Crowd detection using Gabor filters and K-mean clustering

Nhi Nguyen
CPSC 475, Professor Zucker

December 18, 2020

Abstract

The analysis of human crowds has a wide range of application, from urban planning, traffic management, and even social distancing enforcement in a pandemic time. In order to be analyzed, a crowd must first be detected. This paper proposes and examines a simple algorithm to perform this task. Given an aerial image of a crowd, the algorithm segments the image into crowd and non-crowd regions. On a large scale, we expect a crowd to contain some repetitive visual elements or textures that are significantly different from that of a non-crowd region. The proposed algorithm uses multiple Gabor filters to capture these different textures in an image and uses k-mean clustering to segment the image into 2 groups corresponding to crowd and non-crowd regions.

1 Introduction

This paper attempts to detect crowds of humans in still images. Given an image, the proposed algorithm segments out the regions that the crowd occupies.

The task of detecting the presence of a crowd is important in itself. The formation of crowds in public places such as on the streets or in the malls can cause congestion and delays or be a sign of hazard or civil disturbance. In the current pandemic, being able to detect a crowd can also helps with social distancing enforcement.

The natural next step after crowd identification is crowd segmentation. This is a necessary preprocessing step for more complex, higher level tasks, which includes counting the number of individuals in a crowd, estimating crowd density, and analyze the flow and behavioral interaction of crowds. As crowd is an integrated part in humans' society, studying its behaviors reveal characteristics of humans, and the result can be applied to a range of areas, from psychological research, crime detection and prevention, and even even for entertainment purposes such as realistic crowd simulations.

Nevertheless, crowd detection can be a challenging task. Images of crowds, often taken from surveillance cameras, often have limited resolution, which makes it difficult to detect single individuals. In a larger crowds, there can be variation in gender, pose, clothing, or lighting from one person to another. If crowd detection purposefully look for multiple people at a time instead of individual person or faces,

Figure 1: Example of crowd detection in a still image

the same variation happens and its range even increases in the combined appearance of many individuals, not to mention crowd specific characteristics such as density or spacing. In addition, the background of the images themselves can also be confusing for an algorithm that attempts to detect a crowd.

In an attempt to tackle the challenge of crowd detection, this paper loosely defines a crowd as a group of spatially proximate objects with repetitive visual elements. The proposed algorithm makes use of a bank of Gabor filters, often used to analyze textures, to decompose an image to inspect these repetitive visual elements. Then, the crowd can be segmented from the background by clustering algorithm, specifically k-mean in this paper. This algorithm does not have any inherent knowledge about what is a crowd, let alone a crowd of humans. It only attempts to detect and segment the regions that a crowd occupied in images that are known to have a crowd in it.

1.1 Data set description

The data set consists of 16 aerial images of crowds taken from the internet. Each images are tagged with a range 4 properties:

- whether there are `AFEW` or `MANY` people;
- if the spacing is sparse or close, or if the crowd is `SCAT`tered or `CLUS`tered;
- the background is `UNIF`orm or have some `VARI`ances;
- the individuals in the crowd cast `SHAD`ow or `NONE`.

By testing the algorithm on a range of images with varying properties, this paper aims to choose a good set of parameters that can detect crowd well despite the diverse characteristics of crowds. The full set of data with their corresponding properties can be found in the Appendix in Figure 8.

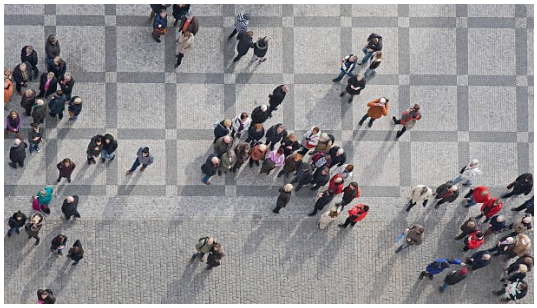


Figure 2: The previous image in the introduction is taken from the data set, with many people in clusters standing in a non-uniform background and there is some shadow. the filename of the image is `many-clus-vari-shad.jpg`.

2 Method details

In this paper, the process of crowd segmentation consists of the following three steps:

1. Decomposing the input image using a bank of Gabor filters;
2. Feature extraction using Gaussian smoothing function, and
3. k-mean clustering.

In addition, before the first step, we would resize the images to a fixed size 288×512 for ease of determining the set of filter parameters later on.

2.1 Gabor filters

A two-dimensional Gabor function consists of a two-dimensional Gaussian modulated by a sinusoidal plane wave of a frequency and an orientation. In this paper, we present a bank of two-dimensional even-symnetric Gabor filters, given by the formula (see [Zuc20c]):

$$gb(x, y, \lambda, \theta, \phi, \sigma, \gamma) = \exp\left(-\frac{\hat{x}^2 + \gamma^2\hat{y}^2}{2\sigma^2}\right)\cos\left(2\pi\frac{\hat{x}}{\lambda} + \phi\right) \tag{2.1}$$

where

$$\hat{x} = x\cos(\theta) + y\sin(\theta), \quad \hat{y} = y\cos(\theta) - x\sin(\theta),$$

and

- λ is the wavelength, or inverse of the frequency, of the cosine,
- θ is the orientation in degrees of the filter, which is normal to the parallel stripes of a Gabor function,
- ϕ is the phase offset in degrees of the center peak from the Gabor center,
- σ is standard deviation of the Gaussian envelope, and
- γ is the spatial aspect ratio, which specifies the elliptical shape of the Gabor support.

The ratio σ/λ determines the spatial frequency bandwidth and hence the number of parallel excitatory and inhibitory stripes in the Gabor filter. The half-response spatial frequency bandwidth b (in octave) related to the ratio σ/λ as follows:

$$b = \log_2 \frac{\frac{\sigma}{\lambda}\pi + \sqrt{\frac{\ln(2)}{2}}}{\frac{\sigma}{\lambda}\pi - \sqrt{\frac{\ln(2)}{2}}} \quad , \quad \frac{\sigma}{\lambda} = \frac{1}{\pi}\sqrt{\frac{\ln(2)}{2}}\frac{2^b+1}{2^b-1}. \tag{2.2}$$

In order to capture the repetitive texture of a crowd from many perspectives, we use 6 orientations with orientation separation angles of $d_\theta = 30°$:

$$\theta : 0°, 30°, 60°, 90°, 120°, 150°$$

We also use a range of wavelengths, evenly spaced in $\log_2$-space, ranging from some minimum wavelength to the radius of the image (or half its diagonal length). The choice of the minimum wavelength is adjusted when we apply the algorithm to some initial images. The general formula for the chosen wavelengths is

$$\lambda : \lambda_{\min} \times 2^k \quad , k \in \mathbb{N}$$

For example, if we choose both $\lambda_{\min}$ and r_λ equal to 2 for a 288×512 image, there would be a total of 42 Gabor filters used from 6 orientations and 7 wavelengths.

In this paper, we set the value of the bandwidth b by default to 1 octave. In that case, the Equation 2.2 gives the approximation $\sigma = 0.5 \times \lambda$.

2.2 Feature Extraction

For each filtered image, we use a Gaussian smoothing function given by (see [Zuc20b])

$$g(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \tag{2.3}$$

where σ is the standard deviation that determines the windown size. The ratio σ/σ_g (where σ_g is the standard deviation parameter of Gabor filter) is estimated and adjusted when we apply the algorithm to some initial images.

2.3 k-mean clustering

Finally, the algorithm cluster the pixels into 2 clusters, corresponding to crowd and non-crowd regions. The general algorithm of k-mean clustering is as follows (see [Zuc20a]):

Step 1. Initialize the k centroids of k clusters randomly.

Step 2. Assign each pixel a label corresponding to the nearest centroid.

Step 3. Calculate the means, or the new centroids, of k clusters.

Step 4. If the centroids are unchanged, the algorithm terminates. Otherwise, go back to Step 2.

Besides the feature extracted from the Gabor bank of filters and Gaussian smoothing, we also include the coordinates of the pixels as two additional features to account for the spatial adjacency of pixels in the clustering process.

Since k-means clustering may output varying and possibly undesired segmentation because of the random initialization, we run the clustering process 5 times and choose the best results.

The code for the general crowd detection algorithm can be found in the Appendix in Listing 1.

3 Test parameters

We choose to test our parameters on two images with opposite characteristic tags, namely `afew-clus-vari-none.jpg` and `many-scat-unif-shad.jpg`. The images are as follows:

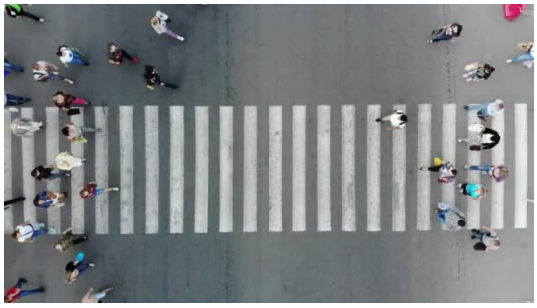


Figure 3: `afew-clus-vari-none.jpg`

Figure 4: `many-scat-unif-shad.jpg`

We first test them on minimum wavelength $\lambda_{\min} = 3$ and the gaussian vs gabor standard deviation ratio (see Equation 2.3) $\sigma/\sigma_g = 3$. The resulting segmentation is in Figure 5.

The algorithm does decently well with both of the picture. For both images, it pinpoints the correct regions where the crowds of people are. In the first image, it seems slightly over estimate the size of each crowd on the left and right. But the crosswalk stripes do not seem to confuse the algorithm. With the second image, the algorithm does a slightly worse job, as the shadow makes it overestimates the regions that the crowd occupies, and there are quite a few people who are not captured as belonging to the crowd.

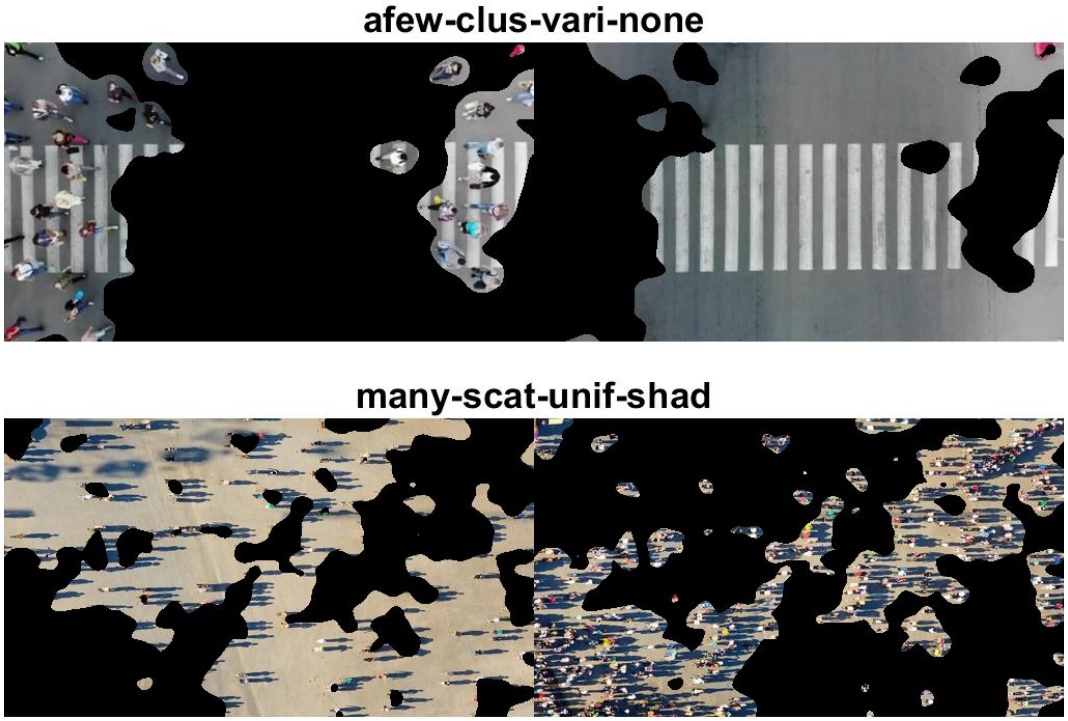


Figure 5: First trial

In order to lessen the algorithm's overestimation and be able to detect more people in a scattered crowd, we will reduce the value of both the minimum wavelength and the standard deviation ratio. The goal is that the algorithm can pick up smaller details in the picture and thus segment more precisely all the regions of the crowd.

In the second trial, we change the minimum wavelength to 2 and the standard deviation ratio to 1.6. The resulting segmentation is in Figure 6.

The algorithm seems to improve for both images. For the first image, the algorithm seems to reduce the algorithm overestimation, although it seems to confuse a tiny part of the crosswalk stripes as parts of the crowd. For the second image, the algorithm seems to no longer include the majority of the shadow as parts of the crowd, and there are only 1-2 people who are no included as belonging to the crowd.

As a result, we choose minimum wavelength equal 2 and standard deviation ratio equal 1.6 as the parameters for our algorithm, in addition to the other parameters chosen in Section 2.

All the code for the first and second trials can be found in the Appendix in Listing 2.

4 Result

The complete code and results when apply the algorithm to all 16 images can be found in the Appendix in Listing 3 and Figure 9.

The algorithm seems to work decently well for the majority of the images. The 3 images that the algorithm do not seem to work well with this are in Figure 7.

For `afew-scat-unif-shad.jpg`, the algorithm seems to laregly overestimate the size of the crowd do to a large shadow cast by a nearby tree. This problem can be

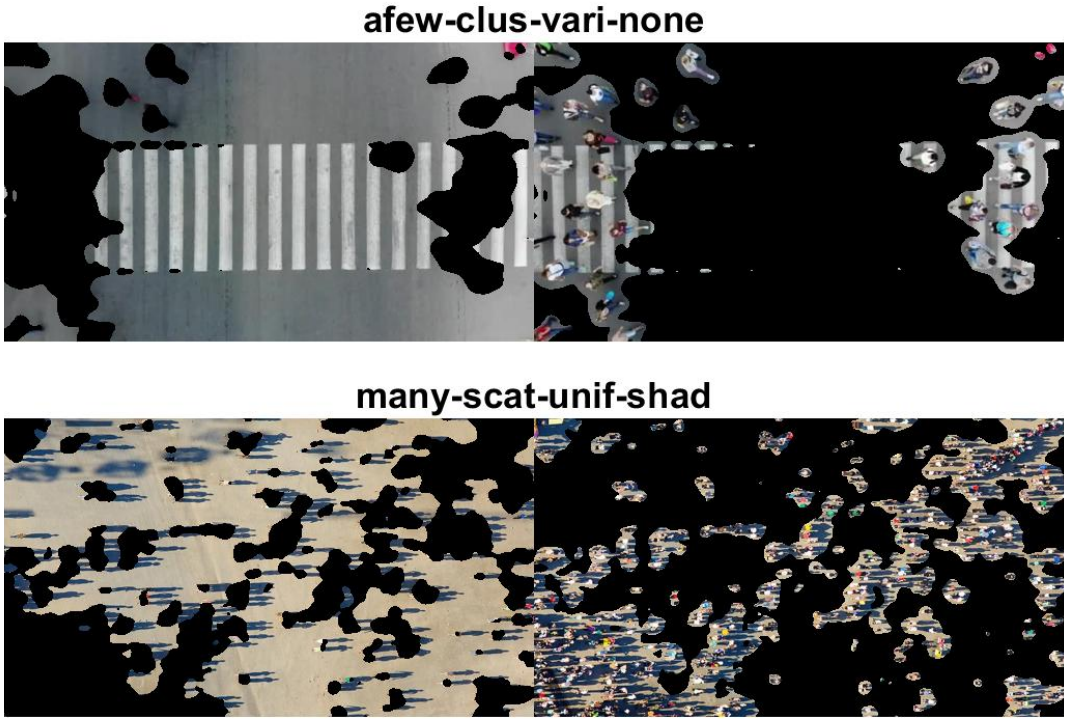


Figure 6: Second trial

reduced by further reduce the minimum wavelength and/or the standard deviation ratio.

Similar problem occurs in `many-clus-vari-none.jpg`, as half of the image is in the sun and half of the image is in the shade. This causes the algorithm to think that everything in the sun is in a group and everything in the shade is in the another group, which is not true for the purpose of crowd detection. Reducing the value of minimum wavelength and standard deviation ratio seems to increase the number of people in the shade recognized as belonging to the crowd, but it also underestimate the size of the crowd/cut off people in other regions.

The most surprising result is probably with `many-clus-unif-shad.jpg`. By eyes, the image seems to clearly show 2 distinct regions corresponding precisely to crowd and non-crowd regions. However, this algorithm appears not capable of picking up on this information, and segment the large crowd on the top of the image against the smaller crowd at the bottom. Adjusting the parameters does not seem to help with this image in particular. I suspect that the extremely high density of the crowd and the somewhat weirdly shape they are standing in may confuse the algorithm. Further inspection may needed to improve the algorithm so that it can segment crowd in images like this one.

5 Discussion

The data set for this paper is certainly not comprehensive. The number of properties is limited to 4 and the categorization is binary. Additional images with more properties, each with a range of magnitude/intensity may help better evaluate the performance of this algorithm.

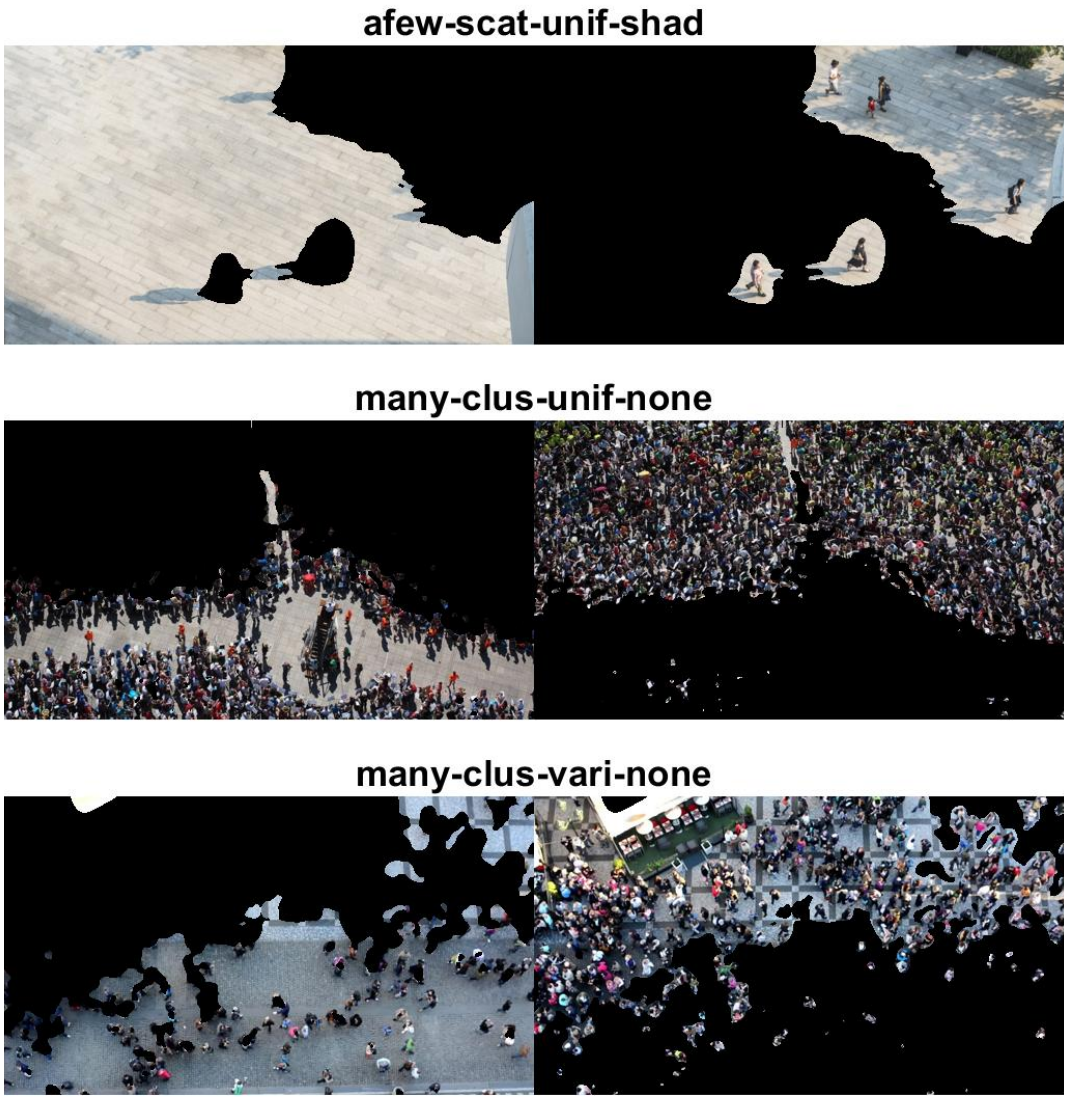


Figure 7: Images where the algorithm did not segment well

There are also a number of other parameters of the filters that this paper fixes and didn't adjust (see the variables in Equation 2.1, 2.2, and 2.3) Study how their variances effect the result may help improve the algorithm. For K-means clustering, we chose to run the programs several (5) times and obtain best results among those. Increase the number of replications or using another criteria might make the clustering process work better and more efficient.

There are some defects inherent in Matlab average filters such as Gabor and Gaussian. In particular, they assume that pixels out of the image has intensity of 0, and thus it is possible the algorithm does not work well for pixels at the circumference of images. This problem did not arise with the 16 images in this paper's data set, but it is a problem that may be needed to deal with when applying to more images in different circumstances.

This program worked reasonably fast, needed from 20.839009 to 31.543316 seconds for each image of size 288×512. However, the time does add up when we want to process all the images multiple times when testing for different parameters.

Acknowledgement

I would like to express my gratitude to my professor Steven Zucker, who gave me many pieces of advice and guided me throughout this project.

References

[Zuc20a] Steven Zucker. Lecture 05: Chapter 07: Object recognition-1: Template matching, lecture notes. *CPSC 475, Yale University*, pages 17–20, 2020.

[Zuc20b] Steven Zucker. Lecture 09: Chapter 1: Statistical view of principle components, lecture notes. *CPSC 475, Yale University*, pages 24–28, 2020.

[Zuc20c] Steven Zucker. Lecture 13: Chapter 17: Hierarchies in the small: The microstructure of lines and edges, lecture notes. *CPSC 475, Yale University*, pages 30–31, 2020.

A Matlab code

Listing 1: `crowd.m` : source code for function `crowd()`

```
function [im, seg1, seg2] = crowd(imOrig, nrows, ncols, dtheta,
    lambdaMin, sratio, nreps)

%CROWD Segment an image into crowd and non-crowd regions

%   [IM, SEG1, SEG2] = CROWD(IMORIG, NROWS, NCOLS, DTHETA, LAMBDAMIN,
%   SRATIO, NREPS) segments a color image IMORIG into 2 regions SEG1
%   and SEG2, one is a crowd region and the other is a non-crowd
%   region.

%   The function resizes the image IMORIG into a NROWS-by-NCOLS image
%   IM, and then converts it into grayscale. It then applys a set of
%   gabor filters with orientations evenly spaced from 0 (inclusively)
```

```
%   to 180 (exclusively) degrees with step DTHETA and wavelengths
%   log2-space evenly spaced from LAMBDAMIN (inclusively) to the radius
%   of the image (exclusively). With the resulting magnitudes for each
%   gabor filter, it applys a gaussian filter with standard deviation
%   SRATIO times the standard deviation of the corresponding gabor
%   filter. The results are flattened and normalized before being
%   segmented into 2 regions using kmean with NREPS replicates.

% resize the image
im = imresize(imOrig, [nrows ncols]);
imGray = rgb2gray(im);

% orientations (thetas)
thetas = 0:dtheta:(180 - dtheta);

% wavelengths (lambdas)
lambdaMax = hypot(nrows, ncols) / 2;        % radius of the image
n = floor(log2(lambdaMax / lambdaMin));
lambdas = 2 .^ (0:(n-1)) * lambdaMin;

% gabor filter bank
gabors = gabor(lambdas, thetas);
gabormags = imgaborfilt(imGray, gabors);

% feature extraction
features = zeros(size(gabormags));
for i = 1:length(gabors)
    gabormag = gabormags(:,:,i);
    sigma = 0.5 * gabors(i).Wavelength;
    features(:,:,i) = imgaussfilt(gabormag, sratio * sigma);
end

% spatial coordinations
X = 1:ncols;
Y = 1:nrows;
[X, Y] = meshgrid(X, Y);
features = cat(3, features, X);
features = cat(3, features, Y);

% flatten and normalize the filtered image
features = reshape(features, nrows * ncols, []);
features = (features - mean(features)) ./ std(features);

% segment the image into crowd and non-crowd
labels = kmeans(features, 2, 'Replicates', nreps);
labels = reshape(labels, [nrows ncols]);

% display the segmentations using the original image
seg1 = zeros(size(im), 'like', im);
seg2 = zeros(size(im), 'like', im);
mask = labels == 1;
mask = repmat(mask, [1 1 3]);
seg1(mask) = im(mask);
seg2(~mask) = im(~mask);

end
```

Listing 2: `testParam.m` : test parameters for some images

```
%% First trial

% fixed parameters
nrows = 288;
ncols = 512;
dtheta = 30;
nreps = 5;

% adjusted parameters
lambdaMin = 3;
sratio = 3;

% First image
file = 'img\afew-clus-vari-none.jpg';
dispC(file, nrows, ncols, dtheta, lambdaMin, sratio, nreps);

% Second image
file = 'img\many-scat-unif-shad.jpg';
dispC(file, nrows, ncols, dtheta, lambdaMin, sratio, nreps);


%% Second trial

% adjusted parameters
lambdaMin = 2;
sratio = 1.6;

% Second image
file = 'img\many-scat-unif-shad.jpg';
dispC(file, nrows, ncols, dtheta, lambdaMin, sratio, nreps);


% First image
file = 'img\afew-clus-vari-none.jpg';
dispC(file, nrows, ncols, dtheta, lambdaMin, sratio, nreps);


%% Display function

function dispC(file, nrows, ncols, dtheta, lambdaMin, sratio, nreps)

%DISPC Display the results from the function crowd

    % process the image
    imOrig = imread(file);
    [im, seg1, seg2] = crowd(imOrig, nrows, ncols, dtheta, lambdaMin,
        sratio, nreps);

    % display the (resized) original image
    figure;
    imshow(im);

    % display the segmentation side by side
    figure;
    imshowpair(seg1, seg2, 'montage');
    title(extractBetween(file,5,23));
end
```

Listing 3: `main.m` : main file

```
% get filenames with corresponding properties
props = ["afew" "many";
         "clus" "scat";
         "unif" "vari";
         "none" "shad"];
files = strings(16);
ind = 1;
for p1=props(1,:)
for p2=props(2,:)
for p3=props(3,:)
for p4=props(4,:)
    files(ind) = strcat('img\',p1,'-',p2,'-',p3,'-',p4,'.jpg');
    ind = ind + 1;
end
end
end
end

% parameters
nrows = 288;
ncols = 512;
dtheta = 30;
lambdaMin = 2;
sratio = 1.6;
nreps = 5;

% store processed images
ims = [];
segs = [];
for i=1:16
    imOrig = imread(files(i));
    tic
    [im, seg1, seg2] = crowd(imOrig, nrows, ncols, dtheta, lambdaMin,
        sratio, nreps);
    toc
    ims = cat(4, ims, im);
    segs = cat(4, segs, seg1);
    segs = cat(4, segs, seg2);
end

% display the results
for i=1:16
    figure;
    imshow(ims(:,:,:,i));
    figure;
    imshowpair(segs(:,:,:,2*i-1), segs(:,:,:,2*i),'montage');
    title(extractBetween(files(i),5,23));
end
```

B Images

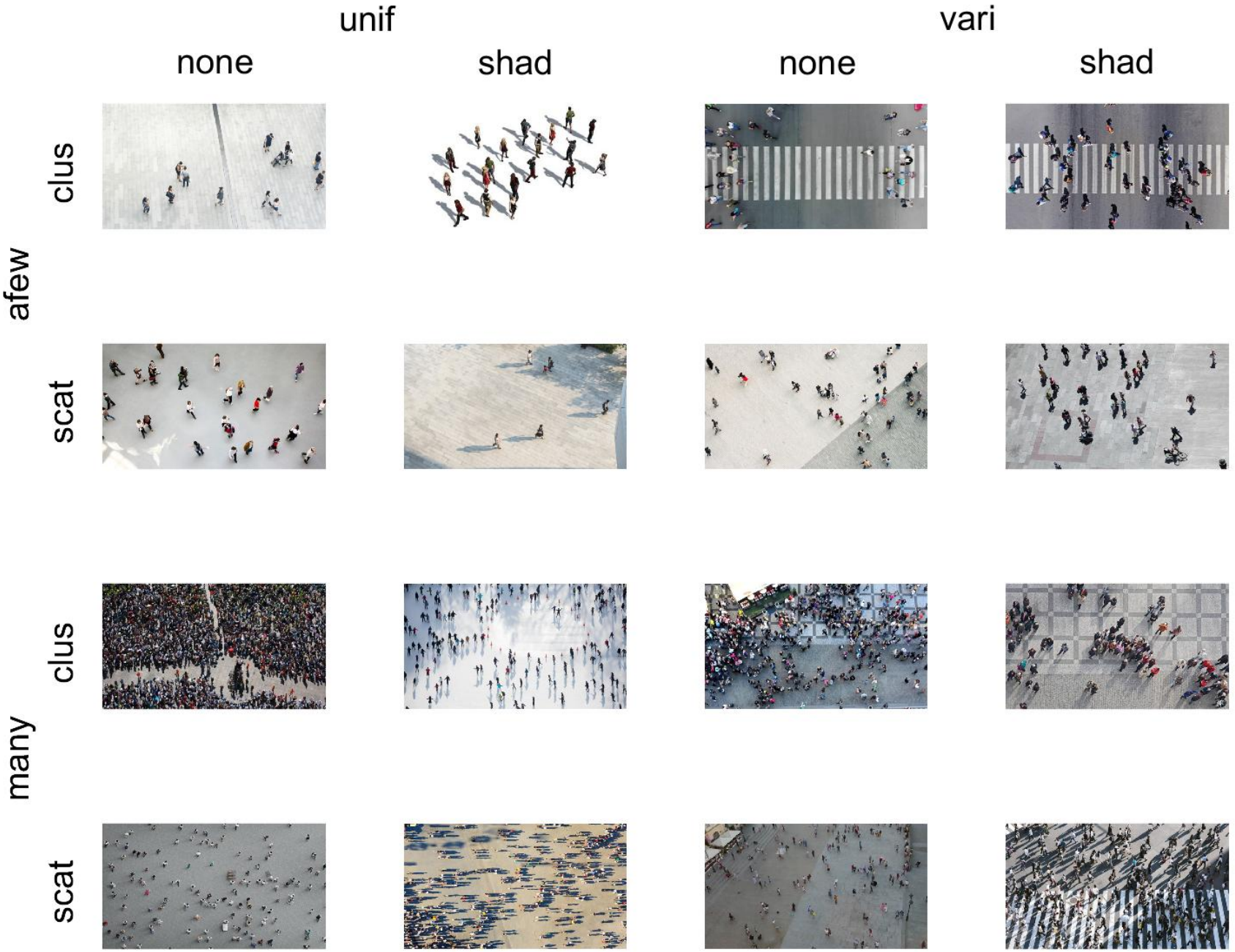
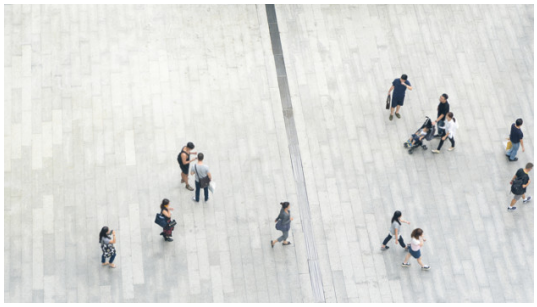
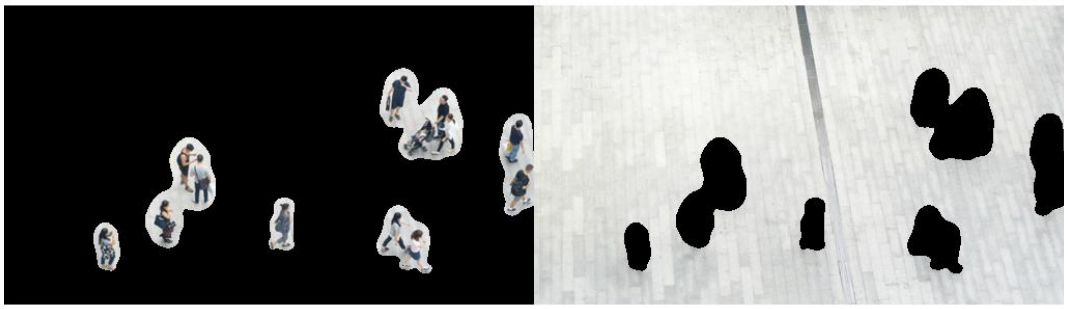


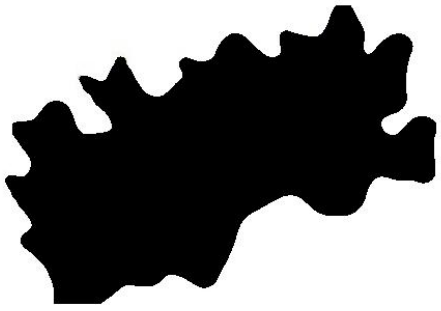
Figure 8: 16 (resized) images in the data set



afew-clus-unif-none

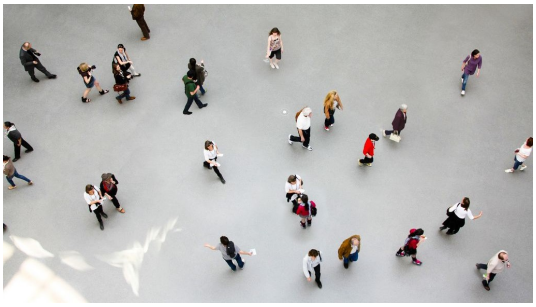


afew-clus-unif-shad



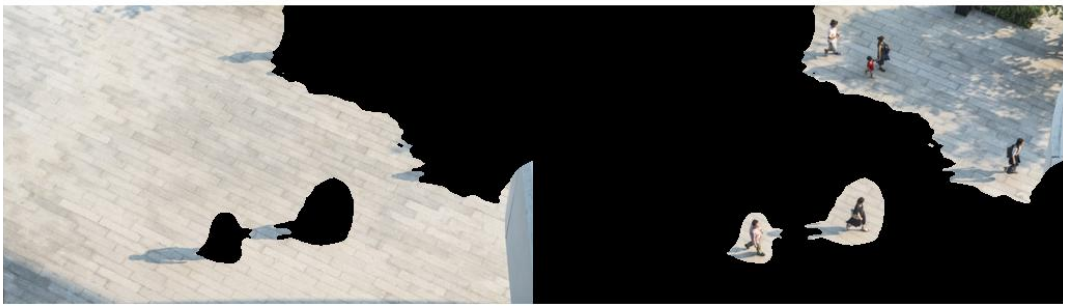
afew-clus-vari-none

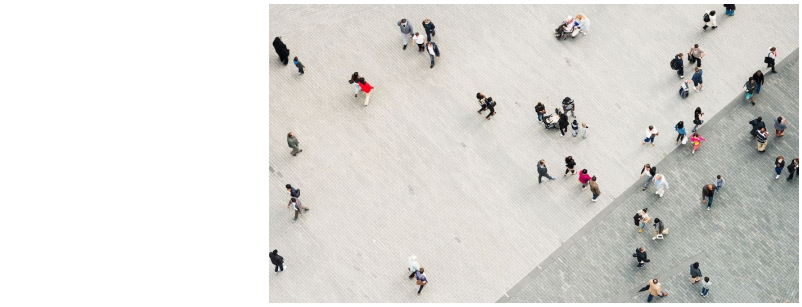
afew-clus-vari-shad



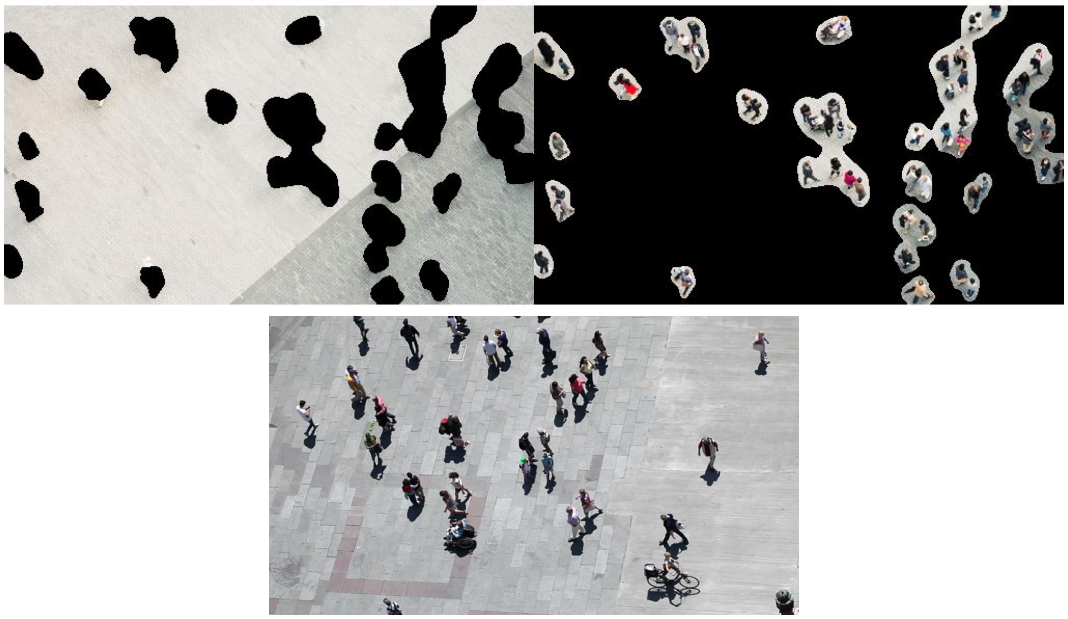
afew-scat-unif-none

afew-scat-unif-shad

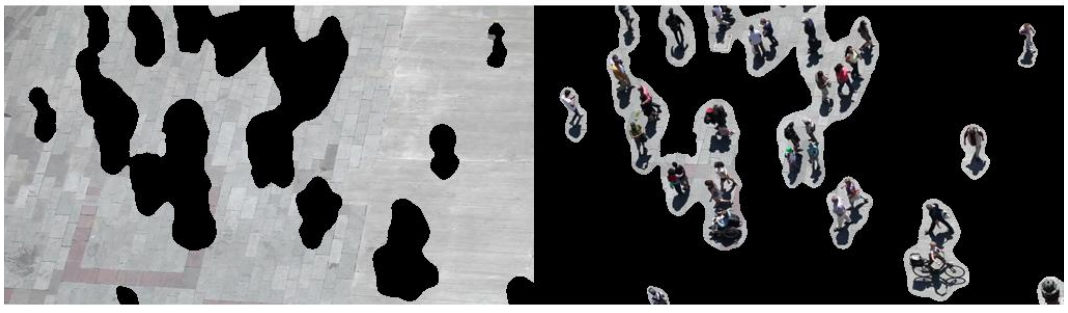


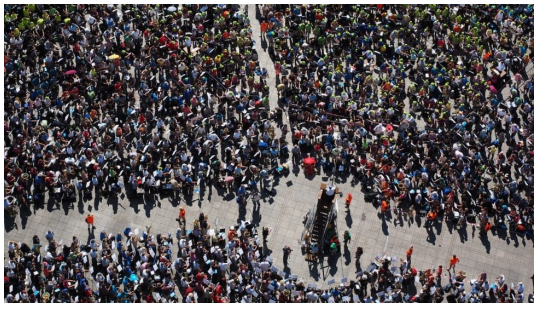


afew-scat-vari-none



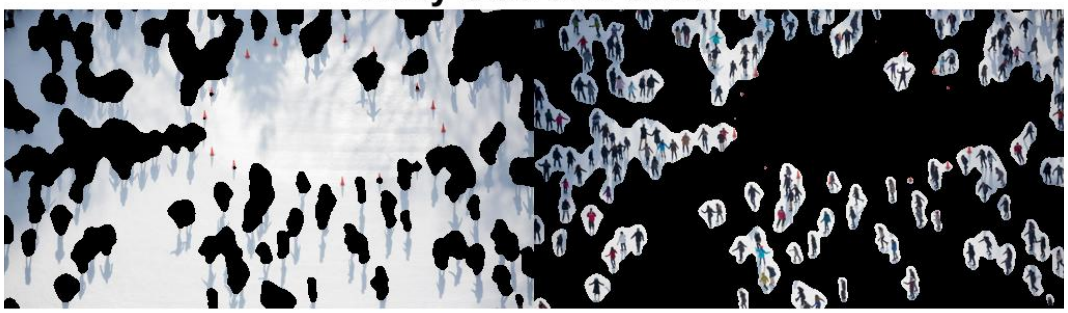
afew-scat-vari-shad





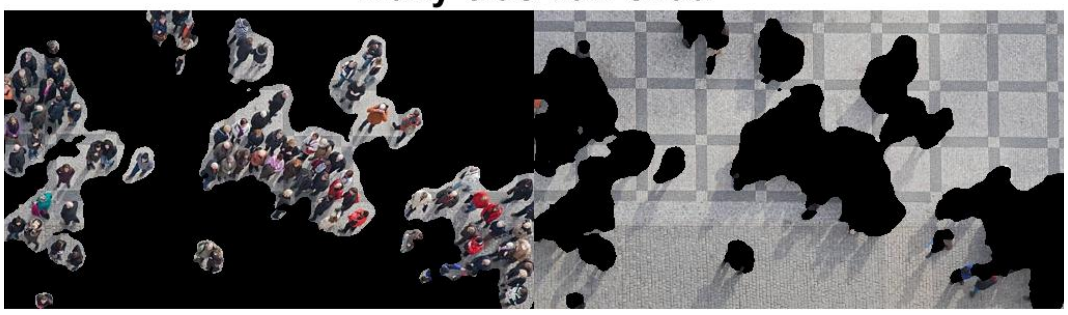
many-clus-unif-none

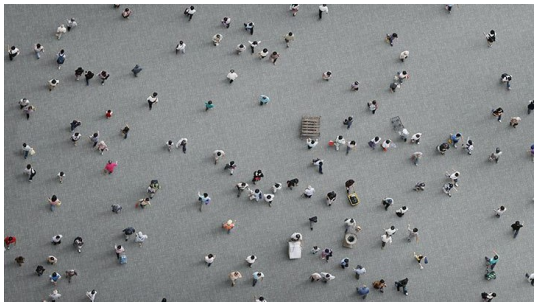
many-clus-unif-shad



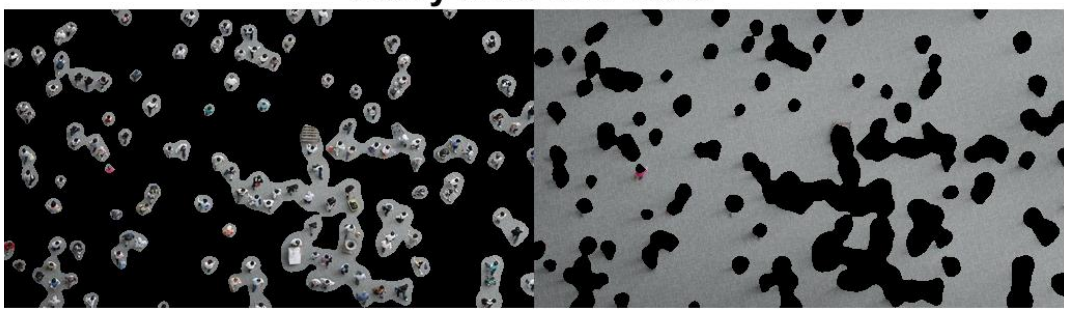
many-clus-vari-none

many-clus-vari-shad

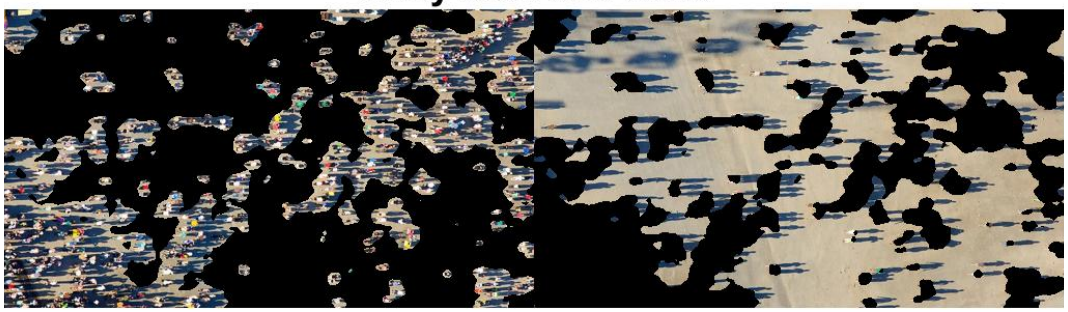




many-scat-unif-none



many-scat-unif-shad



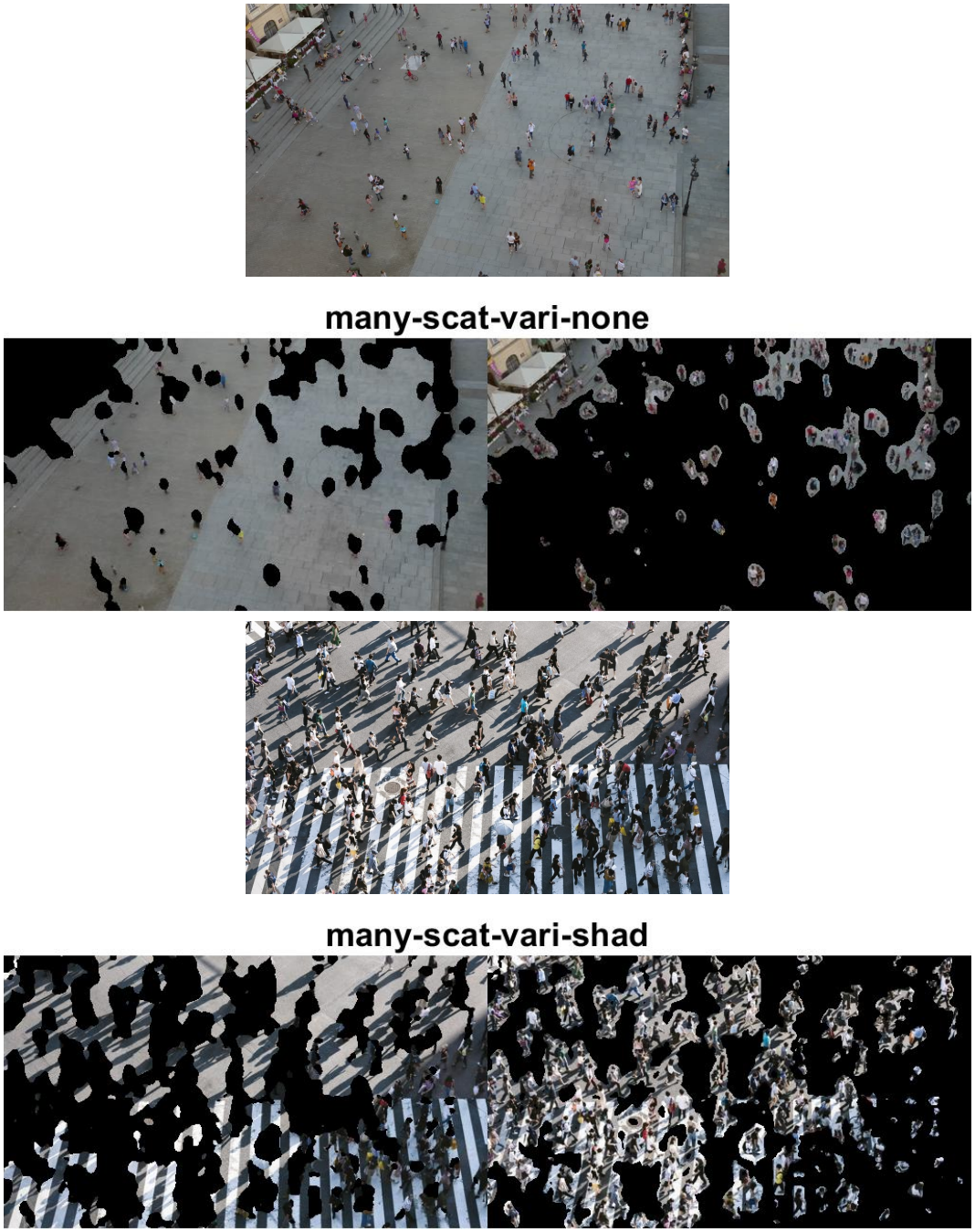


Figure 9: Results