Real-Time Crowd Counting with Depth-Embedded Lightweight Neural Networks

Anurag Perakalapudi, Aaryan Sumesh

2/16/2025

1. Abstract

Accurately estimating crowd density in images is of importance in applications such as public safety and event management. Our study proposes a new approach to crowd counting using a depth-embedding LCDnet (Lightweight Crowd Density Estimation Model), which combines information regarding image depth with image features to predict crowd size. Our model improves on crowd counting accuracy, as measured by Mean Absolute Error (MAE) as compared to Lightweight models that do not use depth-embedding, while also improving runtime as compared to models that incorporate depth-embedding. After testing, our depth-embedded LCDnet model achieved a MAE of 112.64, which was lower than the MAE of only the LCDnet of 181.8, and expectedly higher than the MAE of the depth-embedding model which was 57.55.

2. Introduction

Crowd counting is an important logistical step when event planning. It is commonly used for public safety and urban planning purposes. However, traditional methods often struggle with problems like occlusions (objects being hidden), varying crowd densities, and lighting changes. Using depth information in combination with standard images can improve accuracy by providing a better sense of the space and arrangement of people. This is especially useful for real-time applications, where computational efficiency is critical. The goal of this proposal is to create a lightweight model that uses both depth and image data to improve crowd counting in real-time settings.

Head annotations refer to the labeled locations of people's heads within an image. The 2D coordinates (x, y) that correspond to each person's head location in the picture are commonly used to express these comments. The coordinates for each head in the image are included in the annotation data, which is saved in.mat files with the file name matching the image file. The head annotations are used to generate density maps that represent the number of people in different regions of the image. These annotations serve as ground truth data for training and evaluating the crowd counting model.

Depth-embeddings refer to incorporating depth information into the crowd counting model. In traditional 2D image processing, depth data is not readily available, and the model relies solely on the visual appearance of the crowd to create the depth embeddings. Depth embeddings help by introducing information about the relative distance of objects (people) from the camera.

The input to our algorithm is an image with associated head annotations, which are stored in .mat files associated with the .jpg (image) files. We then use a light-weight model that uses depth-embeddings to output a predicted crowd density map.

3. Related work

Crowd Counting has evolved through the use of technology. Where counting by hand was common, computer science techniques like object detection and object counting took hold. However, this wasn't perfect. In dense and low contrast crowds, numbers would not be accurate (Zhang et al., 2016). A later approach consisted of counting by regression which aimed to use image features like edges and textures to count more accurately (Sindagi & Patel, 2018). Other techniques include dilated convolutions in order to improve feature extraction and lower parameter count (Li et al., 2018) and attention-based models that focus on specific regions in the images (Bai et al., 2020).

A more recent paper on research for crowd counting used depth-embedding feature maps to improve accuracy. A specific example would be the Multi-Column CNN from Zhang et al. However, these models are extremely large, being millions of parameters. This causes longer training time and higher computing cost to run the model.

This brings into question the size of the models, which is what LCDnet by Khan et al. tried to solve. With only 500,000 parameters, this model can be used for real-time video surveillance. However, LCDnet has relatively low accuracy because of its low computing power.

Our paper tries to create a mix of these two by incorporating the depth embedding into the LCDnet model.

4. Dataset and Features

For the performance analysis of our crowd density estimation model, we use a dataset composed of images paired with corresponding head annotations. The dataset is split into training and test sets as follows:

- Training Data: 300 images with corresponding head annotations
- Test Data: 50 images with corresponding head annotations

Each image in the dataset represents a crowd scene, with the head annotations indicating the general locations of individuals within the crowd.

Preprocessing:

Image Loading: The images are loaded using OpenCV's cv2.imread() function. The images are then converted from BGR to RGB format using cv2.cvtColor().

Density Map Generation: The density map is generated based on the head coordinates using the generate_density_map() function. This function creates a heatmap where the density of the crowd is represented spatially. The density maps are generated from the application of a Gaussian filter to the ground truth file.

Normalization: The density map is normalized such that the sum of the density values remains consistent across different images, which is achieved by scaling the density map based on the ratio of the sum of its values to the sum of the original density map.

Tensor Conversion: The density map is converted into a tensor format using torch.tensor() and reshaped to match the expected input format for the model.

Features: The features used in our model are derived from two main sources: the images and the head annotations. The model extracts visual patterns like edges, shapes, and gradients from the images, which help identify people in crowds. The head annotations serve as ground truth (like a baseline) and are used to generate density maps, which the model learns to associate with the visual features in the images for accurate crowd density estimation. In total, we have 100977 trainable parameters (features) generated. Dataset Citation: [1] (see bibliography for full citation)

5. Methods

Our proposed algorithm uses a lightweight convolutional neural network (CNN) model that uses depth embeddings.

Convolutional Neural Network:

CNNs recognize characteristics like edges, textures, and patterns by applying a sequence of convolutional layers, where tiny filters move over the input picture. The input is converted into a collection of feature maps by each convolutional layer, highlighting the existence of certain patterns in various areas of the picture. The convolution operation is mathematically represented as:

$$y(i,\ j) = \sum_{m=1}^{M} \sum_{n=1}^{N} x(i + m,\ j + n) \cdot w(m,\ n) + b$$

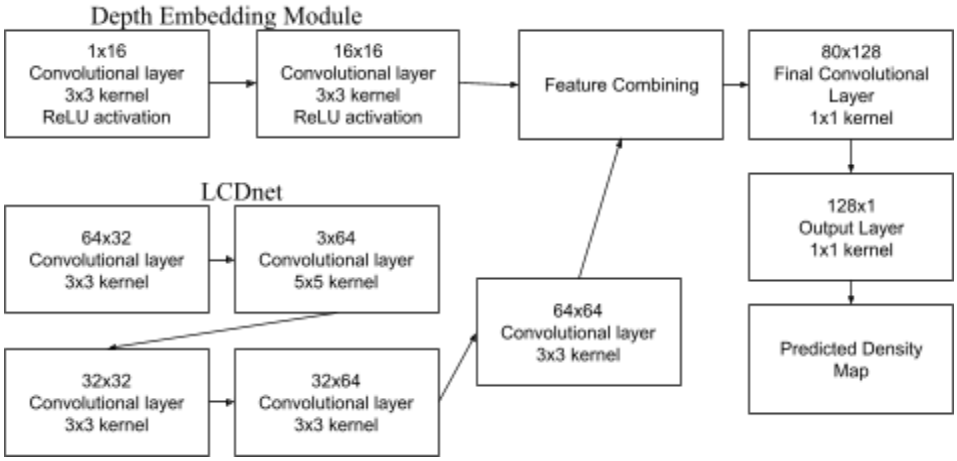
where $x(i,\ j)$ is the input image, $w(m, n)$ is the kernel, b is bias, $y(i,\ j)$ is the resulting feature map, and M and N represent the size of the kernel [4].

Depth Embedding:

Our depth embedding module itself is a CNN. It starts by taking the depth image as input, and then applies a series of convolutional layers to extract features from the depth data. The first convolutional layer processes the depth image and outputs a set of feature maps, which are then passed through additional convolutional layers to refine the features further. The resulting depth embedding is a tensor of weights representing the depth of various points in the image.

Model Architecture:

Our model has the following architecture:



6. Experiments/Results/Discussion

Hyperparameter tuning:

There were two main hyperparameters that were part of our model. They were σ and LR. σ is the scale parameter in the Gaussian kernel. The value of σ is dependent on the sizes of the heads in the image, and the value of σ is calculated as the average distance to k-nearest neighbors. The value of k selected for this is typically 10 [3]. However, this is extremely computationally expensive, especially without a state of the art GPU, like those (Nvidia GTX Titan X GPU and Intel Core i7 6700 processor) [3] used (we used google colab). As such, we decided to set the value of σ constant at 15. This value was empirically determined to work well for most cases in our model. Using larger values of sigma (20 and 25) resulted in a larger MAE, and smaller values of sigma (5) also resulted in a larger MAE.

For LR, or learning rate, we empirically determined that the best value was 1×10^{-4}. The papers which we based our project on used a learning rate of 1e-5 [4]. We tested learning rates of 1e-6, 1e-5, 1e-4, and 1e-3. Below is our mean absolute error vs learning rate graph (on the testing dataset):

Results

Since the value we are predicting (number of people) is continuous and nominal, we decided to use mean absolute error (MAE) as our metric of error. MAE is given by:

$$\text{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$$

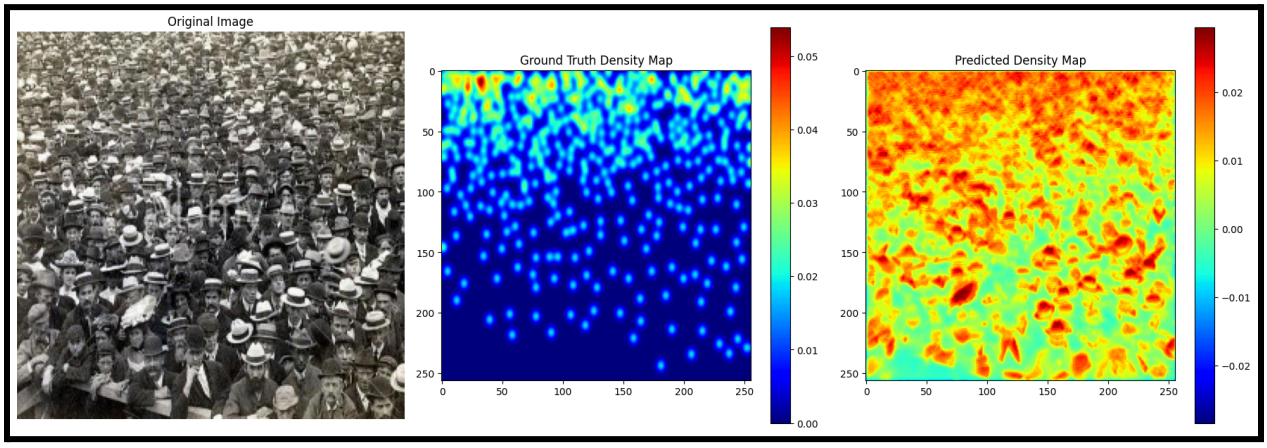
Where y_i is the sum of all points in the ground truth density map and x_i is the sum of all points in our models predicted density map, and n is the number of instances (images). After training our model on 3 epochs, we found that our model had a mean absolute error of 112.64.

We found our result to be satisfactory. Because our model was a compromise between a model that included depth information to predict crowd size and a faster, lightweight CNN model that did not use depth information, we expected to see an MAE of between the faster and slower models. The mean absolute error of the model that used depth embedding and the lightweight CNN, in comparison with our model are shown in the table below:

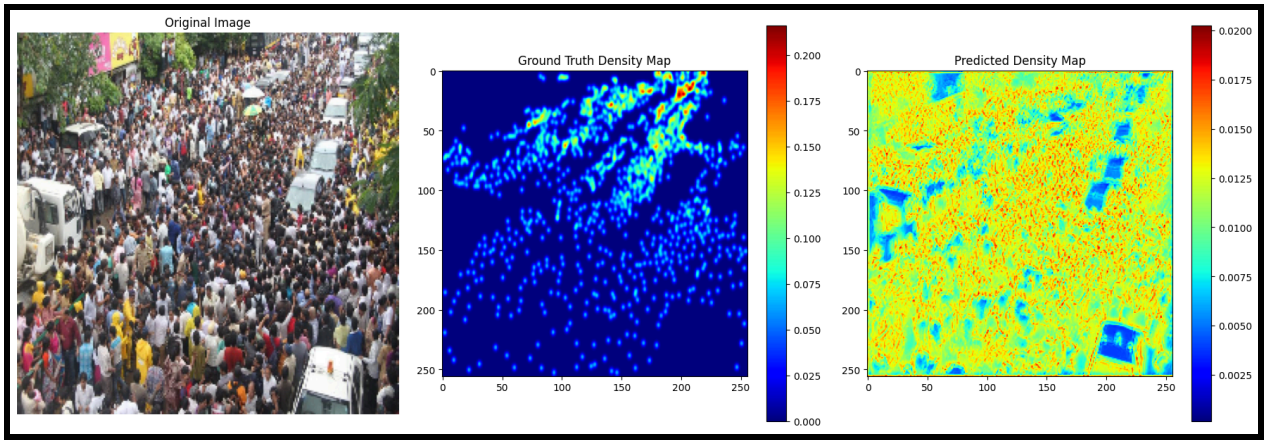
model	best mean absolute error
Our Model (intermediate)	112.64
Lightweight CNN Model (fast)	181.8 [1]
Model with depth embedding (slow)	57.55 [1]

Discussion

We noticed that our model worked well if there were no intervening objects in the image, like the sky or trees. When the image was mainly (>95%) covered with people, our model seemed to be performing the best. For example, in the picture below, the actual value of the number of people was 2320, while our predicted value was 2318.5449. We believe that our model performed well on these scenes without distracting objects because our model didn't have to differentiate between non-people objects and people.



However, when there were objects like trees and cars in the way, we saw that our model was performing worse. In the images below, the actual value of the number of people was 2799, while our model predicted 2647. We believe that our model was worse on these scenes because our model seemed to occasionally confuse these distracting objects like trees, cars, and buildings as people (indicated by nonzero values on the density map).



We believe that our model may have overfit our dataset, though not because of the nature of our model, but because of the hyperparameters that we chose. Ideally, the value of sigma in the Gaussian filter should be dependent on the images in the dataset. Factors like lighting and orientation/perspective will significantly alter the value of sigma. In our case, we "overfit" our dataset by selecting the value of sigma to be 15.

7. Conclusion/Future Work

Overall, our model, which combined depth-embedding with a light-weight CNN model to predict crowd density, performed better than the lightweight CNN model without depth-embedding, and performed worse than the scale-aware depth-embedding model, according to MAE. Specifically, our model performed better on images of crowds that were highly dense and had limited obscuring objects like trees, cars, and buildings.

If we had more computational resources and time, we would further explore choosing a better and adaptable sigma value for the Gaussian filter to improve the quality and accuracy of our density maps. Additionally, we would like to continue to explore other methods of depth embedding. We used a CNN to do the depth embedding, but we would like to explore other methods that have been used for depth embedding, and see which of those methods would be most compatible with our hybrid model.

9. References/Bibliography

[1] "Papers with Code - ShanghaiTech Dataset." Papers With Code, https://paperswithcode.com/dataset/shanghaitech. Accessed 26 Jan. 2025.

[2] Zhao, M., Zhang, C., Zhang, J., Porikli, F., Ni, B., & Zhang, W. (2019). "Scale-aware crowd counting via depth-embedded convolutional neural networks." IEEE Transactions on Image Processing. Accessed 26 Jan. 2025.

[3] Khan, Muhammad Asif, et al. "LCDnet: A Lightweight Crowd Density Estimation Model for Real-Time Video Surveillance." Journal of Real-Time Image Processing, vol. 20, no. 2, Mar. 2023, pp. 1–11, https://doi.org/10.1007/s11554-023-01286-8.

[4] Gillis, Alexander S., et al. “What Is a Convolutional Neural Network (CNN)?” TechTarget, 25 Nov. 2024, https://www.techtarget.com/searchenterpriseai/definition/convolutional-neural-network.

[5] Bai, H., Mao, J., & Chan, S.-H. G. (2020). A Survey on Deep Learning-based Single Image Crowd Counting: Network Design, Loss Function and Supervisory Signal.

[6] Li, Y., Zhang, X., & Chen, D. (2018). CSRNet: Dilated Convolutional Neural Networks for Understanding the Highly Congested Scenes. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 1091–1100.

[7] Zhang, Y., Zhou, D., Chen, S., Gao, S., & Ma, Y. (2016). Single-Image Crowd Counting via Multi-Column Convolutional Neural Network. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 589–597.