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

Bolakonda Anvitha

Table of Contents

1. Project Overview

The “Deep Vision Crowd Monitor” project focuses on preparing data for a deep-learning model that estimates crowd density in highly congested environments.

This preprocessing phase transforms raw images and their annotations into standardized tensors that a neural network can learn from.

The overall goals of this stage are to:

- Understand the dataset through visualization.
- Convert all images into a consistent format.
- Generate reliable density maps from head-point annotations.
- Downsample maps to match CNN output structure.
- Build a PyTorch-compatible dataset for easy training.

This phase establishes the foundation for accurate crowd-count prediction in the advanced stages of the project.

2. Description of Dataset

For this milestone, the ShanghaiTech Part A dataset is used. It contains:

- Images captured in dense urban environments.
- Several hundreds to thousands of people per scene.

- Ground-truth annotations stored in .mat files.
- Each annotation file includes a list of all visible head coordinates.

Part A is challenging due to extremely crowded scenes and large variations in perspective, making it an ideal benchmark for learning crowd estimation.

3. System/Environment Setup

The processing pipeline is built using:

- Python 3.9
- **OpenCV :** computer vision toolkit for handling videos and images
  -it is used to handle video frame extraction and displaying bounding boxes or heat maps in real time.
- **SciPy** for handling **MATLAB** .mat annotation files.
- **NumPy** for vectorized numerical operations
  -it is used to manage images data as a risk and perform quick operations like normalisation and matrix multiplication.
- **PyTorch** and **torchvision** for tensor creation and normalization.
- **Matplotlib** for visualization
  -so showing crowd images density maps and training progress curve.
- **Pillow(pl):** image manipulation
  -image preprocessing before feeding them into the model.

- **Scikit-learn:** machine learning and matrix toolkit
  -to evaluate models accuracy and analyse prediction performance.

- **Plotly:** interactive visualisation library
  -you can use it in streamlit or flask dashboard to show life crowd count graphs or heat maps.

- **tqdm:** for monitoring progress
  -while training a model or looping over images tqdm will help to show progress bar.

- **Torch vision:** pytorch helper library for computer vision using it to load images apply transformation and potentially use transfer learning from pre- trained cnn.

- **Pandas :** data handling and analysis
  -might be used to track model performance log results or manage metadata like image path and predicted crowd count.

- **Flask:** lightweight backend frame
  -can be used to create a real time monitoring dashboard that display crowd count a lots and heat maps in the browser.

- **Streamlit:** fastest way to make web or data apps with python
  -it is used to build a interactive crowd monitoring dashboard using streamlit showing live frames density maps and alerts .

- **Twilio:** communication API for sending SMS calls or WhatsApp alerts
  -when an overcrowded area is detected twilio can send an automatic SMS alert to the control team.

- **Requests:** http library for making web request
  -used for API calls example sending alerts retrieving data from and online feed or integrating with web development.

- **Pyyaml:** reading configuration files
  -used to load configuration parameters examples the growth threshold value data path or model weights.

All steps are executed inside a Jupyter Notebook environment.

4. Exploration of Training Data

Before beginning preprocessing, the dataset was explored to verify structure and consistency.

Image Inspection

A few images were displayed to understand:

- Lighting and clarity
- Crowd density levels
- Range of resolutions

Annotation Verification

Each .mat file was opened and the head-point coordinates were visualized over the image.

This validated that:

- Annotation files are correctly linked
- The points align accurately with visible heads

- All images contain valid ground-truth data

This step ensures there are no missing or corrupted files.

5. Preprocessing Workflow

This section describes how every image is converted into training-ready tensors.

5.1 Loading and Inspecting Images

Images are read using OpenCV.

Since OpenCV loads images in BGR, they are converted into RGB to match the expected input format of deep-learning models.

Pixel values are scaled to the 0–1 range, making them suitable for normalization.

5.2 Reading Ground-Truth Annotation Files

Every image has a corresponding .mat file named in the format:

- GT_IMG_xxx.mat
- These files store all head-point coordinates.
- The coordinates are extracted and used to create Gaussian-based density maps.

5.3 Image Resizing (512 × 512)

To maintain uniform size across the dataset:

- Every image is resized to 512 × 512 pixels.
- All annotation coordinates are rescaled proportionally.
- Resizing ensures that both the input images and their density maps share the same dimensions, which is essential for stable CNN training.

5.4 Generating Gaussian Density Maps

A density map highlights how heavily populated each region is.

The process involves:

1. Creating a blank matrix of zeros.
2. Marking each head location with a value of 1.
3. Applying a Gaussian filter (sigma = 15).

This converts each marked point into a smooth density distribution.

The integral (sum) of the density map equals the number of people in the image.

5.5 Downsampling Density Maps

Neural networks such as CSRNet produce a feature map output that is 1/8th the resolution of the input.

Therefore, each full-resolution (512×512) density map is downsampled to:

➔ 64 × 64

To preserve the total person count, the map is scaled by:

➔ $8 \times 8 = 64$

This step ensures the model's predicted density sum correctly reflects the real count.

5.6 Image Normalization

Before converting images into tensors, standard normalization is applied using ImageNet statistics:

- Mean: [0.485, 0.456, 0.406]
- Standard deviation: [0.229, 0.224, 0.225]

This step aligns the images with the distribution expected by pretrained convolutional layers.

5.7 Converting to PyTorch Tensors

Two tensors are created for each sample:

1. Image Tensor

- Shape: [3, 512, 512]
- Contains normalized RGB values.

2. Ground-truth Density Tensor

- Shape: [64, 64]
- Contains the downsampled density map.

These tensors are saved using torch.save() for efficient loading.

5.8 Building the PyTorch Dataset and Dataloader

A custom dataset class is implemented to load paired tensors:

- Input image tensor
- Ground-truth density tensor

The dataloader enables:

- Mini-batching
- Shuffling
- Efficient GPU/CPU data feeding

This structure is essential for training deep CNN models.

6. Conclusion

The preprocessing stage of the Deep Vision Crowd Monitor project successfully transformed the raw ShanghaiTech Part A train dataset into a structured, model-ready format. Through visualization, annotation verification, density-map generation, image normalization, and dataloader creation, the dataset is now fully prepared for deep-learning based crowd estimation.This phase provided a clear understanding of the dataset characteristics, the behavior of Gaussian density maps, and the importance of consistent preprocessing for reliable model training. The processed tensors, downsampled density maps, and organized dataloader collectively form a strong foundation for the next phase: designing and training the neural network to predict crowd density and detect overcrowding.With the preprocessing pipeline complete, the project is now ready to progress toward model development, evaluation, and deployment.