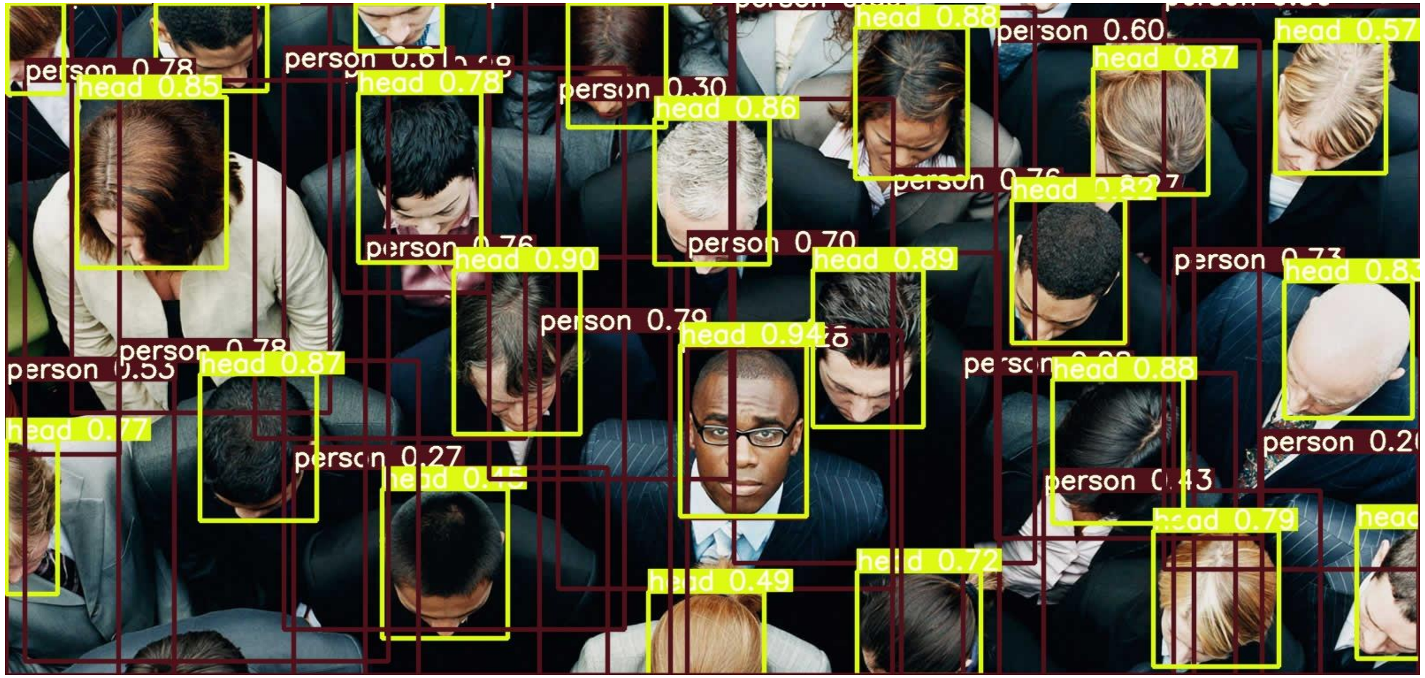
# Crowd Detection

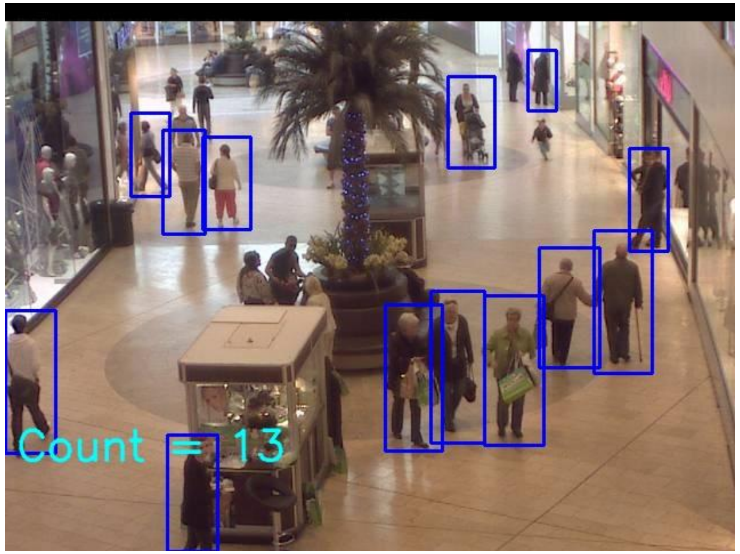
A computer vision study



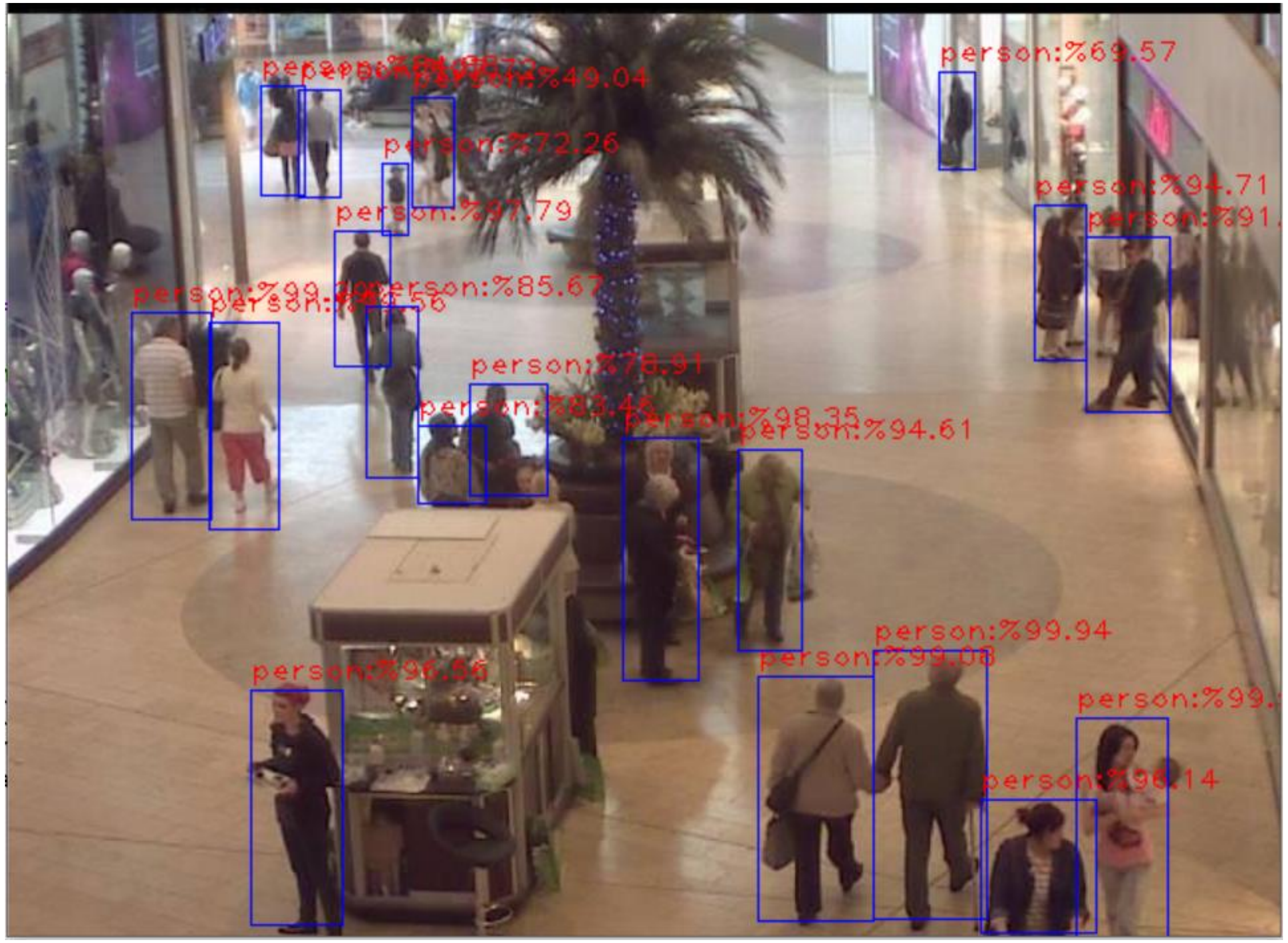

Sara Tohidi
Summer 1401

**Object Detection** **mapping**

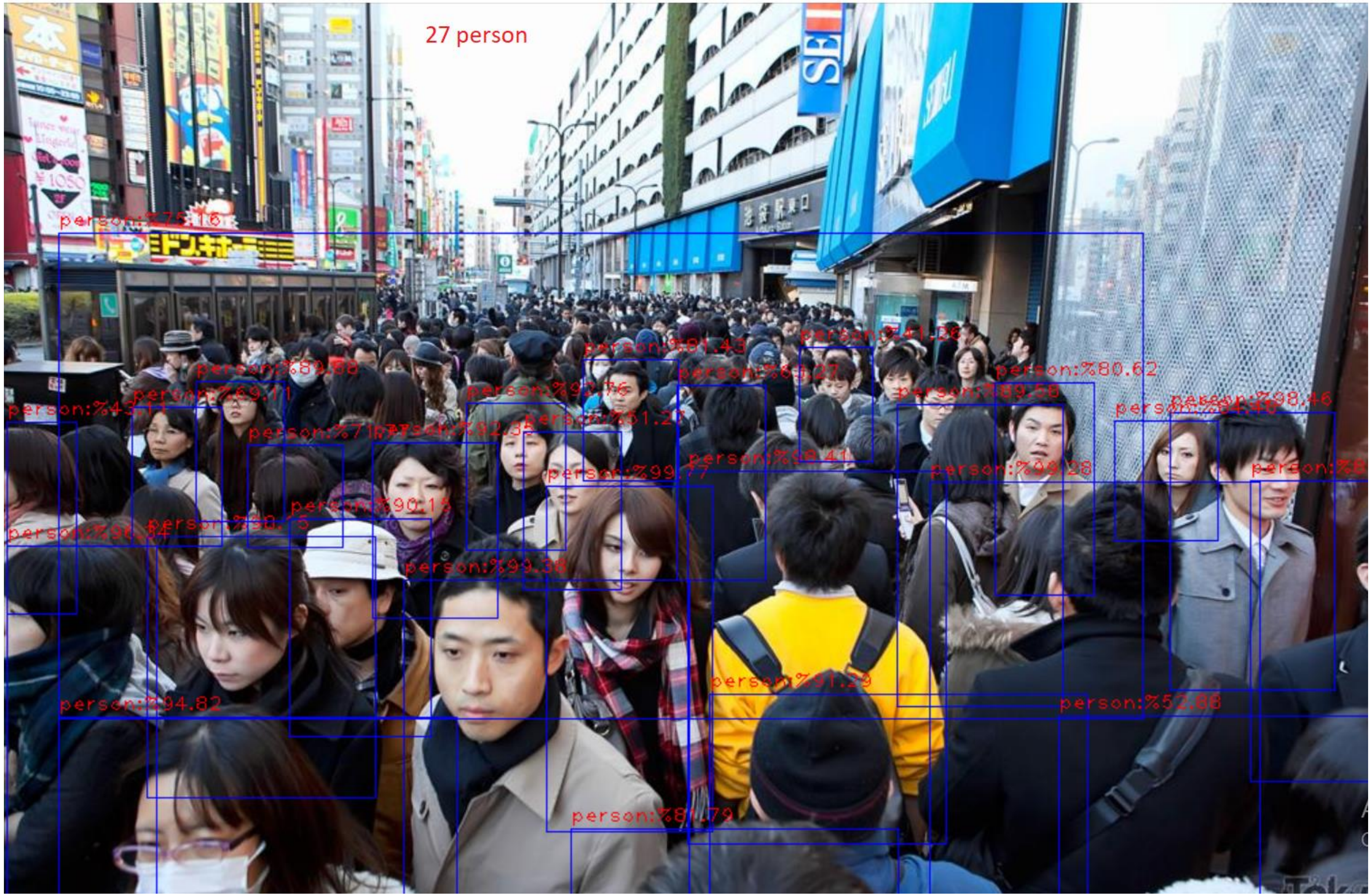
<u>Two</u>
types of
crowd detection:



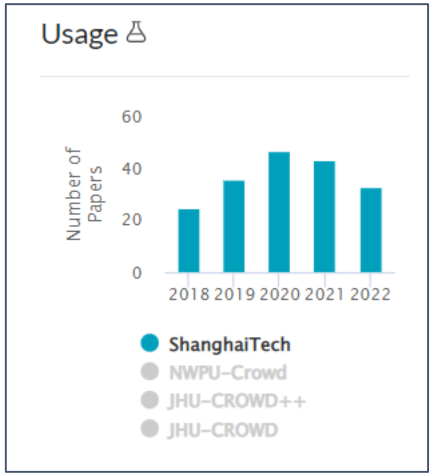

# YOLO v3 : 20


person:%69.57
person:%72.26
person:%97.79
person:%85.67
person:%94.71
person:%78.91
person:%94.61
person:%99.94

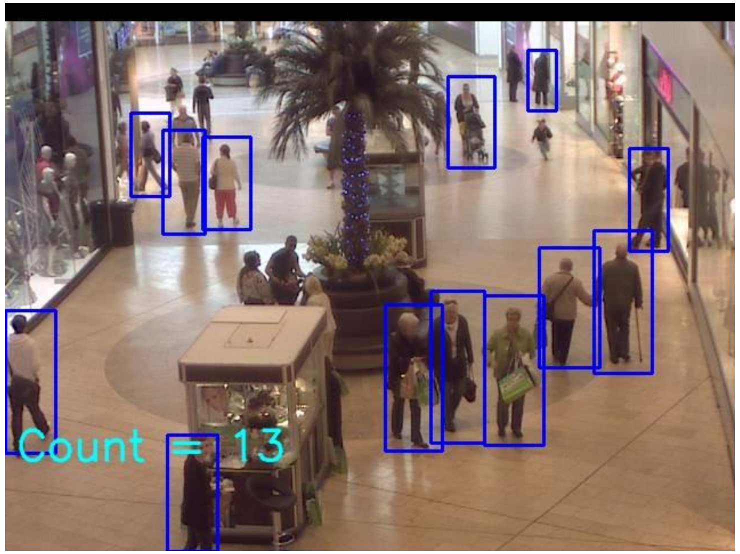
# YOLO v3: 27



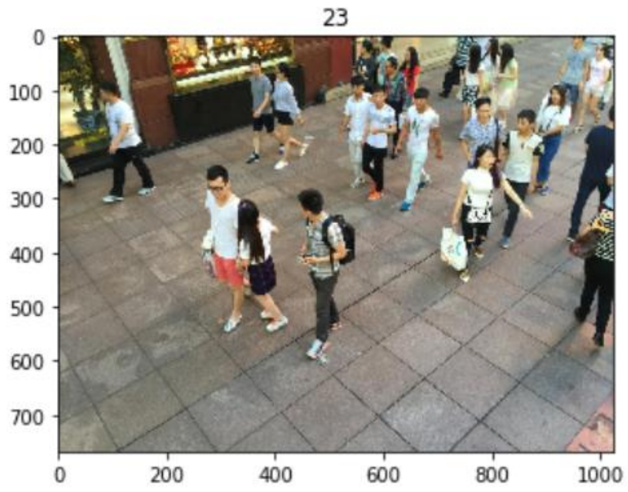
# ShanghaiTech



Object Detection

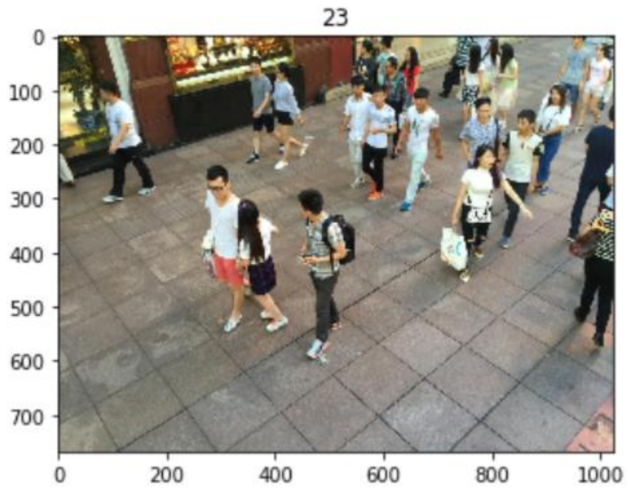



**Mapping**



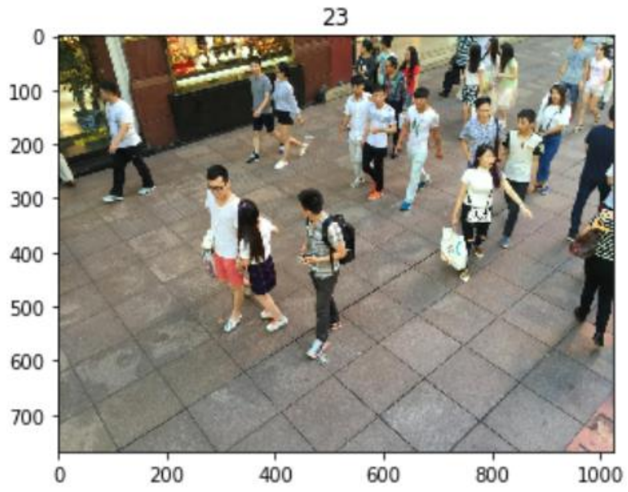

CNN Regression

24

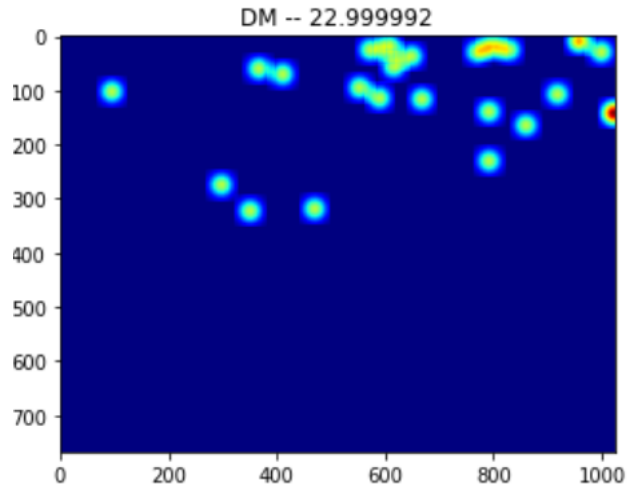



CNN Regression

24




FCN Regression



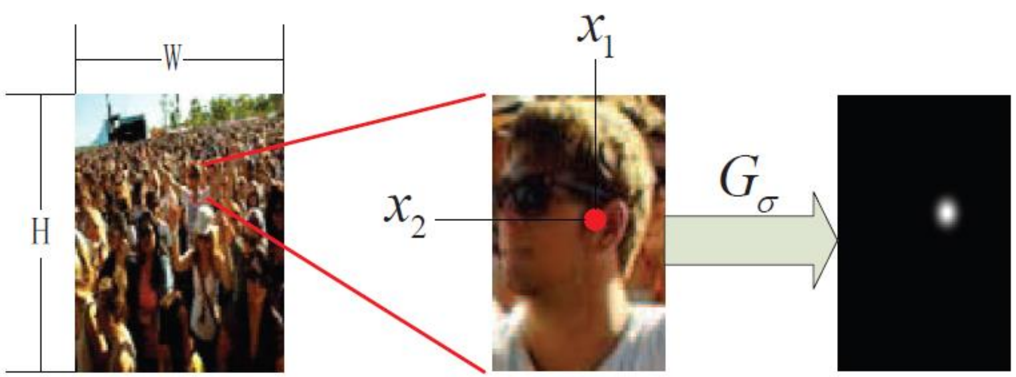



**Figure 3:** The process of generating density map with Gaussian kernel.

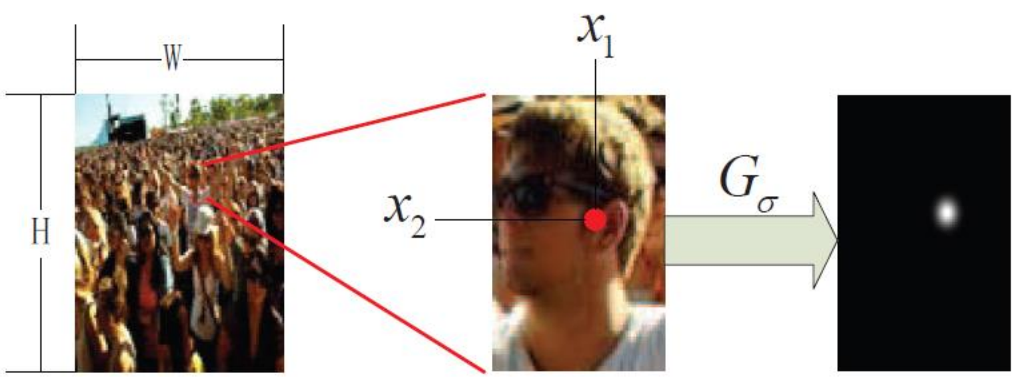



**Figure 3:** The process of generating density map with Gaussian kernel.

```
def gen_density_map_gaussian(im, points, sigma=4):
    density_map = np.zeros(im.shape[:2], dtype=np.float32)
    h, w = density_map.shape[:2]
    num_gt = np.squeeze(points).shape[0]
    if num_gt == 0:
        return density_map
    if sigma == 4:
        # Adaptive sigma in CSRNet.
        leafsize = 2048
        tree = scipy.spatial.KDTree(points.copy(), leafsize=leafsize)
        distances, _ = tree.query(points, k=4)
    for idx_p, p in enumerate(points):
        p = np.round(p).astype(int)
        p[0], p[1] = min(h-1, p[1]), min(w-1, p[0])
        gaussian_radius = sigma * 2 - 1
        if sigma == 4:
            sigma = max(int(np.sum(distances[idx_p][1:4]) * 0.1), 1)
            gaussian_radius = sigma * 3
        gaussian_map = np.multiply(
            cv2.getGaussianKernel(int(gaussian_radius*2+1), sigma),
            cv2.getGaussianKernel(int(gaussian_radius*2+1), sigma).T
        )
…
    return density_map
```

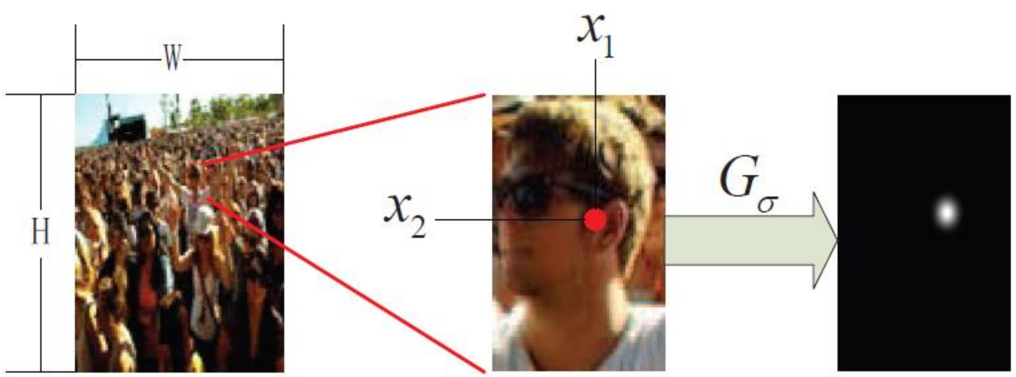
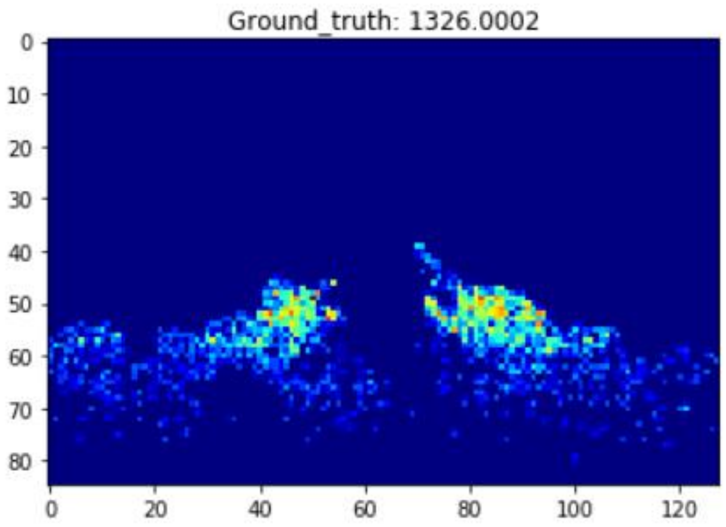



**Figure 3:** The process of generating density map with Gaussian kernel.


```
def gen_density_map_gaussian(im, points, sigma=4):
    density_map = np.zeros(im.shape[:2], dtype=np.float32)
    h, w = density_map.shape[:2]
    num_gt = np.squeeze(points).shape[0]
    if num_gt == 0:
        return density_map
    if sigma == 4:
        # Adaptive sigma in CSRNet.
        leafsize = 2048
        tree = scipy.spatial.KDTree(points.copy(), leafsize=leafsize)
        distances, _ = tree.query(points, k=4)
    for idx_p, p in enumerate(points):
        p = np.round(p).astype(int)
        p[0], p[1] = min(h-1, p[1]), min(w-1, p[0])
        gaussian_radius = sigma * 2 - 1
        if sigma == 4:
            sigma = max(int(np.sum(distances[idx_p][1:4]) * 0.1), 1)
            gaussian_radius = sigma * 3
        gaussian_map = np.multiply(
            cv2.getGaussianKernel(int(gaussian_radius*2+1), sigma),
            cv2.getGaussianKernel(int(gaussian_radius*2+1), sigma).T
        )
…
    return density_map
```

# CNN Regression:

DM = gen_density_map_gaussian(k, gt, sigma=sigma)

Crowd=(np.sum(DM)  #labels of regression

# CNN Regression::

Lines 46_65
Network Architecture

```
46 def create_model() -> tf.keras.Model:
47     """Function initializes and compiles a regression model
48     with pretrained feature extractor.
49     :return: TF Model object
50     """
51     feature_model = tf.keras.applications.InceptionResNetV2(
52         include_top=False, pooling='avg')
53     feature_model.trainable = False
54
55     model = tf.keras.Sequential([
56         tf.keras.Input((IMAGE_SIZE, IMAGE_SIZE, 3)),
57         feature_model,
58         tf.keras.layers.Dense(512, activation='selu'),
59         tf.keras.layers.Dense(1)
60     ])
61     model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=LEARNING_RATE),
62                   loss=tf.keras.losses.MeanSquaredError(),
63                   metrics=[tf.keras.metrics.MeanAbsoluteError()])
64
65     return model
```

# CNN Regression

# CNN Regression

Custom Network Architecture such as :

```
x = Conv2D(64, (3, 3), strides=(1, 1), padding='same', activation='relu')(input_flow)
x = Conv2D(64, (3, 3), strides=(1, 1), padding='same', activation='relu')(x)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Conv2D(128, (3, 3), strides=(1, 1), padding='same', activation='relu')(x)
x = Conv2D(128, (3, 3), strides=(1, 1), padding='same', activation='relu')(x)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Conv2D(256, (3, 3), strides=(1, 1), padding='same', dilation_rate=2, activation='relu')(x)
x = Conv2D(128, (3, 3), strides=(1, 1), padding='same', dilation_rate=2, activation='relu’)(x)
x = Conv2D(64, (3, 3), strides=(1, 1), padding='same', dilation_rate=2, activation='relu')(x)
x= Flatten()(x)
x= Dense(512, activation='relu')(x)
output_flow=Dense(1, activation='linear')(x)
```

# CNN Regression

Custom Network Architecture such as :

```
x = Conv2D(64, (3, 3), strides=(1, 1), padding='same', activation='relu')(input_flow)
x = Conv2D(64, (3, 3), strides=(1, 1), padding='same', activation='relu')(x)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Conv2D(128, (3, 3), strides=(1, 1), padding='same', activation='relu')(x)
x = Conv2D(128, (3, 3), strides=(1, 1), padding='same', activation='relu')(x)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Conv2D(256, (3, 3), strides=(1, 1), padding='same', dilation_rate=2, activation='relu')(x)
x = Conv2D(128, (3, 3), strides=(1, 1), padding='same', dilation_rate=2, activation='relu’)(x)
x = Conv2D(64, (3, 3), strides=(1, 1), padding='same', dilation_rate=2, activation='relu')(x)
x= Flatten()(x)
x= Dense(512, activation='relu')(x)
output_flow=Dense(1, activation='linear')(x)
```

# FCN Regression

## 1-1

Density map dim=( 683,1024,1) ⟶ Feature resulted from CNN:=( 683,1024,1)

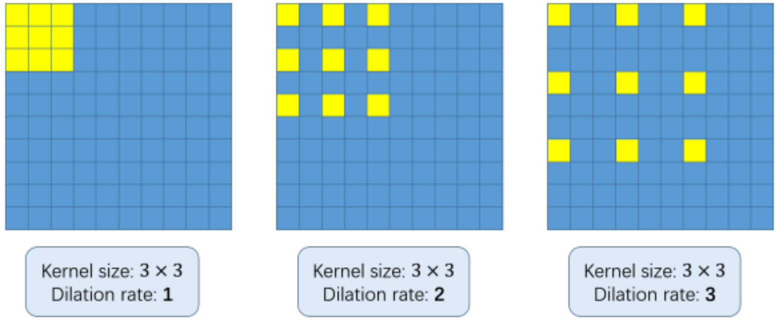


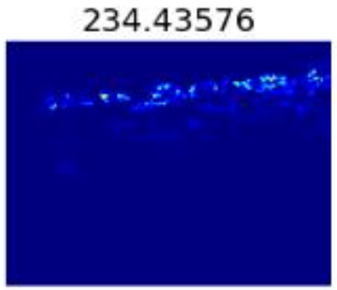

Figure 3. $3 \times 3$ convolution kernels with different dilation rate as 1, 2, and 3.

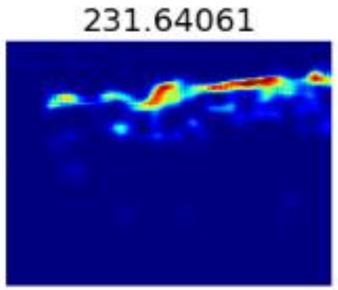
Dilation preserves the size.

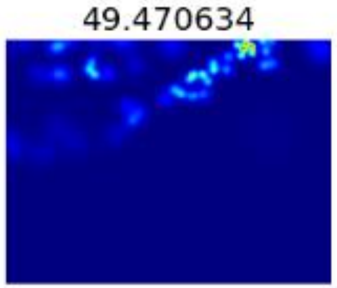
| Configurations of CSRNet | | | |
|---|---|---|---|
| A | B | C | D |
| input(unfixed-resolution color image) | | | |
| front-end (fine-tuned from VGG-16) | | | |
| conv3-64-1<br>conv3-64-1 | | | |
| max-pooling | | | |
| conv3-128-1<br>conv3-128-1 | | | |
| max-pooling | | | |
| conv3-256-1<br>conv3-256-1<br>conv3-256-1 | | | |
| max-pooling | | | |
| conv3-512-1<br>conv3-512-1<br>conv3-512-1 | | | |
| | conv3-512-2<br>conv3-512-2<br>conv3-512-2<br>conv3-256-2<br>conv3-128-2<br>conv3-64-2 | | |
| conv1-1-1 | | | |

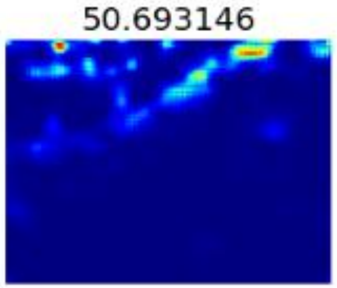
Density map

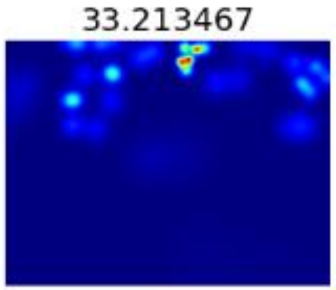
CSRNet

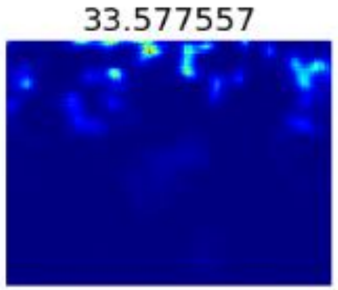


- Fully convolutional regression network for accurate detection of measurement points(2017)

Michal Sofka, Fausto Milletari, Jimmy Jia, and Alex Rothberg

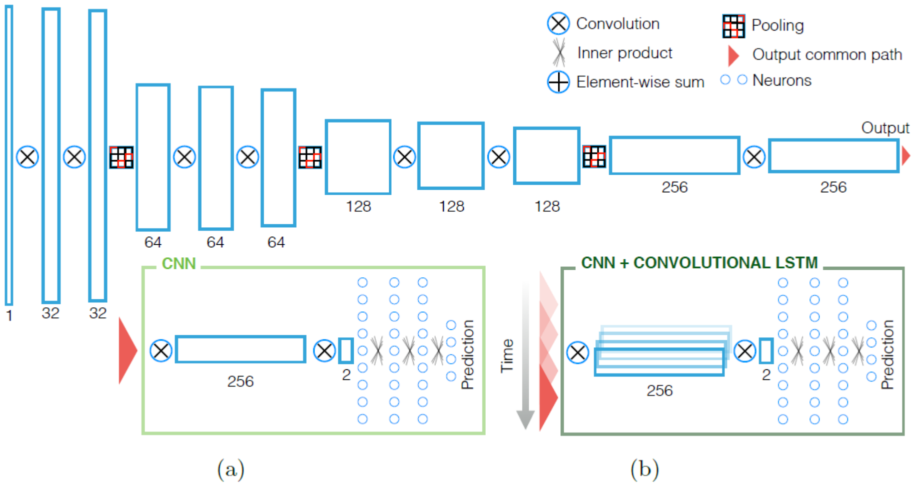



**Fig. 3.** (a) Convolutional Neural Network (CNN) architecture to regress the keypoint locations. (b) CNN with feature maps processed by a Convolutional LSTM to model temporal constraints. CLSTM processes 256 feature maps and its output is used to compute the point location estimate.



The temporal consistency of the estimates is achieved by a long-short term memory cells which process several previous frames to refine estimate of the current frame.

- Rethinking Spatial Invariance of Convolutional Networks for Object Counting(2021)

$$\mathbf{Y}_s = \sum_{i=0}^{N} G(\mu_i, \mathbf{\Sigma}_i) * \mathbf{X}_s + \mathbf{b}_s,$$

# References:


- **CNN-based Density Estimation and Crowd Counting: A Survey(**Guangshuai Gao, Junyu Gao, et all, 2020**)**
- **Locate, Size and Count: Accurately Resolving People in Dense Crowds via Detection(**Deepak Babu Sam, Skand Vishwanath Peri,et all. 2020**)**
- **DPDnet: A Robust People Detector using Deep Learning with an Overhead Depth Camera(**David Fuentes-Jimenez, Roberto Martin-Lopez**)**
- **CSRNet: Dilated Convolutional Neural Networks for Understanding the Highly Congested Scenes (**Yuhong Li1,2 , Xiaofan Zhang, et all. 2018**)**
- **Single-Image Crowd Counting via Multi-Column Convolutional Neural Network**
- **(**Yingying Zhang, Desen Zhou et all. 2016**)**

Give my github repo a

@sara-git

if you found this presentation useful.