CrowdDiffKDE : Multi-hypothesis Crowd Density Estimation using Diffusion Models and Kernel Density Estimation

Github Repo Link

1 Introduction

The Crowddiff authors present a novel, state-of-the-art approach for generating crowd maps using a diffusion model. First, a narrow kernel convolves with point data, and a threshold is applied to the resulting map, which is then used as ground truth for the model-generated map.
Additionally, they perform stochastic crowd map fusion, combining multiple realizations from the diffusion model using SSIM.
Due to the high computational complexity of diffusion models, generating multiple realizations leads to high inference times. To address this issue, I propose using kernel density estimation to produce new crowd map realizations. This approach significantly reduces computation time while generating new realizations quickly.

2 Implementation

Kernel density fit: Using the Crowddiff model, we generate a single crowd map, or realization, for an image, and the resulting points are then thresholded. A Gaussian kernel is applied to fit a kernel density function, modeling the probability distribution across each point in the crowd map. The bandwidth is carefully chosen to achieve an optimal spread, ensuring a smooth distribution without restricting it to specific points.
Removing outliers: The generated points from the model may include outliers or noisy points with extremely low-density values in the kernel density estimate. We remove those in the bottom 5th percentile to filter these points, effectively discarding the least probable points and improving the overall quality of the generated crowd map.
Sample and fusion: Using the learned density distribution, we can sample new points and realizations that could be included in the density map.

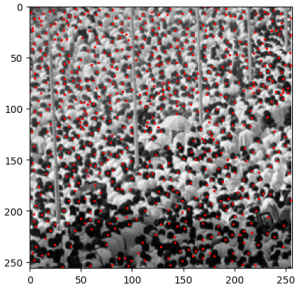
1. First, we randomly sample four times the initial number of points from the crowd map. These sampled points are then filtered to ensure that their density distribution falls within the mid-range.
2. We apply the k-nearest neighbors algorithm on the initial points to calculate a radius of rejection for each point.
3. For each newly generated sample, we compute its distance to each point in the map.
4. If a sample lies outside all radii of rejection for the existing head locations in the map, it is included; otherwise, it is discarded.
5. If the point is included, we repeat this process for each sample on the updated crowd map, now containing the initial points and the newly added samples. Additionally, beta and k can be updated to avoid overcounting.

3 Pseudocode

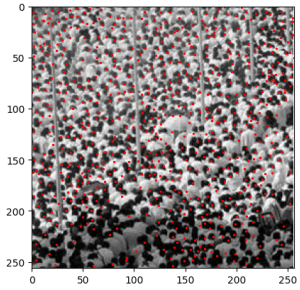
Algorithm 1 Fuse_With_KDE

```
function FUSE_WITH_KDE(crowd_map)
    head_points = DENOISE_AND_THRESHOLD(crowd_map)
    KDE = KDE.FIT(head_points)
    define k and β value and their respective increments
    Neighbours = NEARESTNEIGHBOURS(k + 1)
    NEIGHBOURS.FIT(head_points)
    r = calculate initial radius
    for each point in 4 x head_points_count do
        dist = EUCLIDEAN(point - head_points)
        if dist > r for all head_points then
            APPEND(head_points, point)
            NEIGHBOURS.FIT(head_points)
            UPDATE r, k and β as required
        end if
    end for
    return head_points
end function
```

4 Results



Original Realizations Count, higher computation: 616



Modified with KDE Count, Lower Computation: 653

Metrics	**JHU-Crowd**	**Shtech-A**	**Shtech-B**	**UCF-CC**	**UCF-QNRF**
MAE	52.96	44.352	7.02	87.22	70.19
MSE	201.325	79.24	13.19	196.98	99.32

Table 1: Implemented values for 5 benchmarks

5 Analysis

Fusing with KDE enhances accuracy for images with extremely high densities, such as those in UCF-CC, where crowd densities are high. However, for low-density images, KDE does not improve accuracy and can result in incorrect predictions for locations. Therefore, the KDE method serves as a valuable alternative specifically in high-density regions.