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

Kavya Mehta

Table of Contents

1. Introduction

We uses the ShanghaiTech Crowd Counting Dataset to build a density-based crowd counting model. Before training any deep learning model, the input images and ground-truth annotations must be preprocessed into a format compatible with deep learning architectures such as VGG-based models.

2. Loading the Data

Images were loaded using OpenCV's imread function. Since OpenCV loads images in BGR format but PyTorch and Matplotlib expect RGB, a conversion step was applied. This ensures accurate color representation during visualization and training.

3. Normalizing the Image

The image was first scaled to a floating-point range of 0–1. This helps the neural network handle input values more efficiently. ImageNet normalization was also applied, using mean = [0.485, 0.456, 0.406] and std = [0.229, 0.224, 0.225], because VGG was originally trained on ImageNet with these exact statistics.

4. Loading the Ground-Truth Data

ShanghaiTech provides head annotations in .mat files. These files contain (x, y) coordinates for each person's head. These points serve as the foundation for generating the density map used as ground truth during training, they also help in generating the head count

5. Creating the Density Map

An empty density map with the same spatial dimensions as the image was initialized. Each head annotation was placed as a pixel with intensity 1. A Gaussian filter (sigma = 7) was applied to convert the discrete points into a smooth density distribution. A larger sigma value (7) is suitable for Part A of the dataset where crowd density is high and clusters are closer.

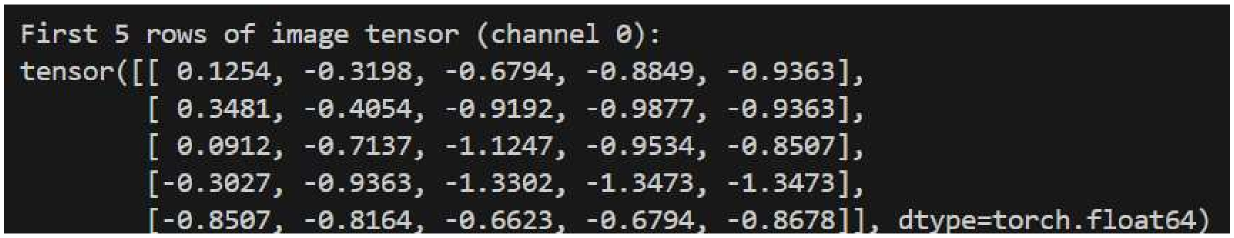
6. Downsampling the Density Map

The density map was downsampled by a factor of 8 in both height and width to match the output resolution of VGG-based models (which inherently downsample using pooling). Since downsampling reduces area by 1/64, the density map was multiplied by 64 to preserve the total person count.

7. Converting to PyTorch Tensors

The normalized image was converted into a tensor with shape (3, H, W). The density map was also converted into a tensor with shape (1, H/8, W/8). This ensures compatibility with PyTorch models during training and inference.

Sample tensors:

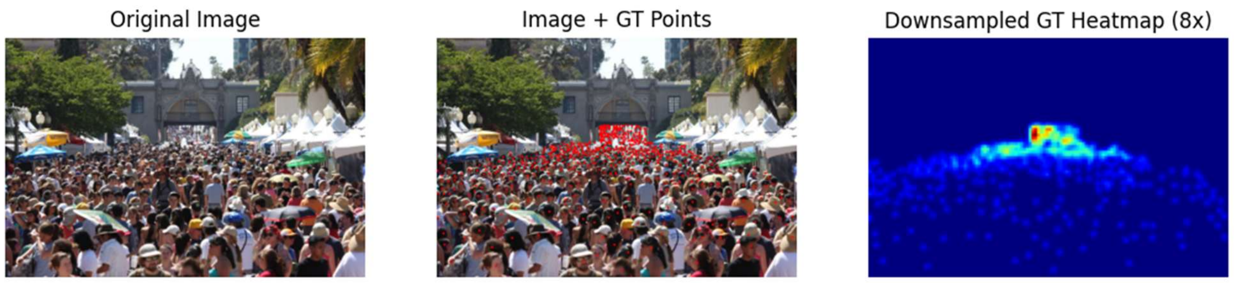


```
First 5 rows of image tensor (channel 0):
tensor([[ 0.1254, -0.3198, -0.6794, -0.8849, -0.9363],
        [ 0.3481, -0.4054, -0.9192, -0.9877, -0.9363],
        [ 0.0912, -0.7137, -1.1247, -0.9534, -0.8507],
        [-0.3027, -0.9363, -1.3302, -1.3473, -1.3473],
        [-0.8507, -0.8164, -0.6623, -0.6794, -0.8678]], dtype=torch.float64)
```

8. Visualizations

Three visual outputs were generated: (1) the original image, (2) the image with ground-truth points overlaid, and (3) the downsampled density heatmap. These visualizations help verify that preprocessing was performed correctly.

Sample:



9. Summary of Preprocessing Pipeline

The preprocessing pipeline includes: loading image and GT data, color space conversion, scaling, ImageNet normalization, density map generation, Gaussian smoothing, downsampling with count preservation, and conversion to tensors. This ensures the model receives consistent, meaningful inputs.