FINAL TERM PRESENTATION

COSE474 DEEP LEARNING

CRSNET+CBAM

SOLVING SCALE VARIATION IN CROWD DENSITY ESTIMATION

Git hub Link: https://github.com/whitepieridae/2025IR0136COSE47400

GROUP 6

MAHIRAH SOFEA

JULIA IRSALINA[redacted]

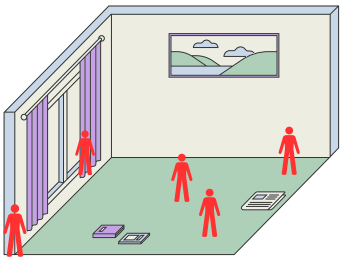
NUR MUSHIRA

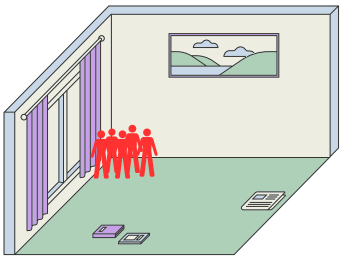
INTRODUCTION

STAMPEDE

SAFETY?

not just about number, but how **dense** the crowd !





density estimation > crowd counting

Our baseline model: CSRNet

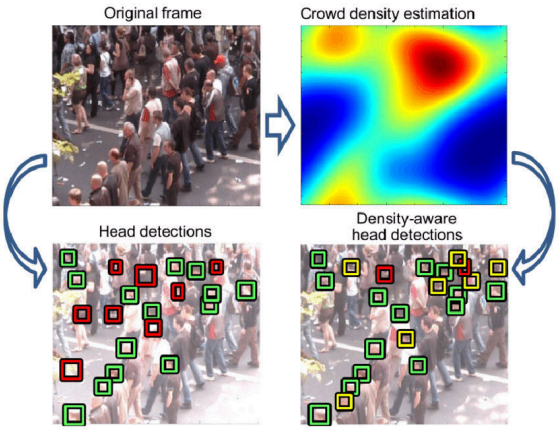
Our Justification:

CSRNet: Dilated Convolutional Neural Networks for Understanding the Highly Congested Scenes

Yuhong Li[1,2], Xiaofan Zhang[1], Deming Chen[1]
[1]University of Illinois at Urbana-Champaign
[2]Beijing University of Posts and Telecommunications
{leeyh,xiaofan3,dchen}@illinois.edu

In the research paper:

- generate high-quality density map
- perform accurate count estimation
- dilated convolutional layers:
  - capture multi-scale contextual information in crowded scenes
  - expands receptive field without losing resolution



Our baseline model: CSRNet

Scale variation issue!

Problem Identification:
CSRNet lacks the ability to adapt to varying object scales due to fixed receptive fields.

smaller

bigger

(Li et al., 2018)

Training Configurations

	Midterm	Current
Optimizer	**SGD**	**Adam**
Learning Rate	0.00001	**0.0001**
Epochs	10	**30**

- Switched to **Adam** for faster and more stable convergence
- **Increased learning rate and training duration** to improve performance
- we train on **200 image** → validate on **50 images**

Learning Rate - Fine Tuning

lr = 1e-3

```
 * MAE: 330.793, RMSE: 500.340, SSIM: 0.217, PSNR: 18.738
 * Best MAE so far: 306.400
epoch 16, processed 6400 samples, lr 0.0010000000
Epoch: [16][0/400]      Time 0.041 (0.041)      Data 0.014 (0.014)      Loss 11.4216 (11.4216)
Epoch: [16][100/400]    Time 0.071 (0.106)      Data 0.030 (0.071)      Loss 55.0545 (32.0633)
Epoch: [16][200/400]    Time 0.123 (0.104)      Data 0.091 (0.069)      Loss 21.9780 (33.0604)
Epoch: [16][300/400]    Time 0.088 (0.103)      Data 0.044 (0.069)      Loss 52.8140 (31.5606)
 * MAE: 323.031, RMSE: 503.476, SSIM: 0.224, PSNR: 18.833
 * Best MAE so far: 306.400
epoch 17, processed 6800 samples, lr 0.0010000000
Epoch: [17][0/400]      Time 0.057 (0.057)      Data 0.021 (0.021)      Loss 5.5991 (5.5991)
Epoch: [17][100/400]    Time 0.121 (0.102)      Data 0.084 (0.068)      Loss 8.1267 (34.3471)
Epoch: [17][200/400]    Time 0.108 (0.101)      Data 0.078 (0.067)      Loss 15.4338 (35.0282)
Epoch: [17][300/400]    Time 0.126 (0.102)      Data 0.092 (0.068)      Loss 11.4062 (34.6163)
 * MAE: 320.663, RMSE: 504.824, SSIM: 0.227, PSNR: 18.862
 * Best MAE so far: 306.400
```

Learning Rate - Fine Tuning

lr = 1e-5 **mae is reduced!**

```
Epoch: [15][0/400]      Time 0.077 (0.077)      Data 0.029 (0.029)      Loss 3135.8740 (3135.8740)
Epoch: [15][100/400]    Time 0.118 (0.104)      Data 0.078 (0.067)      Loss 3390.9250 (928.4084)
Epoch: [15][200/400]    Time 0.132 (0.103)      Data 0.091 (0.067)      Loss 14.8435 (947.9044)
Epoch: [15][300/400]    Time 0.129 (0.104)      Data 0.092 (0.068)      Loss 2713.9680 (3335.0351)
 * MAE: 112.852, RMSE: 196.065, SSIM: 0.472, PSNR: 21.010
 * Best MAE so far: 110.689
epoch 16, processed 6400 samples, lr 0.0000100000
Epoch: [16][0/400]      Time 0.073 (0.073)      Data 0.029 (0.029)      Loss 366.8820 (366.8820)
Epoch: [16][100/400]    Time 0.122 (0.105)      Data 0.084 (0.070)      Loss 138.5269 (531.8713)
Epoch: [16][200/400]    Time 0.084 (0.108)      Data 0.046 (0.071)      Loss 178.9455 (518.4971)
Epoch: [16][300/400]    Time 0.134 (0.105)      Data 0.093 (0.069)      Loss 115.0331 (420.6795)
 * MAE: 106.824, RMSE: 185.702, SSIM: 0.462, PSNR: 21.052
Saved best model at epoch 16 with MAE: 106.824, SSIM: 0.462, PSNR: 21.052
 * Best MAE so far: 106.824
epoch 17, processed 6800 samples, lr 0.0000100000
Epoch: [17][0/400]      Time 0.065 (0.065)      Data 0.019 (0.019)      Loss 0.5753 (0.5753)
```

Output Model shape = Target shape

```
target = cv2.resize(target,(target.shape[1]//8,target.shape[0]//8),interpolation = cv2.INTER_CUBIC)*64
```

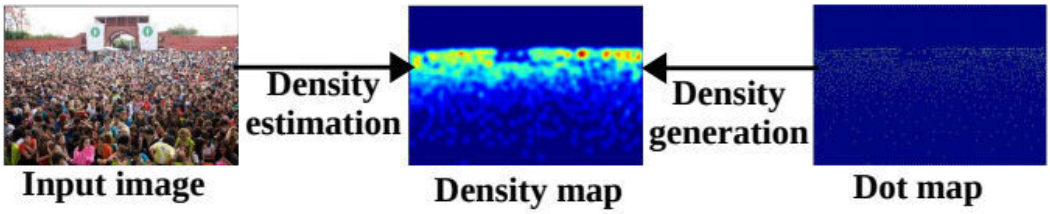
```
epoch 0, processed 0 samples, lr 0.0001000000
[DEBUG] img shape: torch.Size([1, 3, 680, 1024]), target shape (before): torch.Size([1, 1, 680, 1024])
Model output shape: torch.Size([1, 1, 85, 128])
[DEBUG] target shape (after resize): torch.Size([1, 1, 85, 128])
```

DATASET PREPARATION:
DENSITY MAP OF GROUND TRUTH

Our Experiment:

Our reference: (Li et al., 2018)

Dataset	Generating method
ShanghaiTech Part_A [18]	Geometry-adaptive kernels
UCF_CC_50 [22]	
ShanghaiTech Part_B [18]	Fixed kernel: $\sigma = 15$



ShanghaiTech A: using Geometry Adaptive Gaussian

Formula reference:

Single-Image Crowd Counting via Multi-Column Convolutional Neural Network

Yingying Zhang Desen Zhou Siqin Chen Shenghua Gao Yi Ma
Shanghaitech University
{zhangyy2,zhouds,chensq,gaoshh,mayi}@shanghaitech.edu.cn

- Each person's head is blurred using a custom Gaussian
- The blur size changes depending on how crowded the area is
- **Sparse areas → bigger blur, dense areas → smaller blur**
- For each head, we:
  - Find its k-nearest neighbors using KDTree
  - Calculate the average distance to those neighbors

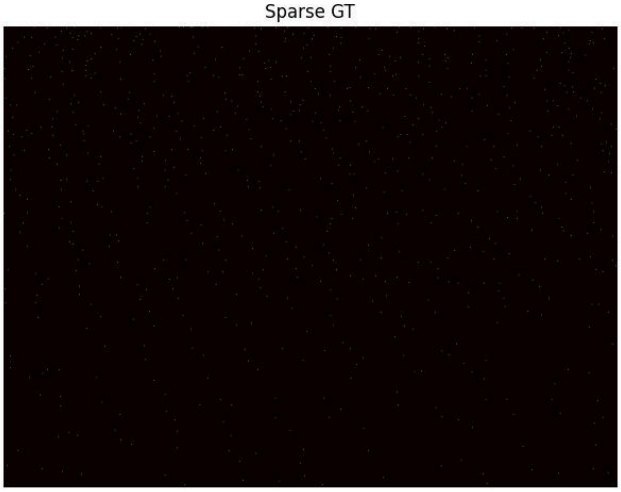
$$F(\boldsymbol{x}) = \sum_{i=1}^{N} \delta(\boldsymbol{x} - \boldsymbol{x}_i) * G_{\sigma_i}(\boldsymbol{x}), \quad \text{with} \quad \sigma_i = \beta \bar{d}^i$$

```python
#there is N gaussian blur->slow

# inspired from https://github.com/davideverona/deep-crowd-counting_crowdnet
def geoadaptive_gaussian(gt):
    print (gt.shape)#takes a 2d binary with 1s marking head positions
    density = np.zeros(gt.shape, dtype=np.float32)

    gt_count = np.count_nonzero(gt)
    if gt_count == 0:
        return density

    pts = np.array(list(zip(np.nonzero(gt)[1], np.nonzero(gt)[0])))
    #find all coordinates where gt[y,x]==1 and turn into list of tuples (x,y)


    # build kdtree for nearest neighbour search
    # for each headpoint, query its 3 neighbours (exc itself)
    # which are distances[i][1], distances[i][2], distances[i][3].

    leafsize = 2048  #larger leafsize reduces the tree's depth for faster query
    tree = scipy.spatial.KDTree(pts.copy(), leafsize=leafsize)
    distances, locations = tree.query(pts, k=4)
    print ('generate density...')


    # loop over each head and generate its density
    # create a blank image, then set the value to 1 at that head's location.
    for i, pt in enumerate(pts):
        pt2d = np.zeros(gt.shape, dtype=np.float32)
        pt2d[pt[1],pt[0]] = 1. #pt[1] is y, pt[0] is x

        if gt_count > 1:
            # compute adaptive part
            # calc sigma based on avg distance to the 3 nearest neighbours and scale w 0.1
            #sigma = (distances[i][1]+distances[i][2]+distances[i][3])*0.1
            sigma = np.mean(distances[i][1:4]) * 0.3

        else:
            sigma = np.average(np.array(gt.shape))/4. # asign default sigma

        #apply gausian blur on single-point image and add to total density maps
        density += scipy.ndimage.gaussian_filter(pt2d, sigma, mode='constant')
    print ('done.')
    return density


#add up all the blurry circles into one final heatmap
```

ShanghaiTech A: using Geometry Adaptive Gaussian

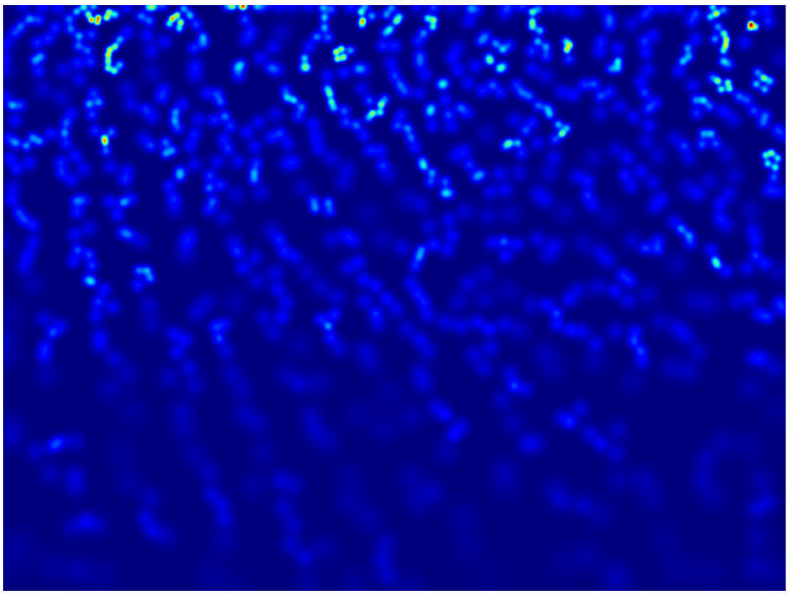
Original Image
Actual Count: 817

Sparse GT



ShanghaiTech A: using Geometry Adaptive Gaussian

Density Map (Geo-Adaptive)
GT Count: 798.25

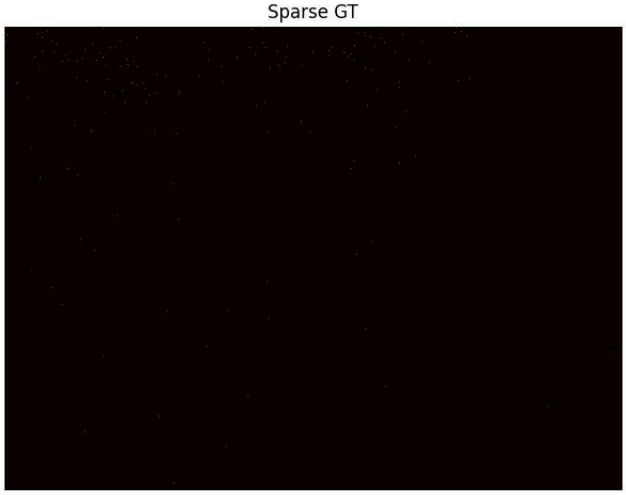


ShanghaiTech B: using Fixed Gaussian = 15

```
from scipy.ndimage import gaussian_filter
```

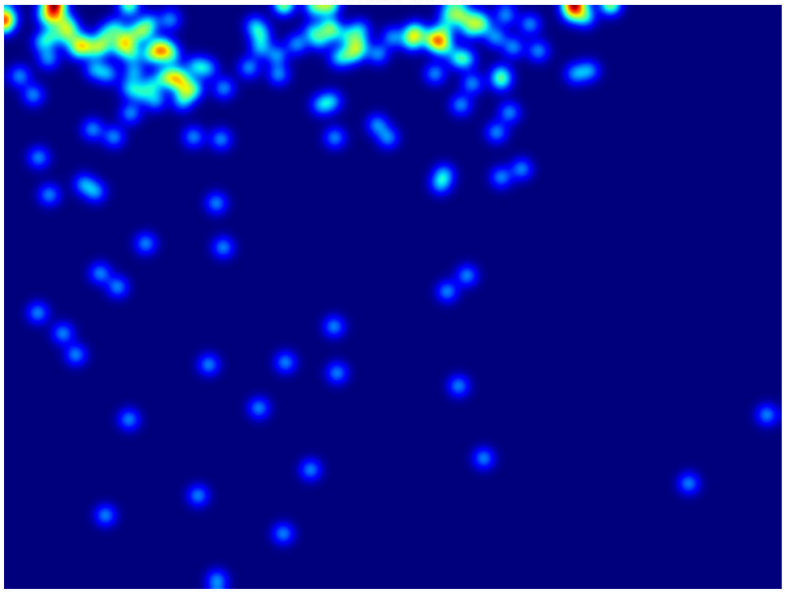
Original Image
Actual Count: 152

Sparse GT



ShanghaiTech B: using Fixed Gaussian = 15

Density Map (Fixed)
GT Count: 152.00



EXPERIMENT 1 : IMPROVE OUR PROPOSED TECHNIQUE

Proposed Techniques: global_gaussian

Goal?

To generate ground truth density maps that reflect both **dense vs. sparse** regions and **near vs. far** object in the scene.

Our Hypothesis:

Using **a single global Gaussian** kernel for density map generation enforces **uniform importance** across all head annotations, regardless of their scale or spatial location. This uniform supervision may guide the model to generalize better across varying object scales, thereby improving robustness to scale variation and perspective distortion in crowd scenes.

ShanghaiTech A: using Global Gaussian

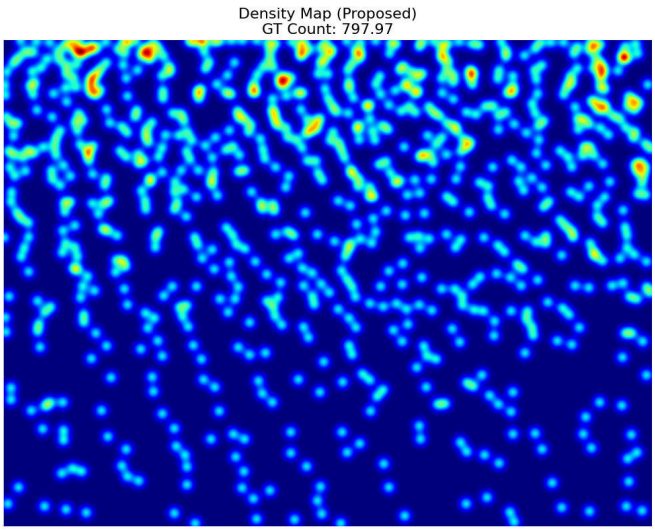
- Entire density map is blurred using a single global Gaussian
- Blur size (σ) depends on the average distance between all heads
- For the whole image, we:
  - Build a KDTree on all head points
  - Compute the average distance to the 3 nearest neighbors for each head
  - Take the mean of all distances and scale by a constant β = 0.3

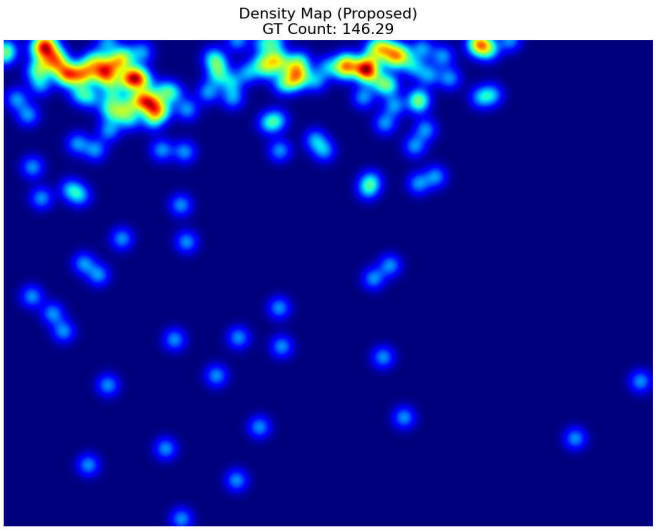
$$F(x) = \sum_{i=1}^{N} \delta(x - x_i) * G_\sigma(x), \quad \text{with} \quad \sigma = \beta \cdot \frac{1}{N} \sum_{i=1}^{N} \bar{d}_i$$

Only 1 Gaussian blur applied → **very fast**

```python
def proposed_gaussian(gt):
    print(gt.shape)
    density = np.zeros(gt.shape, dtype=np.float32) # initiliazing by creating density map of

    gt_count = np.count_nonzero(gt)#count #head annotations in the img
    if gt_count == 0: #if none, return empty density
        return density

    # extract coordinate of all head points in the img
    pts = np.array(list(zip(np.nonzero(gt)[1], np.nonzero(gt)[0])))
    #this return (y_indices, x_indices) of where gt==1
    #so zip(..[1])..[0]) creates (x,y) pairs


    for x, y in pts:
        density[y, x] = 1. # mark each head point in new density map w a value of 1
    print('generate density...')

    #if there is more than one head:
    #   build a KDTree ONCE for fast nearest-neighbour
    #   for each headpoint, query its 3 neighbours (exc itself) at ONCE
    #   get avg dist of those 3 neighbours
    #   take avg of all dist and scale by 0.1-> our single sigma

    # no loop

    if gt_count > 1:
        leafsize = 2048
        tree = scipy.spatial.KDTree(pts.copy(), leafsize=leafsize)#default metric used is Eu
        distances, locations = tree.query(pts, k=4) # "for each head point in pts, give its
        sigma = np.mean(distances[:, 1:4]) * 0.3 #skip the first column of distance to itsel
    else:
        sigma = np.average(np.array(gt.shape))/4. #if there is only one head, use a fixed σ

    # apply one global Gaussian blur using the computed σ
    # this spreads the 1s of head locations into smooth blobs based on spacing
    density = scipy.ndimage.gaussian_filter(density, sigma, mode='constant')

    print('done.')
    return density
```

Our proposed technique: global_gaussian





Comparison Time (One Input)

Part A

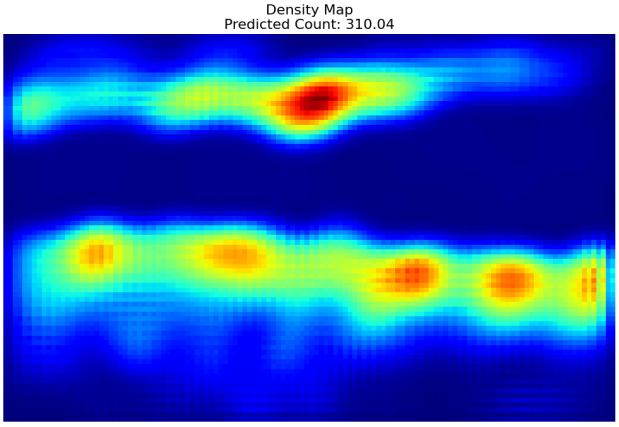
```
[A 1/1] Processing: /home/elicer/.cache/kagglehub/datasets/tthien/shanghaitech/versions/1/ShanghaiTech/part_A/test_
data/images/IMG_5.jpg
[GEOADAPTIVE] Starting...
(768, 1024)
generate density...
done.
        Saved to: /home/elicer/.cache/kagglehub/datasets/tthien/shanghaitech/versions/1/ShanghaiTech/part_A/test_da
ta/generated-h5/IMG_5_geoadaptive.h5
        Time taken: 38.7780 seconds

[FIXED] Starting...
        Saved to: /home/elicer/.cache/kagglehub/datasets/tthien/shanghaitech/versions/1/ShanghaiTech/part_A/test_da
ta/generated-h5/IMG_5_fixed.h5
        Time taken: 0.0719 seconds

[PROPOSED] Starting...
(768, 1024)
generate density...
done.
        Saved to: /home/elicer/.cache/kagglehub/datasets/tthien/shanghaitech/versions/1/ShanghaiTech/part_A/test_da
ta/generated-h5/IMG_5_proposed.h5
        Time taken: 0.0635 seconds
```

Part B

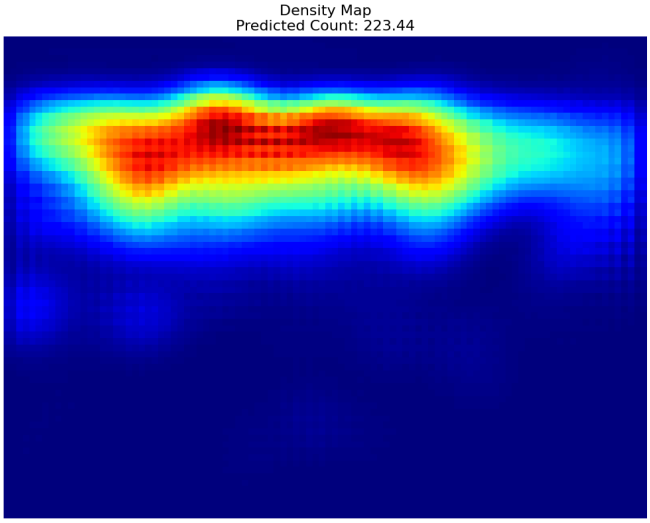
```
[A 1/1] Processing: /home/elicer/.cache/kagglehub/datasets/tthien/shanghaitech/versions/1/ShanghaiTech/part_B/train
_data/images/IMG_305.jpg
[GEOADAPTIVE] Starting...
(768, 1024)
generate density...
done.
        Saved to: /home/elicer/.cache/kagglehub/datasets/tthien/shanghaitech/versions/1/ShanghaiTech/part_B/train_d
ata/generated-h5/IMG_305_geoadaptive.h5
        Time taken: 14.1958 seconds

[FIXED] Starting...
        Saved to: /home/elicer/.cache/kagglehub/datasets/tthien/shanghaitech/versions/1/ShanghaiTech/part_B/train_d
ata/generated-h5/IMG_305_fixed.h5
        Time taken: 0.0740 seconds

[PROPOSED] Starting...
(768, 1024)
generate density...
done.
        Saved to: /home/elicer/.cache/kagglehub/datasets/tthien/shanghaitech/versions/1/ShanghaiTech/part_B/train_d
ata/generated-h5/IMG_305_proposed.h5
        Time taken: 0.0984 seconds
```

	Part A (seconds)	Part B (seconds)
Geometry-Adaptive	38.778	14.1958
Fixed=15	0.0719	0.074
Global (=Proposed)	0.0635	0.0984

CSRNet + Geometry Adaptive Gaussian



CSRNet + Geometry Adaptive Gaussian

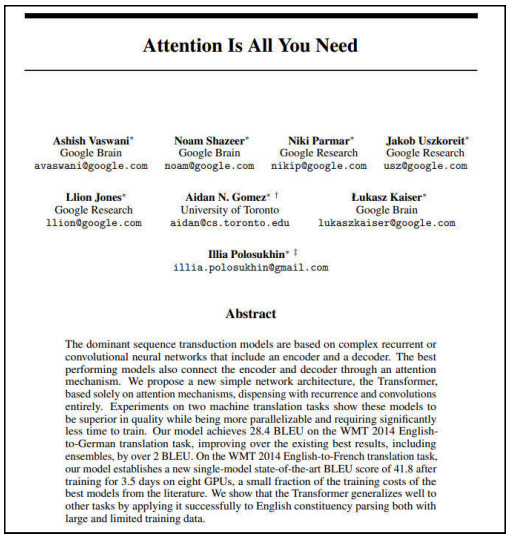


EXPERIMENT 2 :
IMPROVE MODEL ARCHITECTURE
ADD ATTENTION LAYER

CSRNet + CBAM

Why add CBAM ?

To generate ground truth density maps that reflect both **dense vs. sparse** regions and **near vs. far** object in the scene.



Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

CSRNET

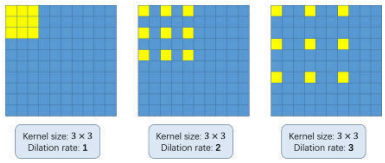


Figure 3. 3 × 3 convolution kernels with different dilation rate as 1, 2, and 3.

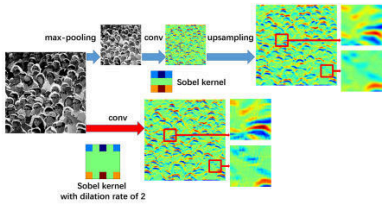
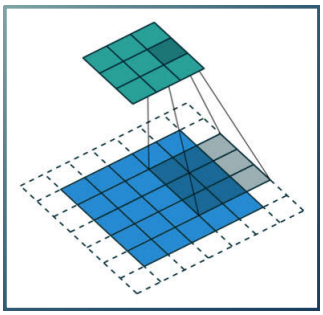
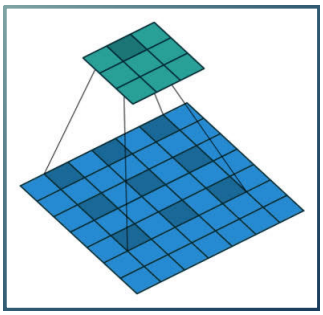


Figure 4. Comparison between dilated convolution and max-pooling, convolution, upsampling. The 3 × 3 Sobel kernel is used in both operations while the dilation rate is 2.





preserve the high-resolution density map without reducing feature map size

CONVOLUTIONAL BLOCK ATTENTION MODULE

CBAM: Convolutional Block Attention Module

Sanghyun Woo*[1], Jongchan Park*†[2], Joon-Young Lee[3], and In So Kweon[1]

[1] Korea Advanced Institute of Science and Technology, Daejeon, Korea
{shwoo93, iskweon77}@kaist.ac.kr
[2] Lunit Inc., Seoul, Korea
jcpark@lunit.io
[3] Adobe Research, San Jose, CA, USA
jolee@adobe.com

Abstract. We propose Convolutional Block Attention Module (CBAM), a simple yet effective attention module for feed-forward convolutional neural networks. Given an intermediate feature map, our module sequentially infers attention maps along two separate dimensions, channel and spatial, then the attention maps are multiplied to the input feature map for adaptive feature refinement. Because CBAM is a lightweight and general module, it can be integrated into any CNN architectures seamlessly with negligible overheads and is end-to-end trainable along with base CNNs. We validate our CBAM through extensive experiments on ImageNet-1K, MS COCO detection, and VOC 2007 detection datasets. Our experiments show consistent improvements in classification and detection performances with various models, demonstrating the wide applicability of CBAM. The code and models will be publicly available.

3 Convolutional Block Attention Module

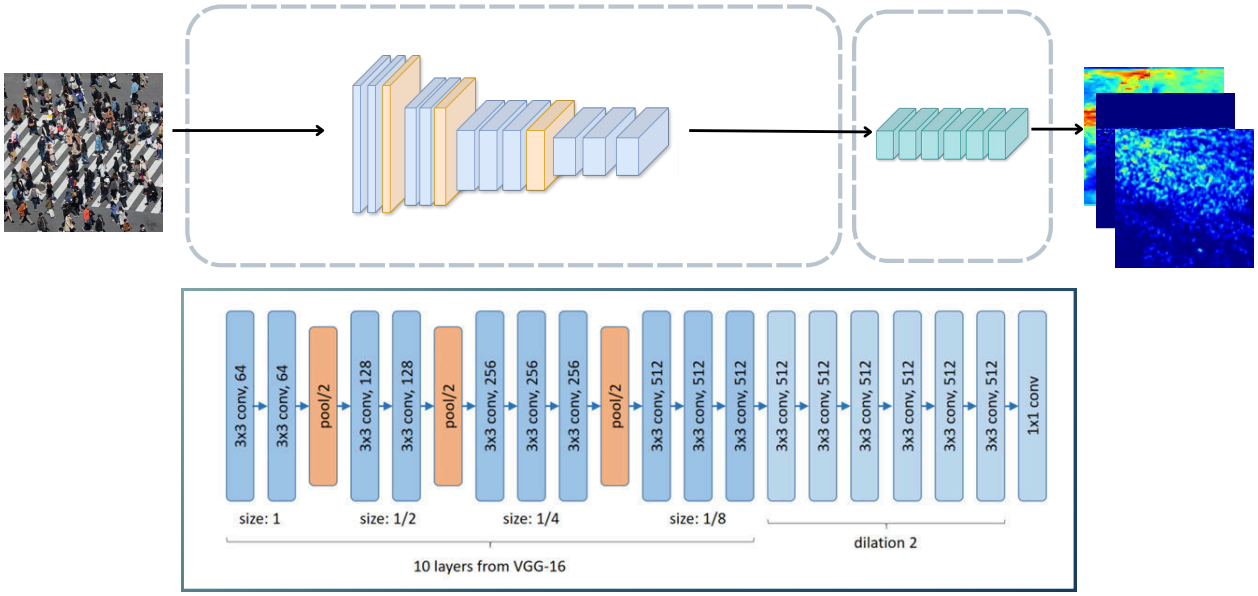
Given an intermediate feature map $\mathbf{F} \in \mathbb{R}^{C\times H\times W}$ as input, CBAM sequentially infers a 1D channel attention map $\mathbf{M_c} \in \mathbb{R}^{C\times 1\times 1}$ and a 2D spatial attention map $\mathbf{M_s} \in \mathbb{R}^{1\times H\times W}$ as illustrated in Fig. 1. The overall attention process can be summarized as:

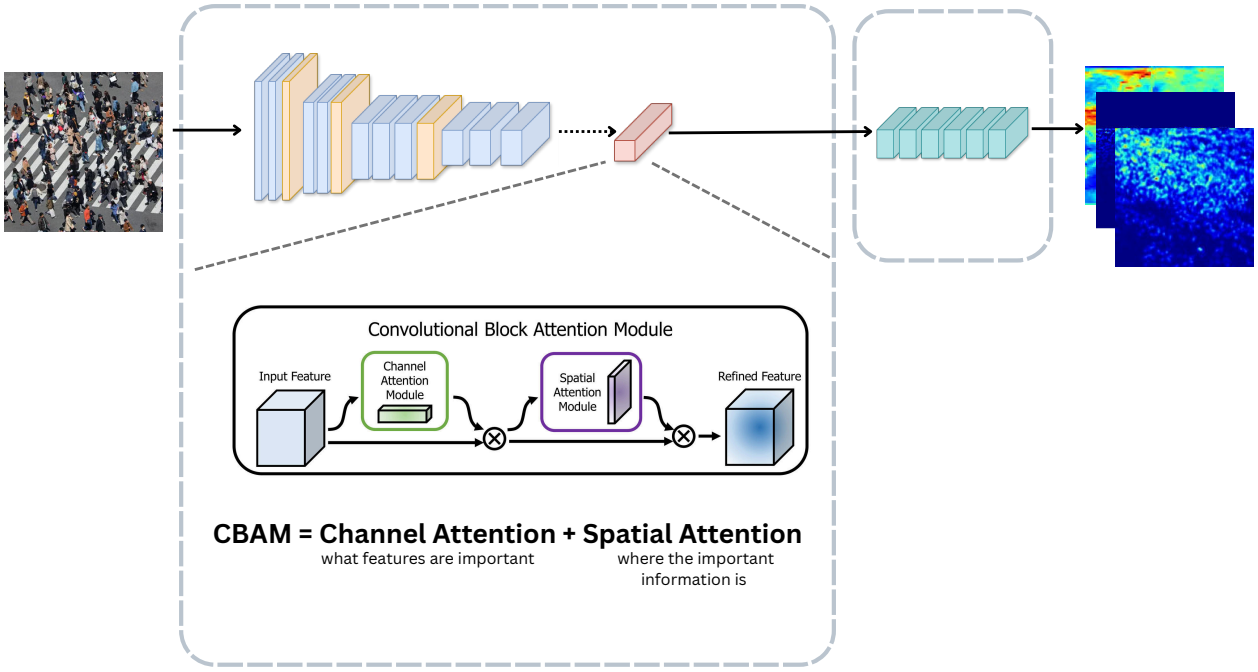
$$\begin{aligned} \mathbf{F}' &= \mathbf{M_c}(\mathbf{F}) \otimes \mathbf{F}, \\ \mathbf{F}'' &= \mathbf{M_s}(\mathbf{F}') \otimes \mathbf{F}', \end{aligned} \tag{1}$$

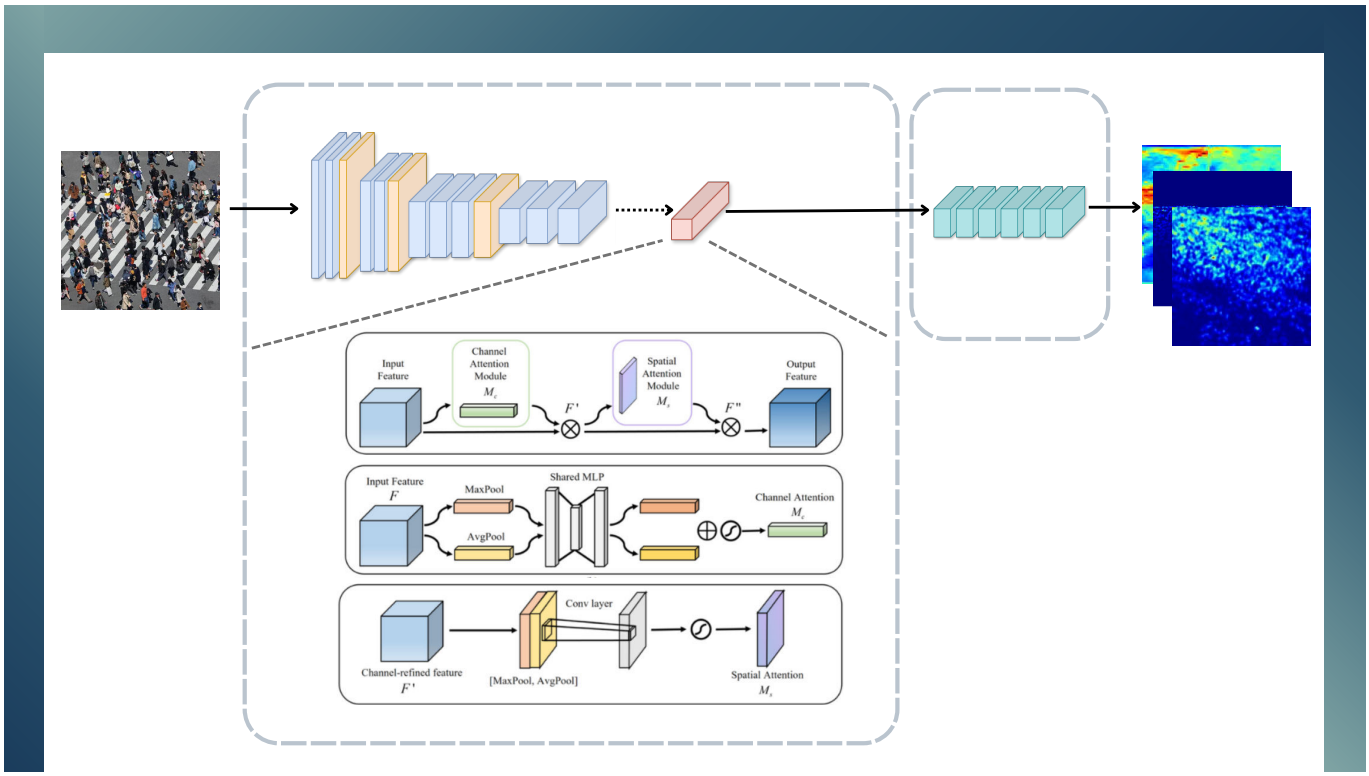
where $\otimes$ denotes element-wise multiplication. During multiplication, the attention values are broadcasted (copied) accordingly: channel attention values are broadcasted along the spatial dimension, and vice versa. $\mathbf{F}''$ is the final refined output. Fig. 2 depicts the computation process of each attention map. The following describes the details of each attention module.

no external labels / masks required hence no extra supervision

CSRNET

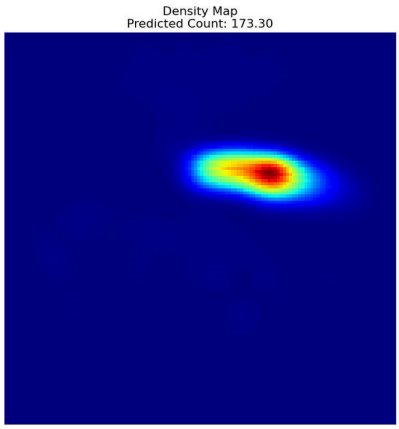

3x3 conv, 64
3x3 conv, 64
pool/2
3x3 conv, 128
3x3 conv, 128
pool/2
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
pool/2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
1x1 conv
size: 1
size: 1/2
size: 1/4
size: 1/8
dilation 2
10 layers from VGG-16


Convolutional Block Attention Module
Input Feature
Channel Attention Module
Spatial Attention Module
Refined Feature
CBAM = Channel Attention + Spatial Attention
what features are important
where the important information is

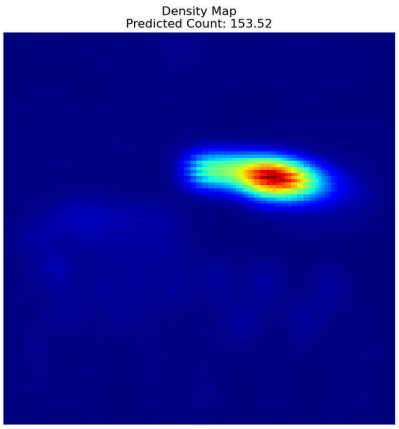


CSRNet vs CSRNet + CBAM

CSRNet



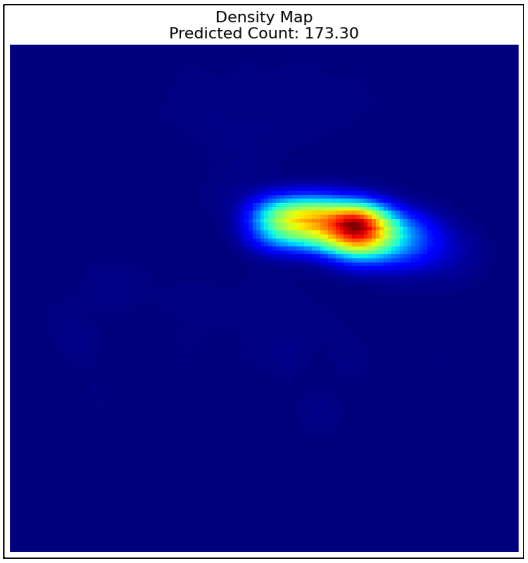
CSRNet_CBAM



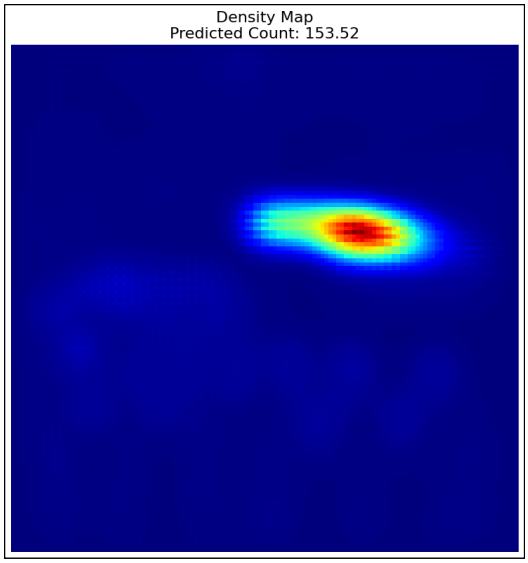
With CBAM, the model starts to detect people in the front, not just the crowded back !

CSRNet vs CSRNet + CBAM

CSRNet



CSRNet_CBAM



EXPERIMENT 3 :
IMPROVE LOSS FUNCTION

COMPOSITE LOSS + COUNT LOSS

Loss Function

TABLE IV: Comparison of crowd counting models: model architecture, inputs and outputs

Model	Input	Input size	Output size	loss	Metric
CrowdCNN [29]	patch	72×72	18×18	combination l_2 loss	MSE, MAE
MCNN [18]	image	arbitrary	input size/4	l_2 loss	MSE, MAE
CrowdNet [63]	patch	225×225	input size/8	l_2 loss	MSE, MAE
MSCNN [54]	patch	225×225	6×6	l_2 loss	MSE, MAE
SCNN [64]	patch	image size/3	-	l_2 loss	MSE, MAE
CMTL [65]	patch	image size/4	input size	l_2 loss, cross-entropy	MSE, MAE
CSRNet [19]	patch	image size/8	input size/8	l_2 loss	MSE, MAE, PSNR, SSIM
SANet [20]	patch	image size/4	input size	composite $(l_2 + l^{SSIM})$ loss	MSE, MAE
SaCNN [61]	patch	image size/8	l_2 loss	l_2 loss	MSE, MAE
DecideNet [66]	patch	$4x3$	-	composite $(l_1 + l_2)$ loss	MSE, MAE

Khan, M. A., Menouar, H., & Hamila, R. (2023). Revisiting crowd counting: State-of-the-art, trends, and future perspectives. Image and Vision Computing, 129, 104597. https://doi.org/10.1016/j.imavis.2022.104597

- CSRNet uses **MSE (L2)** loss for density map regression
- Evaluated with **MAE, RMSE, SSIM, and PSNR**
- Newer works (e.g., SANet) add **SSIM** to improve structural similarity

Our project extends CSRNet with:

Hybrid Loss (Composite Loss)
MSE (L2) + SSIM

(improves pixel accuracy and map structure)

+

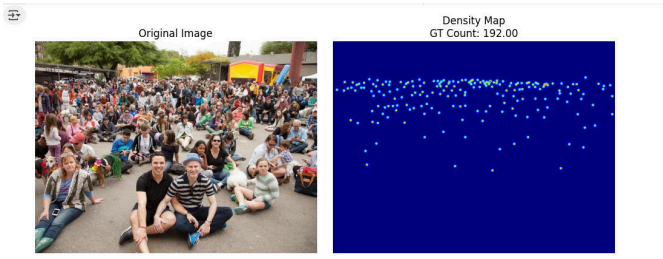
Count Loss
MAE/MSE??

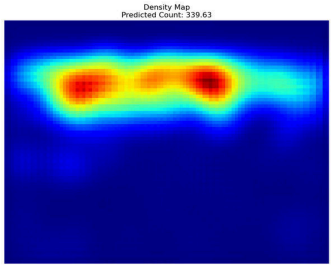
(helping reduce under/overestimation)

Count Loss - MAE or MSE ?

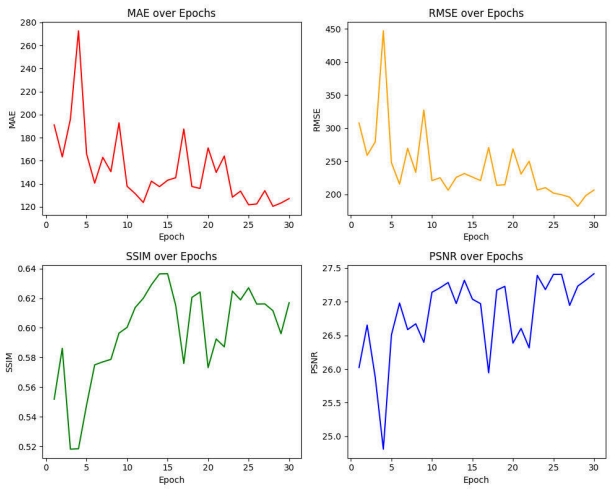
MSE

CSRNet CBAM + Geometry Adaptive Gaussian, 30 epochs





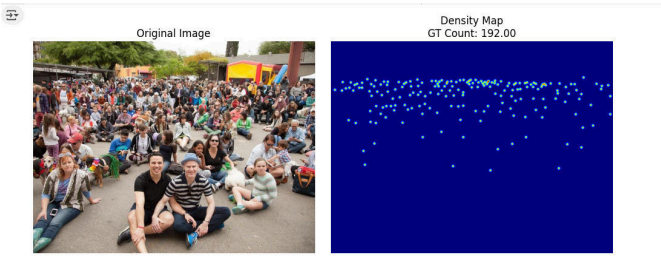

```
==== Best Metrics Summary ====
Best MAE : 120.466
Best RMSE : 182.083
Best SSIM : 0.518
Best PSNR : 24.811
Plot saved as 1. Loss_history.png
Plot saved as 2. Count Accuracy.png
Plot saved as 3. Map Structure.png

==== Final Accuracy Summary ====
Final Train Accuracy: 82.18%
Final Validation Accuracy: 61.12%
Best Validation Accuracy: 69.33%
Plot saved as 4. Evaluation Metrics.png
```

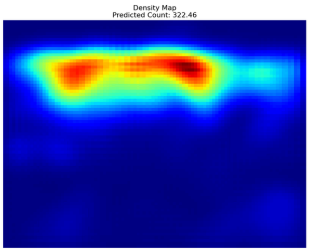


Count Loss - MAE or MSE ?

MAE

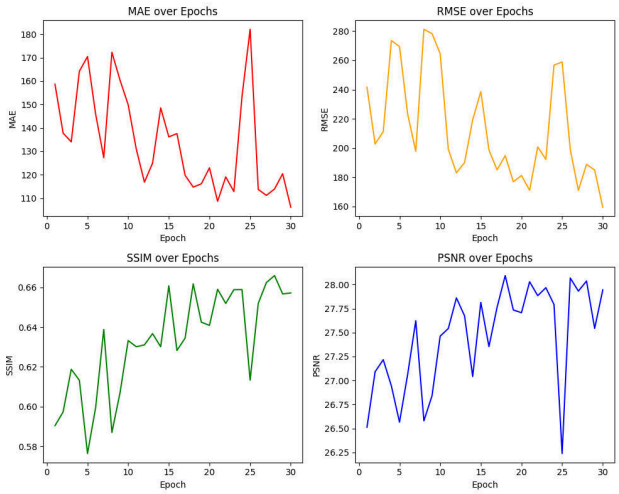
CSRNet CBAM + Geometry Adaptive Gaussian, 30 epochs





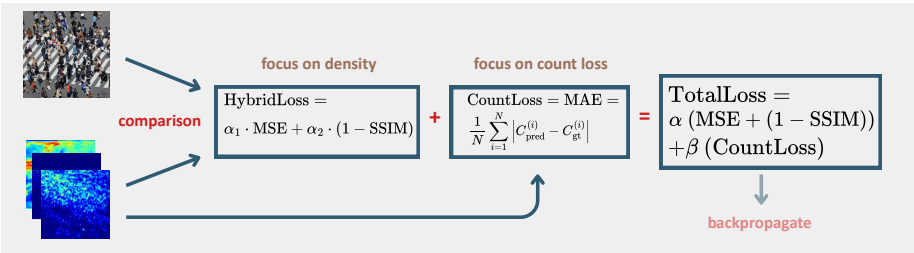

```
==== Best Metrics Summary ====
Best MAE : 106.150
Best RMSE : 159.459
Best SSIM : 0.576
Best PSNR : 26.238
Plot saved as 1. Loss_history.png
Plot saved as 2. Count Accuracy.png
Plot saved as 3. Map Structure.png

==== Final Accuracy Summary ====
Final Train Accuracy: 90.75%
Final Validation Accuracy: 67.19%
Best Validation Accuracy: 71.36%
Plot saved as 4. Evaluation Metrics.png
```

MAE achieved lower MAE and better SSIM/PSNR !



Loss Function

Hybrid loss (MSE + SSIM)



- CSRNet uses **MSE (L2)** loss for density map regression
- However, MSE alone **may cause blurry predictions** and ignores local structural details [1]
- **Structural Similarity Index (SSIM)** improves local consistency of density maps [1]

HybridLoss = MSE(pred, gt) + SSIMLoss(pred, gt)

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(p_i - g_i)^2$$

$$SSIMLoss = 1 - SSIM(pred, gt)$$

SSIM is computed over local patches:

$$\mathrm{SSIM}(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

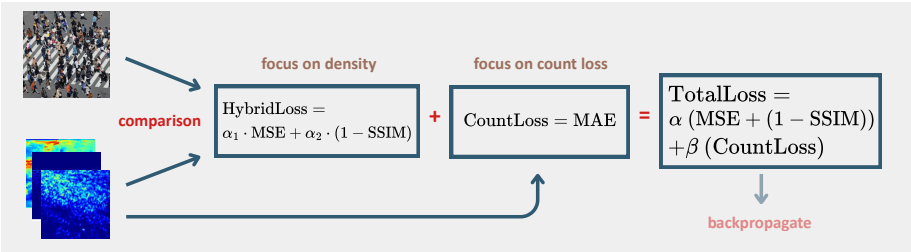
- **MSE** ensures **pixel-wise accuracy** : it pushes the model to predict the correct density at every location, regardless of size

- **SSIM** adds **local structural supervision** : it guides the model to capture contextual texture and patterns, like small blurry heads vs. sharper large ones

1. Cao, X., Wang, Z., Zhao, Y., & Su, F. (2018). Scale aggregation network for accurate and efficient crowd counting. Lecture Notes in Computer Science, 757–773. https://doi.org/10.1007/978-3-030-01228-1_45

Loss Function

CountLoss (MAE)



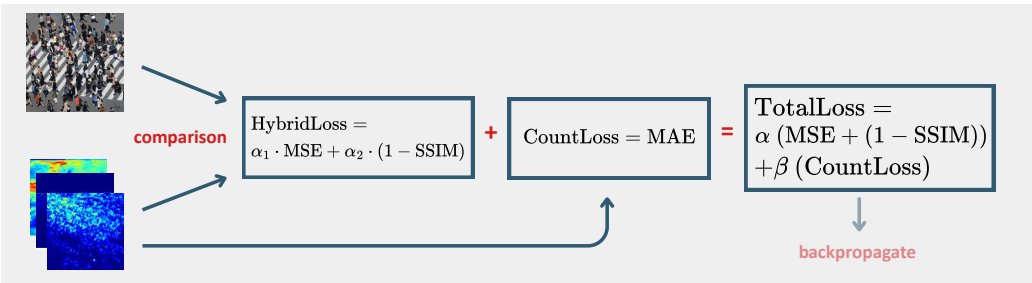
- **MSE + SSIM** focus on pixel-level accuracy
- But they **don't ensure the total count is correct**
- Count Loss compares **predicted value vs. ground truth value**

CountLoss = MAE(sum(pred), sum(gt))

$$MAE = \frac{1}{N}\sum_{i=1}^{N}\left|C_{\text{pred}}^{(i)} - C_{\text{gt}}^{(i)}\right|$$

- Count Loss (MAE) penalizes incorrect total predictions ：
guides the model to **match the actual number of people in the image**

Loss Function



Our project extends CSRNet with:

Hybrid Loss
MSE (L2) + SSIM
(improves pixel accuracy and map structure)

+

Count Loss
MAE
(help reduce under/overestimation)

$$\text{TotalLoss} = \alpha \cdot (\text{MSE} + (1 - \text{SSIM})) + \beta \cdot \text{MAE}$$

Loss Function

***Ablation table or experimentation if Total loss depends on MSE sahaja, Hybridloss sahaja, total loss on just one image !! to justify our loss design choice

```
 ==== Best Metrics Summary ====
Best MAE : 434.297
Best RMSE : 647.870
Best SSIM : 0.420
Best PSNR : 18.748
Plot saved as 1. Loss_history.png
Plot saved as 2. Count Accuracy.png
Plot saved as 3. Map Structure.png

==== Final Accuracy Summary ====
Final Train Accuracy: 11.85%
Final Validation Accuracy: 12.81%
Best Validation Accuracy: 15.80%
Plot saved as 4. Evaluation Metrics.png
```

EVALUATION METRICS

MAE, RMSE, SSIM, PSNR

Evaluation Metrics - Count

1. Mean Absolute Error (MAE)

$$MAE = \frac{1}{N}\sum_{i=1}^{N}\left|C_{\text{pred}}^{(i)} - C_{\text{gt}}^{(i)}\right|$$

- Measures **average error** in predicted crowd counts
- **Lower MAE → better count accuracy**
- Key metric in almost all crowd counting benchmarks

2. Root Mean Squared Error (RMSE)

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(C_{\text{pred}} - C_{\text{gt}})^2}$$

- Similar to MAE but gives **more weight to larger mistakes**
- Helps detect if the model occasionally makes **big errors**, especially in very **crowded images**

Evaluation Metrics - Density Map

3. Structural Similarity Index (SSIM)

$$\text{SSIM}(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

μ_x, μ_y : local means of image patches x and y
σ_x^2, σ_y^2 : local variances
σ_{xy} : local covariance between x and y
C_1, C_2 : small constants to stabilize the division

- Measures **visual similarity** between predicted and ground-truth density maps
- **Higher SSIM → more realistic crowd structure and spatial layout**

4. Peak Signal-to-Noise Ratio (PSNR)

$$\text{PSNR} = 10 \cdot \log_{10}\left(\frac{\text{MAX}^2}{\text{MSE}}\right)$$

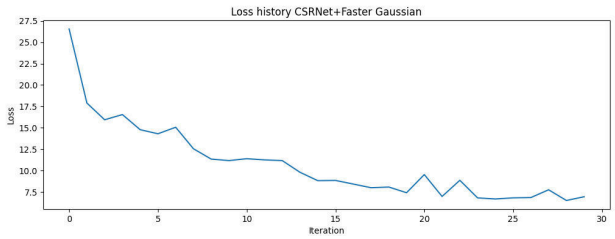
- Measures how much noise (difference) exists between predicted and ground truth map
- **Higher PSNR → better quality (less difference)**

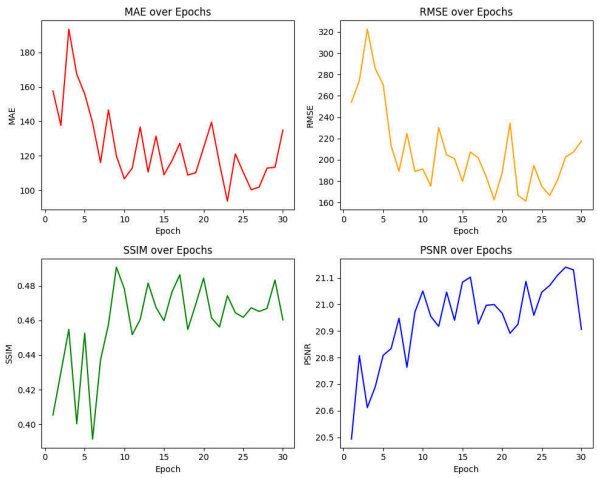
✔ **Together, these metrics help us understand not just how many people the model counts, but also how good the predicted map looks**

RESULT ANALYSIS

CSRNet + Geometry Adaptive Gaussian

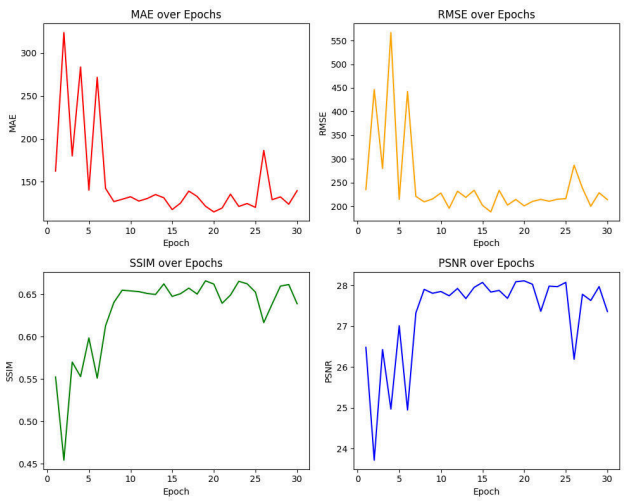
```
 ==== Best Metrics Summary ====
Best MAE : 93.657
Best RMSE : 161.241
Best SSIM : 0.391
Best PSNR : 20.493

Plot saved as 1. Loss_history.png
Plot saved as 2. Count Accuracy.png
Plot saved as 3. Map Structure.png

==== Final Accuracy Summary ====
Final Train Accuracy: 87.21%
Final Validation Accuracy: 69.40%
Best Validation Accuracy: 78.26%
Plot saved as 4. Evaluation Metrics.png
```

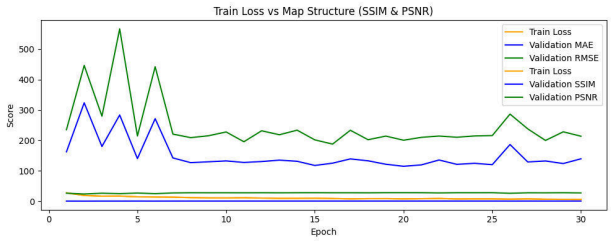




CSRNet + Global Gaussian

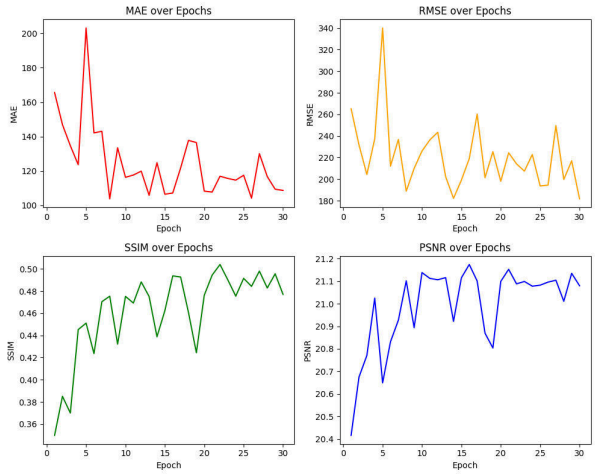
```
==== Best Metrics Summary ====
Best MAE : 114.992
Best RMSE : 187.923
Best SSIM : 0.454
Best PSNR : 23.716
Plot saved as 1. Loss_history.png
Plot saved as 2. Count Accuracy.png
Plot saved as 3. Map Structure.png

==== Final Accuracy Summary ====
Final Train Accuracy: 89.01%
Final Validation Accuracy: 56.05%
Best Validation Accuracy: 73.39%
Plot saved as 4. Evaluation Metrics.png
```

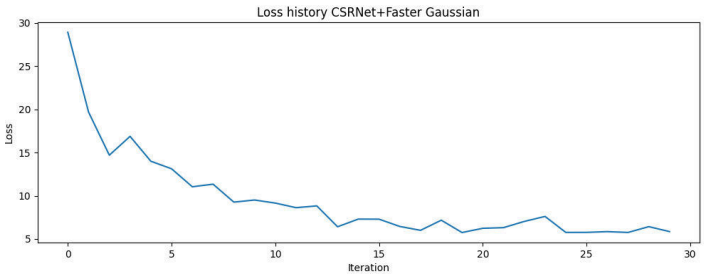




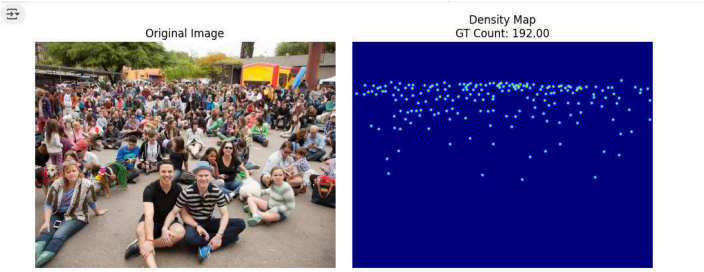
CSRNet + CBAM + Adaptive Gaussian

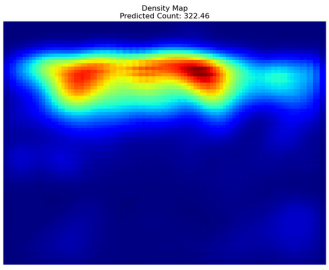
```
==== Best Metrics Summary ====
Best MAE : 103.755
Best RMSE : 181.668
Best SSIM : 0.350
Best PSNR : 20.416

Plot saved as 1. Loss_history.png
Plot saved as 2. Count Accuracy.png
Plot saved as 3. Map Structure.png

==== Final Accuracy Summary ====
Final Train Accuracy: 89.73%
Final Validation Accuracy: 75.09%
Best Validation Accuracy: 79.71%
Plot saved as 4. Evaluation Metrics.png
```





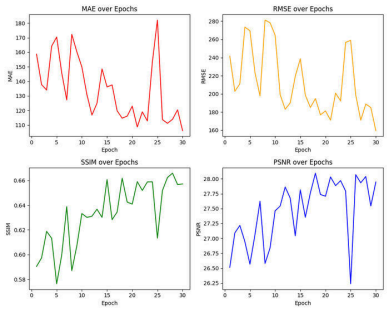
CSRNet + CBAM + Global Gaussian






```
==== Best Metrics Summary ====
Best MAE : 106.150
Best RMSE : 159.459
Best SSIM : 0.576
Best PSNR : 26.238
Plot saved as 1. Loss_history.png
Plot saved as 2. Count Accuracy.png
Plot saved as 3. Map Structure.png

==== Final Accuracy Summary ====
Final Train Accuracy: 90.75%
Final Validation Accuracy: 67.19%
Best Validation Accuracy: 71.36%
Plot saved as 4. Evaluation Metrics.png
```

- Model **correctly focuses** on d**ense regions at the back**, where scale is small
- Despite count error, the density map shows meaningful **spatial attention**
- Loss curves converge well, indicating the model is learning
- Results suggest **progress in structure learning**, but scale adaptation needs further refinement



Ablation Table

		Count Accuracy		Density Map Quality	
Model	**Gaussian Technique**	**MAE ↓**	**RMSE ↓**	**SSIM ↑**	**PSNR ↑**
CSRNet (baseline)	Geometry Adaptive Gaussian	93.657	161.241	0.391	20.493
CSRNet	Global Gaussian	114.92	187.923	0.454	23.716
CSRNet + CBAM	Geometry Adaptive Gaussian	103.755	181.668	0.35	20.416
CSRNet + CBAM	Global Gaussian	106.15	159.459	0.576	26.238

Conclusion

Overall, we believe our project, based on the ablation table for 30 epoch using csrnet +cbam + global gaussian if it is trained with more epoch it can be successfull. with that we conclude that using global gaussian, adding CBAM attention, and using our proposed loss function, can enhance density map and tackle scale variation.

REFERENCE

Cao, X., Wang, Z., Zhao, Y., & Su, F. (2018). Scale aggregation network for accurate and efficient crowd counting. Lecture Notes in Computer Science, 757–773. https://doi.org/10.1007/978-3-030-01228-1_45

Li, Y., Zhang, X., & Chen, D. (2018, April 11). CSRNet: Dilated Convolutional Neural Networks for understanding the highly congested scenes. arXiv.org. https://arxiv.org/abs/1802.10062

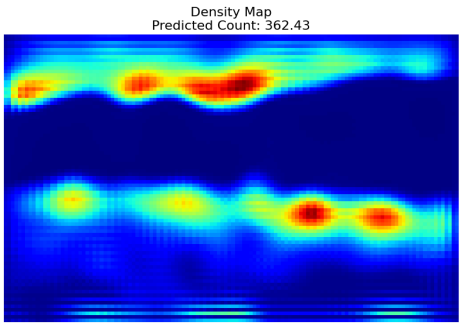
Khan, M. A., Menouar, H., & Hamila, R. (2023). Revisiting crowd counting: State-of-the-art, trends, and future perspectives. Image and Vision Computing, 129, 104597. https://doi.org/10.1016/j.imavis.2022.104597

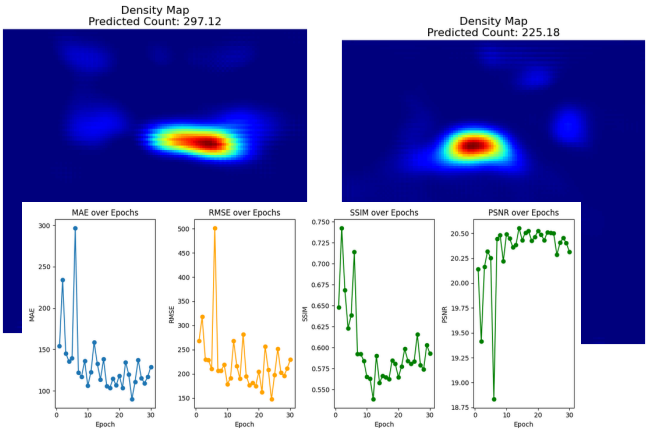
THANK YOU

OTHERS

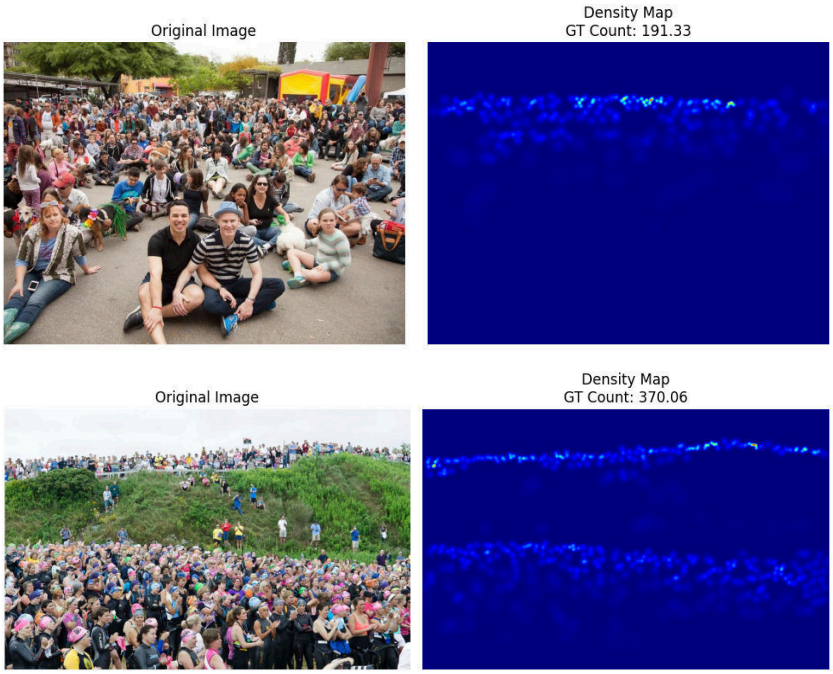
Rubbish

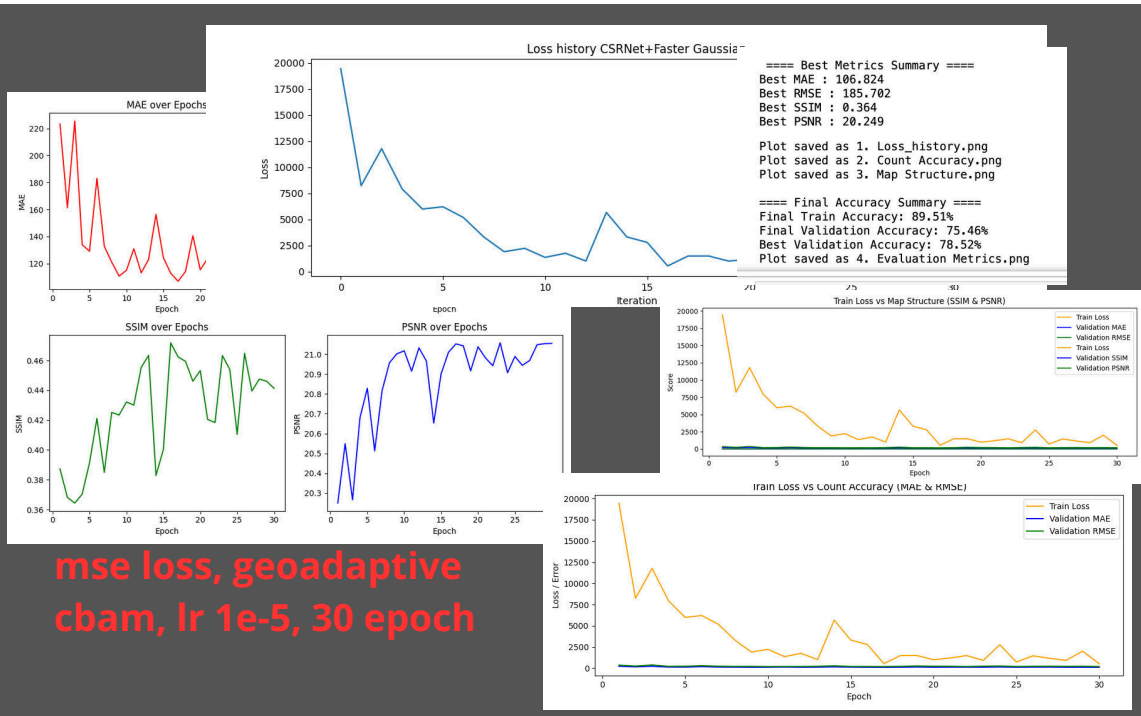
Geometry-Adaptive + CSRNet



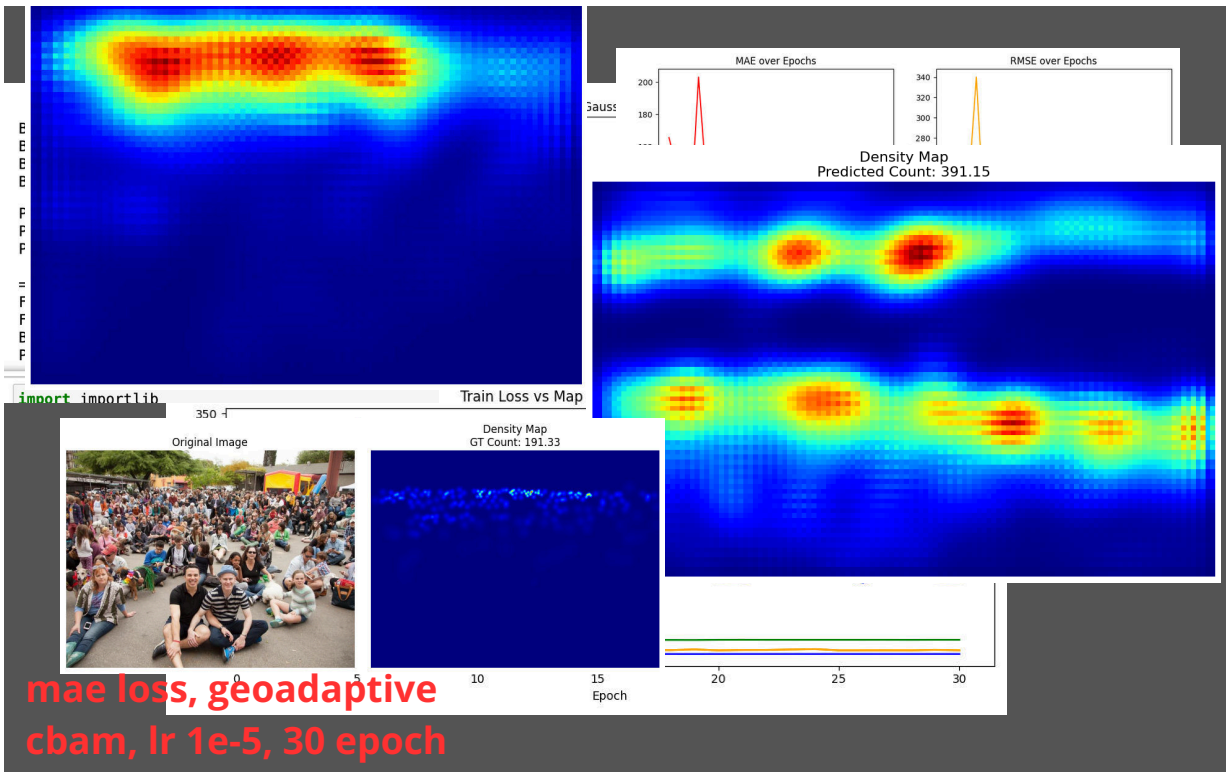


* MAE: 129.067, RMSE: 229.345, SSIM: 0.593, PSNR: 20.314
* best MAE 89.764

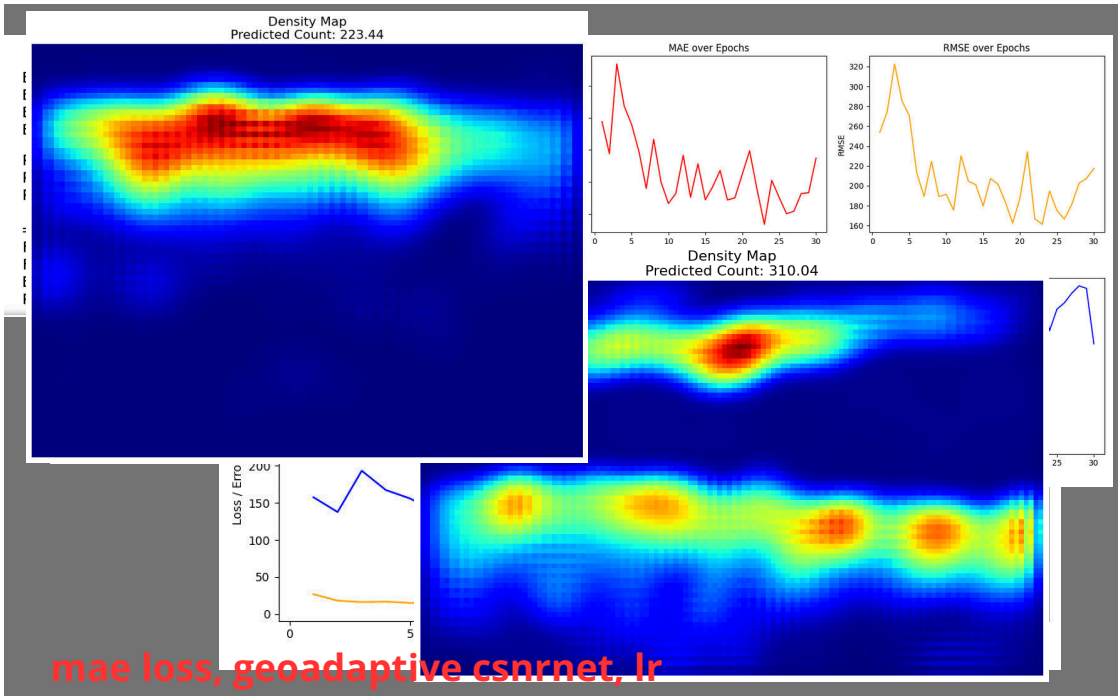




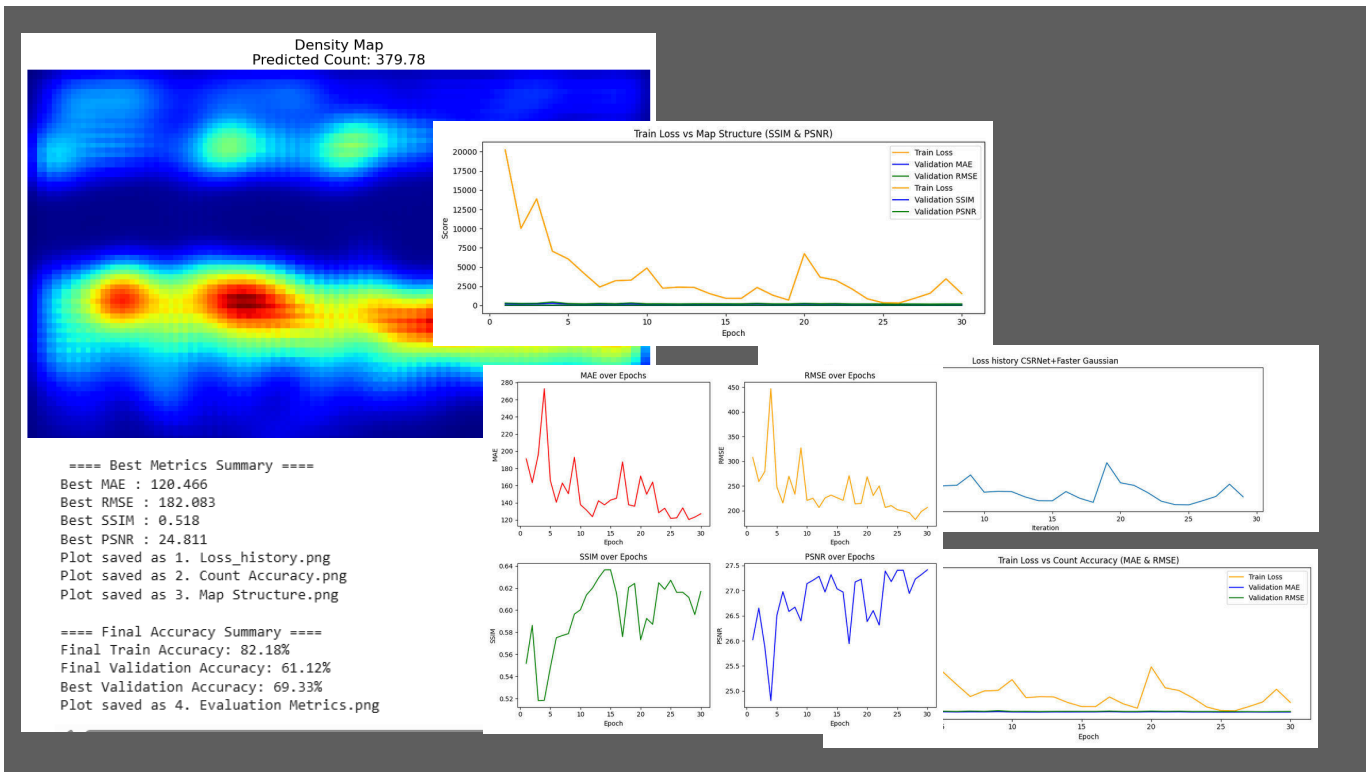
mse loss, geoadaptive cbam, lr 1e-5, 30 epoch

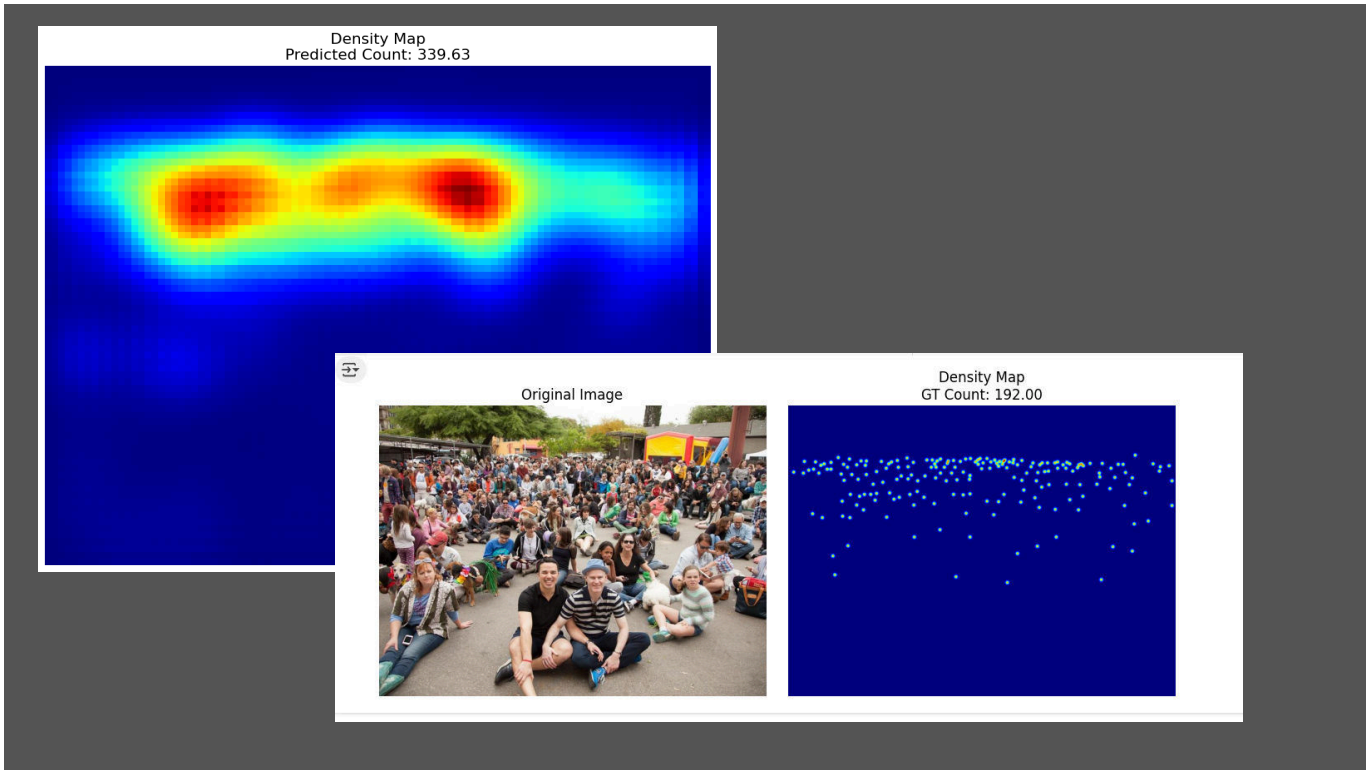


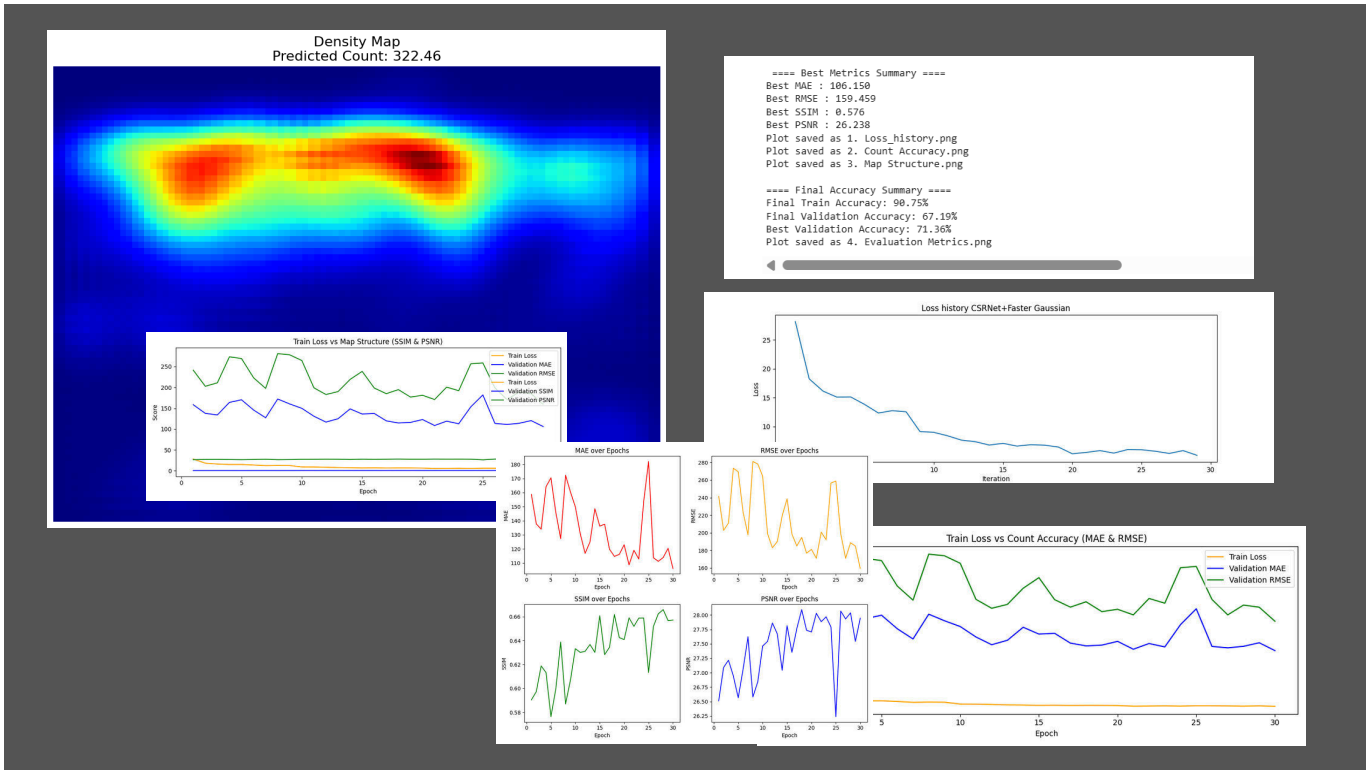
mae loss, geoadaptive cbam, lr 1e-5, 30 epoch

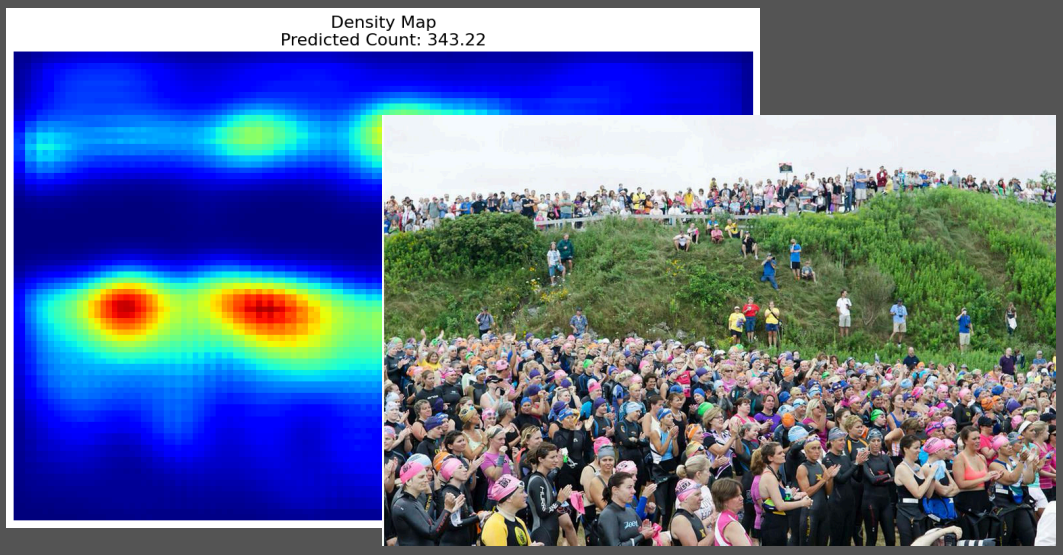


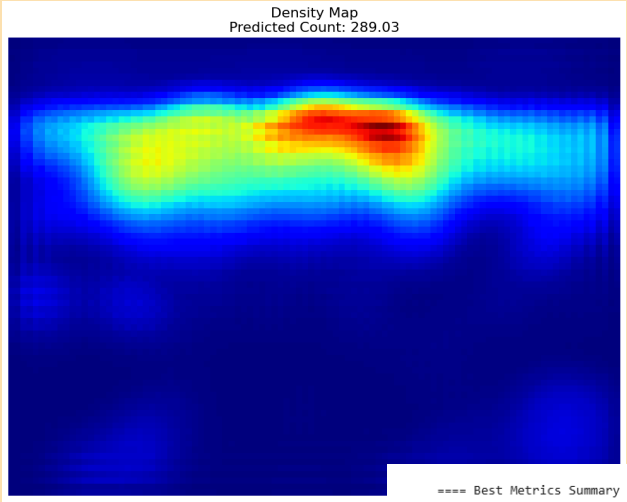
mae loss, geoadaptive csnrnet, lr 1e-5, 30 epoch




Density Map
Predicted Count: 339.63
Original Image
Density Map
GT Count: 192.00

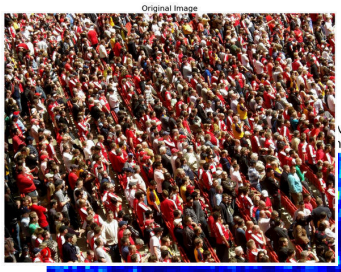

Density Map
Predicted Count: 322.46
==== Best Metrics Summary ====
Best MAE : 106.150
Best RMSE : 159.459
Best SSIM : 0.576
Best PSNR : 26.238
Plot saved as 1. Loss_history.png
Plot saved as 2. Count Accuracy.png
Plot saved as 3. Map Structure.png
==== Final Accuracy Summary ====
Final Train Accuracy: 90.75%
Final Validation Accuracy: 67.19%
Best Validation Accuracy: 71.36%
Plot saved as 4. Evaluation Metrics.png
Train Loss vs Map Structure (SSIM & PSNR)
Loss history CSRNet+Faster Gaussian
MAE over Epochs
RMSE over Epochs
SSIM over Epochs
PSNR over Epochs
Train Loss vs Count Accuracy (MAE & RMSE)
Train Loss
Validation MAE
Validation RMSE
Epoch

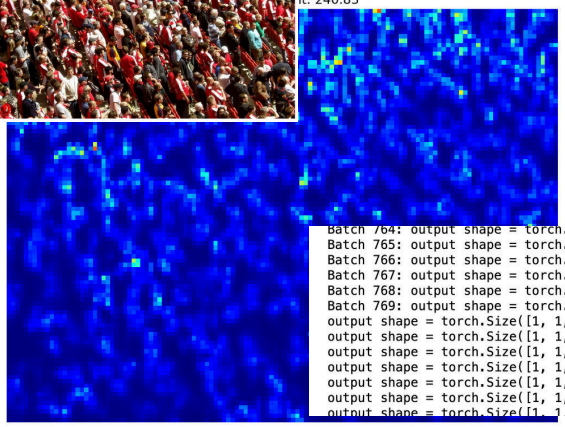

Density Map
Predicted Count: 343.22

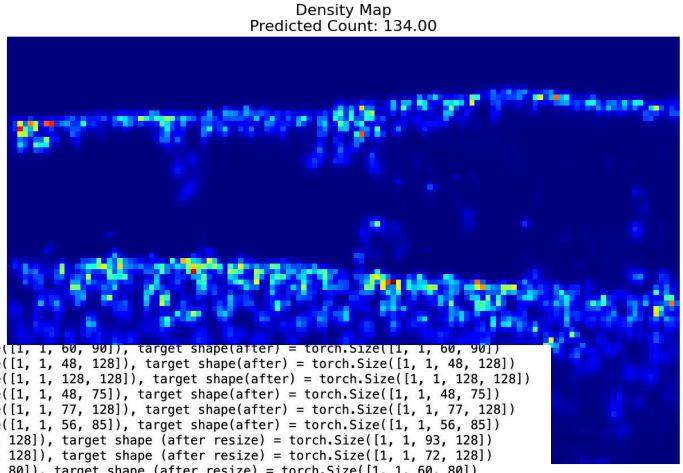



```
 ==== Best Metrics Summary ====
Best MAE : 114.992
Best RMSE : 187.923
Best SSIM : 0.454
Best PSNR : 23.716
Plot saved as 1. Loss_history.png
Plot saved as 2. Count Accuracy.png
Plot saved as 3. Map Structure.png

==== Final Accuracy Summary ====
Final Train Accuracy: 89.01%
Final Validation Accuracy: 56.05%
Best Validation Accuracy: 73.39%
Plot saved as 4. Evaluation Metrics.png
```

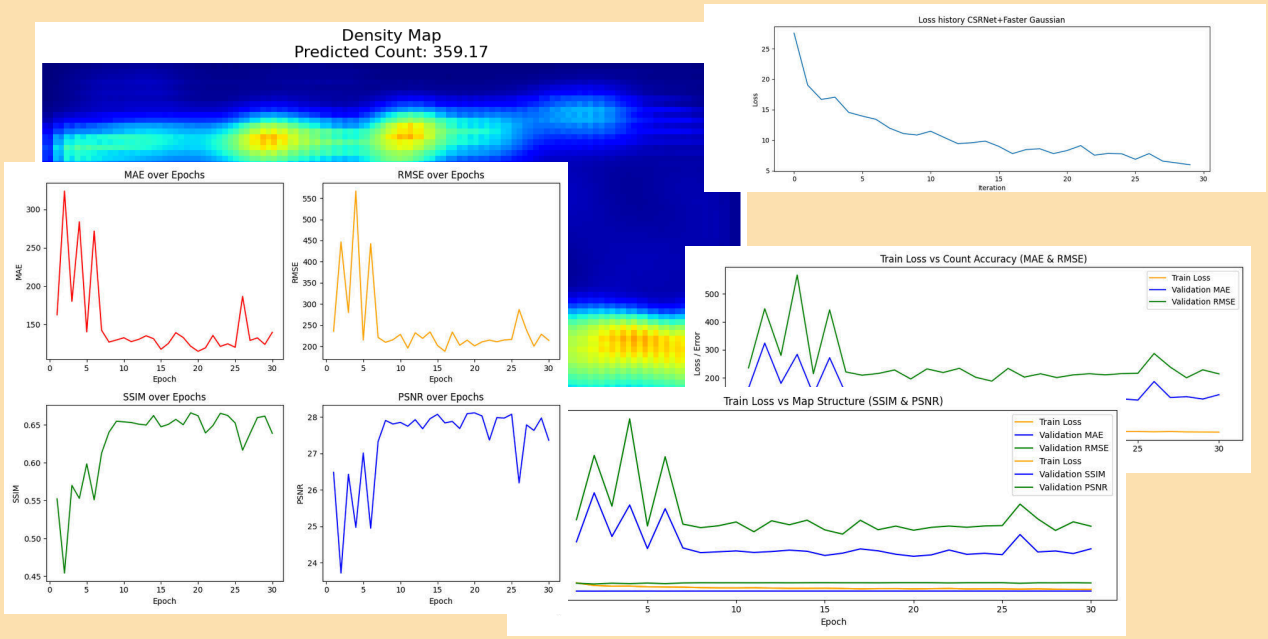
GEOADAPTIVE +CSRNet+CBAM








```
Batch 764: output shape = torch.Size([1, 1, 60, 90]), target shape(after) = torch.Size([1, 1, 60, 90])
Batch 765: output shape = torch.Size([1, 1, 48, 128]), target shape(after) = torch.Size([1, 1, 48, 128])
Batch 766: output shape = torch.Size([1, 1, 128, 128]), target shape(after) = torch.Size([1, 1, 128, 128])
Batch 767: output shape = torch.Size([1, 1, 48, 75]), target shape(after) = torch.Size([1, 1, 48, 75])
Batch 768: output shape = torch.Size([1, 1, 77, 128]), target shape(after) = torch.Size([1, 1, 77, 128])
Batch 769: output shape = torch.Size([1, 1, 56, 85]), target shape(after) = torch.Size([1, 1, 56, 85])
output shape = torch.Size([1, 1, 93, 128]), target shape (after resize) = torch.Size([1, 1, 93, 128])
output shape = torch.Size([1, 1, 72, 128]), target shape (after resize) = torch.Size([1, 1, 72, 128])
output shape = torch.Size([1, 1, 60, 80]), target shape (after resize) = torch.Size([1, 1, 60, 80])
output shape = torch.Size([1, 1, 85, 128]), target shape (after resize) = torch.Size([1, 1, 85, 128])
output shape = torch.Size([1, 1, 66, 100]), target shape (after resize) = torch.Size([1, 1, 66, 100])
output shape = torch.Size([1, 1, 38, 62]), target shape (after resize) = torch.Size([1, 1, 38, 62])
output shape = torch.Size([1, 1, 86, 128]), target shape (after resize) = torch.Size([1, 1, 86, 128])
```


Density Map
Predicted Count: 359.17
Loss history CSRNet+Faster Gaussian
Loss
Iteration
MAE over Epochs
MAE
Epoch
RMSE over Epochs
RMSE
Epoch
Train Loss vs Count Accuracy (MAE & RMSE)
Loss / Error
Train Loss
Validation MAE
Validation RMSE
SSIM over Epochs
SSIM
Epoch
PSNR over Epochs
PSNR
Epoch
Train Loss vs Map Structure (SSIM & PSNR)
Train Loss
Validation MAE
Validation RMSE
Train Loss
Validation SSIM
Validation PSNR
Epoch