Table of Contents

AI DeepVision – Crowd Monitoring Project

1. Introduction

Crowd monitoring is an important task for public safety in areas such as malls, stadiums, festivals, and public transport. Manual monitoring is slow and inaccurate. Modern computer vision enables automated crowd density estimation using deep learning.

This project builds an AI-based system that:

- Estimates number of people in an image
- Generates density heatmaps showing crowd distribution
- Detects overcrowding levels

2. Dataset Description

We use the ShanghaiTech Crowd Counting Dataset and it has two parts:

part	characteristics	difficulty
Part-A	Highly crowded scenes, dense crowds	Hard
Part-B	Outdoor scenes, fewer people	Easier

Each sample contains:

- RGB Image
- Ground truth (.mat file) containing **(x, y)** head locations

The folder structure looks like:

|- ShanghaiTech

|- part-A

|- test_data

|- ground-truth/

|- images/

|- train_data

|- ground-truth/

|- images/

|- part-B

|- test_data

|- ground-truth/

|- images/

|- train_data

|- ground-truth/

|- images/

3. Problem With Raw Ground Truth

Ground-truth gives points only => not suitable for training CNN directly.

So, we convert points => **density maps**

Each head point contributes to a **Gaussian peak** => sum of density = **count of people**

4. Preprocessing Pipeline

Performed on both Part-A and Part-B data using the following steps:

Step	Purpose
Convert BGR – RGB	Standard colour format for deep learning
Resize Images (512×512)	Uniform input size
Create Impulse Map (zeros + head locations=1)	Binary head map
Gaussian Blur (σ=1.5–4)	Converts impulses → density distribution
Downsample to 1/8 resolution	To match CSRNet output
Save as .npy	Efficient numeric storage

Sum of density map = number of people in image

5. Density Map Visualization

We displayed:

- Image with head annotations
- Impulse map (binary points)
- Density map (heat style)
- Heatmap overlay on original image

This helps verify dataset correctness.

6. Why Downsample Density Map?

CSRNet has pooling layers → output is **1/8 size** of input.

Input: 512 × 512

Output: 64 × 64

So, GT density must be downsampled to same size.

To preserve crowd count, multiply by 64 (8×8).

7. Dataset Splitting

For training the model, split the training data to 80% for training and 20% for testing

And Testing data keeps separate.

8. Model Architecture – CSRNet

CSRNet = **Convolutional Neural Network** for crowd counting.

Component	Description
Frontend	VGG-16 style feature extractor
Backend	Dilated CNN to enlarge receptive field
Output Layer	1-channel density map

CSRNet is used because It

- Works for both dense & sparse crowds
- No detection required → faster
- Accurate counting from density estimation

9. Training Setup

Feature	Value
Loss Function	MSE Loss (pixel-wise density difference)
Optimizer	Adam
Learning Rate	1e-5
Batch Size	4
Epochs	100

Monitoring Metrics

- Train Loss
- Validation Loss
- Train MAE
- Validation MAE

MAE → Measures counting accuracy:

MAE = |Predicted Count – Actual Count|

For part-A, the test results are
MAE : 98.13
MSE : 19226.94
RMSE : 138.66

For part-B, the test results are
MAE : 22.83
MSE : 1190.72
RMSE : 34.51

10. CSRNet Crowd Counting — Model Loading, Image Testing, Video Processing & Webcam Inference

a. Loading the CSRNet Model & Trained Weights

After training CSRNet on ShanghaiTech Part-A/Part-B datasets, the trained .pth model files are loaded into the system.
This allows the model to perform inference without retraining.

Steps:

1. Import CSRNet model architecture.
2. Move the model to CPU device.

3. Load the trained weights (CSRNet_best.pth or CSRNet_best_2.pth).
4. Set model to evaluation mode.

Purpose:

✓ Load the trained model

✓ Avoid backpropagation

✓ Enable prediction on new images/videos

b. Testing Prediction on a Single Image

To verify that the model works correctly after loading, a simple image is tested.

Steps:

1. Read image using OpenCV.
2. Convert BGR → RGB (CSRNet expects RGB).
3. Normalize using ImageNet mean and std.
4. Convert image into PyTorch tensor.
5. Feed the tensor into CSRNet.
6. Get the density map and compute total count.
7. Overlay heatmap on original image for visualization.

Purpose:

✓ Confirm the model predicts count properly

✓ Understand density map behavior

✓ Visualize crowd intensity on image

c. Video Processing (Frame-by-Frame Crowd Counting)

CSRNet is applied on each frame of a video to generate real-time predictions.

Steps:

1. Open a video file using cv2.VideoCapture()
2. Loop through video frame-by-frame
3. Preprocess each frame
4. Predict density map
5. Sum density values to get person count
6. Generate heatmap and overlay on frame
7. Display frame inside notebook
8. Continue until video ends

Purpose:

✓ Handle dynamic scenes

✓ Demonstrate CSRNet on real video

✓ Produce visual heatmaps for each frame

d. Real-Time Crowd Counting with Live Webcam

After video prediction works, the next step is live webcam inference.

Steps:

1. Open webcam feed (VideoCapture(0))
2. Read frames continuously
3. Resize frame to match CSRNet crowd scale

4. Preprocess frame
5. Predict density map
6. Clip negative density values (avoid negative counts)
7. Compute final count
8. Overlay heatmap + display live
9. Exit when user presses q

Purpose:

✓ Create a real-time crowd monitoring system

✓ Display real-time heatmap and person count

✓ Demonstrate final working model in practical use