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

Intern Name: Nithya Shree S

TABLE OF CONTENT

1. Project Overview

The project “Deep Vision Crowd Monitor” is designed to accurately estimate crowd density and detect overcrowding in public spaces using AI-based computer vision. By leveraging deep learning techniques, the system can process images or video frames to generate density maps, which indicate the number of people in each region of an image.

Key Objectives:

- Accurate Crowd Estimation: Measure the number of people in a scene for safety and planning purposes.
- Overcrowding Detection: Identify high-density areas in real-time to prevent hazards during events or public gatherings.
- AI Model Training: Develop a CSRNet (Convolutional Neural Network) to predict crowd density from input images.
- Scalable Framework: Provide a modular workflow for training, testing, and evaluation suitable for future expansion to real-time systems or multiple datasets.

Rationale:

Crowd analysis is critical in urban safety, traffic management, and event monitoring. Traditional manual counting is time-consuming and inaccurate, whereas deep learning models like CSRNet allow automatic, scalable, and efficient crowd estimation.

2. Required Modules

The project uses Python and popular deep learning, data processing, and visualization libraries:

Library	**Purpose**
PyTorch & torchvision	Core deep learning framework for model definition, training, and evaluation
OpenCV & Pillow	Image loading, preprocessing, resizing, and visualization
NumPy & SciPy	Numerical computation, matrix operations, and scientific calculations
Pandas	Dataset handling, CSV or metadata management
Matplotlib & Plotly	Visualization of density maps, losses, and plots
scikit-learn	Evaluation metrics, normalization, and preprocessing utilities
tqdm	Progress bars for training loops
Others (Optional for future)	Flask/Streamlit for dashboards, Twilio for notifications, PyYAML for configuration

Notes:

- Environment reproducibility is ensured via requirements.txt.
- GPU acceleration is recommended for faster training.

Rationale:

Selecting these libraries allows efficient model development, easy visualization of results, and compatibility with both CPU and GPU environments.

3. Preprocessing Pipeline

The preprocessing step is critical for high-quality model performance. For both Part A and Part B datasets, the following procedures are applied:

3.1 Image Loading and Normalization

- Images are loaded in RGB format to ensure compatibility with pretrained CNNs.
- Pixel values are scaled between 0–1 to normalize intensity.
- ImageNet normalization is applied: mean = [0.485, 0.456, 0.406], std = [0.229, 0.224, 0.225]. This is essential because CSRNet uses a VGG16 backbone pretrained on ImageNet.

3.2 Ground Truth Preparation

- Density maps (.mat or .npy) are loaded for each image.
- Downsampled by 8× to reduce computational load while retaining spatial density information.
- Multiplying by 64 preserves total crowd counts after downsampling.

3.3 Conversion to PyTorch Tensors

- Both images and density maps are converted to PyTorch tensors for GPU compatibility.
- Tensors maintain channel-first convention (C×H×W), as required by PyTorch models.

Rationale:

Proper preprocessing ensures model convergence, faster training, and accurate density predictions. Without normalization and GT downsampling, the model may fail to learn patterns or produce inaccurate counts.

4. Dataset Preparation (Part A & Part B)

4.1 Folder Structure

The datasets are organized as:

```
Dataset/
  part_A/
    train_data/
      images/
      ground-truth/
    test_data/
      images/
      ground-truth/
  part_B/
    train_data/
      images/
      ground-truth/
    test_data/
      images/
      ground-truth/
```

4.2 PyTorch Dataset & DataLoader

- A custom Dataset class reads images and GT density maps.
- Training DataLoader supports batching, shuffling, and GPU compatibility.
- Part A and Part B datasets can be combined for training to increase model robustness.
- Test sets are loaded separately for evaluation without shuffling.

Rationale:

Combining both datasets improves generalization and allows the model to handle diverse crowd densities and perspectives, which is critical for accurate predictions across real-world scenarios.

5. CSRNet Model Overview

CSRNet consists of two main parts:

1. Frontend (Feature Extractor)
   - Uses pretrained VGG16 convolutional layers (first 23 layers).
   - Extracts low-level and mid-level features from input images.
2. Backend (Dilated Convolution Layers)
   - Multiple dilated convolution layers capture global context and handle dense crowd regions.
   - Produces single-channel density map corresponding to crowd counts.

Output:

- A density map where the sum of pixel values estimates the total number of people in the image.

Rationale:
CSRNet is designed to handle highly congested scenes and provides accurate density estimation, even when people are occluded or partially visible.

6. Training Process

6.1 Workflow

1. Load Part A + Part B training datasets.
2. Initialize CSRNet model and send to GPU.
3. Define loss function: Mean Squared Error (MSELoss).
4. Optimizer: Adam with learning rate = 1e-5.
5. Train for recommended epochs:
    - 50–150 epochs for early results
    - 300 epochs for full convergence
6. Track batch and epoch loss for monitoring.

6.2 Observations

- Loss should gradually decrease.
- Avoid NaN or Inf values.
- Check that training does not freeze or diverge.

6.3 Model Saving

- Save final weights as csrnet_weights.pth.
- Retain the file for testing and further experiments.

Rationale:

Proper training ensures that the model learns accurate crowd representations across both datasets. Loss tracking helps detect overfitting or underfitting.

7. Testing and Evaluation

7.1 Evaluation Steps

1. Load trained model (csrnet_weights.pth).
2. Test on both Part A and Part B test sets.
3. Compute metrics:
   - MAE (Mean Absolute Error)
   - RMSE (Root Mean Squared Error)
4. Compare predicted vs GT counts.
5. Visualize predicted density maps for qualitative assessment.

Target:

- Achieve >80% accuracy in density estimation across both datasets.

Rationale:

Testing on both datasets ensures robustness across multiple environments and crowd scenarios, validating the model's real-world applicability.

8. Next Steps

- Integrate preprocessing, training, and testing notebooks for end-to-end workflow.
- Expand to real-time video inputs in future updates.
- Combine Part A + Part B datasets for higher accuracy.
- Save predictions for future visualization or integration into dashboards.

Rationale:

These steps make the project scalable and adaptable for real-world crowd monitoring applications.