Monte Carlo Stochastic Depth for Uncertainty Estimation in Deep Learning

Supplementary Material

8. Appendix A: Reproducibility

Complete source code reproducing all methods (MCD, MCDB, MCSD) and models (Faster R-CNN, YOLOv8x, RT-DETRx) is provided in the GitHub[1] repository.

Hardware and Software. Experiments were conducted using Python 3.10.12, PyTorch 2.6.0, and Ultralytics 8.3.171. The codebase is hardware-agnostic; however, all reported results were generated on a single NVIDIA GeForce RTX 3090 GPU. Run-times may vary across different hardware configurations. Installation guides and Quick Start instructions are included in the repository, with dependencies listed in `requirements.txt`.

8.1. Framework Adaptations

Network Architectures. Integrating stochastic regularization into the standard architectures of Faster R-CNN, YOLOv8x, and RT-DETRx (as visualized in Fig. 1) necessitated specific modifications to the underlying framework definitions.

- **Faster R-CNN:** We extended the `Bottleneck` class within Torchvision's `resnet.py`. To incorporate DropBlock and Stochastic Depth, we leveraged existing implementations from the `timm` [52] library. This custom `Bottleneck` was then integrated into the backbone instantiation via the `fasterrcnn_resnet50_fpn` function in `faster_rcnn.py`.
- **YOLOv8x / RT-DETRx:** For the Ultralytics-based networks, we modified `nn/modules/block.py` to support the proposed stochastic methods. We integrated the MCDB implementation by Yelleni *et al.* [54] and the MCSD logic defined in Algorithm 1. These stochastic mechanisms were injected directly into the `Bottleneck` class for YOLOv8x and the `HGBlock` class for RT-DETRx. For MCD, we utilized the framework's native Dropout modules.

Inference Outputs. To compute the entropy-based uncertainty scores described in Sec. 5.1, access to the full class probability vectors, which are typically discarded during post-processing, was required. We adapted the inference pipelines of both frameworks as follows:

- **Ultralytics:** We modified the inference flow to retain the full class probability vector prior to thresholding. This involved updating the `postprocess` method within the `DetectionPredictor` class (found in the model-specific `predict.py` files), the `BasePredictor` class in `engine/predictor.py`, and the `non_max_suppression` logic in `utils/ops.py`.
- **Torchvision:** We modified the `postprocess_detections` function within the `RoIHeads` class (located in `detection/roi_heads.py`) to preserve and return the full probability tensors alongside the standard detection outputs.

8.2. Uncertainty and Calibration Metrics

We evaluated model calibration and predictive uncertainty using four primary metrics. Aligning with Sec. 5.1 the formulations are detailed below.

Predictive Entropy. Entropy measures the uncertainty inherent in a probability distribution. As some detectors (YOLOv8x, RT-DETRx) produce independent sigmoid probabilities $\mathbf{p}_i = (p_{i,1}, \ldots, p_{i,C})$ for each class, where probabilities do not sum to 1, we distinguish between two formulations:

1. **Mean Binary Entropy:** We compute the entropy for each class c as an independent Bernoulli trial and then average these entropies. The binary entropy for a single class probability $p_{i,c}$ is:

$$H(p_{i,c}) = -p_{i,c}\log_2(p_{i,c}) - (1-p_{i,c})\log_2(1-p_{i,c})$$

   The final metric for detection i, is the mean of these binary entropies:

$$H_{\text{binary}}(\mathbf{p}_i) = \frac{1}{C}\sum_{c=1}^{C} H(p_{i,c})$$

2. **Shannon Entropy:** The standard Shannon entropy is used for models that output a single multinomial probability distribution where $\sum_{c=1}^{C} p_{i,c} = 1$ (i.e., a softmax output):

$$H_{\text{shannon}}(\mathbf{p}_i) = -\sum_{c=1}^{C} p_{i,c}\log_2(p_{i,c})$$

Predictive Uncertainty. We assessed the quality and practical utility of the predictive uncertainties. This was evaluated using metrics that measure calibration (Brier Score, ECE) and the effectiveness of the uncertainty score in identifying potential errors (AUARC).

- **Brier Score (BS):** The Brier Score measures the accuracy of probabilistic predictions by computing the mean squared error (MSE) between the predicted probability

[1] https://github.com/code-supplement-2026/mc-val

vector and the one-hot encoded true label vector. Our implementation computes the average MSE over all N detections and all C classes.

$$\text{BS} = \frac{1}{N \cdot C} \sum_{i=1}^{N} \sum_{c=1}^{C} (p_{i,c} - y_{i,c})^2$$

where N is the total number of detections (True Positives in this context), C is the number of classes, $p_{i,c}$ is the predicted probability for detection i of class c, and $y_{i,c}$ is the one-hot true label (1 if c is the true class for detection i, 0 otherwise).

- **Expected Calibration Error (ECE):** ECE measures calibration by partitioning all N detections (True Positives and False Positives) into M equally-spaced confidence bins. It then computes a weighted average of the absolute difference between the mean accuracy and mean confidence in each bin.

$$\text{ECE} = \sum_{m=1}^{M} \frac{|B_m|}{N} \left|\text{acc}(B_m) - \text{conf}(B_m)\right|$$

where B_m is the set of detections whose confidence falls into bin m, $\text{acc}(B_m)$ is the accuracy (fraction of True Positives) of the detections in B_m, and $\text{conf}(B_m)$ is the average confidence of detections in B_m.

- **Area Under the Accuracy-Rejection Curve (AUARC):** AUARC evaluates the trade-off between predictive accuracy and the fraction of predictions rejected based on an uncertainty score. Following [56], detections are sorted by their uncertainty score (e.g., entropy) in descending order.

$$\text{AUARC} = \int_0^1 \text{Acc}(r) dr$$

where r is the rejection fraction (from 0 to 1). $\text{Acc}(r)$ is the precision (i.e., $\frac{TP}{TP+FP}$) of the *retained* (non-rejected) predictions after rejecting the top r fraction of most uncertain detections. A higher AUARC value indicates that the uncertainty measure is a good proxy for error, allowing for the effective rejection of false positives.

9. Appendix B: Theoretical Derivations

In this section, we provide the formal derivation connecting the Stochastic Depth (SD) training objective to the Variational Inference (VI) framework utilized in the main paper.

9.1. Derivation of the ELBO Objective for MCSD

As defined in the main paper, our objective is to maximize the Evidence Lower Bound (ELBO):

$$\mathcal{L}_{\text{VI}}(\theta) = \underbrace{\mathbb{E}_{q_\theta(W)}[\log p(\mathcal{D}|W)]}_{\text{Expected Log-Likelihood}} - \underbrace{\text{KL}(q_\theta(W)||p(W))}_{\text{Complexity Penalty}}$$

Here, we demonstrate that the standard SD training procedure constitutes a stochastic gradient optimization of this objective.

9.1.1. The Expected Log-Likelihood Term

Let the training dataset be $\mathcal{D} = \{(x_n, y_n)\}_{n=1}^N$. In the MCSD framework, the variational distribution $q_\theta(W)$ represents the stochastic process of selecting a sub-network $W^{(B)}$ conditioned on a Bernoulli vector $B \in \{0,1\}^L$, as defined in Eq. (12) of the main text.

The expected log-likelihood term of $\mathcal{L}_{\text{VI}}(\theta)$ can be expanded as:

$$\mathbb{E}_{q_\theta(W)}[\log p(\mathcal{D}|W)] = \mathbb{E}_{B\sim p(B)} \left[\sum_{n=1}^{N} \log p(y_n|x_n, W^{(B)}) \right]$$

Exact computation of this expectation requires summation over 2^L possible sub-networks, which is intractable. We approximate this via Monte Carlo sampling. For a mini-batch of size M, we sample a mask $B_m \sim p(B)$ and compute the estimator:

$$\mathbb{E}_{B\sim p(B)} \left[\sum_{n=1}^{N} \log p(..) \right] \approx \frac{N}{M} \sum_{m=1}^{M} \log p(y_m|x_m, W^{(B_m)})$$

For standard discriminative tasks, the model's log-likelihood corresponds to the negative of the task loss function (e.g., Cross-Entropy or Focal Loss), such that $\log p(y|x, W) \propto -\mathcal{L}_{\text{task}}(y, f(x; W))$.

Consequently, maximizing the expected log-likelihood is equivalent to minimizing the expected task loss:

$$\begin{aligned} &\arg\max_\theta \mathbb{E}_{q_\theta(W)}[\log p(\mathcal{D}|W)] \\ &\iff \arg\min_\theta \mathbb{E}_{B\sim p(B)}[\mathcal{L}_{\text{task}}(y, f(x \mid W^{(B)}))] \end{aligned}$$

The standard SD training algorithm, which performs a forward pass with a sampled path B and minimizes $\mathcal{L}_{\text{task}}$, is therefore a direct stochastic gradient ascent optimization of the first term of the ELBO.

9.2. The KL Divergence as L2 Regularization

Having established the optimization of the likelihood term, we address the complexity penalty, $\text{KL}(q_\theta(W)||p(W))$.

In our framework, $q_\theta(W)$ is a discrete mixture distribution over sub-networks, while the prior $p(W)$ is assumed to be a standard Gaussian $\mathcal{N}(0, I)$. Direct computation of the KL divergence between a discrete mixture and a continuous prior is ill-posed.

Following the established methodology for approximate Bayesian inference in deep learning [11], we approximate the complexity penalty via L_2 regularization (weight decay). We assume a prior length-scale, such that the minimization of the KL divergence corresponds to minimizing

the L_2 norm of the variational parameters (the weights W_l of the residual blocks).

The resulting surrogate training objective becomes:

$$\mathcal{L}_{\text{final}} = \frac{1}{M}\sum_{m=1}^{M}\mathcal{L}_{\text{task}}(y_m, f(x_m \mid W^{(B_m)})) + \lambda\sum_{l=1}^{L}||W_l||_2^2$$

This confirms that training with Stochastic Depth and weight decay optimizes a valid proxy for the full ELBO, justifying MCSD as a theoretically grounded Bayesian approximation.

9.3. Justification for Probabilistic Scaling

Algorithm 1 details the implementation of the MCSD residual block. A critical component is the scaling of the residual features A_{res} by the inverse survival probability $1/p_l$ during the forward pass.

This scaling ensures that the stochastic gradients computed during training (and the predictions during MCSD inference) act as unbiased estimators of the full network. Let x_{l+1} be the output of block l. The expected output over the distribution of masks b_l is:

$$\begin{aligned}\mathbb{E}_{b_l}[x_{l+1}] &= \mathbb{E}_{b_l}\left[x_l + \frac{b_l}{p_l}\cdot\mathcal{F}_l(x_l; W_l)\right] \\ &= x_l + \frac{\mathbb{E}[b_l]}{p_l}\cdot\mathcal{F}_l(x_l; W_l) \\ &= x_l + \mathcal{F}_l(x_l; W_l)\end{aligned}$$

Since $\mathbb{E}[b_l] = p_l$, the term cancels out, ensuring the expected output of the stochastic pass equals the output of the deterministic, fully active block. This preserves the feature magnitude distribution between training and Monte Carlo inference, preventing distributional shift when sampling T predictions for uncertainty quantification.

10. Appendix C: Additional Results

10.1. Extended Pareto Analysis

Complementing Fig. 2 in the main text, we provide the complete Pareto trade-off plots for the Faster R-CNN (Fig. 4) and YOLOv8x (Fig. 5) architectures.

Consistent with the RT-DETR results discussed in the main paper, we observe that the **confidence threshold** acts as the dominant variable governing the trade-off between predictive performance (mAP) and uncertainty ranking (AUARC). Higher thresholds generally improve uncertainty ranking at the cost of precision, while lower thresholds maximize mAP but degrade calibration and ranking capabilities.

Sensitivity to Regularization Magnitude. In the case of Faster R-CNN (Fig. 4), we observe a distinct drop in predictive performance based on the regularization magnitude

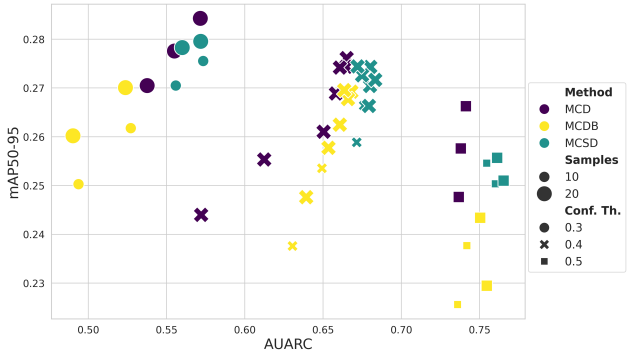

Figure 4. Pareto front analysis of the accuracy (mAP) vs. uncertainty ranking (AUARC) trade-off for Faster R-CNN on the COCO validation set. Each point represents a unique hyperparameter configuration. Non-competitive configurations are omitted.

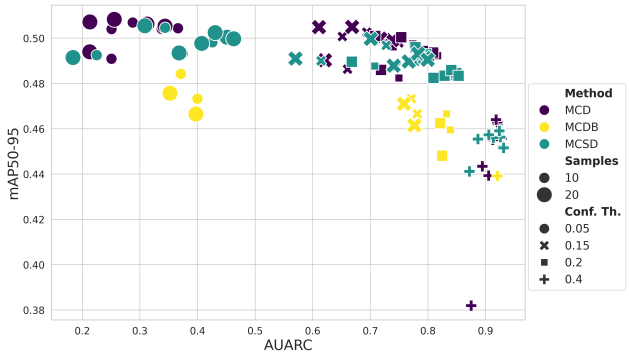

Figure 5. Pareto front analysis of the accuracy (mAP) vs. uncertainty ranking (AUARC) trade-off for YOLOv8x on the COCO validation set. Each point represents a unique hyperparameter configuration. Non-competitive configurations are omitted.

(drop rate). We evaluated a wider range of regularization intensities to capture this behavior. The analysis reveals that lower drop rates (≈ 0.1) yield the highest mAP, whereas increasing the rate (≈ 0.3) significantly degrades predictive performance without a commensurate gain in uncertainty quality.

Selection of Pareto-Optimal Configurations To facilitate the tabular comparison in Tab. 1 of the main paper, we required a principled criterion to select a single representative configuration from the Pareto frontiers shown in Fig. 4, Fig. 5, and Fig. 2.

We define the Ideal Performance Point (IPP) as the coordinate combining perfect accuracy and perfect uncertainty ranking, assuming normalized metrics where mAP $\in [0, 1]$ (higher is better) and AUARC $\in [0, 1]$ (higher is better).

For each method (MCD, MCDB, MCSD), we selected the configuration that minimizes the Euclidean distance d

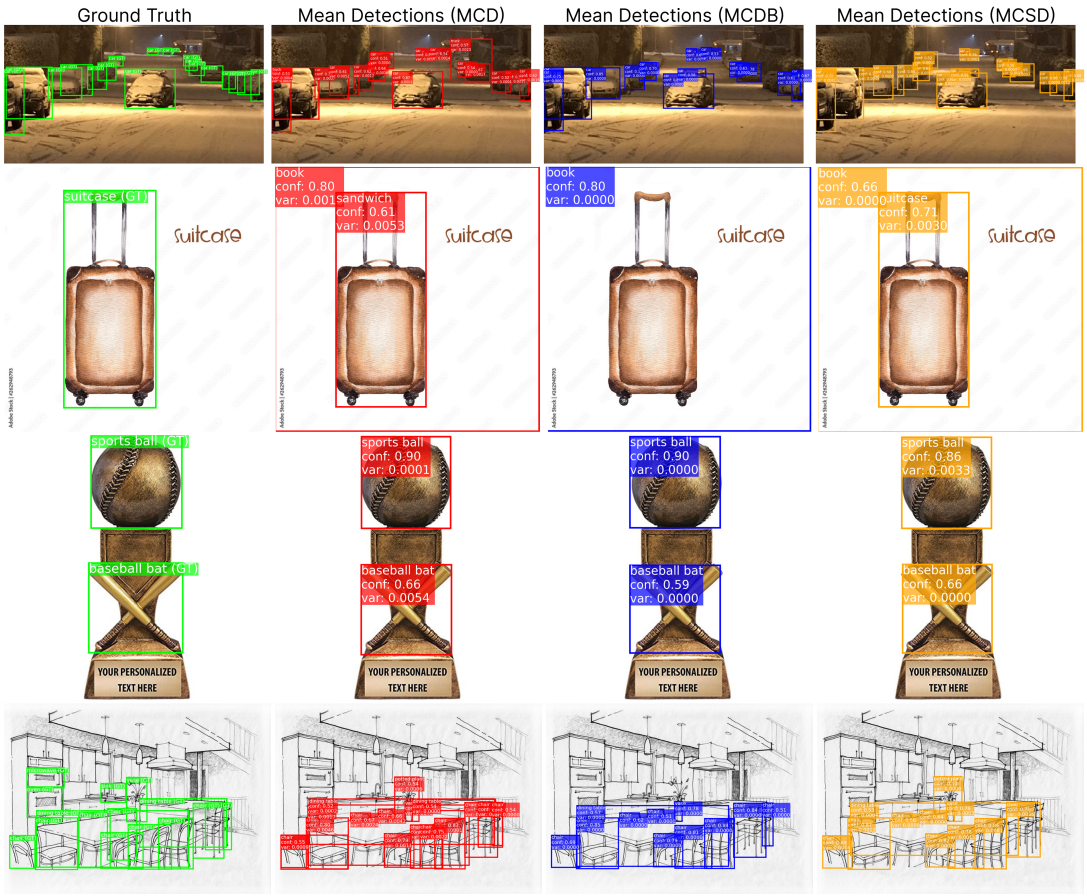

Figure 6. Qualitative comparison of detection and uncertainty for data under distribution shift, using RT-DETRx. Examples from the COCO-O dataset, comparing detection outputs across different uncertainty quantification methods. Each row represents a distinct domain shift: Row 1 (*Weather*), Row 2 (*Painting*), Row 3 (*Handmake*), and Row 4 (*Sketch*). Columns from left to right show: Ground Truth (green bounding boxes), Mean Detections (MCD) (red), Mean Detections (MCDB) (blue), and Mean Detections (MCSD) (orange).

to this IPP:

$$d(c) = \sqrt{(x_{IPP} - \text{AUARC}_c)^2 + (y_{IPP} - \text{mAP}_c)^2}$$

where c represents a specific hyperparameter configuration. This approach balances the two competing objectives, ensuring the selected configuration represents the best overall compromise between predictive power and safety-critical reliability.

10.2. Qualitative Evaluation

To complement the quantitative aggregate metrics presented in the main paper, we provide illustrative qualitative examples in Fig. 6. These samples are drawn from the COCO-O dataset, representing distribution-shifted scenarios where reliable uncertainty quantification is most critical.

Comparison with Prior Work. Previous work by Yelleni *et al*. [54] posited, based on a limited set of examples, that MCDB is superior to MCD because it eliminates specific misclassifications (e.g., misclassifying a loaf of bread as a sheep). While our visual analysis corroborates that MCDB produces fewer false positive detections (hallucinations or misclassifications) than MCD, we observe that this behavior stems from a significantly more conservative detection threshold.

The Precision-Recall Trade-off. As illustrated in Fig. 6, this conservatism often extends to false negatives. In the first row of the figure (Car scenario), while MCD and MCSD successfully identify the majority of the vehicles despite the domain shift, MCDB suppresses some predictions entirely. This suggests that the ”safety” observed in [54] comes at the cost of reduced recall. In contrast, MCSD maintains a detection density comparable to MCD, thus capturing the true positives while offering the calibration benefits detailed in our quantitative tables. This visual evidence aligns with the quantitative findings where MCDB generally yielded lower mAP scores compared to MCD and MCSD across the tested architectures.

Table 2. Impact of stochastic layer placement. Comparison of stochasticity applied to earlier vs. later stages in single and multiple (*Half Block*) layers. *N/A* denotes configurations that failed to meet minimum viability thresholds.

			Adapted Layers			
			Single Layer		Half Block	
Model	**Method**	**Metric**	**First**	**Last**	**First**	**Last**
Faster R-CNN	MCD	**ECE** ↓	**0.236**	0.254	**0.223**	0.234
		AUARC ↑	**0.680**	0.672	**0.683**	0.675
	MCDB	**ECE** ↓	**0.260**	0.263	0.257	**0.224**
		AUARC ↑	0.572	**0.668**	0.612	**0.661**
	MCSD	**ECE** ↓	**0.231**	0.244	0.226	**0.218**
		AUARC ↑	**0.668**	0.664	**0.666**	0.661
YOLOv8x	MCD	**ECE** ↓	**0.057**	0.064	**0.056**	0.059
		AUARC ↑	0.725	**0.824**	0.757	**0.781**
	MCDB	**ECE** ↓	N/A	**0.060**	N/A	N/A
		AUARC ↑	N/A	**0.801**	N/A	N/A
	MCSD	**ECE** ↓	**0.044**	0.064	**0.048**	0.060
		AUARC ↑	0.7890	**0.826**	0.771	**0.823**
RT-DETRx	MCD	**ECE** ↓	0.073	**0.042**	0.109	**0.077**
		AUARC ↑	0.822	**0.894**	0.731	**0.818**
	MCDB	**ECE** ↓	0.454	**0.037**	N/A	N/A
		AUARC ↑	0.197	**0.897**	N/A	N/A
	MCSD	**ECE** ↓	0.063	**0.037**	0.060	**0.050**
		AUARC ↑	0.874	**0.908**	0.835	**0.865**

10.3. Sensitivity to Stochastic Layer Placement

We observed that the efficacy of stochastic regularization for variational inference is dependent on the architectural depth at which it is applied. To quantify this architectural inductive bias, we compare performance when stochastic layers are restricted to the earlier stages of the adjusted blocks against the later stages. Table 2 presents the mean calibration (ECE) and uncertainty ranking (AUARC) metrics across these configurations, verifying the trends reported in the main paper.

Data Inclusion Criteria. Unlike the Pareto-optimal results reported in the main text (which isolate the single best trade-off configuration), Table 2 reports the mean performance across all configurations that met a minimum viability threshold in mAP and AUARC. Entries marked as **N/A** indicate configurations that resulted in severe performance degradation, rendering them non-viable for deployment.

It is important to note that because this table averages over a broader set of "viable" models rather than selecting the strict Pareto-best, certain uncertainty metrics (specifically AUARC) may appear higher here than in Tab. 1 in the main paper.

10.4. Distribution Shift Results

To supplement the visual trends presented in Fig. 3 of the main paper, Tab. 3 provides the granular quantitative metrics comparing In-Distribution (ID) (COCO) performance against the six distribution shifts in COCO-O.

Quantification of Degradation. The tabular data highlights the severity of the domain shift. Across all architectures, we observe a precipitous drop in predictive performance as the domain shifts from photorealistic ID (COCO) to abstract representations. For instance, on the 'tattoo' split, which represents one of the most severe shifts, mAP_{50-95} degrades by approximately 76% for Faster R-CNN and 69% for RT-DETRx compared to their in-distribution baselines.

Architecture-Specific Uncertainty Scaling. A critical insight revealed by Tab. 3 is the disparity in the *magnitude* of the uncertainty response between studied detectors. Faster R-CNN exhibits a dynamic range of entropy, nearly doubling from 0.796 (ID) to 1.569 (Tattoo), indicating a robust ability to express epistemic uncertainty. In contrast, the dense detectors (YOLOv8x, RT-DETRx) exhibit only a marginal entropy range even when detection performance collapses. While their relative trend is correct (entropy rises as mAP falls), their apparent overconfidence suggests that dense detectors require stronger calibration scaling to be interpretable in safety-critical OOD contexts.

Methodological Robustness. Finally, the data confirms that MCSD maintains strict parity with MCD under shift. On the hardest split (Tattoo), the performance gap between MCD and MCSD is negligible ($< 0.5\%$ mAP difference on RT-DETRx), confirming that the computational efficiency of MCSD does not come at the cost of OOD robustness.

10.5. Further Limitations

While MCSD offers a theoretically grounded and empirically robust method for uncertainty quantification, we acknowledge specific limitations inherent to the approach.

First, unlike MCD, which can be applied to almost any neural architecture, MCSD imposes a strict architectural dependency: it requires the presence of residual skip-connections to function as a valid Bayesian approximation.

Second, consistent with all ensemble-based and Monte Carlo methods, MCSD incurs a computational overhead at inference time. Generating a calibrated uncertainty estimate requires T forward passes, scaling the inference latency linearly. While this is acceptable for safety-critical offline analysis, it poses challenges for real-time constraints compared to single-pass deterministic methods.

Finally, our empirical analysis is currently restricted to the COCO and COCO-O datasets. Applying MCSD across a wider variety of object detection datasets is necessary to fully confirm the generalizability of these findings.

Table 3. Quantitative results under distribution shift. Comparison of mAP and Predictive Entropy across the COCO-O domains. Note the difference in Entropy measures between Faster R-CNN (Shannon Entropy) and the dense detectors (Binary Entropy).

			Dataset						
Model	**Method**	**Metric**	**COCO (ID)**	**weather**	**painting**	**handmake**	**cartoon**	**sketch**	**tattoo**
Faster R-CNN	MCD	**mAP.5.95** ↑	0.270	0.168	0.148	0.102	0.090	0.059	0.064
		Entropy ↓	0.796	1.101	1.289	1.505	1.374	1.310	1.569
	MCDB	**mAP.5.95** ↑	0.258	0.156	0.157	0.091	0.090	0.061	0.061
		Entropy ↓	0.894	1.155	1.340	1.531	1.375	1.371	1.546
	MCSD	**mAP.5.95** ↑	0.266	0.161	0.147	0.085	0.088	0.059	0.054
		Entropy ↓	0.646	0.944	1.100	1.26	1.134	1.111	1.375
YOLOv8x	MCD	**mAP.5.95** ↑	0.499	0.413	0.382	0.261	0.211	0.201	0.156
		Entropy ↓	0.00920	0.01005	0.01026	0.01024	0.01040	0.01030	0.01121
	MCDB	**mAP.5.95** ↑	0.462	0.379	0.344	0.231	0.181	0.174	0.143
		Entropy ↓	0.00958	0.01014	0.01041	0.01055	0.01044	0.01068	0.01105
	MCSD	**mAP.5.95** ↑	0.491	0.407	0.376	0.259	0.201	0.194	0.148
		Entropy ↓	0.00909	0.00990	0.01017	0.01011	0.01025	0.01026	0.01096
RT-DETRx	MCD	**mAP.5.95** ↑	0.449	0.393	0.414	0.284	0.226	0.201	0.137
		Entropy ↓	0.0936	0.0942	0.1000	0.1141	0.1099	0.1051	0.1247
	MCDB	**mAP.5.95** ↑	0.396	0.335	0.372	0.247	0.191	0.179	0.119
		Entropy ↓	0.0864	0.0875	0.0931	0.1062	0.1000	0.0973	0.1198
	MCSD	**mAP.5.95** ↑	0.437	0.380	0.395	0.263	0.217	0.200	0.135
		Entropy ↓	0.0916	0.0939	0.0989	0.1143	0.1071	0.1039	0.1218