted into Python lists/NumPy arrays. This ensures compatibility even if annotation formats differ.

5.3 Resizing and Downscaling

- To standardize data:
  Initial resize - The image is resized to a larger fixed resolution (e.g., 1024×1024) to standardize shape
- **Downscaling**
  The image is then reduced to 512×512 or your preferred working size.

This reduces memory usage and speeds up further processing.

5.4 Image Loading

Images are loaded using cv2.imread (BGR format), then:

- converted to RGB for consistency
- optionally transformed to floating-point values
- resized to a uniform working size

This ensures consistency and prepares images for normalization.

5.5 Annotation Extraction

Using RobustMatReader, .mat annotation files are opened safely. Head coordinates are extracted from the 'image_info' structure and converted into Python lists/NumPy arrays. This ensures compatibility even if annotation formats differ.

5.6 Resizing and Downscaling

To standardize data:

- **Initial resize** - The image is resized to a larger fixed resolution (e.g., 1024×1024) to standardize shape
- **Downscaling**
  The image is then reduced to 512×512 or your preferred working size.

This reduces memory usage and speeds up further processing.

5.7 Implementing Robust mat Reader

A RobustMatReader (SciPy-based safe loader) is used to ensure error-free extraction, even if annotation structures vary. These points serve as the foundation for generating Gaussian density maps.

5.8 Density Map Generation

Density maps convert sparse head points into smooth distributions. Steps:

- Create a blank map filled with zeros.
- For each head coordinate, place a value of 1 at that location (using integer index rounding).
- Apply a Gaussian filter over the entire grid to convert sharp dots into smooth clusters.

Gaussian sigma defines the spread of each density blob.
The sum of the density map approximates the crowd count.

5.9 Tensor Conversion & Normalization

Images are converted to tensors in shape C × H × W.

Normalized using ImageNet

statistics: mean = [0.485, 0.456, 0.406]

std = [0.229, 0.224, 0.225]

Density maps become 1 × H × W tensors.

This ensures compatibility with PyTorch-based crowd counting models.

6.Training & Testing pipeline

Shanghai Tech Dataset – Part A & Part B

Part A

- Contains very dense crowds with many overlapping people.
- Harder for the model → lower accuracy, higher MAE/RMSE.
- More training is needed because heads are small and tightly packed.

Part B

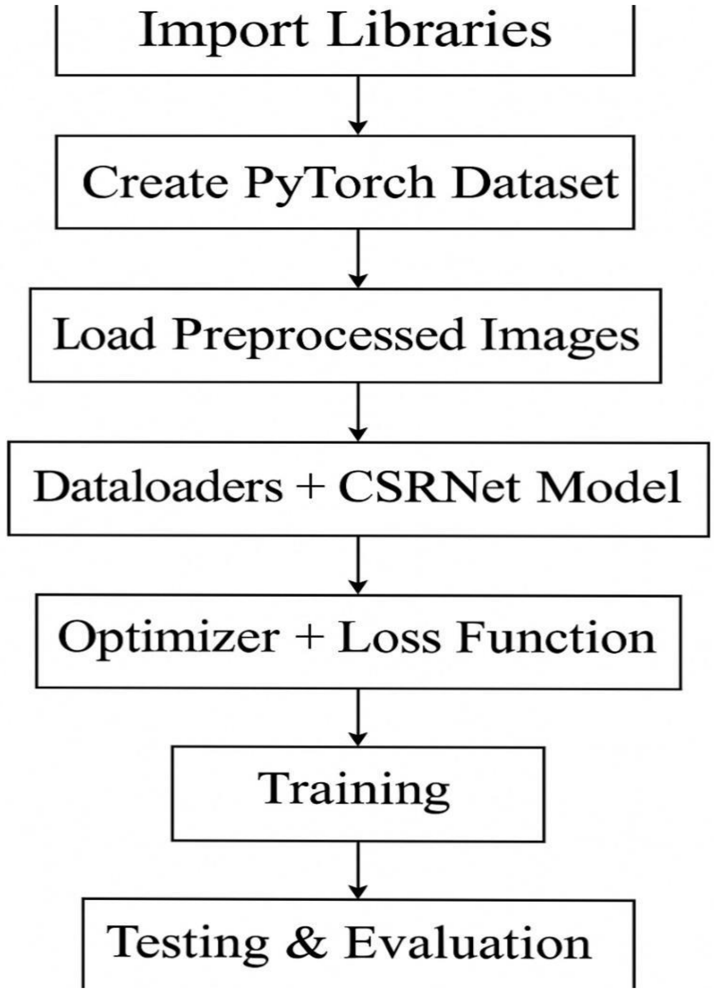
- Contains less dense, clearer outdoor scenes.
- Easier for the model → higher accuracy, lower MAE/RMSE.
- People are more visible and spaced out, so predictions are more reliable.

What I did during training

- Loaded preprocessed images and density maps.
- Created PyTorch Dataset & Dataloaders for batching and efficient training.
- Loaded the CSRNet model for density map prediction.

Why CSRNet is used

- CSRNet uses a VGG-16 frontend for strong feature extraction.
- Uses dilated convolution backend to capture large-scale crowd information.
- Designed specifically for crowd counting, making it well-suited for Part A & B.



Flow diagram

Additional training details

- Used MSE Loss and Adam Optimizer.
- Trained for multiple epochs until the loss stabilized.
- Saved the best checkpoint based on validation accuracy.

Testing Process

- Loaded test images from Part A and Part B.
- Generated predicted density maps and summed them to get counts.
- Calculated MAE and RMSE to measure performance.

Observed that:

1. Part A → lower accuracy (high crowd density).

Results —

- MAE: 78.89
- RMSE: 122.78
- Counting Accuracy: 80.47 %

2. Part B → higher accuracy (compared to Part A - less crowd density).

Results--

- MAE: 11.59
- RMSE: 19.57
- Counting Accuracy: 91.12%

7.Conclusion

In this project, the Shanghai Tech Part A and Part B datasets were used to build a complete crowd-counting pipeline. The preprocessing phase converted images and annotations into normalized tensors and density maps. CSRNet was chosen for training because its VGG16 frontend and dilated backend work well for both dense and sparse crowds.

During testing, the model performed better on Part B due to lower crowd density, while Part A showed higher errors because of heavy occlusion. Overall, the system successfully learned to predict density maps and estimate crowd counts, demonstrating an effective deep-learning approach for crowd monitoring.