Evaluating Supervision Levels Trade-Offs for Infrared-Based People Counting
Supplementary Material

1. Hyper-parameters

The training hyper-parameters utilized for training the people counting models using image-level architectures, point-wise localization, and object detectors are outlined in Tab. 1, Tab. 2, and Tab. 3, respectively. These hyper-parameters were applied consistently across all reported results in Section 4. In the case of image-level tasks, the loss functions employed were either Mean Square Error (MSE) for regression or Cross Entropy (CE) for classification. On the other hand, point-wise localization loss functions encompassed multiple terms within the loss function, such as Euclidean distance between points (EUC), Smooth L1 distance (SL1), or Split loss (L-S). For object detectors, YoloV8 utilized Varifocal loss (VFL) and Distribution Focal loss, while DINO employed L1 distance and Generalized Intersection over Union (GIOU).

Table 1. Image-level training hyper-parameters

	Image-Level models from Scratch		**Image-Level models from MAE Pretraining**		**Image-Level models from Fine-Tuning**	
	ConvNeXt-Micro/Tiny	ViT-3L/4L	ConvNeXt-Micro/Tiny	ViT-3L/4L	ConvNeXt-Micro/Tiny	ViT-3L/4L
Learning Rate	1.00e-4	1.00e-4	1.50e-6	1.50e-4	2.00e-4	1.00e-5
Epochs	450	450	500	500	450	450
Batch Size	64	64	64	64	64	64
Optimizer	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW
Momentum (beta1, beta2)	(0.9; 0.999)	(0.9; 0.95)	(0.9; 0.95)	(0.9; 0.95)	(0.9; 0.999)	(0.9; 0.999)
Weight Decay	0.3	0.3	0.05	0.05	0.05	0.3
Scheduler	Cosine	Cosine	Cosine	Cosine	Cosine	Cosine
Warmup Epochs	40	20	40	40	40	5
Loss Function	MSE/CE	MSE/CE	MSE	MSE	MSE/CE	MSE/CE

Table 2. Point-wise localization training hyper-parameters

	Point-Level Localizers	
	P2PNet	PET
Learning Rate	1.00e-4	1.00e-4
Epochs	1500	1500
Batch Size	8	8
Optimizer	Adam	AdamW
Momentum (beta1, beta2)	(0.9, 0.999)	(0.9, 0.999)
Weight Decay	N/A	N/A
Scheduler	None	None
Warmup Epochs	N/A	N/A
Loss Function	EUC + CE	SL1 + CE + L-S

Table 3. Object Detection training hyper-parameters

	Object Detectors	
	YoloV8-S/M/L	DINO-SWIN-Tiny
Learning Rate	1.00e-3	1.00e-4
Epochs	400	12
Batch Size	16	4
Optimizer	SGD	AdamW
Momentum [beta or (beta1, beta2)]	0.937	(0.9; 0.999)
Weight Decay	0.0005	N/A
Scheduler	Cosine	None
Warmup Epochs	3	N/A
Loss Function	VFL + DFL	L1 + GIOU

Fig. 1 displays the count accuracy versus threshold curves for object detectors. These curves were generated after the completion of training, and the validation set was utilized to identify the optimal score threshold for evaluation purposes. The determined best thresholds for the LLVIP and Distech IR datasets are outlined in Tab. 4.

2. Effect of class imbalance on image-level counting

Tab. 5 and Tab. 6 illustrate the achieved count accuracy per number of people in each image for LLVIP and Distech IR, respectively. The tables also present the class distribution, aiding in the analysis of class imbalance within this framework. As expected, higher occurrences of a category correlate with higher count accuracy. Classes represented by only one image

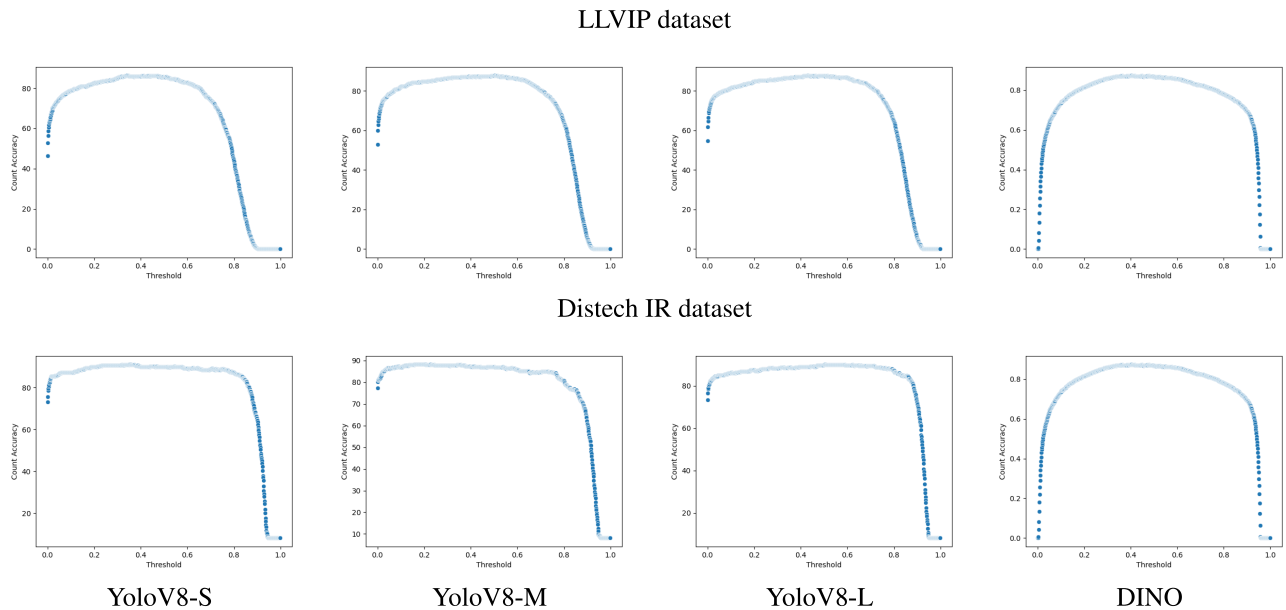

Figure 1. Count accuracy at various thresholds for the LLVIP and Distech IR datasets using YoloV8-S, YoloV8-M, YoloV8-L, and DINO models. The selected best threshold represents the value that yields the highest count accuracy.

Table 4. Best thresholds per model per dataset

Dataset	Yolov8-S	Yolov8-M	Yolov8-L	DINO
LLVIP	0.343	0.503	0.43	0.405
Distech IR	0.351	0.159	0.504	0.405

displayed a binary performance outcome, typically 0% or 100%. This trend aligns with findings in existing literature on image-level tasks characterized by severe class imbalance. Notably, regression-based methods were equally impacted by this class imbalance. These results are a notable drawback of such people counting techniques. We hypothesize that employing stratified versions of datasets or techniques designed to mitigate the effects of class imbalance might enhance performance and potentially match the performance of object detectors.

Table 5. Count Accuracy Per Class LLVIP

		Accuracy Count :	0	1	2	3	4	5	6	7	8	9	10	11	12	13
		Occurences	1	1203	990	602	333	171	81	45	24	9	3	2	1	1
Model	**Pretrain.**	**Head Type**														
ConvNeXt	None	Classification	100.00	91.81	83.52	74.61	73.32	65.99	48.78	57.50	29.41	7.14	25.00	0.00	100.00	0.00
		Regression	100.00	92.47	85.86	74.61	68.27	69.23	53.66	66.25	44.12	21.43	37.50	0.00	100.00	100.00
	MAE	Classification	100.00	92.79	86.16	73.05	70.91	67.61	57.72	60.00	35.29	7.14	0.00	0.00	100.00	100.00
		Regression	100.00	93.56	85.15	75.70	71.15	69.23	52.03	56.25	38.24	28.57	25.00	0.00	100.00	100.00
ViT	None	Classification	0.00	80.79	65.41	54.98	56.01	42.51	38.21	36.25	26.47	0.00	25.00	0.00	100.00	0.00
		Regression	0.00	78.82	66.73	58.26	55.53	48.99	34.96	35.00	20.59	0.00	0.00	0.00	0.00	0.00
	MAE	Classification	0.00	86.79	64.90	55.30	57.21	36.84	26.83	50.00	2.94	0.00	0.00	0.00	0.00	0.00
		Regression	100.00	81.66	63.58	61.99	56.25	45.34	39.84	42.50	29.41	14.29	12.50	0.00	0.00	100.00

3. Localization details

The localization results on the main manuscript are reported in terms of mean Absolute Euclidean Distance (mAED). We calculate the mAED between the predicted coordinates and the ground truth for all images in the testing set. Both the x and y position coordinates are normalized within the range of 0 to 1.

Table 6. Count Accuracy Per Class Distech IR

		Accuracy Count :	0	2	3	4	5	6
		Occurences	19	50	36	5	144	1
Model	**Pretraining**	**Head Type**						
ConvNeXt	None	Classification	89.47	86.00	44.44	20.00	95.83	0.00
		Regression	94.74	62.00	83.33	20.00	93.75	0.00
	MAE	Classification	84.21	82.00	75.00	0.00	97.22	100.00
		Regression	89.47	78.00	77.78	20.00	93.75	0.00
ViT	None	Classification	94.74	68.00	30.56	40.00	96.53	0.00
		Regression	94.74	56.00	52.78	0.00	86.81	0.00
	MAE	Classification	89.47	76.00	33.33	0.00	91.67	100.00
		Regression	78.95	42.00	33.33	20.00	84.72	0.00

The mAED is computed as:

$$mAED = \frac{1}{M}\sum_{j=1}^{M}\frac{1}{N_j}\sum_{i=1}^{N_j}||p_{ij} - \hat{p}_{ij}||_2^2 \tag{1}$$

where $\hat{p}_{ij}$ and p_{ij} represent the coordinates of the predicted and ground truth positions for the i^{th} point within the j^{th} image. As previously outlined, the predicted points and ground truth are paired using the Hungarian matching algorithm. In this context, M signifies the total number of images within the test set, while N_j indicates the total number of points per image. When the estimated points and ground truth locations fail to align, a penalty distance of 1 is assigned.

Since image-level techniques lack localization information, we use class activation maps to obtain the locations of individuals. Our proposed algorithm leverages the identified count of people ($numInstances$) to guide te localization process. Empirically, a threshold of 27 was determined to effectively binarize the activation map and extract the relevant regions of interest. For specifics regarding the employed algorithm, refer to Algorithm 1. Several examples illustrating the localization obtained using the aforementioned algorithm from the activation maps are presented in Fig. 2.

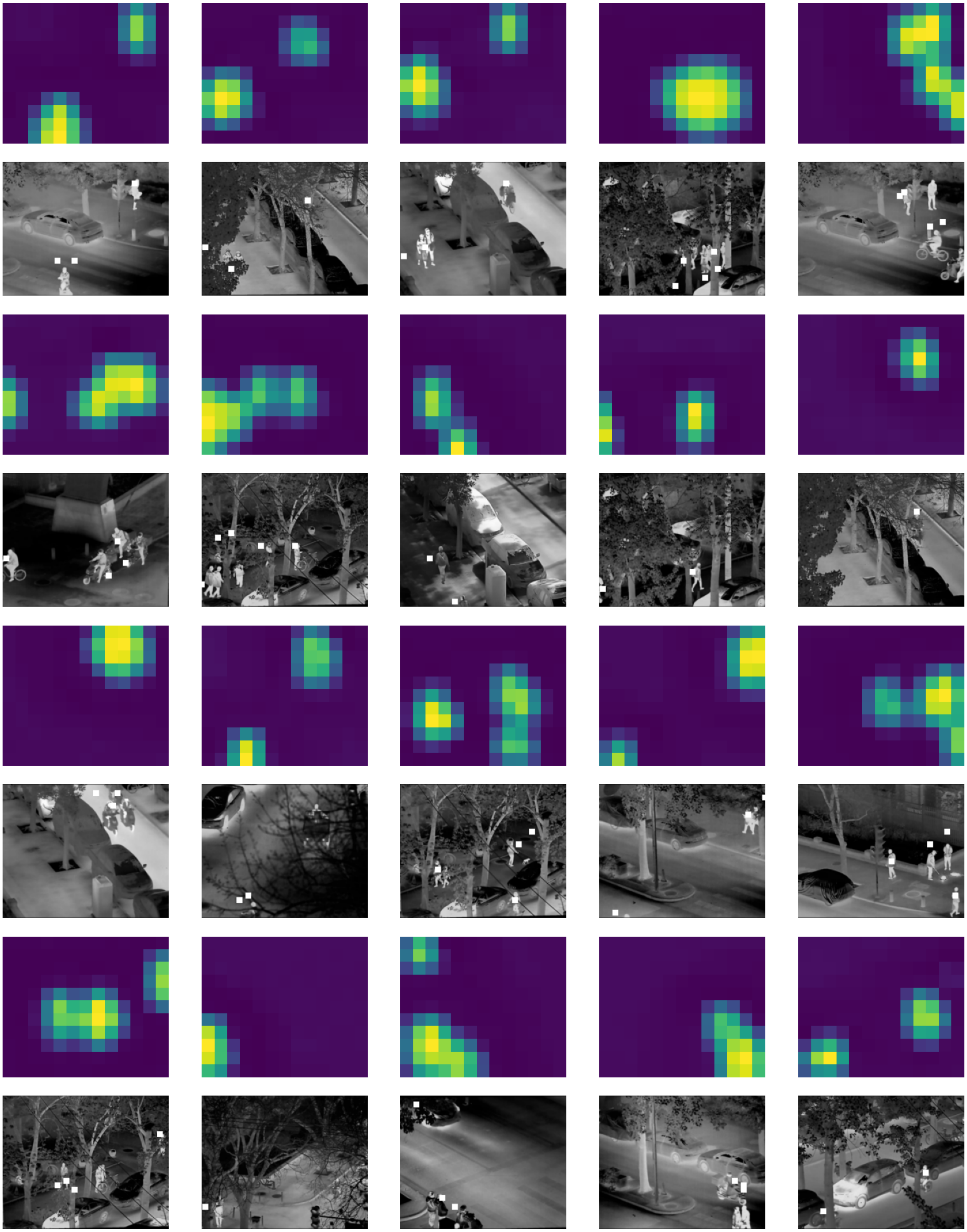

Figure 2. Examples of people localization using ConvNeXt attention maps.

Algorithm 1 People's location from ConvNeXt activation maps

```
procedure LocatePeople(activationMap, binaryThreshold, numInstances)
    activationMap ← binarize(activationMap, binaryThreshold)
    countours ← findCountours(activationMap).
    x, y ← findCoordinates(countours).
    if numInstances = len(countours) then
        return x, y
    else if numInstances < len(countours) then
        countours ← reverseSortByAreaSize(countours)
        x, y ← findCoordinates(countours).
        x, y ← sliceList(countours, start = 0, end = numInstances).
        return x, y
    else
        x, y ← List(), List()
        avgInstanceSize ← totalArea(countours)/numInstances
        for contour in countours do
            numPeoples ← round(area(contour)/avgInstanceSize)
            if numPeoples ≤ 1 then
                px, py ← findCoordinates(countour)
                x ← x.append(px)
                y ← y.append(py)
            else
                randomCoordinates ← uniformRandomSample(contour, numSamples = numPeoples)
                for px, py in randomCoordinates do
                    x ← x.append(px)
                    y ← y.append(py)
        return x, y
```