Faculty of Engineering Sciences
Heidelberg University

Master Thesis
in Computer Engineering
submitted by
Maximilian Mielke
born in Duisburg, Germany
30/11/2025

UNCERTAINTY ESTIMATION FOR SINGLE STAGE OBJECT DETECTION

This Master thesis has been carried out by Maximilian Mielke
at the
Hardware and Artificial Intelligence (HAWAII) Lab
at the Institute of Computer Engineering
Supervisor: Prof. Dr. Holger Fröning
Second Examiner: Prof. Dr. Ullrich Köthe

ABSTRACT

While deep neural networks deliver state-of-the-art performance in object detection, their inherent tendency toward overconfidence compromises their reliability in safety-critical applications, necessitating robust methods for uncertainty quantification. Although full Bayesian inference would provide the most principled treatment of uncertainty, it is computationally impractical or even infeasiblefor modern large-scale models and real-time detection pipelines. However, the application of Bayesian approximation techniques to complex, real-world object detection scenarios remains significantly underexplored, as existing literature focuses predominantly on simplified toy problems and lower-dimensional datasets.

To address this gap, this thesis implements and evaluates Deep Ensembles and Monte Carlo Dropout within the state-of-the-art YOLOv8 architecture, assessing their ability to capture aleatoric and epistemic uncertainty across a corruption-augmented COCO dataset.

Various Monte Carlo Dropout configurations with different dropout locations were explored; however, Deep Ensembles offer superior robustness and epistemic uncertainty estimation compared to Monte Carlo Dropout, which requires aggressive dropout in the detection head to remain effective.

ZUSAMMENFASSUNG

Während tiefe neuronale Netze im Bereich der Objekterkennung eine leistungsstarke, dem Stand der Technik entsprechende Performance liefern, beeinträchtigt ihre inhärente Tendenz zur Überkonfidenz ihre Zuverlässigkeit in sicherheitskritischen Anwendungen und macht robuste Methoden zur Unsicherheitsquantifizierung erforderlich. Obwohl eine vollständige Bayes'sche Inferenz die prinzipiell fundierteste Behandlung von Unsicherheit bieten würde, ist sie für moderne großskalige Modelle und Echtzeit-Detektionspipelines rechnerisch unpraktikabel oder sogar unmöglich. Die Anwendung von bayesianischen Approximationsverfahren auf komplexe Objekterkennungsszenarien aus der realen Welt ist jedoch bislang nur unzureichend erforscht, da sich die bestehende Literatur überwiegend auf vereinfachte Probleme und niedrigdimensionale Datensätze konzentriert.

Um diese Lücke zu schließen, implementiert und bewertet diese Arbeit Deep Ensembles und Monte-Carlo-Dropout innerhalb der modernen YOLOv8-Architektur und untersucht deren Fähigkeit, aleatorische und epistemische Unsicherheit über ein durch Korruptionen erweitertes COCO-Dataset hinweg zu erfassen.

Es wurden verschiedene Monte-Carlo-Dropout-Konfigurationen mit unterschiedlichen Dropout-Positionen untersucht; allerdings bieten Deep Ensembles eine überlegene Robustheit und epistemische Unsicherheitsschätzung im Vergleich zu Monte-Carlo-Dropout, das einen aggressiven Dropout im Detection Head erfordert, um effektiv zu bleiben.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to Prof. Dr. Holger Fröning for giving me the opportunity to work alongside the highly talented and kind members of the HAWAII group. It has been a privilege to conduct my thesis in such an inspiring environment.

I also extend my heartfelt thanks to Hendrik Borras. His consistent support and guidance were invaluable to me throughout this process. I am deeply grateful for his mentorship, which provided a encouraging space where I could openly discuss my ideas and navigate any challenges I encountered.

Lastly, I want to thank my family and my partner for their unwavering patience and support. Because of you, I was able to pursue this remarkable master's program so far from home. Thank you for making this journey possible.

CONTENTS

1 INTRODUCTION AND MOTIVATION

Deep neural networks (DNNs) have emerged as state of the art solutions for a variety of tasks such as Natural Language Processing [27] [103] [12] and object detection [13] [15] [79]. However, a well-documented limitation of these models is their tendency to produce overly confident predictions, making internally calculated confidence scores unreliable [85] [122]. This overconfidence can lead to uncertainty regarding the reliability of the model's predictions, as it does not provide a clear measure of the uncertainty associated with its outputs.

These issues have driven the development of techniques to quantify uncertainty in model predictions based on Bayesian statistics and Bayes' theorem. Bayesian Neural Networks, derived from Bayesian methods, provide a framework for combining the capabilities of DNNs with uncertainty quantification [17].

Bayesian Neural Networks treat model parameters as probability distributions. However, the process of integrating or sampling from these distributions, which is necessary to make predictions, can add significantly to the computational cost. Furthermore, performing Bayesian inference to compute these probability distributions can itself be intractable or highly impractical for modern large-scale neural networks [55].

Various approximation methods have been proposed to address these challenges. These methods aim to provide more computationally efficient yet approximate ways of estimating the probability distributions of model parameters. One such approach is Variational Inference [6], which seeks to approximate the true posterior distribution. Other methods, such as Deep Ensembles [65] and Monte Carlo Dropout [37], have been introduced to approximate Bayesian methods by performing multiple forward passes through deep learning models with different weight configurations. These techniques allow uncertainty metrics to be estimated without the need for full Bayesian inference.

Although Bayesian inference with small multilayer perceptrons has been explored on relatively simple and low-dimensional datasets, such as MNIST [26] or subsets of ImageNet [25], these studies typically involve toy problems or less challenging object detection tasks [37] [65]. In contrast, the application of Bayesian methods to complex, real-world scenarios, such as object detection on the COCO dataset using

state-of-the-art deep learning models, remains largely underexplored in the literature.

The present work is motivated by the gap between simplified toy problems commonly found in the literature and the application of Bayesian approximation methods in complex, real-world object detection scenarios. The objective of this study is to implement and evaluate Bayesian Neural Network approximation techniques, specifically Deep Ensembles and Monte Carlo Dropout, within a state of the art object detection model, YOLOv8. The effectiveness of these methods is assessed through uncertainty metrics in both regression and classification tasks.

Accordingly, the main contributions of this work are summarized as follows:

(1) Architectural modification of YOLOv8: The YOLOv8 network is modified to incorporate support for Deep Ensembles and Monte Carlo Dropout. In addition, a methodology is developed to compute uncertainty metrics directly from the model's inference outputs.

(2) Assessment of Bayesian approximation techniques: The effectiveness of Deep Ensembles and Monte Carlo Dropout for uncertainty quantification is assessed by comparing various uncertainty metrics, while also discussing the practical trade-offs and computational costs associated with each method.

(3) Evaluation on a corruption-augmented dataset: A modified version of the COCO dataset is employed to more accurately reflect real-world conditions. The dataset includes typical corruption types such as weather effects, noise, blur and digital artifacts in different corruption severities.

This master thesis evaluates Bayesian approximation techniques under a demanding setting, combining a state-of-the-art model with a large-scale and complex dataset. By providing a practical benchmark and implementation guideline, this work aims to facilitate the wider adoption of these techniques for developing safer and more reliable AI systems.

2 BACKGROUND

This chapter establishes the theoretical foundations necessary for the experimental design and analysis presented later in this work. It begins with an introduction to Bayesian Neural Networks in Section 2.1, followed by a detailed discussion of approximation techniques in Subsections 2.1.1 through 2.1.4. Subsequently, the fundamental principles of Object Detection are outlined in Section 2.2, leading into an explanation of the specific YOLO architecture employed in this study in Section 2.3.

2.1 BAYESIAN NEURAL NETWORKS

In Bayesian Neural Networks (BNNs) the model parameters, typically the weights and biases ω, are treated as random variables rather than fixed quantities. This contrasts with the classical approach, where the weights are considered to have a single optimal but unknown value and the observed data $\mathcal{D}$ are regarded as random.

During training, BNNs aim to infer the *posterior distribution* $p(\omega \mid \mathcal{D})$ over the weights, which quantifies the updated belief about the parameters after observing the data. The computation of this posterior is governed by Bayes' theorem:

$$p(\omega \mid \mathcal{D}) = \frac{p(\omega) \cdot p(\mathcal{D} \mid \omega)}{\int p(\omega) \cdot p(\mathcal{D} \mid \omega)\, d\omega} = \frac{p(\omega) \cdot p(\mathcal{D} \mid \omega)}{p(\mathcal{D})} \tag{2.1}$$

where $p(\omega)$ is the *prior* distribution encoding a belief about the parameters before seeing any data, $p(\mathcal{D} \mid \omega)$ is the *likelihood* describing how probable the data is given the parameters and $p(\mathcal{D})$ is the *evidence* or marginal likelihood, which acts as a normalizing constant [43].

In a BNN, the forward pass for a given input x involves computing the output of the model y by integrating over the posterior distribution of the parameters. The predictive distribution as a result of the inference is formulated as [123]:

$$p(y \mid x, \mathcal{D}) = \int p(y \mid x, \omega)\, p(\omega \mid \mathcal{D})\, d\omega \tag{2.2}$$

In practice, the computation of the evidence integral $p(\mathcal{D})$ in equation 2.1, which is required for obtaining the posterior distribution

$p(\omega \mid \mathcal{D})$, is computationally intractable due to the high dimensionality and non-convexity of the underlying probability distributions [55], requiring exponential time for computation [6]. To enable practical applications, various approximation techniques have been developed, including Markov Chain Monte Carlo, Variational Inference and others [60]. These methods are discussed in detail in the following Sections 2.1.1 - 2.1.4.

Furthermore, computing the predictive distribution $p(y \mid x, \mathcal{D})$ as in equation 2.2 is equally challenging, particularly for non-trivial models, as it involves integration over a high-dimensional parameter space. A common strategy is Monte Carlo sampling, where multiple samples s are drawn from the posterior distribution to estimate the predictive distribution, as in equation 2.3. [60].

$$p(y \mid x, \mathcal{D}) \approx \frac{1}{S} \sum_{s=1}^{S} p(y \mid x, \omega_s) \tag{2.3}$$

2.1.1 Markov Chain Monte Carlo

While Markov Chain Monte Carlo (MCMC) methods are not directly employed in this work primarily due to their computational requirements in the context of large-scale models such as YOLOv8 they remain a fundamental baseline tool in Bayesian inference. As such, a brief overview is given to highlight their theoretical importance and application scope.

MCMC methods are designed to draw samples from complex, often high-dimensional probability distributions, particularly when direct computation or integration of the distribution is analytically intractable. MCMC approximates the posterior distribution of the model parameters given the observed data [105]. The MCMC method construct a Markov chain whose equilibrium distribution corresponds to the desired posterior, allowing inference via empirical sampling statistics.

A defining feature of MCMC is the Markov property: each state in the sequence depends only on its immediate predecessor, not on the full trajectory of prior samples. Formally, for a sequence of states $x^{(n)}$, this property is expressed as in equation 2.4:

$$P(x^{(n+1)} \mid x^{(n)}, x^{(t)} : t \in \mathcal{E}) = P(x^{(n+1)} \mid x^{(n)}), \tag{2.4}$$

where $\mathcal{E}$ denotes all preceding successful time steps t [83].

The Monte Carlo aspect refers to the stochastic nature of the method, which uses random sampling to explore the parameter space. Together,

these elements define a versatile family of algorithms for numerically approximating probability distributions through iterative, sample-based procedures [106].

To ensure that the Markov chain correctly represents samples from the target distribution, it must satisfy the property of ergodicity. Ergodicity guaranties that it is possible to transition from any state to any other and that the chain avoids cycles that would prevent convergence. If these conditions are met and as the number of iterations increases, the distribution over the sampled states converges to a stationary distribution. In practice, however, this convergence cannot be proven analytically and is instead assumed to occur after a so-called burn-in period, during which early samples, potentially biased by initial conditions, are discarded [17].

A Markov chain is fully characterized by two elements: the initial distribution over states, $p_0(x)$ and the transition kernel $T_n(x, x')$, which defines the probability of moving from state x to state x' at time step n is given as:

$$p_{n+1}(x') = \sum_{\tilde{x}} p_n(\tilde{x}) T_n(\tilde{x}, x'), \tag{2.5}$$

where $p_n(\tilde{x})$ is the probability of being in state $\tilde{x}$ at time n [83].

Given an initial distribution, this recursion determines the distribution over states for all future time steps.

Various MCMC algorithms differ primarily in how new states are proposed. Common variants include Hamiltonian Monte Carlo [11], Gibbs Sampling [20] or the No-U-Turn Sampler [52].

Once a new sample is proposed, it is either accepted or rejected according to a transition probability that ensures that the Markov chain has the correct stationary distribution. This is typically done using a Metropolis-Hastings acceptance algorithm, which compares the posterior probability of the proposed state with that of the current state [97].

2.1.2 Variational Inference

Variational inference approximates the intractable posterior distribution over model parameters by employing a simpler parameterized family of distributions. This approach has been proposed to address the limitations of MCMC methods, in particular their slow convergence and poor scalability [38].

A technique for approximating a target probability distribution is to minimize the *Kullback–Leibler divergence* (KL), which quantifies the difference between two probability distributions [59]. The KL divergence is based on the Shannon entropy, a fundamental measure

of uncertainty in information theory [101]. Shannon entropy is defined as:

$$H(P) = -\sum_{x \in \mathcal{X}} P(x) \log P(x) \tag{2.6}$$

Building on this, the KL divergence between two probability distributions P and Q is given by [38].:

$$D_{\mathrm{KL}}(P \parallel Q) = \mathbb{E}_{x \sim P(X)} \left[\log \frac{p(x)}{q(x)} \right] \tag{2.7}$$

In the framework of Bayesian inference, the KL divergence can be expressed in terms of the approximate distribution $q(\omega)$ and the target posterior $p(\omega \mid \mathcal{D})$ as follows [6]:

$$D_{\mathrm{KL}}(q(\omega) \parallel p(\omega \mid \mathcal{D})) = \mathbb{E}[\log q(\omega)] - \mathbb{E}[\log p(\omega \mid \mathcal{D})] \tag{2.8}$$

By applying Bayes' theorem, the posterior distribution can be rewritten as [6]:

$$p(\omega \mid \mathcal{D}) = \frac{p(\omega, \mathcal{D})}{p(\mathcal{D})} \tag{2.9}$$

Substituting this into the KL divergence yields [6]:

$$D_{\mathrm{KL}}(q(\omega) \parallel p(\omega \mid \mathcal{D})) = \mathbb{E}[\log q(\omega)] - \mathbb{E}[\log p(\omega, \mathcal{D})] + \log p(\mathcal{D}) \tag{2.10}$$

To find the distribution $q(\omega)$ that best approximates the target posterior $p(\omega \mid \mathcal{D})$, a optimization problem is formulated that minimizes the KL divergence:

$$q^*(\omega) = \arg\min_{q(\omega) \in \mathcal{Q}} D_{\mathrm{KL}}(q(\omega) \parallel p(\omega \mid \mathcal{D})) \tag{2.11}$$

However, as can be seen in equation 2.10 and because evaluating the KL divergence in equation 2.11 requires computing the marginal likelihood $\log p(\mathcal{D})$, which is generally intractable as noted above, the *Evidence Lower Bound* (ELBO) was introduced [38]. By rearranging Equation 2.10 the ELBO is defined as:

$$\mathrm{ELBO}(q) = \mathbb{E}[\log p(\omega, \mathcal{D})] \; - \; \mathbb{E}_{q(\omega)}[\log q(\omega)]$$

which is equivalent to the negative KL divergence plus the constant $\log p(\mathcal{D})$ [38].

This formulation removes the dependence on the intractable term $\log p(\mathcal{D})$ by treating it as a constant, thereby yielding a tractable objective for variational optimization [6].

Equivalently, the ELBO can be expressed in terms of the expected log-likelihood and a regularization term as

$$\text{ELBO}(q) = \mathbb{E}[\log p(\mathcal{D} \mid \omega)] \; - \; D_{\text{KL}}\big(q(\omega) \,\|\, p(\omega)\big)$$

where the first term measures how well the model explains the observed data and the second term penalizes deviation of the variational distribution from the prior [6].

Moreover, the ELBO lower-bounds the log evidence,

$$\log p(\mathcal{D}) = \text{ELBO}(q) + D_{\text{KL}}\big(q(\omega) \,\|\, p(\omega \mid \mathcal{D})\big)$$

and since $D_{\text{KL}}(\cdot) \geq 0$, it implies that $\log p(\mathcal{D}) \geq \text{ELBO}(q)$ [59]. Thus, maximizing the ELBO is exactly equivalent to minimizing the KL divergence to the target posterior [6].

To optimize the variational distribution, the goal is to maximize the ELBO, requiring an estimator for the gradient of the ELBO with respect to the variational parameters λ. In variational inference the gradient of the ELBO can be expressed as an expectation with respect to the parameters of the variational distribution λ, i.e. the mean and variance [90]:

The prior distribution $q(\omega)$ provides the initial guess for the hidden variables in the model. These hidden variables, denoted ω, are typically assumed to belong to a mean-field variational family $q(\omega|\lambda)$. The values of these parameters are iteratively adjusted during the optimization process to approximate the true posterior distribution of the hidden variables as shown in equation 2.12 [51].

$$\nabla_\lambda \mathcal{L} = \mathbb{E}_q \left[\nabla_\lambda \log q(\omega|\lambda) \left(\log p(\omega, \mathcal{D}) - \log q(\omega|\lambda)\right)\right] \tag{2.12}$$

where the expectation is taken with respect to the variational distribution $q(\omega|\lambda)$. However, evaluating this gradient directly can be computationally expensive, especially for complex models or large datasets.

Stochastic Variational Inference (SVI) addresses the limitation of traditional VI, which requires passing through the entire dataset to perform a single gradient calculation and therefore a single optimization iteration. This becomes problematic, especially with large datasets, due to memory constraints and increasing computational cost per iteration, leading to poor scalability. To overcome this issue, SVI introduces a noisy gradient-based method to update the parameters of the variational distribution [51].

Specifically, it employs Monte Carlo sampling to approximate the gradient: by drawing S samples from the variational distribution, a

noisy but unbiased estimate of the gradient is obtained, as defined in equation 2.13 [51, 90].

$$\nabla_\lambda \mathcal{L} \approx \frac{1}{S} \sum_{s=1}^{S} \nabla_\lambda \log q(\omega_s|\lambda) \left(\log p(\omega_s, \mathcal{D}) - \log q(\omega_s|\lambda)\right) \quad (2.13)$$

in which each sample ω_s is drawn from the variational distribution $q(\omega|\lambda)$. This sample-based gradient for a effiencient calculation of the gradient of the ELBO, even for large datasets. The noisy gradients derived from this process can then be used to iteratively update the variational parameters λ improving the approximation of the target distribution. The update rule for λ is typically governed by a learning rate schedule and the process continues until convergence [51].

2.1.3 Deep Ensembles

Ensemble methods combine multiple models to create a more robust and capable model, improving predictive performance [28]. Furthermore, Lakshminarayanan et al. [65] demonstrated that Deep Ensembles are effective in modeling predictive uncertainty. Although diversity among individual model parameters is crucial, traditional methods such as bagging, which trains models on different data subsets, are not strictly necessary. Research shows that other sources of stochasticity, including random weight initialization and mini-batch selection, introduce sufficient diversity [40] [68]. This allows ensemble members to be trained on the full dataset, thereby maximizing accuracy.

Furthermore, adversarial training examples have been used to increase model robustness and Out-Of-Domain detection[44]. After training, inference is performed by combining the predictions of all ensemble members.

The combined prediction is given by equation 2.14:

$$p(y|x) = \frac{1}{M} \sum_{m=1}^{M} p_{\theta_m}(y|x, \theta_m) \quad (2.14)$$

For classification tasks, the mean value of the output distribution is computed, such as the output of a softmax activation. In the case of regression, the value of each bounding box coordinate is averaged. If the model is extended with an additional head to predict variance, these variance estimates are also averaged, forming a Gaussian distribution for the regression problem [65].

When comparing Deep Ensembles with other popular methods, it was found that Deep Ensembles were easier to implement than methods based on Bayes' theorem. Nevertheless, they can achieve performance on a par with Bayesian neural networks, even with relatively small ensemble sizes, such as $M = 5$ models [65].

Repulsive Ensembles extend the Deep Ensemble approach by introducing a repulsive term to the model parameters. This repulsive term encourages functional diversity among the ensemble members and prevents early saturation when adding more models to the ensemble. It has been shown that the training dynamics of repulsive ensembles correspond to a Wasserstein gradient flow that minimizes the KL divergence to the Bayesian posterior [21] [77] [76] [31].

Repulsive ensembles address the issue that deep neural networks are often overparameterized. In such models, different sets of parameters can describe the same function, leading to a false sense of diversity [21]. Therefore, it is necessary to introduce a repulsive term in two spaces: the parameter space and the function space. In the parameter space, the repulsive term pushes the model weights apart, promoting diversity. In the function space, it mitigates the effects of overparameterization by ensuring that the outputs of the models remain distinct.

The parameter space update is formulated in equation 2.15 [65]. In this expression, the first term represents the classic optimization objective, while the second term applies the repulsive force to ensure diversity:

$$\phi(\mathbf{w}_i^t) = \nabla_{\mathbf{w}_i^t} \log p(\mathbf{w}_i^t|\mathcal{D}) - \mathcal{R}\left(\left\{\nabla_{\mathbf{w}_i^t} k(\mathbf{w}_i^t, \mathbf{w}_j^t)\right\}_{j=1}^{n}\right) \tag{2.15}$$

Equation 2.16 specifies the repulsive kernel gradient, where the interaction between model weights is scaled by their distance and modulated by a kernel function to maintain a well-spread approximation of the posterior [65]:

$$\nabla_{\mathbf{w}_i^t} k(\mathbf{w}_i^t, \mathbf{w}_j^t) = \frac{2}{h}(\mathbf{w}_j^t - \mathbf{w}_i^t)\, k(\mathbf{w}_i^t, \mathbf{w}_j^t) \tag{2.16}$$

For the function space, the weight update is formulated by first computing gradients in the function space and then projecting them back into the parameter space using the Jacobian $\frac{\partial f_i^t}{\partial \mathbf{w}_i^t}$, aligning the update of the weights with the functional posterior. The repulsive term operates on projected functions $\pi_B(f_i^t)$, evaluated over a subset B of the input space and encourages diversity among the models outputs by penalizing similarity in function space rather than in weight space [21].

The function space update is given by equation 2.17:

$$\phi(\mathbf{w}_i^t) = \left[\frac{\partial f_i^t}{\partial \mathbf{w}_i^t}\right]^\top \left(\nabla_{f_i^t} \log p(f_i^t|\mathcal{D}) - \mathcal{R}\left(\left\{\nabla_{f_i^t} k(\pi_B(f_i^t), \pi_B(f_j^t))\right\}_{j=1}^{n}\right)\right) \tag{2.17}$$

Since training each ensemble member with its respective repulsive term introduces increased complexity and computational cost, an

alternative approach is to use a shared backbone with a lightweight repulsive ensemble head. Such strategy promotes functional diversity at lower computational cost, improves uncertainty estimation and enables efficient integration with pre-trained models [107] [102].

2.1.4 Monte Carlo Dropout

Monte Carlo Dropout (MCDO) is a another proposed method to approximate Bayesian Inference in neural networks by enabling dropout at both training and test time.

From a Bayesian perspective, MCDO can be seen as an approximation to a deep Gaussian Process [37] [22]. A Gaussian Process is a stochastic process where any finite set of function values follows a joint Gaussian distribution making it a natural Bayesian prior over functions that dropout networks can approximate [32].

Although MCDO uses a single trained network with stochastic forward passes, in practice it behaves similarly to an ensemble method. Each dropout sample corresponds to a different subnetwork. Therefore the technique can be interpreted as sampling from an implicit ensemble of models [65].

Let $f(\cdot)$ denote the predictive function of the neural network. At test time, multiple stochastic forward passes are performed through the network, each time with a different dropout mask applied to the neurons. This produces a set of stochastic predictions $\{f^{(t)}(x)\}_{t=1}^{T}$ for the input x, where each $f^{(t)}(x)$ corresponds to a different sample from the approximate posterior.

When modeling uncertainty, these sets of predictions are used to estimate both the predictive mean and variance of the model's output. Specifically, the predictive mean is approximated by averaging the outputs of the T stochastic forward passes, while the predictive variance captures both the model's inherent noise and the variation across these predictions. Formally, the predictive mean is given by equation 2.18 and the predictive variance is estimated as the sample variance of the predictions [37].

$$\mathbb{E}q(y|x)(y^*) \approx \frac{1}{T}\sum t = 1^T f^{(t)}(x) \tag{2.18}$$

MCDO can also be interpreted as performing variational inference by approximating the intractable posterior distribution $p(\omega|X, Y)$ with a variational distribution $q(\omega)$. This approximation is constructed by applying Bernoulli-distributed dropout masks to the network weights. For each layer i, the variational distribution over weights is defined as [37]:

$$W_i = M_i \cdot \text{diag}([z_{i,j}]_{j=1}^{K_{i-1}}) \tag{2.19}$$

where $z_{i,j} \sim \text{Bernoulli}(p_i)$ for $i = 1, ..., L$ and $j = 1, ..., K_{i-1}$. Here, M_i are variational parameters to be optimized and p_i represents the dropout probability for neurons in layer i. The binary variable $z_{i,j} = 0$ corresponds to unit j in layer $i-1$ being dropped out as an input to layer i.

This variational approximation leads to the standard loss function with L2 regularization [37]:

$$L_{\text{dropout}} = \frac{1}{N} \sum_{i=1}^{N} E(y_i, \hat{y}_i) + \lambda \sum_{i=1}^{L} \|W_i\|_2^2 + \|b_i\|_2^2 \tag{2.20}$$

where $E(y_i, \hat{y}_i)$ is a task-specific loss function (such as softmax loss or Euclidean loss), N is the number of data points and λ is the weight decay parameter controlling regularization strength.

This dropout objective corresponds to minimizing the KL divergence between the variational distribution $q(\omega)$ and the true posterior of a deep Gaussian process which MCDO approximates. The connection to the previously mentioned ELBO is given by equation 2.21:

$$L_{\text{dropout}} \propto -\text{ELBO}(q) = -\mathbb{E}[\log p(\mathcal{D}|\omega)] + D_{\text{KL}}(q(\omega)\|p(\omega)) \tag{2.21}$$

The task-specific loss term acts as an approximation of the negative log-likelihood, representing how well the model explains the observed data. Simultaneously, the L_2 regularization term applied to the network weights approximates the KL divergence between the variational posterior distribution and the prior. Consequently, minimizing this dropout loss function is equivalent to maximizing the ELBO [37].

2.2 OBJECT DETECTION IN MACHINE LEARNING

Computer vision is an umbrella term that describes a range of tasks designed to interpret visual data. These tasks can be as simple as image classification or as complex as object detection and semantic segmentation, which are shown in Figure 2.1. The granularity of the analysis is what primarily differentiates these tasks, ranging from entire image categorization to pixel-level understanding and spatial localization of entities.

2.2.1 Problem Definition

Object detection is a composite task that addresses both *where* objects are located and *what* category they belong to. The objective is to estimate the locations and categories of all instances within a single image, regardless of the total number of objects present [109].

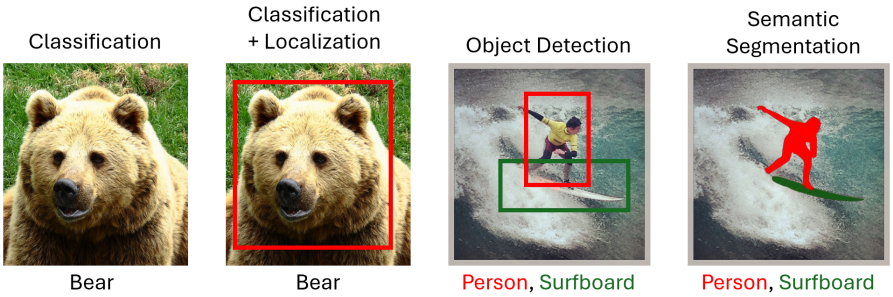


Figure 2.1: Comparison of computer vision tasks showing the progression from single-label Classification and Localization to multi-object Detection and Semantic Segmentation. Images are from the COCO dataset [74].

Formally, the output of a object detection model is a set of predictions where each detected instance is usually represented by a bounding box, a class label and a confidence score. The bounding box is usually expressed as a vector $b = (x, y, w, h)$, where (x, y) denotes the coordinates of the box center and w and h represent the width and height relative to the image dimensions. Alongside localization, the model predicts a class label c from a set of predefined categories and a confidence score $P(Object)$, reflecting the probability that the region contains an object of interest. In modern regression based frameworks, this confidence often accounts for the probability of the object's existence and the Intersection over Union fit between the predicted box and the ground truth [128].

Several variations in visual data complicate achieving robust detection. Detection algorithms must address significant intra-class variations, such as diverse structural features and appearances within the same category [75]. Another challenge is occlusion, in which objects are only partially visible or overlap, requiring the model to infer the entire object with missing information. Additionally, objects in natural scenes exhibit size variations and detecting small objects is particularly challenging due to the coarse nature of feature maps in deep convolutional networks, which can result in a loss of semantic information for small instances [72]. Finally, the detector must be invariant to viewpoint changes and maintain accurate localization and classification despite alterations in object pose, rotation and lighting conditions [128] [75].

2.2.2 Network Architectures

Modern object detection frameworks typically consist of three distinct functional components: a *backbone* network, responsible for extracting semantic feature representations from the input image, a *neck*, which aggregates and refines these features and a *head*, which performs the final localization and classification [33]. Understanding current detec-

tion methods requires examining the fundamental building blocks of these backbone and neck structures and the approaches used to design the detection heads.

2.2.2.1 Building Blocks of Neural Networks

The foundational unit of deep learning is the Perceptron [99]. When connected in multiple layers, these units form the simplest neural network architecture, known as the Multilayer Perceptron (MLP). While MLPs are universal function approximators, they lack the spatial inductive biases required for efficient computer vision. Because every input pixel connects to every neuron in the subsequent layer, MLPs ignore the local spatial structure inherent in visual scenes. Consequently, they suffer from a prohibitive explosion of parameters when processing high-dimensional image data [67].

Convolutional Neural Networks (CNNs) addressed these limitations and became the standard for visual feature extraction. Unlike MLPs, CNNs employ learnable filters called kernels that slide over the input. This architecture enables the network to capture local patterns such as edges and textures in earlier layers and progressively more complex structures in deeper layers [66]. To reduce the amount of trainable parameters, modern backbones such as the ResNet family [47] utilize pooling layers and strided convolutions to downsample the spatial resolution while increasing the depth of the feature channels.

Attention modules have recently emerged as a powerful alternative or complement to convolutions. Originating in natural language processing with the Transformer architecture [117], attention mechanisms calculate the relevance of one part of the input sequence to another, regardless of their positional distance [4] [63]. In computer vision, this allows the model to capture global context and dependencies that localized convolution operators might miss. Both Vision Transformers [29] and detection specific Transformers [13] treat image patches as sequences. They use attention mechanisms to model relationships between different areas of an image, which improves the detector's ability to understand object interactions and occlusions.

2.2.2.2 Detection Heads

Once features are extracted by the backbone and refined by the neck, the detection head determines how these features are translated into final predictions. This process is generally categorized into two primary methods: Two Stage and single stage detectors.

Two-Stage Detectors, popularized by the R-CNN family, operate on a proposal driven paradigm. The detection process begins with a Region Proposal Network, which generates a set of candidate regions that are likely to contain objects, filtering out the vast majority of the background. In the second stage, these proposed regions are pooled

into feature maps of the same size and then passed to a classifier and regressor to refine the coordinates and predict the class label [42]. Because the detection head only processes regions associated with high probability, two-stage detectors typically achieve higher localization accuracy and are more robust to small objects. However, this accuracy comes at the cost of computational efficiency, as the sequential nature of the two steps creates a bottleneck that increases latency and the number of parameters [104].

Single stage Detectors, such as the YOLO series [92] and SSD [79], prioritize speed by framing detection as a single regression problem. Instead of generating proposals, these models predict bounding boxes and class probabilities directly from the feature maps in a single forward pass. However, this architectural efficiency historically required a trade-off between inference speed and detection accuracy. Early single stage models often struggled with difficult examples, particularly manifesting as poor performance on small objects [104]. Consequently, much of the subsequent innovation in single stage detection has focused on bridging the accuracy gap to handle difficult tasks without sacrificing real-time performance. Prominent examples include the introduction of Focal Loss [73] to mitigate foreground-background class imbalance and the development of multi-scale architectures that fuse feature maps of varying resolutions to enhance small object detection [72].

2.2.3 Evaluation Metrics

To assess the performance of object detection models, standardized metrics are required to measure both the localization accuracy and the classification correctness. The evaluation pipeline relies on comparing the predicted bounding boxes against the ground truth annotations using geometric overlap, which subsequently allows for computation of other metrics like precision and recall.

Intersection over Union:

The most commonly used metric for measuring the geometric accuracy of a predicted bounding box is the Intersection over Union (IoU). IoU measures the degree of overlap between the predicted box B_p and the ground truth box B_{gt}. It is calculated as the ratio of the area of their intersection to the area of their union:

$$IoU = \frac{Area(B_p \cap B_{gt})}{Area(B_p \cup B_{gt})} \tag{2.22}$$

This metric produces a value between 0 and 1, where 0 indicates no overlap and 1 indicates a perfect match. In evaluation pipelines, the IoU serves as a threshold to define correctness. If the IoU exceeds a

pre-defined threshold α, the prediction is classified as a True Positive (TP), otherwise, it is considered a False Positive (FP) [96].

Precision and Recall:

Based on the classifications into TP and FP derived from the IoU threshold, the model's performance is further decomposed into Precision and Recall. Precision quantifies the reliability of the model's positive predictions, measuring the percentage of detected objects that are actual ground truth instances. Conversely, Recall measures the coverage of the model, evaluating the percentage of all ground truth objects that were successfully detected. These are formally defined as:

$$Precision = \frac{TP}{TP + FP} \quad Recall = \frac{TP}{TP + FN} \tag{2.23}$$

where FN (False Negative) represents ground truth objects that the model failed to detect. A detector seeks to maximize both metrics, though there is typically a trade-off: lowering the detection confidence threshold increases Recall but often decreases Precision by introducing more False Positives [89].

Mean Average Precision:

To provide a single scalar metric that summarizes performance, Average Precision is utilized. AP is defined as the area under the Precision-Recall curve, which is generated by plotting precision against recall as the prediction confidence threshold is varied. Mathematically, the Average Precision for a specific class corresponds to the integral of the precision $p(r)$ as a function of recall r:

$$AP = \int_0^1 p(r)dr \tag{2.24}$$

In discrete implementations, this integral is approximated using interpolation methods [34]. The Mean Average Precision is subsequently derived by calculating the mean of the AP scores across all N object categories in the dataset.

$$mAP = \frac{1}{N} \sum_{i=1}^{N} \mathrm{AP}_i \tag{2.25}$$

Additional metrics, including the F1 score and deployment constraints such as processing speed and energy efficiency, are acknowledged but remain beyond the scope of this analysis.

2.3 YOU ONLY LOOK ONCE

YOLO (You Only Look Once) models belong to the class of convolutional neural networks and are specifically designed for real-time

object detection. These models typically follow the modular architecture consisting of three main components: backbone, neck and detection head as discussed in Chapter 2.2.2 [33]. The YOLO model series has undergone extensive development over the years by different people, groups and companies, with the aim of improving both detection accuracy and computational efficiency.

YOLOv8 [58], published in 2023, was chosen as the model for this work due to its recent development and its practical advantages. Although its successor, YOLOv11, has been introduced lately, it remains poorly documented and is not yet widely adopted. Furthermore, other members of the institute's research group have been working with YOLOv8, enabling more consistent comparisons and insights across experiments. Notably, similar to YOLOv5, YOLOv8 was released without an accompanying official publication.

2.3.1 Advances in YOLO Architecture

The series of YOLO models originated from the work of Joseph Redmon, who introduced the first YOLOv1 model in 2015 with the paper *You Only Look Once: Unified, Real-Time Object Detection* [92]. The key feature of YOLOv1 was the unification of bounding box regression and object classification in a single neural network, allowing object detection in a single forward pass, categorizing it as a single stage object detector. This eliminated the need for separate region proposal stages, which were common in SOTA methods at the time.

YOLOv1 divides the input image into an $S \times S$ grid. Each grid cell predicts B bounding boxes, where each box encodes its coordinates (x, y, w, h), along with a confidence score. This confidence score reflects both the probability that the box contains an object and the accuracy of the bounding box prediction. Additionally, each grid cell outputs a set of class probability scores, conditioned on the presence of an object. During inference, the class probabilities are multiplied by the bounding box confidence scores to yield class-specific confidence values for a given bounding box. A non-maximum suppression step is then applied to remove duplicate detections [92].

The original YOLO architecture employed 24 convolutional layers followed by 2 fully connected layers as the head for final prediction. With this design, real-time inference is possible while maintaining competitive accuracy compared to other SOTA approaches such as R-CNN [42], Fast R-CNN [41], Faster R-CNN [95] and DPMs [35].

Subsequent versions of YOLO introduced substantial architectural modifications while retaining the single stage detection principle [113]:

YOLOv2 [93] introduced batch normalization [54], which stabilizes and accelerates training by normalizing the inputs of each layer. It also adopted anchor boxes for bounding box predictions, allowing

the network to better predict objects with varying aspect ratios and transitioned to the Darknet-19 backbone.

YOLOv3 [94] replaced the softmax classifier with a binary cross-entropy loss to support multilabel classification within a single bounding box. It introduced the Darknet-53 backbone with residual connections to improve gradient flow in deep networks. Additionally, Feature Pyramid Networks (FPNs) were used to improve multi-scale detection by combining low-resolution, semantically strong features with high-resolution, detailed features. This is done through a top-down process, where the features from deeper layers, which contain strong semantic information but are lower in resolution, are upsampled and merged with the higher resolution features from earlier convolutional layers [72].

YOLOv4 [7] retained the Darknet-53 backbone but modified it using Cross-Stage Partial Networks (CSPNet) [119], which splits the input feature map, processes one part through additional layers and then concatenates it with the other part. The FPN used in YOLOv3 has been replaced by a Path Aggregation Network (PANet) [78]. Building upon the standard FPN structure, PANet refines and merges multi-scale feature maps starting from the lowest resolution upward to higher stages using a series of convolutional building blocks. At each stage, higher-resolution feature maps are combined with coarser ones via lateral connections, followed by convolutional operations that unify and propagate information, thereby improving the localization of objects across scales.

Starting with YOLOv5, development was transferred to the company Ultralytics [115], which maintains and releases updates without accompanying academic papers. Analysis of open source code, documentation and independent studies are necessary to understand these models.

YOLOv5 introduced several practical improvements, including half-precision (float16-based) inference for speed, data augmentation techniques such as mosaic augmentation and deployment in PyTorch [87] instead of Darknet [91]. The model has been released in multiple sizes (e.g. YOLOv5n to YOLOv5x) to accommodate different performance and resource constraints [62].

Other models such as YOLOv6 [70], YOLOv7 [118], YOLOR [120], YOLOX [39] and DAMO-YOLO [125] have been developed by other groups and are not discussed further here. This work focuses exclusively on YOLOv8, developed by Ultralytics, as it is the direct successor to YOLOv5.

2.3.2 YOLOv8

The YOLOv8 architecture, illustrated in Figure 2.2, employs a custom CSPDarknet53 backbone, maintaining conceptual similarities with

YOLOv5. However, the CSP layers have been replaced by C2f blocks (Cross-Stage Partial bottleneck with two convolutions).

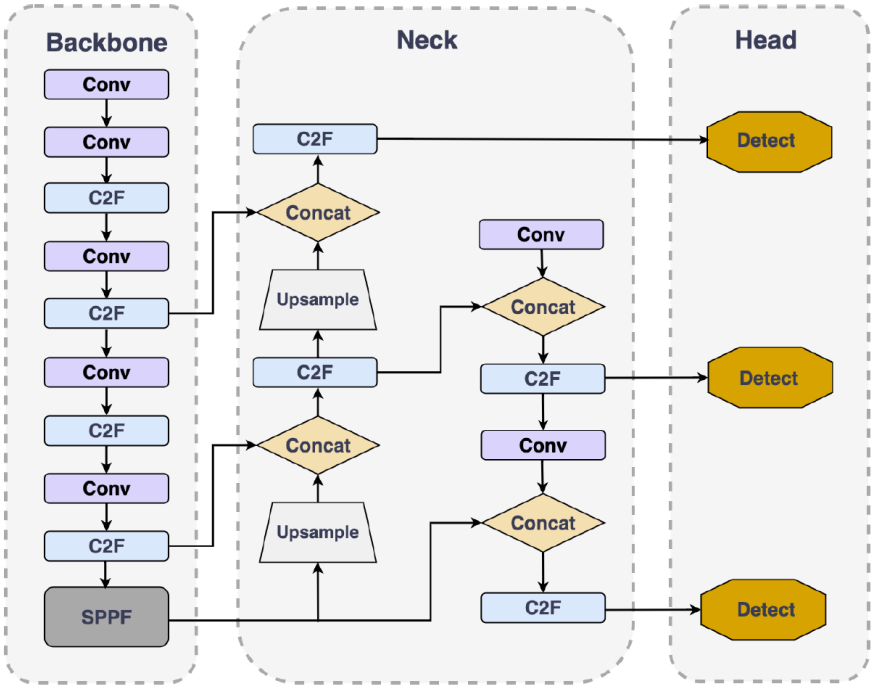


Figure 2.2: Overview of the YOLOv8 architecture comprising a CSPDarknet-based Backbone, a hybrid FPN and PAN Neck and three parallel detection heads for multi-scale object detection. The architecture utilizes C2F modules for efficient feature extraction and aggregation and includes SPPF to support input images of varying sizes and aspect ratios [126]

.

These are shown in detail in Figure 2.3 together with layer-level visualization of the different blocks used in the network.[1] The C2f block processes its input through an initial 1x1 convolutional layer, after which the feature map is split into two separate paths. One path is processed through a bottleneck module involving additional convolutions, while the other bypasses this operation. The outputs of both paths are concatenated and processed by a final 1x1 convolutional layer. The C2f block structure helps preserve low-level spatial detail that is often lost after successive convolutional operations, while still allowing deeper feature extraction through the bottleneck path.

The final component of the backbone is the Spatial Pyramid Pooling Fast (SPPF) module. This block enables the extraction of a fixed-length feature representation regardless of input image size. It does so by pooling feature maps at multiple spatial scales and concatenating them, as described by He et al [46].

1 https://github.com/ultralytics/ultralytics/issues/189#issue-1527158137

The neck structure integrates a hybrid approach combining elements of the FPN top-down and PANet down-up processes [127]. The FPN and PANet implementations have been adapted to also use C2f blocks.

The detection head consists of three separate detection modules, each operating on different feature map resolutions from the neck. These outputs are then combined to produce the final detection predictions. Unlike previous versions of YOLO, YOLOv8 employs an anchor-free detection approach, allowing the model greater flexibility in detecting objects of different scales and aspect ratios [50].

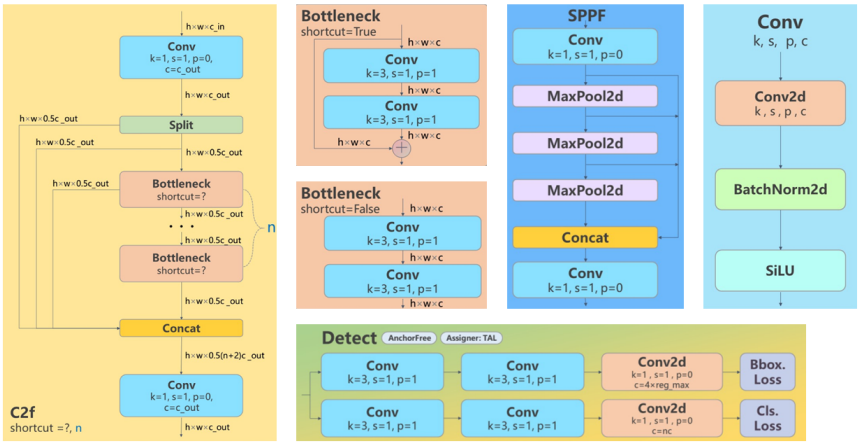


Figure 2.3: Layer-wise representation of key architectural blocks utilized in YOLOv8 including C2f and Bottleneck Modules, Cov Layers, SPPF and Detection Head

The YOLOv8 model is available in five different sizes - YOLOv8n, YOLOv8s, YOLOv8m, YOLOv8l and YOLOv8x - offering a choice between inference speed and detection accuracy. In addition, YOLOv8 supports five task-specific modes, each with a distinct model: object detection, image segmentation, image classification, pose estimation and oriented bounding box detection. For the object detection task, Table 2.1[2] summarizes the performance metrics of each model variant.

Table 2.1: Performance Comparison of YOLOv8 Models

Model	Size [px]	$mAP^{50:95}_{val}$ [%]	CPU ONNX [ms]	A100 TensorRT [ms]	Params [M]	FLOPs [B]
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

Due to the relatively small model size of a few megabytes size with 3.2 million parameters, the nano version of YoloV8 was chosen for this work, as it allows scalability with faster training and inference,

2 https://docs.ultralytics.com/de/models/yolov8/#overview

while still achieving competitive accuracy on the Common Objects in Context [74] dataset.

3 STATE OF THE ART AND RELATED WORKS

This chapter provides an overview of recent advances and the state of the art in object detection and uncertainty estimation. In particular, Section 3.1 discusses the state of the art of object detection models, while Section 3.2 reviews methods and related work for incorporating uncertainty estimation techniques into object detection models.

3.1 OBJECT DETECTION

Object detection is a computer vision task that involves both identifying and localizing instances of predefined object categories within an image [128]. This task can be divided into 2D and 3D object detection: 2D detectors predict the height and width of an object, while 3D detectors additionally estimate its depth. Object detection has been widely applied in various fields, including medical imaging [98] [121], robotics [69] [100] as well as autonomous driving [5] [18].

CNNs are extensively used for feature extraction in object detecion, employing convolutional layers that apply learnable kernels to recognize spatial hierarchies in the input data. These filters capture local patterns such as edges and textures, allowing deeper layers of the network to identify increasingly complex structures [86]. For many years, CNN-based models such as R-CNN [42] and YOLO [92] have been dominant in the field, consistently achieving leading benchmark results.

However, transformer-based networks have recently surpassed CNNs in performance and have taken the lead in the object detection ranking.[1] Their success can be attributed to the fact that transformers do not have the receptive field limitations imposed by kernel size [4] [63]. Notable transformer-based object detection models include Vision Transformers [29], Swin Transformers [80], DINO [15] and DETR [13]. Unlike CNNs, which focus on local patterns, transformers use attention mechanisms, capturing global relationships and allowing the model to better understand the context and structure of the whole image [13].

The field has also advanced towards unified, promptable foundation models that extend beyond the category constraints of traditional

1 https://leaderboard.roboflow.com/

CNN and transformer-based detectors, which require that objects to be detected be part of the training dataset. A prime example at the time of writing this thesis is the release of SAM 3, which introduces Promptable Concept Segmentation to detect, segment, and track objects using text prompts or image examples. Unlike standard architectures that couple recognition and localization within specific class heads, SAM 3 utilizes a shared vision encoder backbone that feeds into both a DETR-based detector and a memory-based video tracker [14].

Despite the emergence of other methods, CNNs are still widely used in practice. This is mainly due to their lower data requirements for training and faster inference speed [81] [24]. Prominent architectures in CNN-based object detection include the YOLO [92] and Region-based CNN families [41], alongside other influential models such as RetinaNet [71] and EfficientDet [110]

3.2 UNCERTAINTY ESTIMATION IN OBJECT DETECTION

Uncertainty estimation in object detection has gained increasing attention as a means to improve the accountability and reliability of AI systems. While traditional object detectors provide point estimates for bounding boxes and class scores, uncertainty-aware approaches aim to complement these predictions with measures of uncertainty [61].

Techniques for estimating these uncertainties can be broadly divided into two paradigms: sampling-based methods, which rely on multiple forward passes to approximate the posterior distribution, offering high-quality estimates at the cost of computational latency and single-pass methods, which are real-time capable and predict uncertainty directly, though often with limitations in capturing full epistemic uncertainty. Alternatively, these techniques can be categorized by their theoretical foundations, distinguishing between principled methods that incorporate a Bayesian prior and those that do not.

Sampling-based Methods

Sampling-based approaches approximate Bayesian inference by treating network parameters or inputs as stochastic, requiring multiple forward passes during inference to estimate the posterior distribution [84] [36].

A prominent technique is Monte Carlo Dropout, which approximates Bayesian inference by performing multiple stochastic forward passes with dropout enabled at test time [37]. This method has been adapted for object detection, for instance, Kraus [64] applied MCDO to YOLOv3 to spatially visualize uncertainty, while Peng [88] utilized MCDO estimates to calibrate confidence scores. To effectively aggregate these stochastic predictions. BayesOD introduces a framework

for uncertainty estimation in object detection by replacing standard Non-Maximum Suppression with Bayesian inference. It treats multiple redundant anchor outputs as evidence to fuse rather than discard, thereby updating probability distributions for both object locations and categories to produce more reliable confidence estimates [45].

Deep Ensembles represent another robust sampling strategy, where multiple independently initialized neural networks are trained and their predictions combined. Ensembles have been shown to effectively capture epistemic uncertainty and are often considered the standard for uncertainty quality [65]. To prevent ensemble members from converging to similar solutions, Repulsive Ensembles encourage function space diversity, further improving performance [21] [107].

Additionally, Test-Time Data Augmentation generates multiple augmented versions of the input image at inference time. Methods such as those by Ayhan and Berens [2] and Wu et al. [124] estimate uncertainty by analyzing the variability of predictions across these augmented views.

Single-pass Methods

Single-pass methods aim to quantify uncertainty within a single forward pass, making them suitable for real-time applications. These methods typically modify the network architecture or loss function to predict distributional parameters directly.

A common approach to capture aleatoric uncertainty is to model the bounding box regression as a probability distribution rather than a deterministic value. He et al. [48] proposed Bounding Box Regression with Uncertainty, utilizing a KL-Divergence loss to learn the variance of bounding box coordinates. Similarly, Choi et al. [19] introduced Gaussian YOLOv3, which modifies the YOLO detection head to predict coordinate uncertainty as Gaussian distributions.

Li et al. [71] introduced Generalized Focal Loss, which models the location of bounding box edges as general probability distributions rather than deterministic points. This is achieved by dividing the prediction into discrete bins describing distances from the anchor, where the network predicts the probability of the edge falling into each bin.

Beyond regression uncertainty, other deterministic methods focus on Out-of-Distribution detection. Zolfi et al. [129] proposed YOLOOD, which leverages internal feature activations from YOLO's detection modules to compute an OOD score. Van Amersfoort et al. [1] introduced Deterministic Uncertainty Quantification, a method that detects OOD samples in a single forward pass by utilizing a Radial Basis Function to measure the distance between input features and learned class centroids.

Despite the growing number of methods, existing studies use dif-

ferent models, datasets, approximation techniques, and evaluation metrics, making a comprehensive comparison challenging. This work addresses this gap by systematically evaluating two popular sample-based uncertainty estimation techniques on a single stage object detection model and guide future research in uncertainty-aware object detection.

4 EXPERIMENTAL DESIGN

The experimental framework designed to quantify and evaluate uncertainty metrics for object detection is detailed in this chapter. It begins by highlighting the methodologies selected to capture distinct uncertainty metrics. Subsequently, the necessary modifications made to the standard YOLOv8 model are described to facilitate these uncertainty estimation techniques. The primary objective of this experimental design is to conduct a relative performance analysis of these methods. This approach is necessitated by the computational infeasibility of establishing a definitive ground truth for model uncertainty using conventional techniques like Markov Chain Monte Carlo, which are not scalable to deep neural networks of this magnitude.

4.1 UNCERTAINTY METRICS

In order to quantify uncertainty in object detection, it is first essential to establish a clear definition of the subject. This section categorizes uncertainty along two primary axes: the nature of the prediction task, regression versus classification and the fundamental source of the uncertainty itself. It then describes how to capture the metrics for these uncertainties.

4.1.1 Regression and Classification Uncertainty

In object detection, a distinction can be made between two primary types of predictive tasks: *regression* and *classification*.

The task of determining the corner coordinates of each bounding box is a regression problem. The objective is to predict continuous numerical values based on input data. The simplest form of this is linear regression. However, in deep learning, this continues value is predicted by the neural network.

Conversely, the classification task involves assigning a predefined label to each detected object. There are different approaches for finalizing the model's classification output. One common method is to apply the softmax function to the model's internal logit values, transforming them into a probability distribution over the class labels. The softmax function is defined in equation 4.1 as:

$$Q_j(\mathbf{x}) = \frac{e^{V_j(\mathbf{x})}}{\sum_k e^{V_k(\mathbf{x})}} \tag{4.1}$$

This function preserves the relative order of the input scores and provides a differentiable approximation of the winner-take-all mechanism by amplifying the most likely class prediction [10].

While the winner-take-all approach is suitable for many applications, it can be limiting in cases where multiple overlapping or hierarchically related labels are present (e.g., both dog and golden retriever). In such cases, it is necessary to employ multi-label classification. By default, YOLOv8 utilizes a multi-label classification framework and replaces the softmax function with the sigmoid activation function, as shown in equation 4.2.[1]

$$\text{Sigmoid}(x) = \frac{1}{1 + e^{-x}} \tag{4.2}$$

The sigmoid function independently maps each logit to a value in the range $(0, 1)$, allowing for multiple classes to be simultaneously active without enforcing a mutual exclusivity constraint. Final label assignments are then determined by applying a predefined threshold to these values [30].

This distinction introduces a challenge: object detection involves two different types of prediction tasks each requiring distinct metrics for uncertainty assessment. While regression uncertainty is concerned with continuous error margins, classification uncertainty is inherently discrete, whether a prediction is true or false and therefore requires different modeling strategies.

4.1.2 Aleatoric and Epistemic Uncertainty

The overall predictive uncertainty of a prediction as described in section 2.1 can be decomposed into two primary sources: *aleatoric* uncertainty, also known as data uncertainty and *epistemic* uncertainty, or model uncertainty [116].

In the context of Bayesian inference, aleatoric uncertainty is associated with the likelihood term $p(\mathcal{D} \mid \omega)$, as it captures the intrinsic noise present in the observed data. This type of uncertainty is considered irreducible because it originates from inherent stochasticity or measurement errors, which are independent of the model parameters [53]. Aleatoric uncertainty can be further classified into two types: *homoscedastic* uncertainty, which remains constant across different input

1 https://github.com/ultralytics/ultralytics/blob/440ff0d975d3a0fad5ad1d76e98250e611b3a896/ultralytics/nn/modules/head.py#L26-L212

conditions and *heteroscedastic* uncertainty, which varies with the input and may depend on specific data features [61].

Epistemic uncertainty instead reflects a lack of knowledge regarding the model parameters and is reducible through the acquisition of additional data. It is encoded in the posterior distribution $p(\omega \mid \mathcal{D})$ [53]. From a Bayesian perspective, this uncertainty quantifies the distribution of beliefs over the parameter space, conditioned on the observed data. Epistemic uncertainty is typically high in regions of the input space that are poorly represented or entirely Out-Of-Domain compared to the training dataset and diminishes as the model is exposed to more diverse and representative data [60].

Therefore, four distinct types of uncertainty metrics need to be captured: aleatoric and epistemic uncertainty for both regression and classification tasks.

4.1.3 Capturing Uncertainties

Having established the theoretical distinctions between the different forms of uncertainty, this section outlines the specific methodologies employed to derive quantitative scores for each.

Regression Uncertainty

For the regression task of bounding box prediction, aleatoric uncertainty is captured by augmenting the model's architecture with an additional prediction head. This *aleatoric head* is trained to directly output a metric associated with data uncertainty during a single forward pass, a technique also used in [57] and [49]. The greater the value predicted by this head, the higher the model's estimation of the inherent noise or stochasticity present in the input data. The specific implementation details of this architectural modification are discussed in Section 4.2.1.

An alternative approach exists within the native YOLOv8 architecture. The model internally employs a Distribution Focal Loss module for predicting the coordinates of the bounding box, as proposed in [71]. This module assigns probability scores to discrete bins that represent offsets from an anchor point. The logits from this module could, in principle, be utilized to derive a measure of aleatoric uncertainty. However, the dedicated aleatoric head approach was selected for this work due to its greater transferability to a wider range of architectures.

Epistemic uncertainty for regression is quantified by measuring the variance across multiple bounding box predictions for the same object, where each prediction is generated from a different model configuration. Given a set of T predicted bounding boxes, $\mathbf{B} = \{\mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(T)}\}$, where each box $\mathbf{b}^{(t)} = (x_1^{(t)}, y_1^{(t)}, x_2^{(t)}, y_2^{(t)})$ represents the corner coor-

dinates, the epistemic uncertainty $U_{\text{epist,reg}}$ is calculated as the mean variance across the four coordinates, as shown in equation 4.3:

$$U_{\text{epist,reg}} = \frac{1}{4}\sum_{j=1}^{4}\left(\frac{1}{T}\sum_{t=1}^{T}(b_j^{(t)} - \bar{b}_j)^2\right) \tag{4.3}$$

Here, $\bar{b}_j$ is the mean value for the j-th coordinate across all T predictions. This metric captures the disagreement among the ensemble of predictions, reflecting the model's uncertainty about its own parameters.

Classification Uncertainty

To quantify classification uncertainty, the total predictive uncertainty is first calculated. This is achieved by computing the entropy of the averaged predictive distribution, which is obtained by averaging the softmax classification probabilities over multiple stochastic forward passes for each detection [16] [36]. The predictive uncertainty is given by:

$$U_{\text{pred}} = -\sum_{c}\bar{p}_c \log(\bar{p}_c) \tag{4.4}$$

where $\bar{p}_c$ represents the mean probability for class c across the T forward passes.

To decompose the total predictive uncertainty from Equation 4.4 into its aleatoric and epistemic components, two additional metrics are calculated. Aleatoric uncertainty is captured by the expected entropy of the predictions. This is estimated by averaging the entropy of each individual softmax output across the multiple forward passes, as described in [82] and [16]. This metric, denoted here as H_{softmax}, is calculated as follows:

$$H_{\text{softmax}} = -\frac{1}{T}\sum_{t=1}^{T}\sum_{c} p_c^{(t)} \log(p_c^{(t)}) \tag{4.5}$$

where $p_c^{(t)}$ is the probability of class c in the t-th forward pass.

Since total predictive uncertainty is the sum of its aleatoric and epistemic components, the epistemic uncertainty can be isolated. This is equivalent to the mutual information (MI) between the predictions and the model parameters. It is calculated as the difference between the total predictive uncertainty and the aleatoric uncertainty [16]:

$$MI = U_{\text{pred}} - H_{\text{softmax}} \tag{4.6}$$

Through these distinct methodologies, quantitative scores for all four required uncertainty types are obtained for subsequent analysis.

4.2 YOLO MODEL MODIFICATIONS

To support the methodologies described, the YOLOv8 architecture had to be modified to deliver the required output, particularly additional outputs from the aleatoric head or the reformatting of existing outputs to match the format expected by the equations. Furthermore, the training process was adapted to support the Deep Ensembles and MCDO concepts.

Initially, general modifications to the networks framework were necessary for both the Deep Ensembles and MCDO techniques, while specific adjustments for each technique were also implemented [58].

Capturing the raw logits from the classification branch of the detection head[2] was the first step. With this modification, these logits can be manually converted into the softmax format, which is required for calculating metrics such as predictive uncertainty and softmax entropy. This logit output is returned additionally, ensuring that the model can still be trained using its default pipeline, which employs a final sigmoid activation. To accommodate these new outputs, the *results.boxes* class[3] was modified to support the return of additional parameters, including the detection logits before sigmoid activation and the aleatoric head predictions. Consequently, the Non-Maximum Suppression[4] (NMS) function also required alteration to correctly handle the additional parameters within the modified *results.boxes* class. The NMS function was further adjusted to retain all per-class probabilities, deviating from its standard behavior of discarding values below a set threshold. This specific modification was based on a solution identified in a GitHub issue.[5]

4.2.1 Aleatoric Head

An aleatoric head was added to the network to directly predict the aleatoric uncertainty associated with the bounding box coordinates. This head predicts the log-variances ($\log \sigma^2$) for each of the four bounding box parameters (x, y, w, h). The corresponding mean values for the bounding box regression are taken directly from the model's existing regression branch, while the new aleatoric head is trained specifically to predict the variances.

2 https://github.com/ultralytics/ultralytics/blob/440ff0d975d3a0fad5ad1d76e98250e611b3a896/ultralytics/nn/modules/head.py#L26-L212

3 https://github.com/ultralytics/ultralytics/blob/440ff0d975d3a0fad5ad1d76e98250e611b3a896/ultralytics/engine/results.py#L177-L813

4 https://github.com/ultralytics/ultralytics/blob/440ff0d975d3a0fad5ad1d76e98250e611b3a896/ultralytics/utils/nms.py#L13-L166

5 https://github.com/ultralytics/ultralytics/issues/2863#issuecomment-2259399447

Optimization of the aleatoric head is performed using a dedicated gradient based on the heteroscedastic loss, as defined in Equation 4.7. This gradient calculation is detached from the other prediction head branches, ensuring that the original classification and regression branches can still be trained unaffected by the new loss component. The heteroscedastic loss function is based on the work of Alex Kendall and Yarin Gal[61] and is defined as:

$$\mathcal{L}_{\text{HSL}} = 0.5 \times \exp(-s_i) \times (y_i - \hat{y}_i)^2 + 0.5 \times s_i \tag{4.7}$$

Here, y_i is the target value and $\hat{y}_i$ is the model's mean prediction. The model is trained to predict the log variance, $s_i := \log \hat{\sigma}_i^2$. This loss function consists of two primary components: a residual regression term weighted by the predicted uncertainty and an uncertainty regularization term. The model learns the variance $\hat{\sigma}_i^2$ implicitly from the regression task without requiring explicit uncertainty labels. The second term, $0.5 \times s_i$, acts as a regularizer, preventing the network from predicting infinite uncertainty and therefore zero loss for all data points. Predicting the log variance s_i rather than the variance σ_i^2 directly provides greater numerical stability, as the loss calculation avoids potential division by zero. Furthermore, the exponential mapping $\exp(s_i)$ ensures the predicted variance is always positive [61].

Integrating this new loss component introduces additional parameters, which slows down the training convergence. The optimal magnitude of the heteroscedastic loss was therefore investigated. Using the default magnitude (a factor of 1.0) forces the model to heavily optimize for the aleatoric head parameters, as this new loss component strongly dominates the other losses used in the model. Consequently, different scaling factors for $\mathcal{L}_{\text{HSL}}$ were tested and the results are presented in Figure 4.1. In principle, a lower magnitude for the heteroscedastic loss results in faster model convergence and a higher final mAP. When inspecting the individual loss components, it becomes visible that choosing a factor greater than 0.1 causes the heteroscedastic loss to dominate the overall loss associated with bounding box regression. By logging the default bounding box loss from the mean prediction branch and the variance loss from the aleatoric head, a scaling factor of 0.1 was found to satisfy a condition where each loss component has a similar influence on the total loss. An alternative approach could involve increasing the influence of the heteroscedastic loss during training using an annealing schedule, similar to KL annealing [9]. This would allow the model to first learn the primary regression and classification tasks before slowly incorporating the variance predictions.

In the final implementation, the four predicted variances (for x, y, w, h) from the aleatoric head are averaged to produce a single scalar variance value for each detected bounding box.

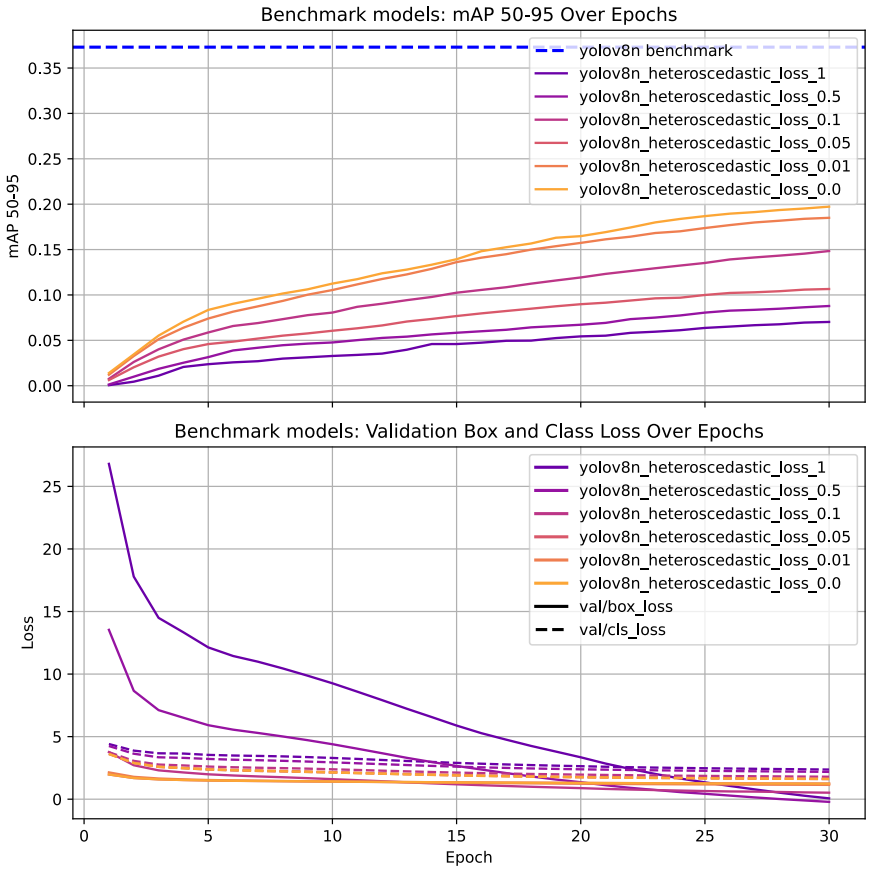


Figure 4.1: mAP scores and loss components during training of the YOLOv8n model with different magnitudes for the heteroscedastic loss for the aleatoric head. A higher magnitude for the heteroscedastic loss correlates with a lower mAP.

4.2.2 Deep Ensembles

To train multiple YOLO models for the Deep Ensemble method, the guidelines from the original proposing paper were followed [65]. This approach builds on the assumption that small differences in weight initialization, combined with random shuffling of the training data for each model, are sufficient to create models that are diverse in their respective weight spaces. This diversity is further encouraged by the stochastic nature of neural network training, which arises from the optimization algorithm itself and non-deterministic operations on modern hardware like GPUs [8].

For weight initialization, the default PyTorch implementation was utilized, along with setting an individual random seed for each ensemble member. Another implementation detail was to avoid loading a pretrained model, which the Ultralytics library does by default. Instead, a model configuration *.yaml* file, which only defines the model's architecture, was used. This forced the weights for each model to be

initialized randomly. The specific *.yaml* file for the YOLOv8 model architecture is available within the Ultralytics library[6].

Regarding the shuffling of training batches, the dataloader function found within the Ultralytics *data/build.py* script[7] required adjustment. The original library implementation contained a hardcoded seed for the shuffling mechanism, which prevented the intended randomness of the training batches across different ensemble models.

In general, the implementation of the Deep Ensemble training strategy was relatively straightforward, requiring only these minor adjustments to the standard training pipeline.

4.2.3 Monte Carlo Dropout

To modify the YOLOv8 model to support MCDO, several modifications were implemented. The model's configuration *.yaml* file was altered to accept an additional parameter for specified layers, namely the dropout probability. In accordance with the original MCDO paper, dropout layers were placed after the activation function of a given layer [37].

Within the Ultralytics library, the classes corresponding to the convolutional layer[8], the C2F modules[9] and the bottleneck layers[10] were modified to accept this new dropout parameter. The dropout functionality was implemented using the standard PyTorch *nn.Dropout* function [111]. While *nn.Dropout2D* exists to drop entire channels for regularization in convolutional networks, the *nn.Dropout* method was chosen [112]. This decision was based on the premise that dropping individual neurons, rather than entire channels, more closely aligns with the concept of sampling from a distribution over the network's weights. A key requirement for MCDO is that the dropout layers must be active during inference. Therefore, all affected layers are explicitly set to training mode before each forward pass during inference.

In existing literature, there is no definitive consensus on the optimal placement of dropout layers within the network architecture for MCDO. Previous experiments on MCDO in CNNs have placed

6 https://github.com/ultralytics/ultralytics/blob/440ff0d975d3a0fad5ad1d76e98250e611b3a896/ultralytics/cfg/models/v8/yolov8.yaml#L1-L49

7 https://github.com/ultralytics/ultralytics/blob/440ff0d975d3a0fad5ad1d76e98250e611b3a896/ultralytics/data/build.py#L325

8 https://github.com/ultralytics/ultralytics/blob/440ff0d975d3a0fad5ad1d76e98250e611b3a896/ultralytics/nn/modules/conv.py#L39-L89

9 https://github.com/ultralytics/ultralytics/blob/440ff0d975d3a0fad5ad1d76e98250e611b3a896/ultralytics/nn/modules/block.py#L283-L314

10 https://github.com/ultralytics/ultralytics/blob/440ff0d975d3a0fad5ad1d76e98250e611b3a896/ultralytics/nn/modules/block.py#L452-L476

dropout for example immediately before the last fully connected layer [37], right before the detection module [23], or inside the detection head itself [3].

Given this disagreement in the literature, this thesis investigates and compares different dropout locations to evaluate their impact on uncertainty estimation. Suitable locations within the YOLOv8 network include the convolutional layers in the model's backbone, in the neck and in the detection head. In the neck and the head, the convolutional layers are part of the C2F modules, which themselves consist of convolutional layers and bottleneck layers, which are also constructed from convolutional layers.

For the uncertainty evaluation, three distinct configurations are tested. Dropout is applied to the convolutional layers within: (1) the *backbone*, (2) the textitneck, specifically right before the detect module, or (3) the detection textithead of the model's architecture.

4.3 CLUSTERING ALGORITHM

The uncertainty estimation metrics described in Section 4.1.3 expect a set of predictions corresponding to the same object instance. As an image may contain multiple objects and each ensemble member or MCDO forward pass generates its own set of detections, it is necessary to group these disparate predictions. A clustering algorithm was implemented for this purpose, where each resulting cluster collects all predictions referring to the same unique object within the image. The flowchart describing the algorithm is shown in detail in Figure **??**. The Flowchart was created with Mermaid [108].

The algorithm functions by comparing all predictions against each other using the Intersection over Union metric. If the IoU between two bounding boxes is greater than or equal to a specified threshold, they are assigned to the same cluster. This process is implemented using a union-find approach, which makes the algorithm order-invariant, where the final clusters are not dependent on the order of the predictions being processed. The IoU threshold itself is a tunable hyperparameter and its influence on uncertainty estimation is investigated.

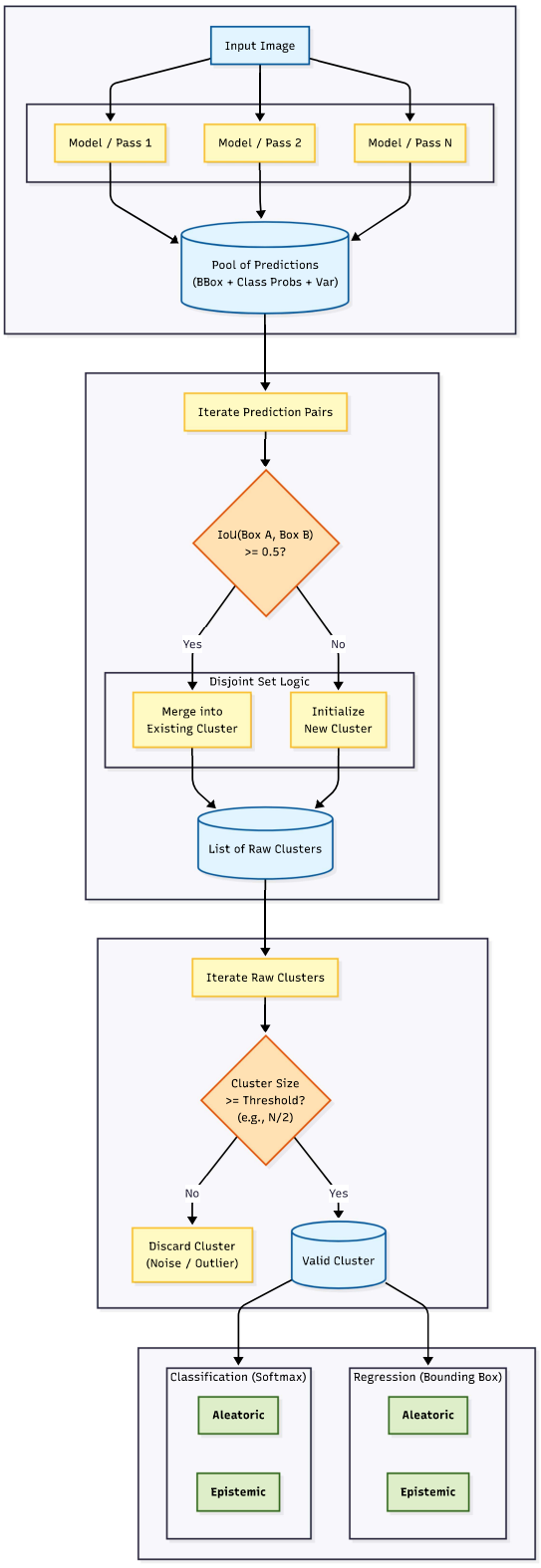


Figure 4.2: The clustering algorithm flowchart. Input images undergo multiple inference passes to create a pool of predictions. These are iteratively compared via IoU metrics and merged into raw clusters using disjoint set logic to ensure order-invariance. Finally, clusters meeting the consensus size threshold are retained for uncertainty quantification in classification and regression tasks.

Furthermore a consensus value ranging from 0.0 to 1.0 is introduced. This value defines the minimum fraction of ensemble members or MCDO based forward passes that must agree on an object's presence for a cluster to be considered valid. This acts as a filter to remove edge detections that are not consistently identified. This consensus threshold is also treated as a tunable parameter for investigation.

4.4 CORRUPTED COCO DATASET

The Microsoft Common Objects in Context (COCO) dataset is a large-scale annotated collection of images designed to support object detection, segmentation and captioning tasks. It consists of over 330,000 images, more than 200,000 of which are labeled, containing a total of 1.5 million object instances. The dataset covers 80 object categories, including common items such as people, vehicles, household objects, animals and food. Images in COCO are typically complex and depict everyday scenes with multiple objects, varying context and substantial occlusion, making the dataset particularly challenging and representative of real-world settings [74].

For experimental analysis, the COCO validation dataset images were corrupted in a controlled manner using a custom Python script [56]. The script introduces multiple types of corruptions such as Gaussian noise, shot noise, impulse noise, defocus blur, motion blur, zoom blur, snow, fog, contrast adjustment, elastic transformations, pixelation, JPEG compression, speckle noise and spatter across five severity levels. The different corruption types at severity 2 for a COCO example image is visualized in Figure 4.3.

The corrupted COCO dataset is used for evaluation of model performance under degraded visual conditions and is used here to investigate how uncertainty metrics from different uncertainty estimation methods vary in response to different corruption types and severities.

Given the computational expense and data volume associated with evaluating all corruption types across five severity levels, a more focused analytical approach was adopted. Therefore, the corruption types were first categorized into four distinct groups based on the nature of the visual degradation they introduce. From each group, a single representative corruption was selected for the uncertainty analysis.

The four corruption groups are defined as follows:

1. Noise: Corruptions that introduce random, pixel-level variations. This group includes Gaussian Noise, Shot Noise, Impulse Noise and Speckle Noise.

2. Blur: Corruptions that reduce high-frequency spatial details, simulating optical or motion-related distortions. This category

Figure 4.3: The various corruption types at severity level 2 from the Corrupted COCO dataset.

comprises Gaussian Blur, Defocus Blur, Motion Blur, Zoom Blur and Glass Blur.

3. Weather: Corruptions that simulate environmental conditions. This group contains Fog, Frost, Snow, Spatter, Brightness, Contrast and Saturate.

4. Others: Corruptions that introduce artifacts from digital processing or spatial distortions. This group includes Pixelate, JPEG Compression and Elastic Transformation.

For the evaluation, one corruption type was selected from each group. The selection was guided by the objective of maximizing the diversity of corruptions, ensuring that the chosen subset represents fundamentally different types of visual degradation. The selected corruptions are:

1. *Gaussian Noise* from the Noise group

2. *Motion Blur* from the Blur group

3. *Brightness* from the Weather group

4. *Pixelate* from the Others group

With the methodology implemented and the model adjustments finalized, the results were collected, which are presented in the subsequent chapter.

4.5 UNCERTAINTY EVALUATION PIPELINE

The uncertainty metrics were computed using a two-stage procedure. First, a producer script performed inference on the corrupted validation dataset for a selected corruption. The outputs from each inference run, whether from an ensemble member or an MCDO forward pass, were serialized into JSON files containing image IDs, bounding box coordinates, class confidences, and variance predictions. Next, the clustering algorithm described in Section 4.3 grouped these predictions to identify unique objects. Finally, a consumer script calculated the four aleatoric and epistemic uncertainty metrics for each object, averaging the results across the full dataset. Key experimental variables analyzed included the *corruption type*, *ensemble size/ number of forward passes*, *clustering thresholds* (IoU and consensus), and the specific MCDO parameters *dropout probability* and *dropout location*.

5 RESULTS

This chapter presents the results obtained from the experimental design for capturing uncertainty metrics using Deep Ensembles and Monte Carlo Dropout. Subsequently, a comparison between the two approaches is drawn.

Inference was performed across the four corruption types selected in Section 4.4 using a custom YOLOv8n model, modified to include an aleatoric head as detailed in Section 4.2. To ensure consistency, the model was trained for 100 epochs, matching the default training settings used for the Deep Ensemble and MCDO models. For each corruption type and severity the average detections per image (*Avg*) and average bounding box diameter (*Diam*) were recorded these results are summarized in Table 5.1.

Table 5.1: Mean detection counts (Avg) and average bounding box diameter (Diam) of the benchmark model across selected image corruptions and severity levels. Brightness and Gaussian noise are highlighted.

	Severity 0		Severity 1		Severity 2		Severity 3		Severity 4		Severity 5	
Corruption	Avg	Diam	Avg	Diam	Avg	Diam	Avg	Diam	Avg	Diam	Avg	Diam
Brightness	4.88	213.51	4.52	217.83	4.27	222.05	4.03	226.13	3.72	232.00	3.38	237.89
Gaussian Noise	4.88	213.51	2.90	236.04	1.99	249.24	1.08	272.09	0.45	305.41	0.08	406.02
Motion Blur	4.88	213.51	3.62	251.51	2.65	291.40	1.69	362.15	1.04	448.89	0.73	506.04
Pixelate	4.88	213.51	4.22	224.26	4.12	225.71	2.76	235.31	1.19	250.60	0.77	234.47

The data indicates that for default validation images without corruption, the model performs consistently, identifying an average of 4.88 detections per image. However, as corruption severity increases, the model's behavior diverges based on the corruption type. The model detection capabilities displays relatively strong robustness against brightness corruption, capturing an average of 3.38 objects per image even at the highest severity, which equals approximately 70% of the original objects.

Conversely, performance degrades significantly under motion blur and pixelate corruptions, capturing only 0.73 to 0.77 objects per image at Severity 5. This corresponds to a recognition rate of approximately 15%. Within this pairing, pixelate corruption appears more stable at moderate severities, though detection counts drop sharply at higher levels. Finally, gaussian noise exerts the most severe influence. At the highest severity, the model averages just 0.08 detections per image, representing a retention rate of only 1.6%.

Furthermore, it is observed that the average bounding box diameter increases with corruption severity. For brightness and pixelate corruptions, this increase is marginal (approximately 10%). However, the impact of Gaussian noise and motion blur is more severe, resulting in average detected diameters up to twice the size of the uncorrupted baseline.

Based on these findings, the subsequent analysis of uncertainty metrics focuses on two distinct cases: the corruption with the highest detection count (brightness) and the corruption with the lowest detection count (gaussian noise), which have been highlighted in the table. This selection enables a comparison between a stable scenario, where survivorship bias is as minimal as possible, and a high degradation scenario, where survivorship bias is expected to be severe.

Model training was performed on a dedicated compute cluster equipped with NVIDIA RTX 2080 Ti and A30 GPUs. All experiments were implemented using the Ultralytics library version 8.3.49. [1]

5.1 DEEP ENSEMBLES

The results collected using the Deep Ensemble method are detailed in this section. First, the training details for the models are presented. Next, the characteristic metrics and plots for uncertainty evaluation are demonstrated and explained.

Deep Ensembles are a straightforward approach to approximating Bayesian inference. The resulting metrics and trends from this method are therefore treated as a reference point against which the MCDO approach, which is more sensitive to factors such as the placement of the dropout layers or the applied dropout rate, is compared.

5.1.1 Functional Verification

Following the experimental design outlined in Chapter 4, an ensemble comprising 20 members was trained for 100 epochs. The ensemble size was selected to maximize diversity within the available computational constraints. All YOLOv8n models were trained from scratch using random weight initialization and the previously described modified dataloader shuffling. Default training augmentations were employed, as hyperparameter optimization fell outside the scope of this work. Accordingly, the training configuration utilized the default automatic optimizer, an adaptive learning rate, and a weight decay of 0.005 for regularization. Figure 5.1 depicts the training progression as measured by the mAP score. As anticipated, all ensemble members exhibited similar convergence trajectories. Notably, the integration of the aleatoric

1 https://pypi.org/project/ultralytics/8.3.49/

head resulted in a lower mAP score compared to the standard Ultralytics reference baseline [114].

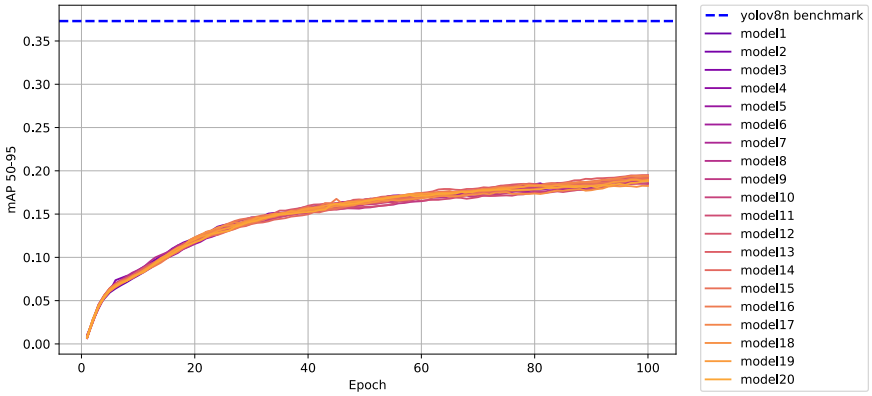


Figure 5.1: Training progression of mAP scores for the YOLOv8n ensemble members. The official Ultralytics benchmark is included as a reference baseline.

The Deep Ensemble method aims to promote diversity in the weight space between models. Figure 5.2 illustrates this by plotting the Kernel Density Estimation of the standard deviations calculated for each individual weight parameter across the ensemble. For representation purposes, this analysis focuses on the first convolutional layer of the main bounding box prediction head branch.

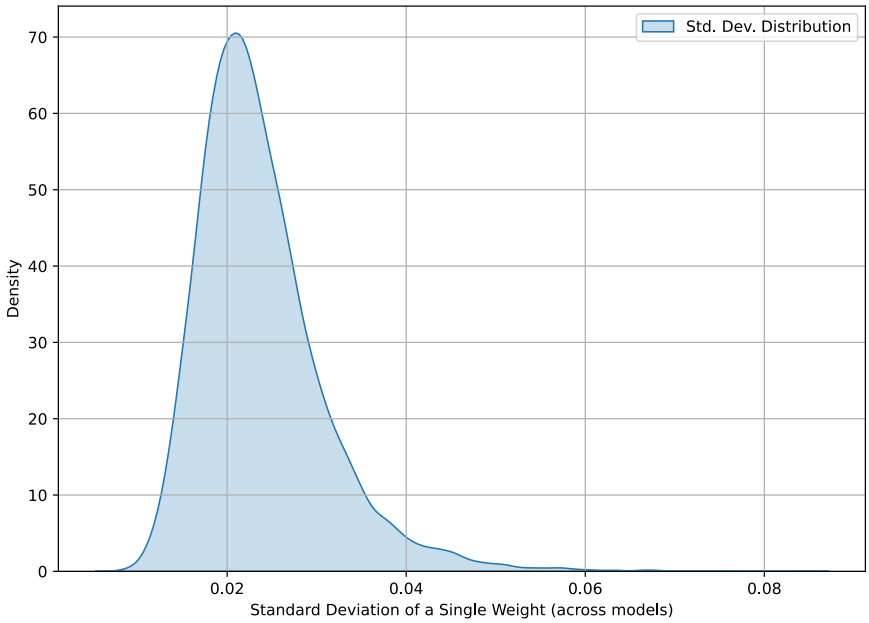


Figure 5.2: Kernel Density Estimation (KDE) of the per-parameter standard deviations for the first convolutional layer of the main bounding box prediction head branch. The distribution aggregates the standard deviation of each individual weight calculated across the entire ensemble of YOLO models.

The visualization highlights the diversity of the learned representations. If all models had converged to the same parameters, we would see a concentration near zero. Instead, there is a clear spread. Given a maximum weight magnitude of 0.33, the ensemble members vary by about 10% of the parameter's total range.

5.1.2 Evaluation of Parameter Tuning

To allow for a concise comparison between the Deep Ensemble method and the MCDO method, the dimensionality of the hyperparameter space must be reduced. Specifically, representative plots for uncertainty metrics require the independent variables to be narrowed down to a set of fixed parameters that can be verified and subsequently applied to the MCDO evaluation. Three primary variables influence the final uncertainty metrics: the *ensemble size* and the two hyperparameters governing the clustering algorithm, *IoU threshold* and *consensus threshold*.

5.1.2.1 Ensemble Size

To quantify the influence of ensemble size on the uncertainty metrics, the four metrics were analyzed under baseline conditions with no data corruption across varying ensemble sizes. During this analysis, the IoU

and consensus thresholds were iterated to observe their interaction with the ensemble size.

The objective was to fix the ensemble size at a value where the metrics demonstrate convergence. Figure 5.3 illustrates the relationship between ensemble size and the IoU hyperparameter. The consensus threshold was disabled for this experiment. It can be observed that the majority of metrics begin to stabilize at an ensemble size of approximately 15 members. Although absolute convergence is not fully achieved, the variance diminishes significantly by 20 members.

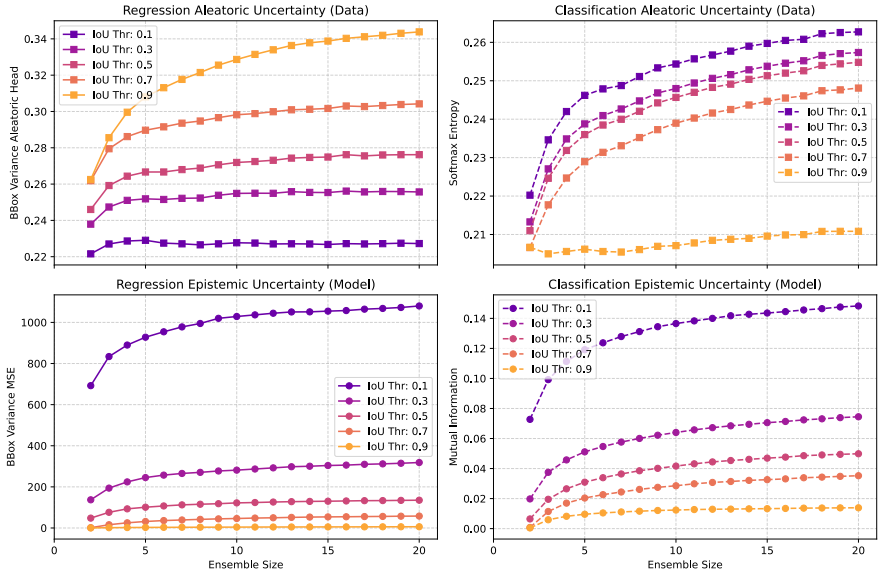


Figure 5.3: Convergence of the four uncertainty metrics as a function of ensemble size plotted against different IoU threshold values.

A similar trend is evident in Figure 5.4, where uncertainty metrics are plotted against ensemble size while varying the consensus threshold. The IoU threshold was set to 50 % for this evaluation. Here, the convergence behavior is more discrete. As the ensemble size increases, the clustering algorithm crosses critical thresholds like permitting valid clusters despite a lack of unanimous agreement among members, causing significant step-changes in the calculated metrics. While these metrics largely stabilize after an ensemble size of 11, the data confirms that larger ensemble sizes approximate the point of convergence more closely.

Based on these observations, a final ensemble size of 20 was selected.

5.1.2.2 *IoU and Consensus Thresholds*

Following the determination of the ensemble size, the two remaining hyperparameters of the clustering algorithm, *IoU threshold* and *consensus threshold*, must be collapsed to fixed values. With the ensemble

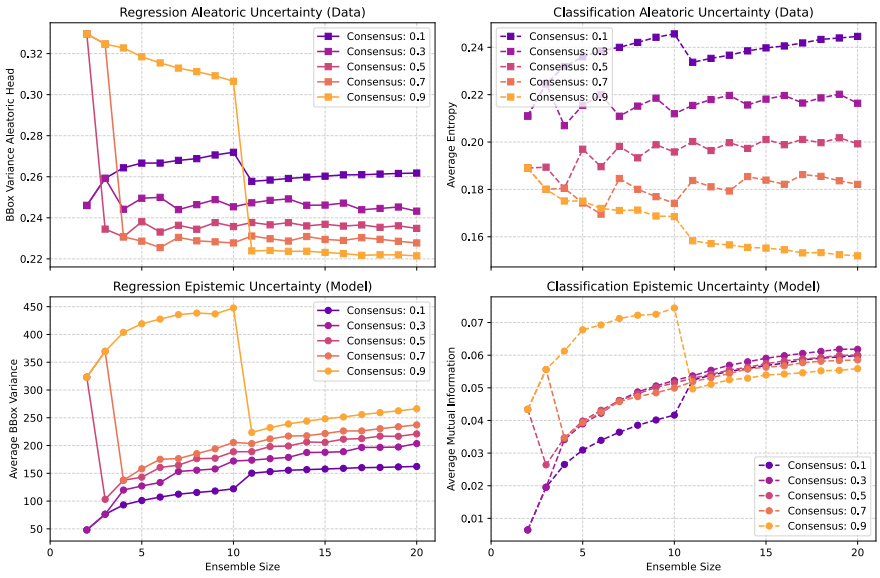


Figure 5.4: Convergence of the four uncertainty metrics as a function of ensemble size plotted against different consensus threshold values.

size fixed at 20, the objective of this phase is to monitor the stability of uncertainty metrics under specific image corruptions and severities. The goal is to identify parameter values that yield representative results while minimizing outliers.

To evaluate these parameters, uncertainty metrics were analyzed relative to corruption severity. The brightness corruption was selected as the primary variable for this tuning process, as it demonstrated the highest stability and detection retention rates, even at high severity levels. The effect of the IoU parameter is visualized in Figure 5.5. The corresponding analysis for gaussian noise corruption is provided in Appendix a.2.1.

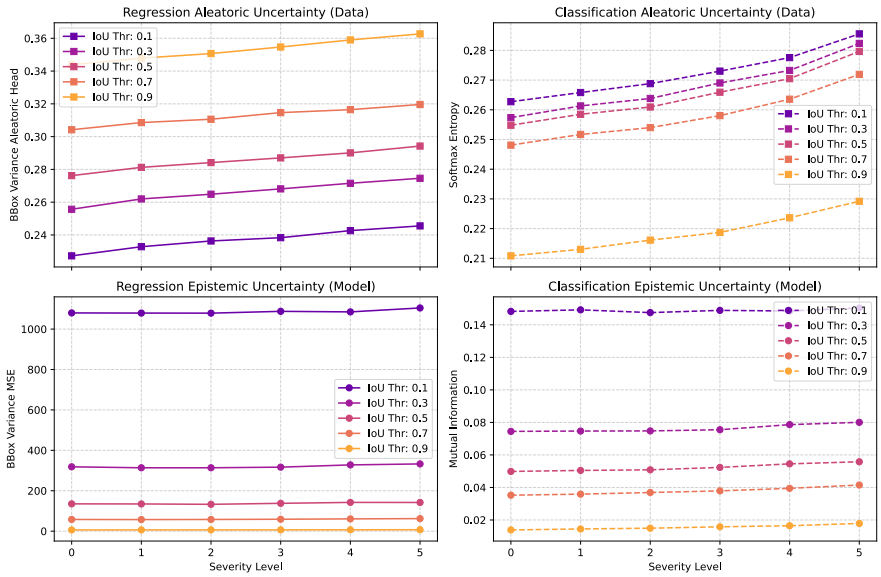


Figure 5.5: Impact of varying IoU thresholds on uncertainty metrics across increasing brightness corruption severities (Ensemble Size = 20, Consensus = 0.0).

The results indicate that the IoU threshold has a marginal impact on aleatoric uncertainty. This behavior is expected, as aleatoric uncertainty is computed per individual prediction, meaning the clustering algorithm only influences this metric indirectly by determining the number of accepted predictions. Conversely, epistemic uncertainty, which is derived from the divergence of predictions between different models, exhibits greater sensitivity to IoU variations. To mitigate outliers, an IoU threshold of 0.5 was selected. This value follows the trends observed in the 0.3 and 0.7 thresholds while avoiding extreme offsets. For this evaluation, the consensus parameter was again disabled.

Finally, the consensus hyperparameter was analyzed. Figure 5.6 illustrates the metrics as a function of corruption severity and consensus values. The observed deviations here are notably smaller than those seen when varying the IoU parameter, suggesting a lower sensitivity. A consensus value of 0.5 was selected for further evaluation, as it exhibiting the lowest variance relative to neighboring plots.

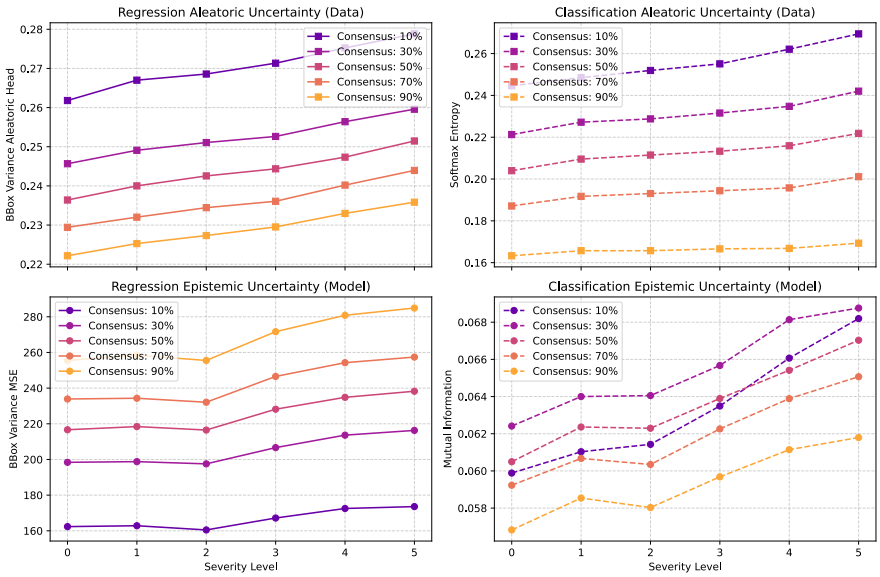


Figure 5.6: Impact of varying consensus thresholds on uncertainty metrics across increasing brightness corruption severities (Ensemble Size = 20, IoU = 0.5).

With the parameter tuning concluded, the final configuration for the evaluation is established as follows:

- Ensemble Size: 20
- IoU Threshold: 0.5
- Consensus Threshold: 0.5

5.1.3 Deep Ensemble Evaluation

The final evaluation of the Deep Ensemble method is conducted using the configuration established in the previous section. The following analysis examines the evolution of uncertainty metrics as a function of increasing corruption severity, focusing specifically on brightness and gaussian noise corruptions.

Figure 5.7 illustrates the trends of the uncertainty metrics under the brightness and gaussian noise corruption. For the brightness corruption the metrics follow the expected behavior: as corruption severity increases, the uncertainty metrics rise overall by approximately 7% to 10% depending on the metric. It is important to note that the brightness corruption represents a high-stability scenario even at the maximum severity level, the majority of objects are successfully detected by the model.

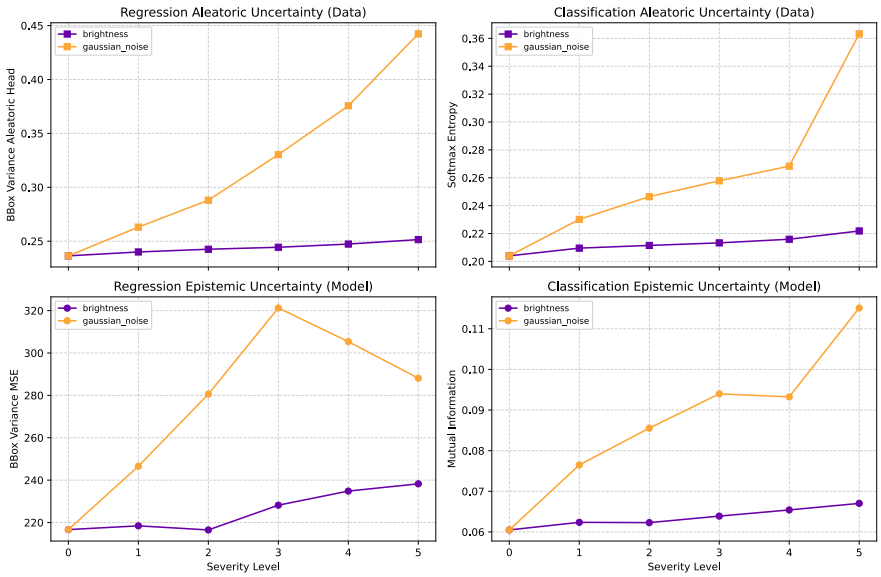


Figure 5.7: Comparative analysis of uncertainty metrics under increasing corruption severity for brightness and gaussian noise corruption for deep ensembles (Ensemble Size = 20, IoU = 0.5, Consensus = 0.5)

In contrast, the gaussian noise corruption, shown shows a different behavioral pattern. The aleatoric uncertainty for both regression and classification exhibits a sharp increase of approximately 45%. However, the epistemic uncertainty diverges: while the classification metric follows a similar upward trajectory of a $\approx 45\%$ increase, the epistemic regression uncertainty peaks at severity level 3 before declining.

This anomaly is likely attributable to survivorship bias. At severity level 4, the object detection rate drops to only 10%. Furthermore, it was observed that the average bounding box diameter of the remaining detected objects increases with corruption severity. This suggests that at high noise levels, the model fails to detect smaller, more difficult objects. This distributional shift and its implications are discussed chapter 6.

Representative plots for the additional corruption types motion blur and pixelate, are provided in Appendix a.2.2.

5.2 MONTE CARLO DROPOUT

Following the evaluation of the Deep Ensemble method, this section details the implementation and results of the MCDO approach. The structure mirrors the previous analysis: first, the detection capabilities are verified to ensure the selected corruption types remain valid. Second, the training performance is validated. Finally, the hyperpa-

rameter reduction is verified to ensure the parameters selected for Deep Ensembles, specifically if the number of forward passes and clustering thresholds remain applicable to the MCDO.

Unlike Deep Ensembles, the MCDO implementation introduces architectural variations. The evaluation covers three distinct dropout injection points: the *backbone*, the *neck*, and the *detection head*. Each configuration is assessed using three dropout probabilities: 0.1, 0.3, and 0.5.

To justify the focus on brightness and gaussian noise corruption for the uncertainty analysis, the average detection counts must first be verified for the MCDO models. Figure 5.8 displays the average detection count across corruption severities for the three dropout locations. For clarity, only the medium dropout rate ($p = 0.3$) is visualized here, as the trends remain consistent across other rates (see Appendix a.3.1 for full data).

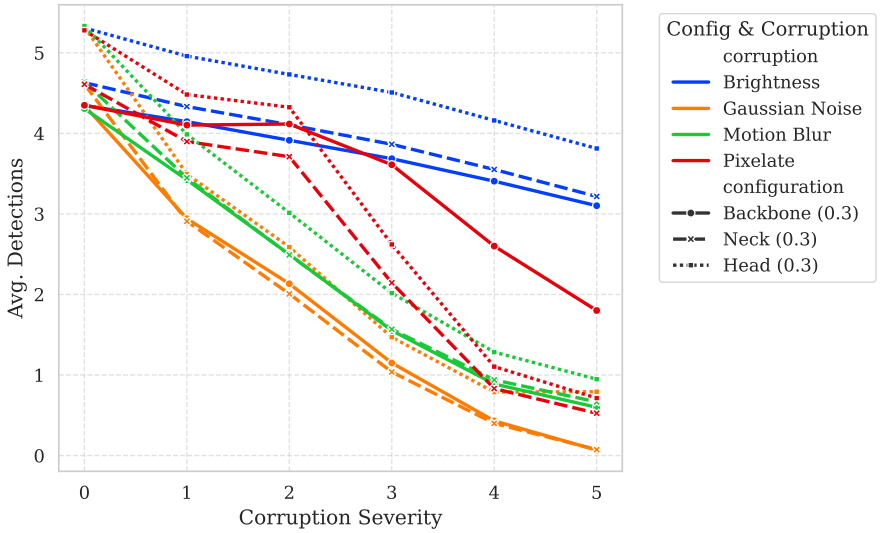


Figure 5.8: Average detection count per image across four corruption types for MCDO models with a dropout rate of 0.3. Comparison distinguishes between dropout applied to the backbone, neck and head.

The data indicates that MCDO models exhibit detection behaviors similar to the baseline. A hierarchy in performance is observable: models with dropout in the detection head capture the highest number of objects, followed by the neck, with the backbone dropout yielding the lowest detection counts. Generally, an increased dropout magnitude correlates with a slight reduction in detections.

However, a notable exception occurs with head dropout under brightness corruption. At the highest corruption severity, the head dropout model with a rate of 0.5 captures significantly more objects (+15%) compared to the rate of 0.1. Furthermore, with an average of 4.17 detections per image, this configuration outperforms the baseline

model with 3.38 detections, suggesting that aggressive dropout in the detection head acts as an effective regularizer, improving robustness against intense corruptions. Based on these results, the selection of brightness and gaussian noise as the bounding cases for stability and degradation remains valid.

5.2.1 Functional Verification

The training of individual MCDO models followed the guidelines introduced in Section 4.2. Similar to the Deep Ensemble members, models were trained from scratch with random weight initialization, incorporating dropout layers at the specified architectural locations. The training hyperparameters remained consistent with the previous experiments to ensure comparability.

Performance validation on the uncorrupted dataset highlights distinct trends regarding dropout placement and magnitude. As illustrated in Figure 5.9, the *head* dropout configuration yields the best performance, while *backbone* dropout results in the lowest mAP scores. Additionally, a clear inverse relationship is observed where higher dropout rates lead to decreased mAP metrics as models converge slower.

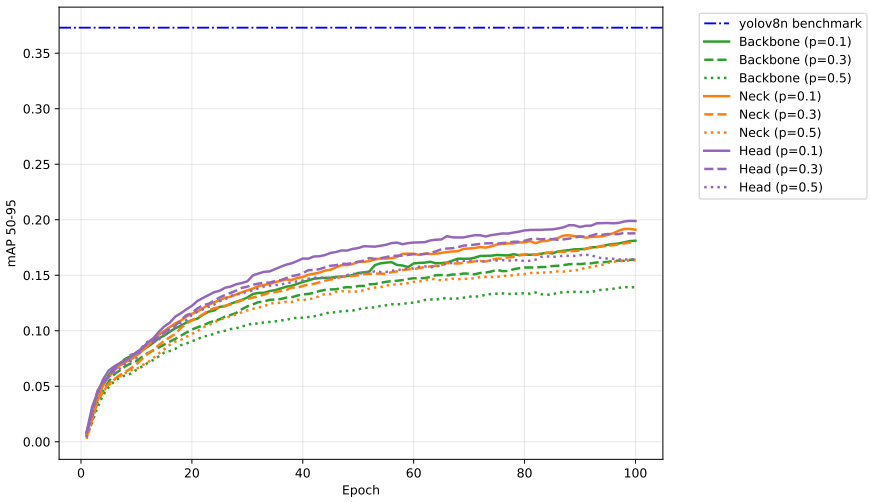


Figure 5.9: Comparison of mAP scores during training across different dropout locations and probabilities on the validation dataset.

5.2.2 Evaluation of Parameter Tuning

To facilitate a direct comparison between Deep Ensembles and MCDO, it is preferable to maintain consistent values for the *ensemble size* (in this case the number of forward passes), *IoU threshold*, and *consensus threshold*. Therefore, this section validates that the values derived in the Deep Ensemble analysis remain robust for the MCDO method.

Given the size of the possible configuration space, the following validation utilizes the best-performing architectural configuration: Head Dropout with a medium rate of 0.3, evaluated under brightness corruption. Plots illustrating other dropout rates and Gaussian noise corruption are provided in the appendix and will be mentioned.

5.2.2.1 Number of Forward Passes

The assumption that 20 forward passes provide a stable approximation of the uncertainty metrics is tested in Figure 5.10. This plot illustrates the change of uncertainty metrics as the number of forward passes increases, plotted against varying IoU thresholds.

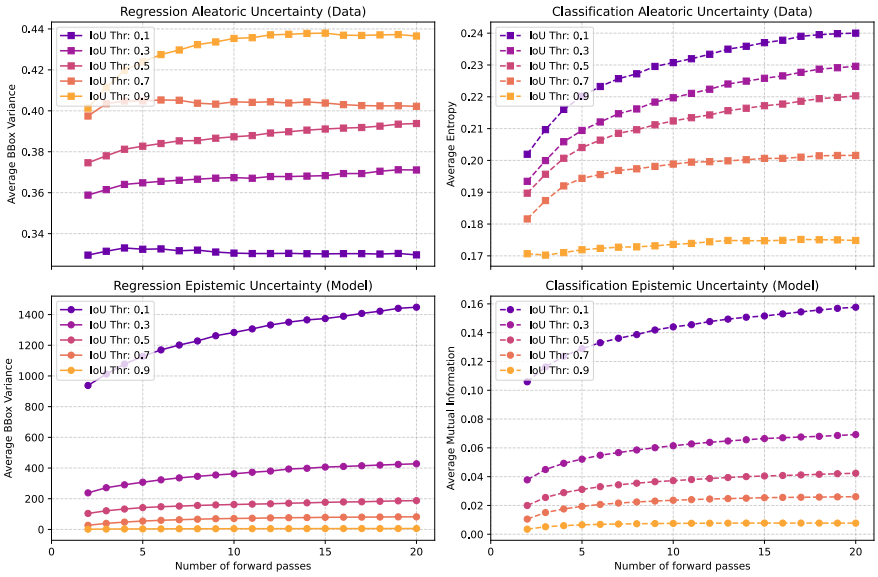


Figure 5.10: Convergence of uncertainty metrics as a function of the number of forward passes, plotted against different IoU threshold values (Head Dropout, p = 0.3, Consenus = 0.0).

The metrics exhibit convergence behavior similar to that of Deep Ensembles. When analyzing the interaction with the consensus threshold in Figure 5.11, the convergence appears slightly slower than in the Deep Ensemble case. However, as the metrics approach stability and the variance decreases significantly, the limit of 20 forward passes is choosen as an upper bound for this study.

Additional validation plots regarding the number of forward passes for gaussian noise and different dropout rates can be found in Appendix a.3.2. The overarching conclusion remains that maximizing the amount of passes yields the most stable results, as convergence is not fully reached.

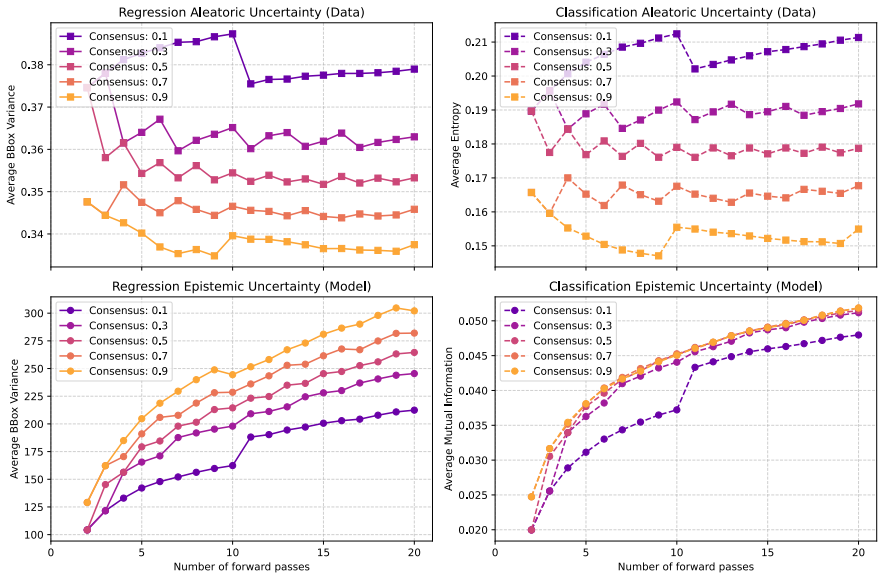


Figure 5.11: Convergence of uncertainty metrics as a function of the number of forward passes, plotted against different Consensus threshold values (Head Dropout, p = 0.3, IoU = 0.5).

5.2.2.2 *IoU and Consensus Thresholds*

With the number of forward passes fixed at 20, the impact of the clustering hyperparameters is validated by analyzing the uncertainty metrics relative to corruption severity. Figure 5.12 demonstrates the effect of the IoU threshold.

The data confirms that the previously selected IoU threshold of 0.5 represents a stable middle ground, balancing the behaviors observed at 0.3 and 0.7 without introducing extreme outliers like 0.1. Similarly, Figure 5.13 plots the metrics against the consensus threshold. The value of 0.5 is again validated as a fitting choice, minimizing variance while maintaining sensitivity to the corruption levels.

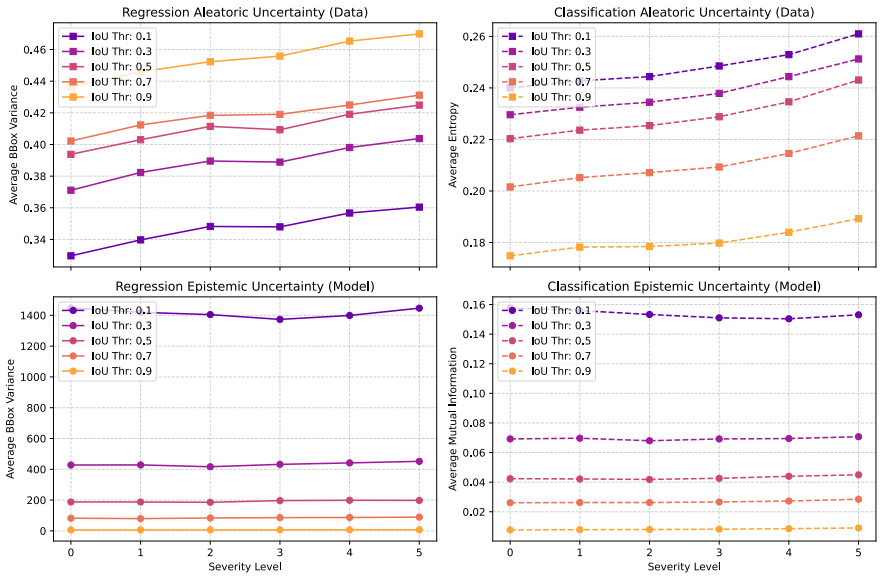


Figure 5.12: Impact of varying IoU thresholds on uncertainty metrics across increasing brightness corruption severities (Forward Passes = 20, Consensus = 0.0).

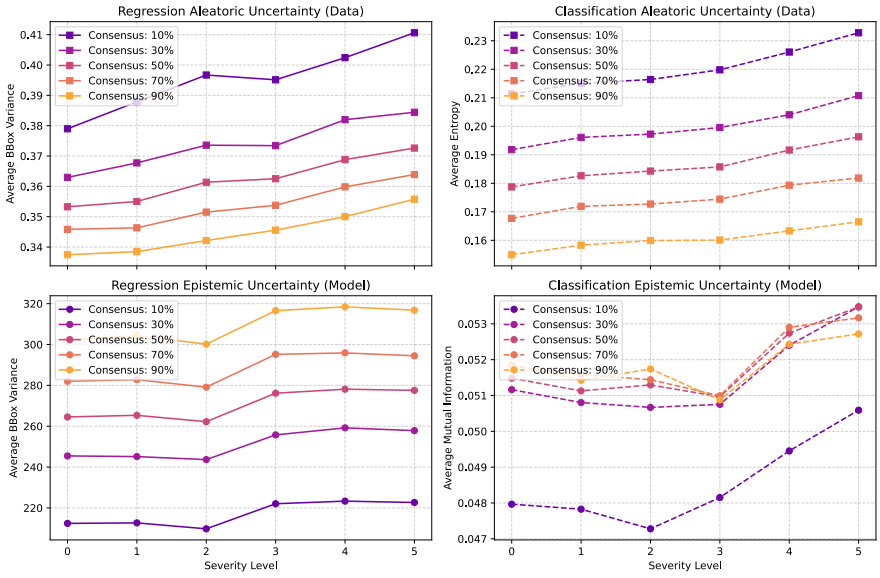


Figure 5.13: Impact of varying Consensus thresholds on uncertainty metrics across increasing brightness corruption severities (Forward Passes = 20, IoU = 0.5).

Plots for other configurations are available in Appendix a.3.3. In conclusion, the assumptions made regarding the three primary variables hold true for the MCDO approach. To ensure a fair comparison

between the two uncertainty estimation methods, the following configuration is retained:

- Number of Forward Passes: 20
- IoU Threshold: 0.5
- Consensus Threshold: 0.5

5.2.3 Monte Carlo Dropout Evaluation

To conduct a comparative analysis of the various MCDO configurations, we evaluate the changes in uncertainty metrics across three dropout locations (Backbone, Neck, Head) and three dropout probabilities ($p \in \{0.1, 0.3, 0.5\}$).

The impact of brightness corruption is depicted in Figure 5.14. For this corruption the object detection recall remained robust, with the majority of objects successfully detected even at the highest severity levels.

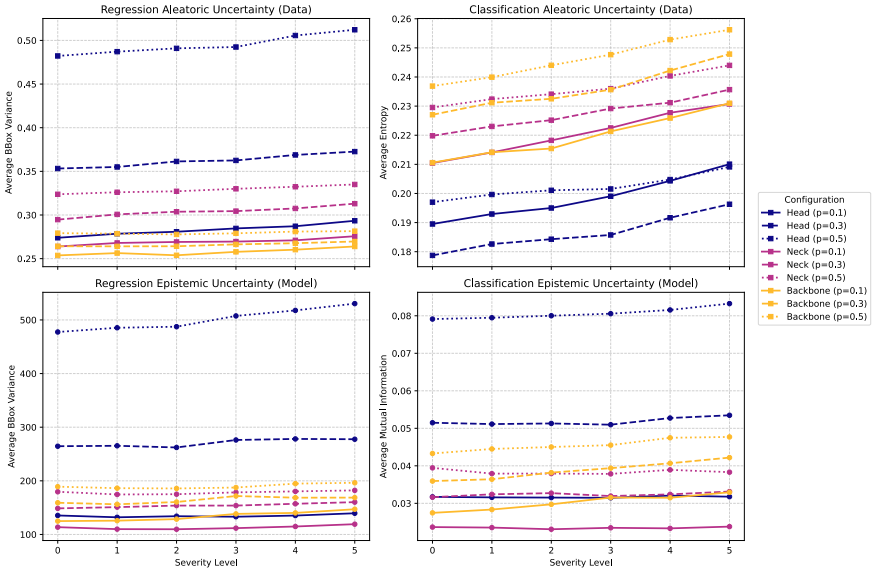


Figure 5.14: Comparative analysis of uncertainty metrics under brightness corruption across different MCDO locations and dropout rates (Forward passes = 20, IoU = 0.5, Consensus = 0.5).

Several key trends can be observed. Overall, all configurations exhibit similar behaviors relative to severity, though they are offset in absolute magnitude. Generally, a higher dropout rate correlates with higher absolute uncertainty values. The relative increase in uncertainty between severity levels 0 and 5 is modest, ranging from approximately 5% to 10% across metrics.

Notably, applying dropout to the detection *head* with a probability of $p = 0.5$ yields the most expected behavior, producing consistently

increasing uncertainty scores across all metrics as corruption severity rises. Conversely, *backbone* dropout appears ineffective at capturing the rising uncertainty associated with the regression task, although it successfully captures the increase in classification uncertainty for all dropout rates. Finally, the *neck* dropout configuration presents mixed results: while aleatoric uncertainties are calculated as expected, epistemic uncertainties remain stagnant. This suggests that applying dropout at the neck location may be insufficient to generate the model diversity required to capture epistemic uncertainty effectively.

The evaluation moves to Gaussian noise corruption in Figure 5.15. In contrast to brightness, this corruption causes severe degradation in detection performance. At the highest severity, only approximately 1% of objects are detected. Consequently, the uncertainty metrics exhibit erratic instability beyond severity level 4.

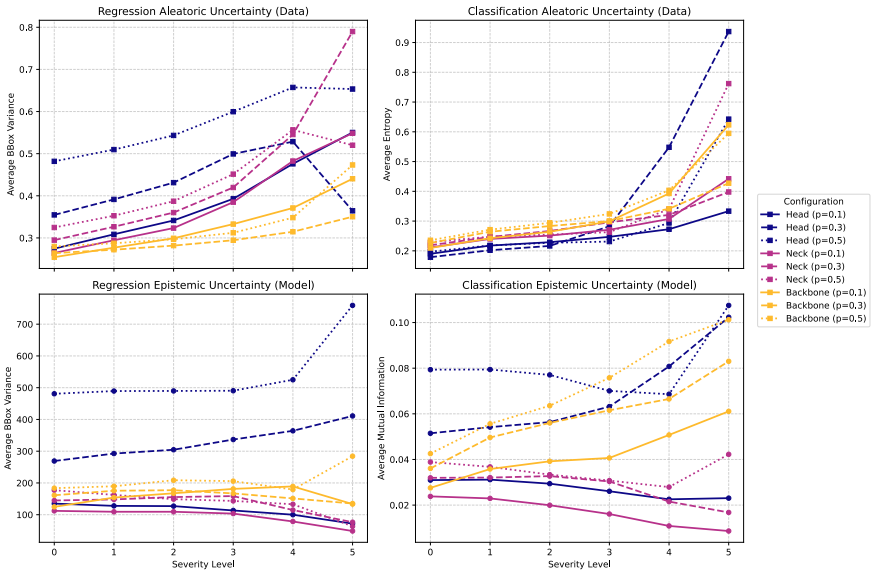


Figure 5.15: Comparative analysis of uncertainty metrics under Gaussian noise corruption (Forward passes = 20, IoU = 0.5, Consensus = 0.5).

Prior to this collapse in detection, the *head* dropout with higher probabilities ($p = 0.3, 0.5$) demonstrates the comparatively best performance, showing a increase in uncertainty across all metrics, but does not rise steadily and has bumps. Similar to the brightness experiment, the *neck* dropout fails to capture epistemic uncertainty. The *backbone* dropout maintains relatively stable performance across metrics.

Additionally the results for motion blur corruption are presented in Figure 5.16. This corruption represents an intermediate scenario regarding detection volume: detections decline at a steady rate, retaining approximately ten times the count of the gaussian noise corruption, though fewer than when applying the brightness corruption.

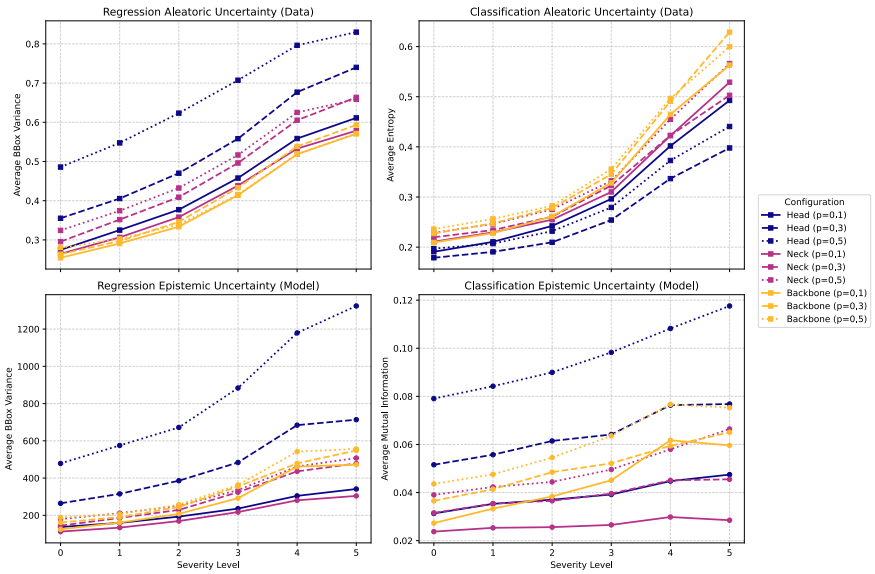


Figure 5.16: Comparative analysis of uncertainty metrics under motion blur corruption (Forward passes = 20, IoU = 0.5, Consensus = 0.5).

Under these conditions, all MCDO configurations perform largely as expected. Furthermore, the rate of increase in uncertainty metrics is significantly more pronounced compared to the previous corruptions.

For completeness, the comparative analysis for pixelate corruption is detailed in Appendix a.3.4.

5.3 COMPARISON

To conclude the experimental evaluation, a direct comparison is drawn between the Deep Ensemble method (established as the reference standard) and the various MMCDO configurations. For this analysis, the metrics obtained from the Deep Ensemble evaluation are added onto the MCDO results. The structure of the plots mirrors the previous section, displaying the three dropout locations and three dropout probabilities alongside the Deep Ensemble baseline.

The comparison under brightness corruption is illustrated in Figure 5.17.

In terms of aleatoric uncertainty, a distinct divergence is observed. The MCDO models generally predict higher aleatoric uncertainty compared to the Deep Ensembles. This suggests that the continuous injection of noise during MCDO training prevents the model from converging as fast as the ensemble members, resulting in higher predicted data noise. The exception is the backbone dropout with the lowest rate ($p = 0.1$). Its performance aligns closely with the ensemble, implying that low-magnitude dropout in the backbone creates minimal interference during training. Furthermore, for aleatoric classification

uncertainty, both neck and backbone dropout exhibit elevated baseline values even at zero severity. This indicates that these models struggle with class discrimination compared to the Deep Ensemble and head dropout models.

Regarding epistemic uncertainty, the MCDO configuration that most closely approximates the Deep Ensemble baseline is the head dropout with a rate of $p = 0.3$. Notably, only the head dropout configurations ($p = 0.3$ and $p = 0.5$) demonstrate the capacity to capture epistemic uncertainty magnitudes comparable to the ensemble method. The neck and backbone configurations consistently underestimate this metric.

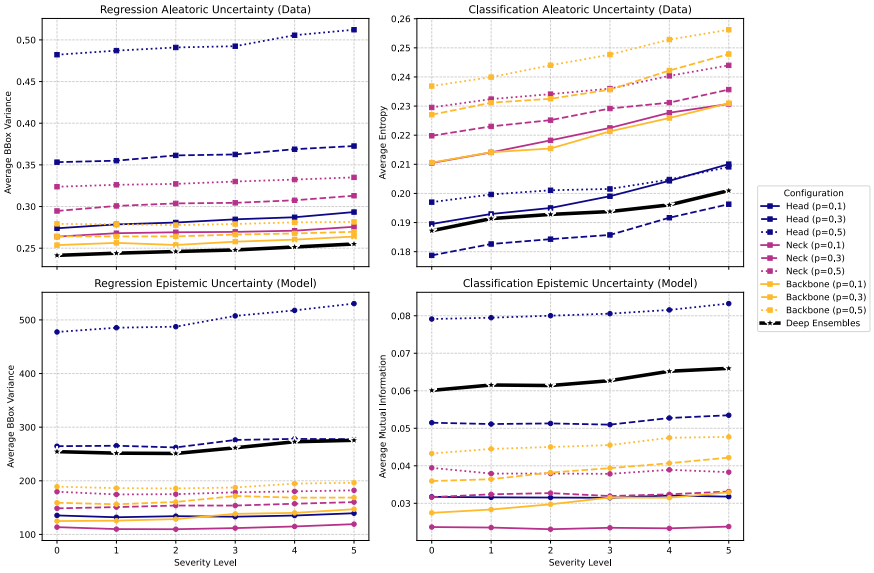


Figure 5.17: Comparative analysis of uncertainty metrics under brightness corruption, contrasting Deep Ensembles against various MCDO configurations (Ensemble Size / Forward Passes = 20, IoU = 0.5, Consensus = 0.5).

The analysis of Gaussian noise corruption, shown in Figure 5.18 highlights model behavior under severe detection degradation. As previously noted, uncertainty metrics exhibit significant instability at high severities due to the collapse in detection counts. In contrast, the Deep Ensemble metrics appear less sensitive to these outliers, maintaining smoother trends even as performance degrades.

For aleatoric uncertainty, the Deep Ensemble method yields again consistently lower absolute values compared to MCDO. Conversely, for epistemic uncertainty, the ensemble reports higher values than the majority of MCDO configurations. While the instability of the plots at high severity makes precise interpretation difficult, the head and backbone dropout configurations with higher probabilities appear to track the Deep Ensemble trends most effectively before the detection collapse occurs.

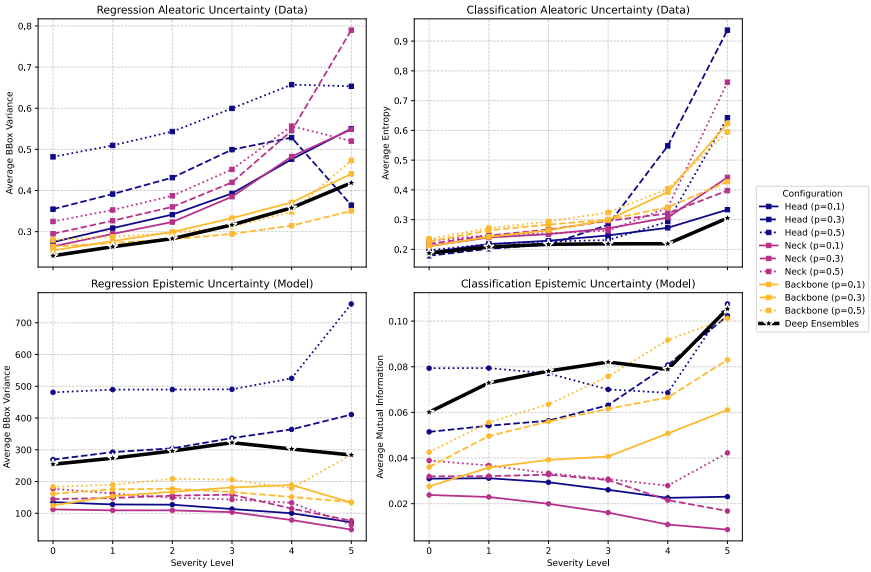


Figure 5.18: Comparative analysis of uncertainty metrics under Gaussian noise corruption, contrasting Deep Ensembles against various MCDO configurations (Ensemble Size / Forward Passes = 20, IoU = 0.5, Consensus = 0.5).

Finally, the motion blur corruption (Figure 5.19) provides a view of an intermediate degradation scenario.

Consistent with previous observations, Deep Ensembles exhibit lower aleatoric uncertainty, indicating predictions with lower variance compared to MCDO. Regarding epistemic uncertainty, the ensemble method again outperforms most MCDO configurations in terms of capturing model diversity. The exception is the head dropout configuration, which not only approaches but, at higher dropout rates, exceeds the epistemic uncertainty levels recorded by the Deep Ensembles.

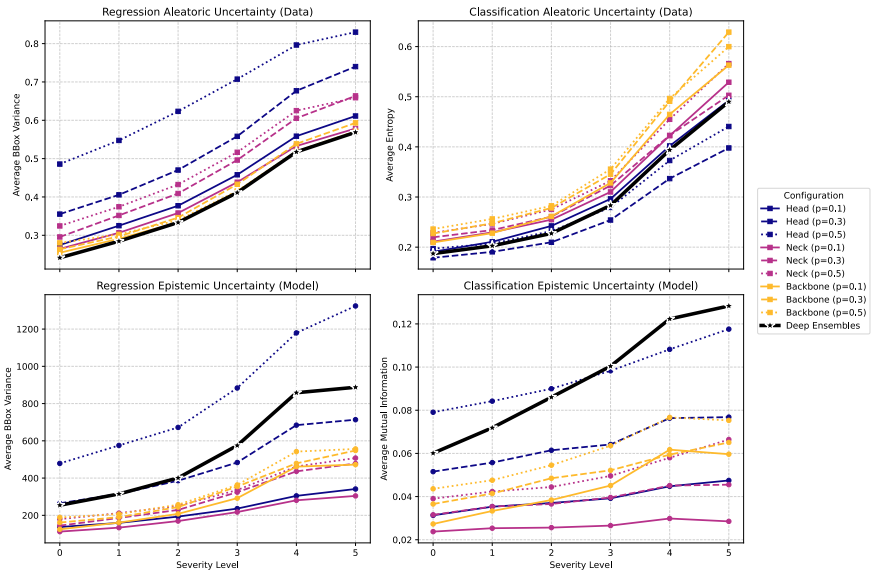


Figure 5.19: Comparative analysis of uncertainty metrics under motion blur corruption, contrasting Deep Ensembles against various MCDO configurations (Ensemble Size / Forward Passes = 20, IoU = 0.5, Consensus = 0.5).

The comparative plots for pixelate corruption follow similar trends and are included in Appendix a.3.5.

CONCLUSION

The experimental results show distinct behavioral differences between Deep Ensembles and MCDO for uncertainty estimation in object detection.

A consistent trend across all corruption types is that MCDO yields higher aleatoric uncertainty values than Deep Ensembles. This can be attributed to the regularization effect of dropout. Introducing stochasticity during training slows convergence, preventing the model from predicting with the same precision as the ensemble members. The only exception is low-rate backbone dropout, where the interference is insufficient to significantly alter the training dynamics. This results in metrics similar to the ensemble baseline.

Conversely, Deep Ensembles generally exhibit higher epistemic uncertainty because they are trained as independent models with random initializations, which allows the ensemble to explore a more diverse range of weights. MCDO, which relies on perturbing a single model, often struggles to generate comparable diversity. Only the head dropout configuration with a high probability ($p \geq 0.3$) succeeds in generating epistemic uncertainty estimates comparable to the en-

semble, suggesting that aggressive perturbation in the final layers is necessary to approximate true model diversity.

Regarding architectural placement, head dropout emerges as the best configuration for MCDO in this study. It most closely tracks Deep Ensemble metrics and robustly responds to increasing corruption. Backbone Dropout, while stable, fails to generate sufficient diversity and consistently underestimates epistemic uncertainty. Neck Dropout was the least effective configuration, exhibiting high baseline variance and poor sensitivity to epistemic changes.

Finally, the stability of the metrics depends heavily on the nature of the corruption. Deep Ensembles demonstrated superior robustness in high-degradation scenarios, whereas MCDO metrics became chaotic once detection counts dropped.

6 DISCUSSION AND OUTLOOK

The primary aim of this thesis was to evaluate and compare Deep Ensembles and Monte Carlo Dropout as Bayesian approximation techniques for the YOLOv8 object detector. The evaluation relied on a corrupted version of the COCO2017 dataset to test model robustness. To facilitate this, an experimental framework was designed to capture both aleatoric and epistemic uncertainty in regression and classification tasks. Key modifications included the addition of an aleatoric head to the YOLO architecture and the implementation of a clustering algorithm to aggregate predictions from stochastic forward passes. A parameter search successfully identified transferable hyperparameters for ensemble size and clustering, ensuring a fair and direct comparison between the two methods.

The experimental results demonstrate that the reliability of uncertainty metrics is intrinsically linked to the stability of the underlying object detector. Under moderate corruption such as brightness changes or motion blur detection counts decreased gradually and uncertainty metrics exhibited stable, interpretable trends. However, under severe degradation scenarios, such as intense Gaussian noise or pixelation, the detector suffered a sharp decline in recall, causing uncertainty metrics to behave unpredictably. These findings suggest that Bayesian approximations in object detection are effective only as long as the detector maintains a baseline of representational capability.

Regarding the configuration of MCDO, this study highlights the critical importance of dropout placement and magnitude. The analysis revealed that applying dropout to the *detection head* with a high probability ($p = 0.5$) yielded the most robust results, closely mirroring the trends observed in Deep Ensembles in most cases. Conversely, applying dropout to the backbone or neck resulted mainly in low, stagnant estimates of epistemic uncertainty. This indicates that stochasticity introduced in early layers is often suppressed by subsequent layers, failing to generate the output diversity necessary to approximate model uncertainty. Therefore, to achieve uncertainty estimates comparable to an ensemble, aggressive perturbation in the final layers of the network is required.

When directly comparing the two methods, Deep Ensembles proved more robust and more successful in capturing epistemic uncertainty. MCDO based models generally predicts higher aleatoric uncertainty,

likely because the added regularization during training prevented the model from converging as quickly as the individual ensemble members. Furthermore, with the exception of the high-rate head dropout configuration, MCDO underestimated epistemic uncertainty, reflecting lower total diversity among the models.

However, these results must be considered in the context of survivorship bias. As corruption severity increased, the average bounding box diameter of detected objects also increased. This implies that the detector preferentially fails on smaller, difficult objects, continuing to track only the larger, easier instances. Consequently, metrics calculated at high corruption levels represent a biased subset of the data rather than the complete dataset. This bias particularly affects regression uncertainty, whereas classification uncertainty remains arguably more robust to object size. Additionally, the drop in detection counts at high severities reduces the sample size available for the clustering algorithm. This mimics the effect of reducing the number of ensemble members, thereby introducing instability into the metrics that is independent of actual image uncertainty.

A significant challenge identified in this study was the non-normalized nature of the applied corruptions. Because different corruptions impact recall at vastly different rates, direct comparison was difficult. For instance, high-severity brightness corruption retained significantly more detections than high-severity Gaussian noise. Furthermore, the implementation of the aleatoric head introduced additional complexity to the model and the new optimization objective slowed convergence and resulted in a decrease in mAP compared to the standard YOLOv8n baseline. Future implementations could employ KL annealing to mitigate this, allowing the model to train standard weights before tuning the aleatoric head in later epochs.

From a practical deployment perspective, the choice between Deep Ensembles and MCDO represents a trade-off between implementation complexity, computational resources, and performance.

Deep Ensembles offer straightforward implementation but require a linear increase in training time and memory proportional to the ensemble size, making them prohibitive for resource-constrained environments.

MCDO requires only a single trained model, offering advantages in storage and latency. However, finding the optimal configuration is non-trivial. Even in this work it cannot be ensured, that the optimal dropout configuration was found. As shown, a suboptimal configuration can fail to capture epistemic uncertainty entirely.

Future research should aim to decouple uncertainty metrics from the survivorship bias inherent in detection tasks. Experiments using

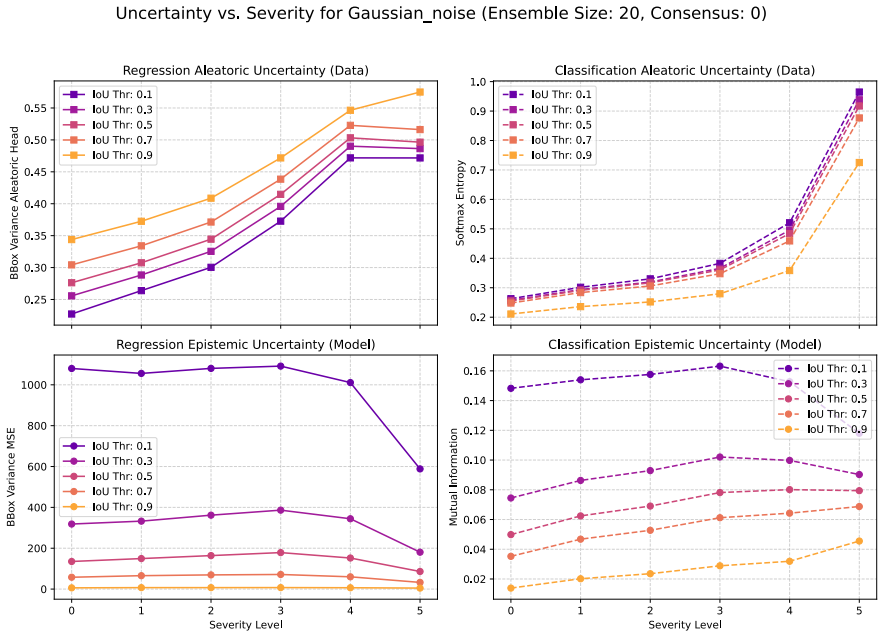
datasets such as DirtyMNIST potentially restricted to single-object instances or synthesized for uniform object sizes could isolate uncertainty metrics from the effects of bounding box regression difficulties. Additionally, creating a corruption dataset normalized by detection count rather than visual intensity would allow for a more rigorous comparison across degradation types. Further investigation into the detector's confidence threshold is also advised. Lowering this threshold to force predictions on out-of-distribution data could provide deeper insights into behavior in high-uncertainty regimes. Finally, expanding the scope to include methods such as Repulsive Ensembles or Last-Layer Variational Inference could provide a more complete view of the reliability of deep learning models in object detection.

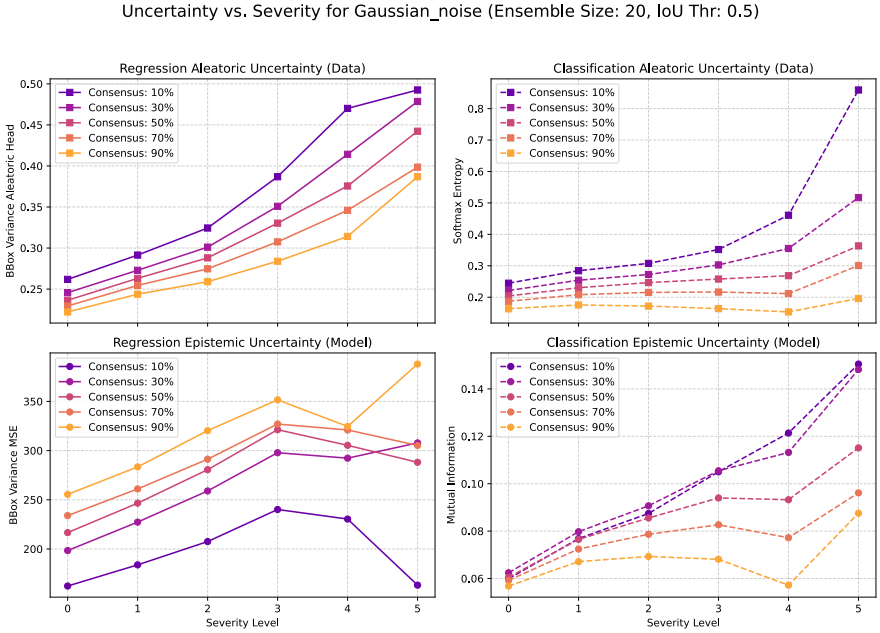
A APPENDIX

A.1 CLUSTERING ALGORITHM

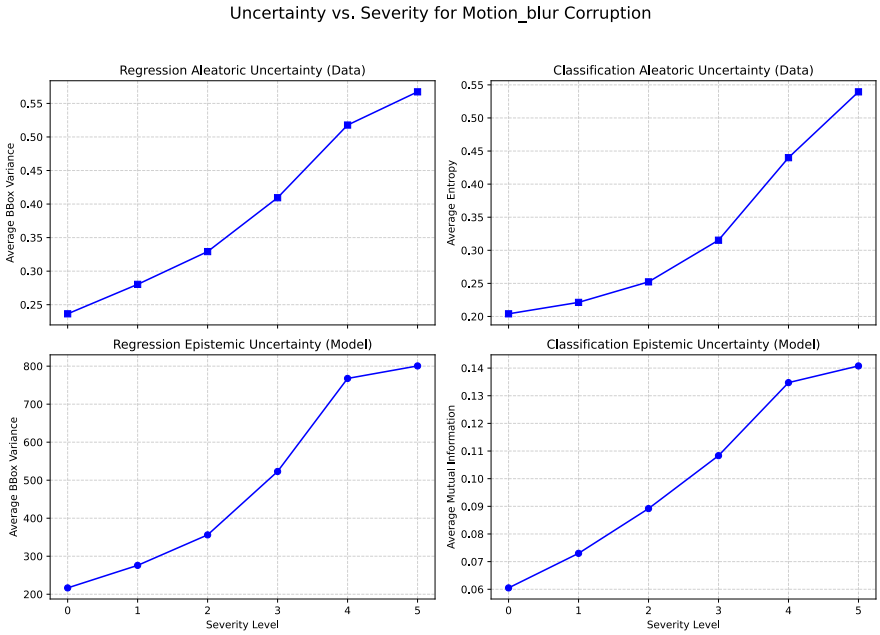
A.2 DEEP ENSEMBLES

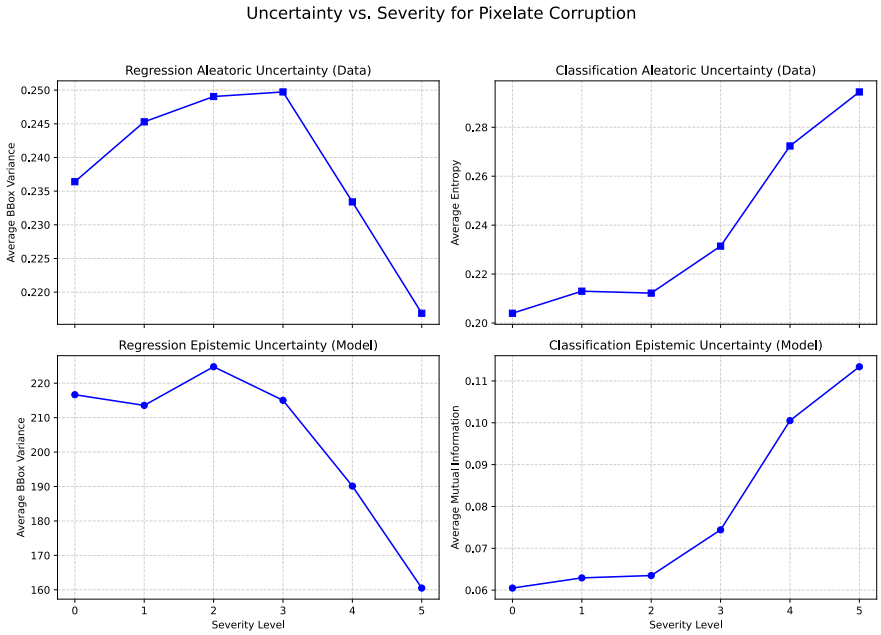
A.2.1 IoU and Consensus Tuning for Gaussian Noise Corruption





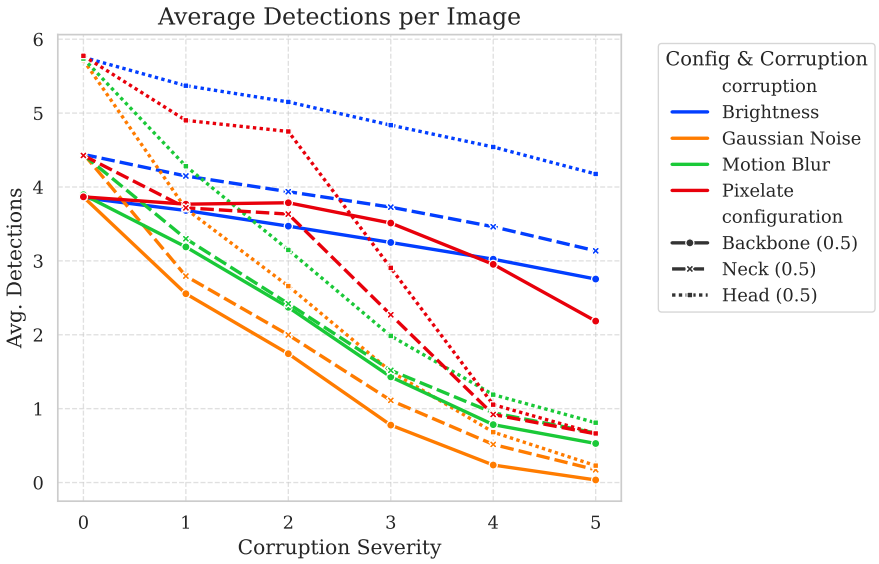
A.2.2 Deep Ensemble Evaluation for Motion Blur and Pixelate Corruption





A.3 MONTE CARLO DROPOUT

A.3.1 Detection Counts Monte Carlo Dropout



A.3.2 Number of Forward Passes Tuning

A.3.2.1 *Brightness Corruption*

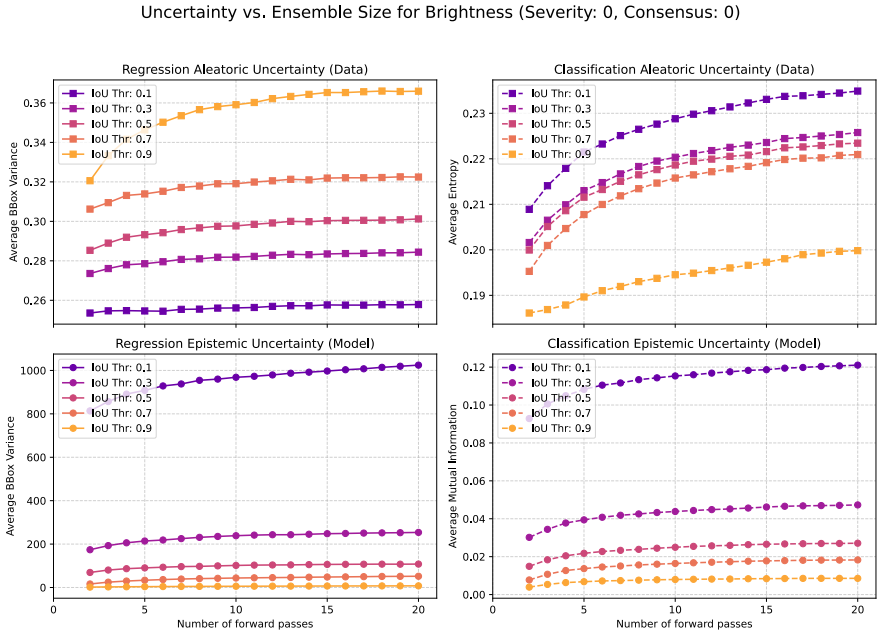


Figure a.1: Dropout Rate: 0.1

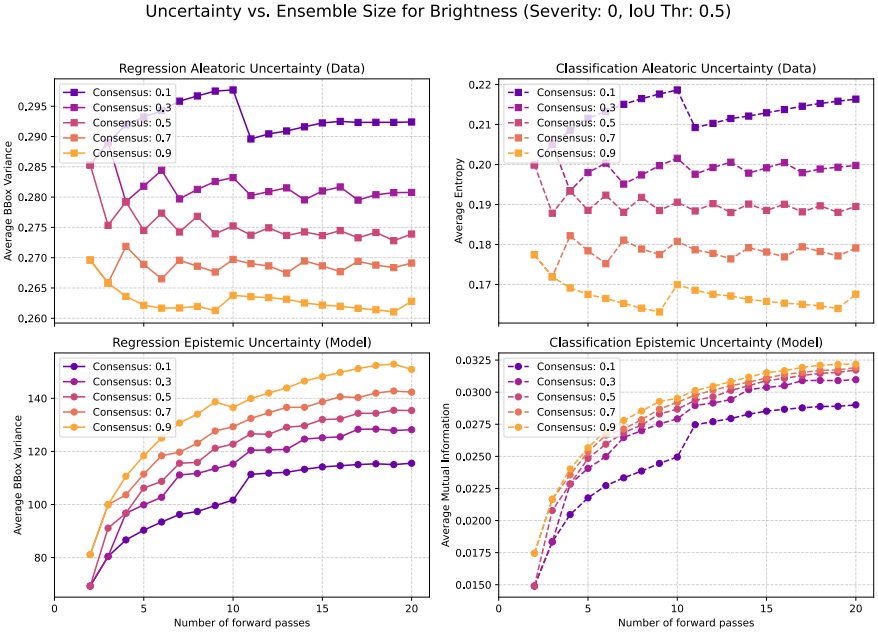


Figure a.2: Dropout Rate: 0.1

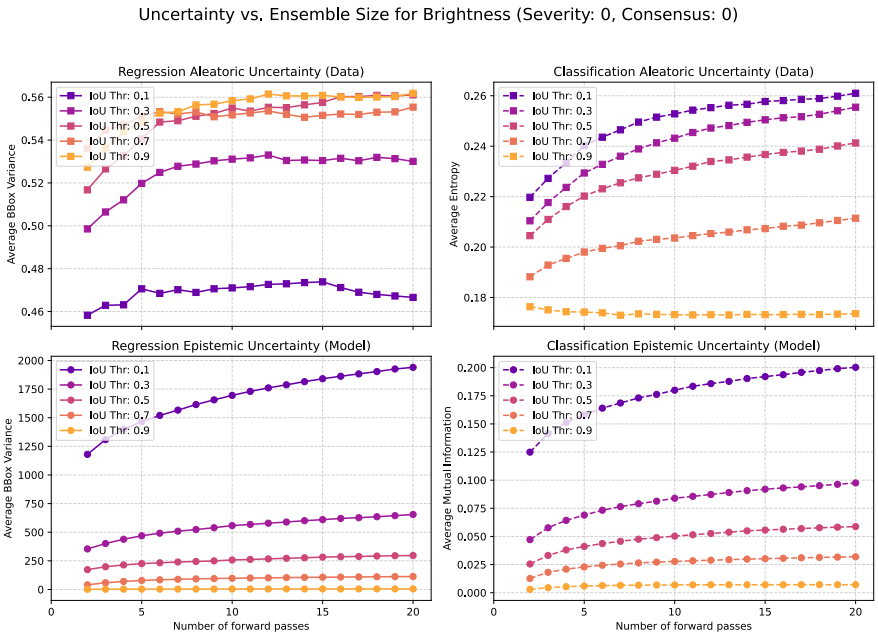


Figure a.3: Dropout Rate: 0.5

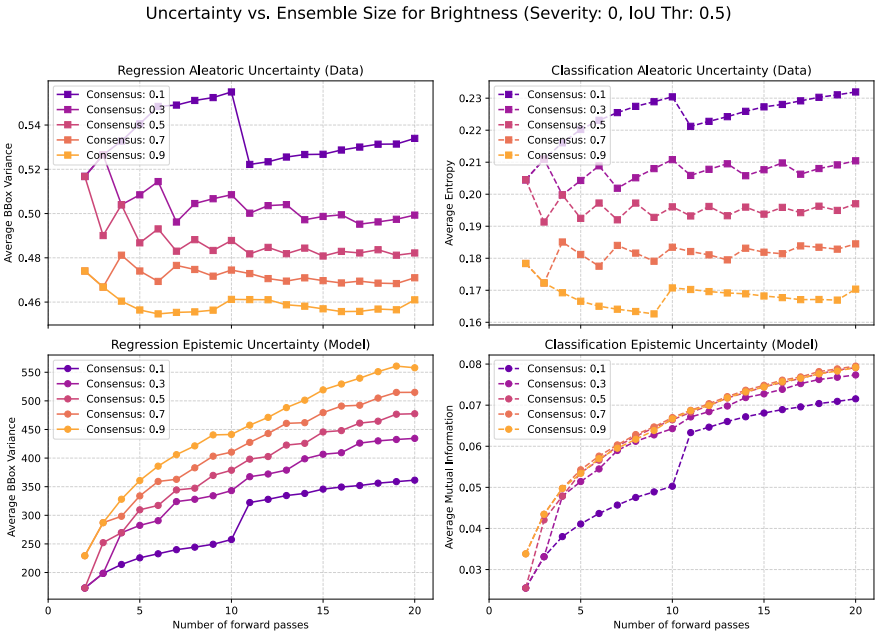


Figure a.4: Dropout Rate: 0.5

A.3.2.2 Gaussian Noise Corruption

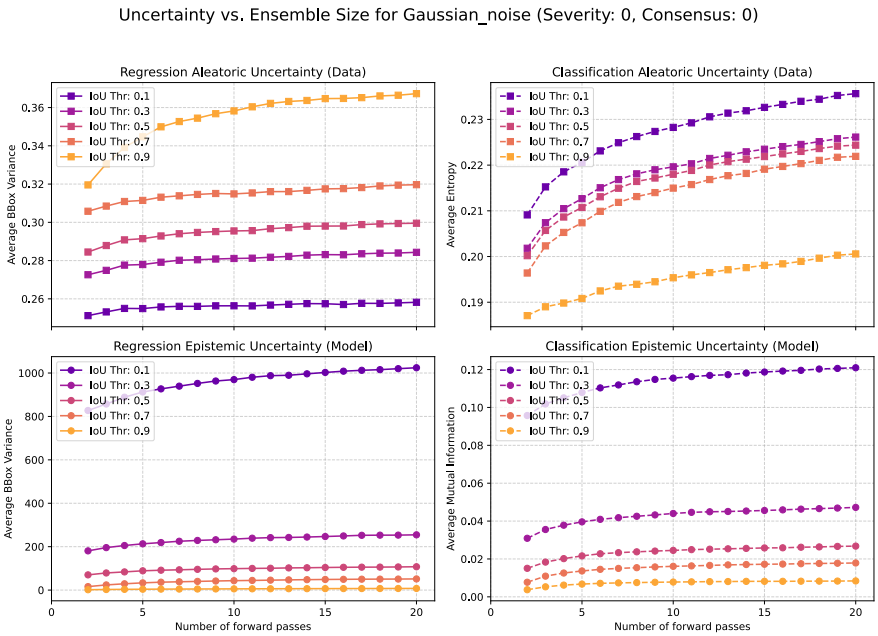


Figure a.5: Dropout Rate: 0.1

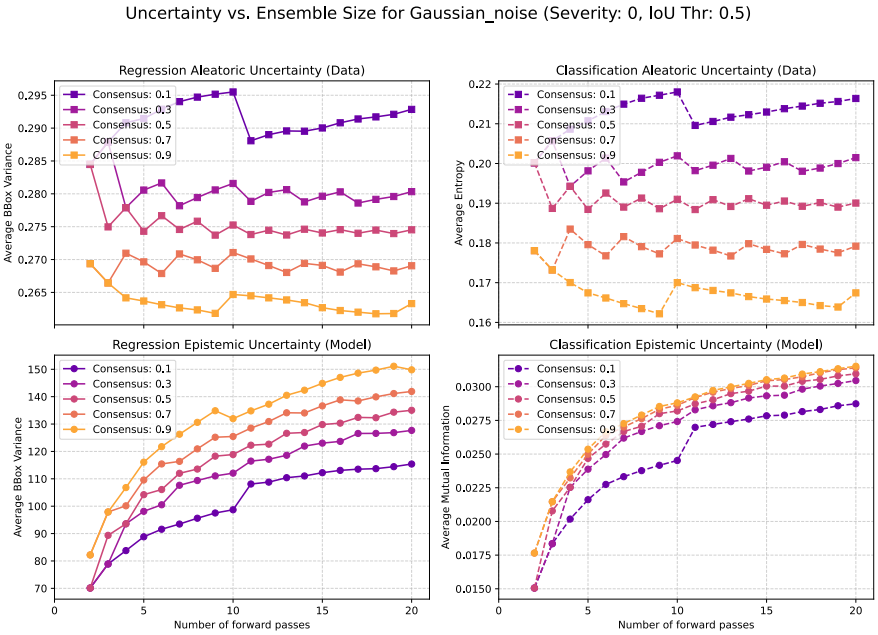


Figure a.6: Dropout Rate: 0.1

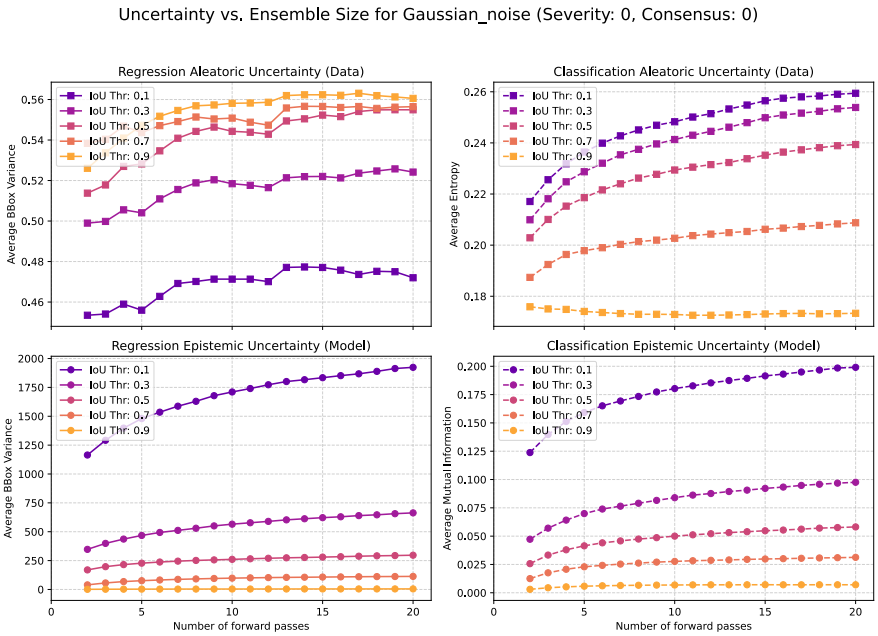


Figure a.7: Dropout Rate: 0.5

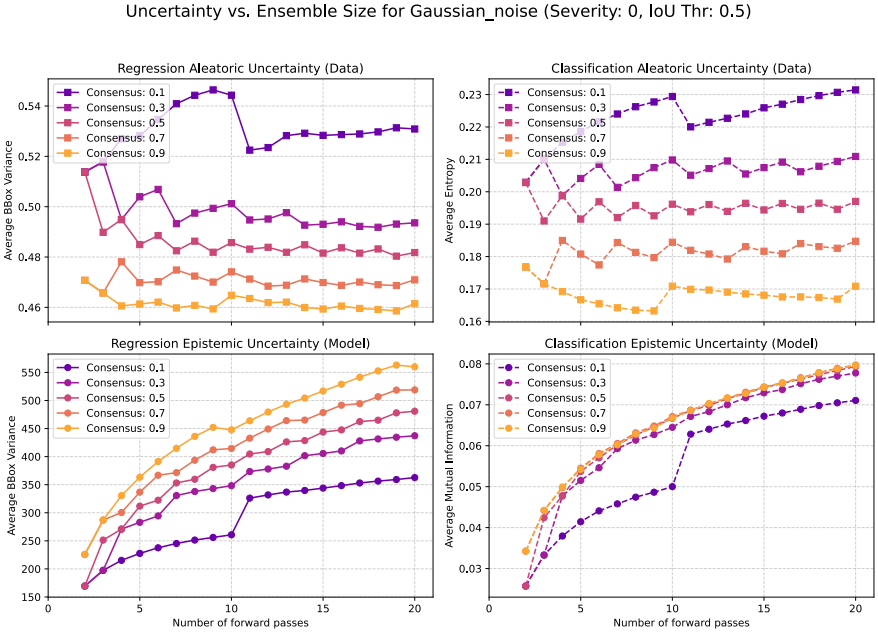


Figure a.8: Dropout Rate: 0.5

A.3.3 IoU and Consensus Tuning

A.3.3.1 IoU and Consensus Tuning for Brightness Corruption

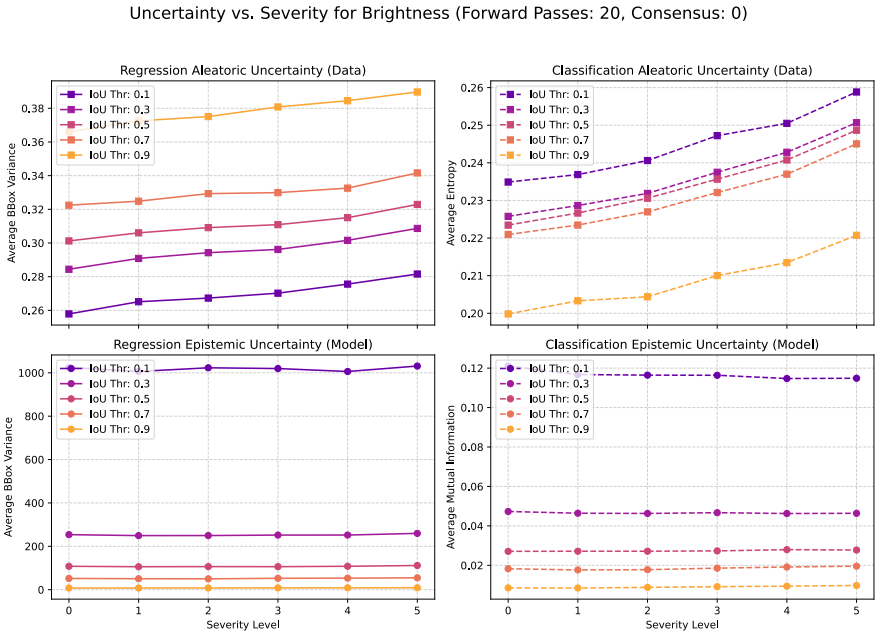


Figure a.9: Dropout Rate: 0.1

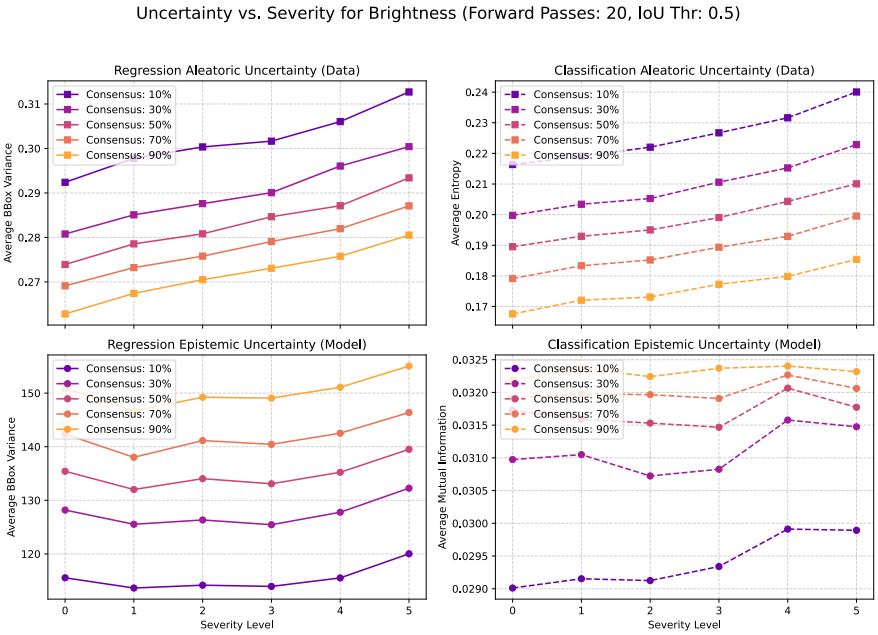


Figure a.10: Dropout Rate: 0.1

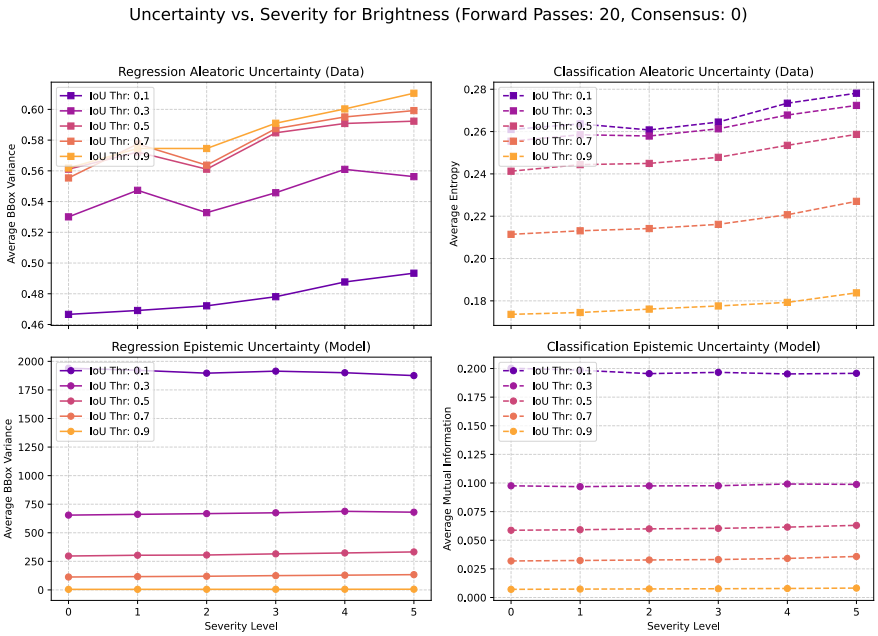


Figure a.11: Dropout Rate: 0.5

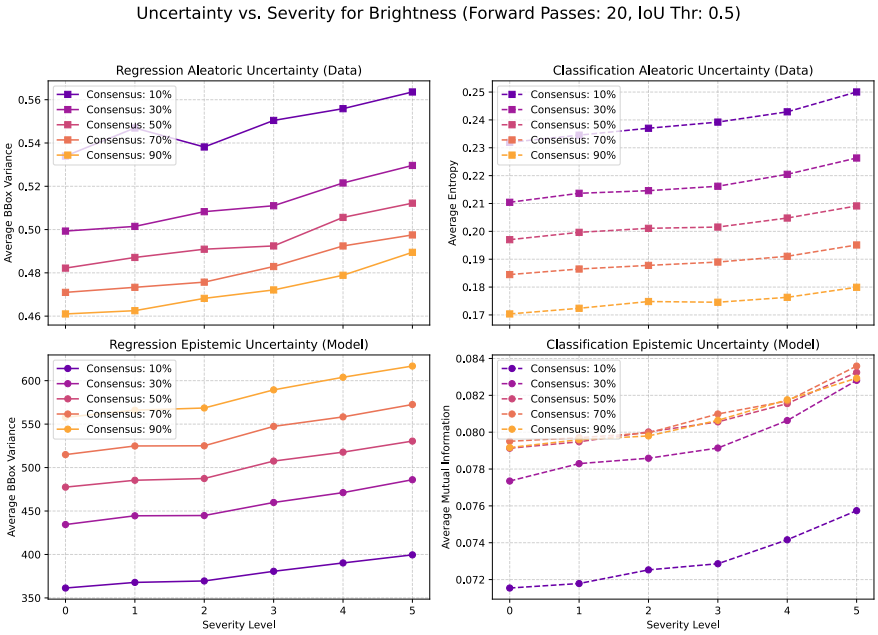


Figure a.12: Dropout Rate: 0.5

A.3.3.2 IoU and Consensus Tuning for Gaussian Noise Corruption

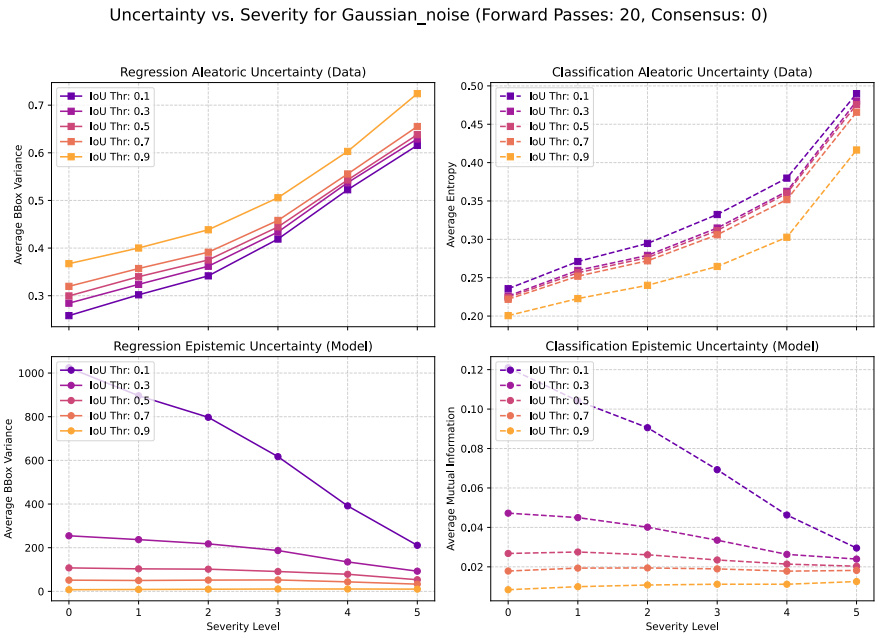


Figure a.13: Dropout Rate: 0.1

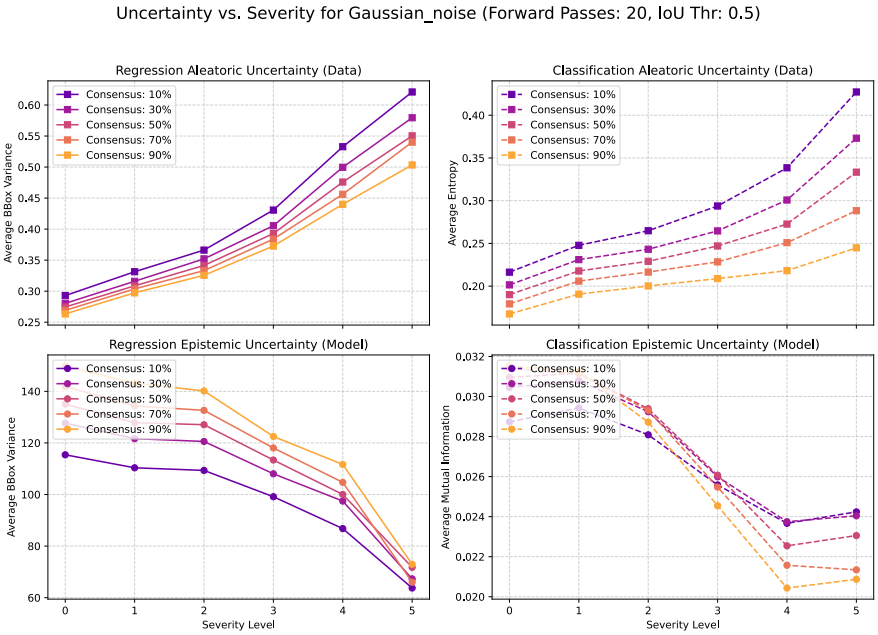


Figure a.14: Dropout Rate: 0.1

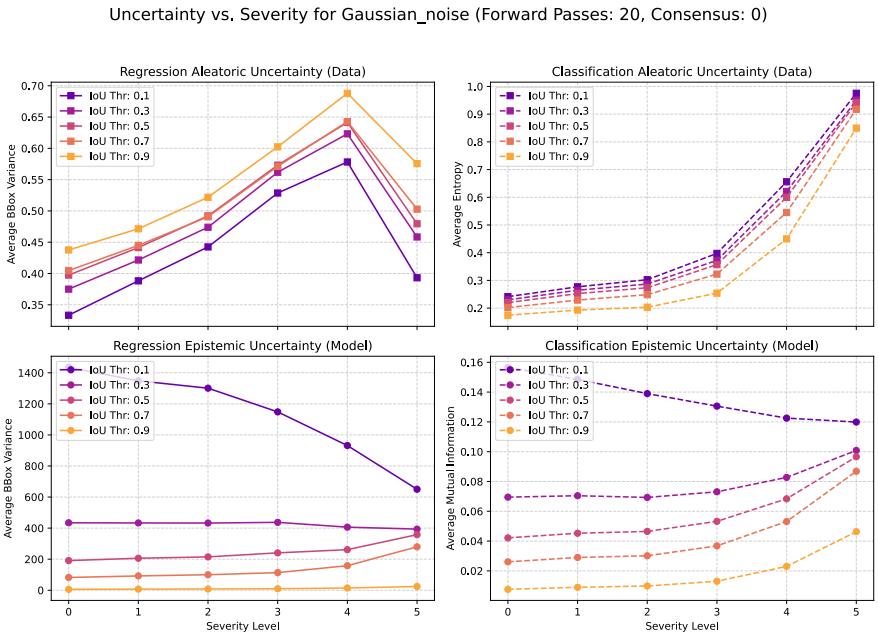


Figure a.15: Dropout Rate: 0.3

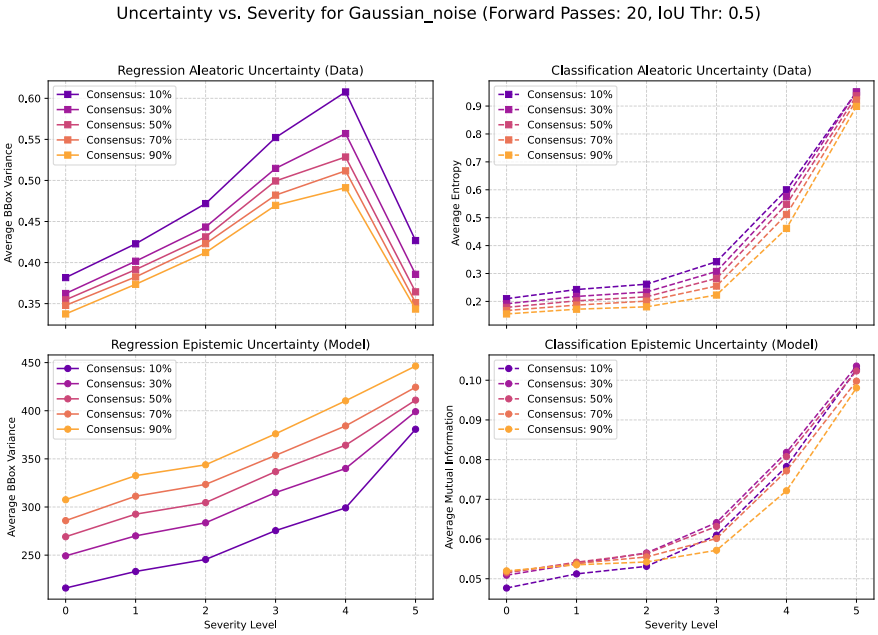


Figure a.16: Dropout Rate: 0.3

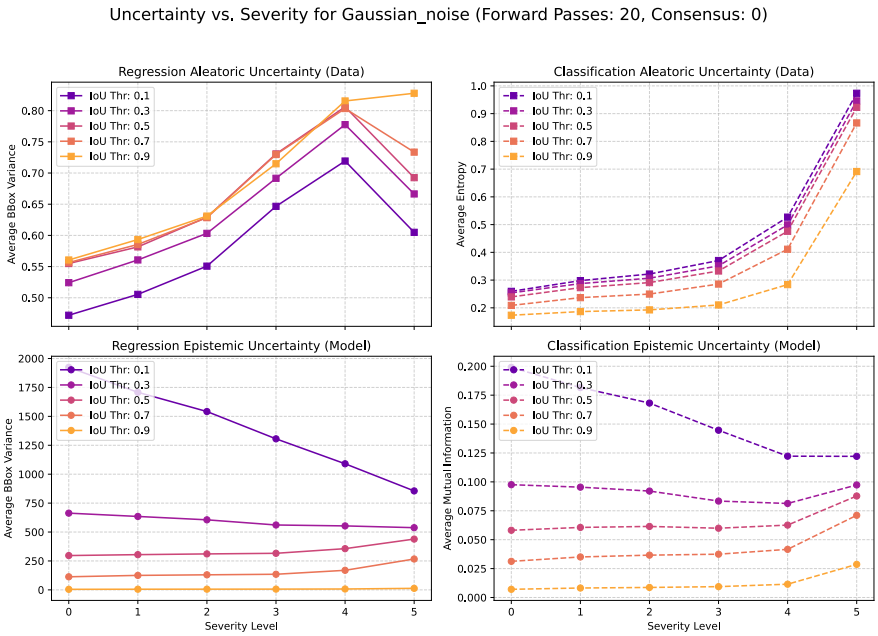


Figure a.17: Dropout Rate: 0.5

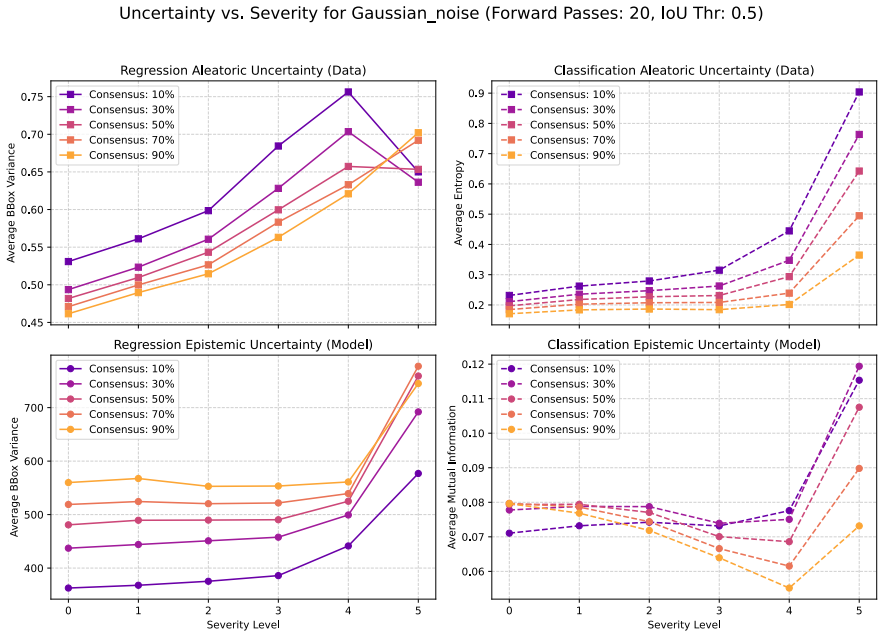
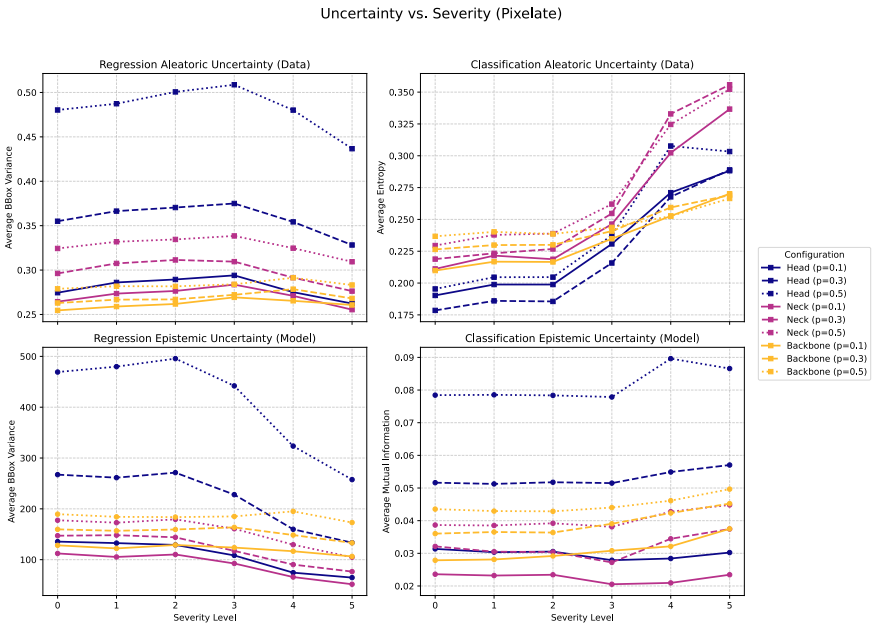
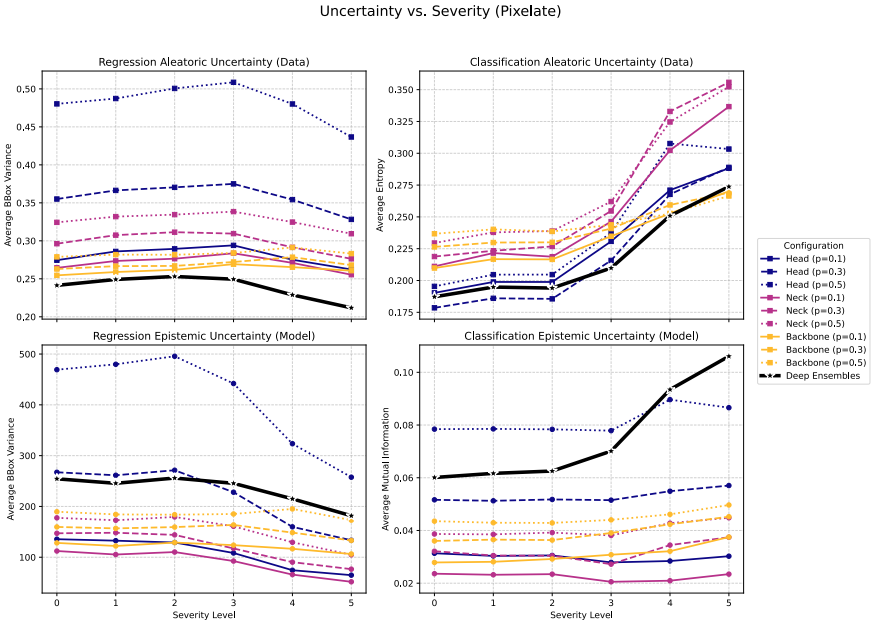


Figure a.18: Dropout Rate: 0.5

A.3.4 Monte Carlo Evaluation for Pixelate Corruption



A.3.5 Comparison between Deep Ensembles and MCDO



BIBLIOGRAPHY

[1] Joost van Amersfoort, Lewis Smith, Yee Whye Teh, and Yarin Gal. *Uncertainty Estimation Using a Single Deep Deterministic Neural Network*. 2020. arXiv: 2003.02037 [cs.LG]. URL: https://arxiv.org/abs/2003.02037.

[2] Murat Seckin Ayhan and Philipp Berens. "Test-time Data Augmentation for Estimation of Heteroscedastic Aleatoric Uncertainty in Deep Neural Networks." In: *International conference on Medical Imaging with Deep Learning*. 2018.

[3] Tiago Azevedo, René de Jong, Matthew Mattina, and Partha Maji. *Stochastic-YOLO: Efficient Probabilistic Object Detection under Dataset Shifts*. 2020. arXiv: 2009.02967 [cs.CV]. URL: https://arxiv.org/abs/2009.02967.

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2016. arXiv: 1409.0473 [cs.CL]. URL: https://arxiv.org/abs/1409.0473.

[5] Abhishek Balasubramaniam and Sudeep Pasricha. *Object Detection in Autonomous Vehicles: Status and Open Challenges*. 2022. arXiv: 2201.07706 [cs.CV]. URL: https://arxiv.org/abs/2201.07706.

[6] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. "Variational Inference: A Review for Statisticians." In: *Journal of the American Statistical Association* 112.518 (Apr. 2017), pp. 859–877. ISSN: 1537-274X. DOI: 10.1080/01621459.2017.1285773. URL: http://dx.doi.org/10.1080/01621459.2017.1285773.

[7] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. 2020. arXiv: 2004.10934 [cs.CV]. URL: https://arxiv.org/abs/2004.10934.

[8] Xavier Bouthillier et al. *Accounting for Variance in Machine Learning Benchmarks*. 2021. arXiv: 2103.03098 [cs.LG]. URL: https://arxiv.org/abs/2103.03098.

[9] Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Jozefowicz, and Samy Bengio. *Generating Sentences from a Continuous Space*. 2016. arXiv: 1511.06349 [cs.LG]. URL: https://arxiv.org/abs/1511.06349.

[10] John Bridle. "Training Stochastic Model Recognition Algorithms as Networks can Lead to Maximum Mutual Information Estimation of Parameters." In: *Advances in Neural Information Processing Systems*. Ed. by D. Touretzky. Vol. 2. Morgan-Kaufmann, 1989. URL: https://proceedings.neurips.cc/paper_files/paper/1989/file/0336dcbab05b9d5ad24f4333c7658a0e-Paper.pdf.

[11] Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng. *Handbook of Markov Chain Monte Carlo*. Chapman and Hall/CRC, May 2011. ISBN: 9780429138508. DOI: 10.1201/b10905. URL: http://dx.doi.org/10.1201/b10905.

[12] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL]. URL: https://arxiv.org/abs/2005.14165.

[13] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. *End-to-End Object Detection with Transformers*. 2020. arXiv: 2005.12872 [cs.CV]. URL: https://arxiv.org/abs/2005.12872.

[14] Nicolas Carion et al. *SAM 3: Segment Anything with Concepts*. 2025. arXiv: 2511.16719 [cs.CV]. URL: https://arxiv.org/abs/2511.16719.

[15] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. *Emerging Properties in Self-Supervised Vision Transformers*. 2021. arXiv: 2104.14294 [cs.CV]. URL: https://arxiv.org/abs/2104.14294.

[16] Lucy R. Chai. "Uncertainty Estimation in Bayesian Neural Networks and Links to Interpretability." Dissertation submitted for the degree of Master of Philosophy, Churchill College. MA thesis. Cambridge, UK: University of Cambridge, Aug. 2018. URL: https://people.csail.mit.edu/lrchai/files/Chai_thesis.pdf.

[17] Tom Charnock, Laurence Perreault-Levasseur, and François Lanusse. *Bayesian Neural Networks*. 2020. arXiv: 2006.01490 [stat.ML]. URL: https://arxiv.org/abs/2006.01490.

[18] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. *Multi-View 3D Object Detection Network for Autonomous Driving*. 2017. arXiv: 1611.07759 [cs.CV]. URL: https://arxiv.org/abs/1611.07759.

[19] Jiwoong Choi, Dayoung Chun, Hyun Kim, and Hyuk-Jae Lee. *Gaussian YOLOv3: An Accurate and Fast Object Detector Using Localization Uncertainty for Autonomous Driving*. 2019. arXiv: 1904.04620 [cs.CV]. URL: https://arxiv.org/abs/1904.04620.

[20] Nicolas Chopin and Sumeetpal S. Singh. "On particle Gibbs sampling." In: *Bernoulli* 21.3 (Aug. 2015). ISSN: 1350-7265. DOI: 10.3150/14-bej629. URL: http://dx.doi.org/10.3150/14-BEJ629.

[21] Francesco D'Angelo and Vincent Fortuin. *Repulsive Deep Ensembles are Bayesian*. 2023. arXiv: 2106.11642 [cs.LG]. URL: https://arxiv.org/abs/2106.11642.

[22] Andreas C. Damianou and Neil D. Lawrence. *Deep Gaussian Processes*. 2013. arXiv: 1211.0358 [stat.ML]. URL: https://arxiv.org/abs/1211.0358.

[23] Kumari Deepshikha, Sai Harsha Yelleni, P. K. Srijith, and C Krishna Mohan. *Monte Carlo DropBlock for Modelling Uncertainty in Object Detection*. 2021. arXiv: 2108.03614 [cs.CV]. URL: https://arxiv.org/abs/2108.03614.

[24] Luca Deininger, Bernhard Stimpel, Anil Yuce, Samaneh Abbasi-Sureshjani, Simon Schönenberger, Paolo Ocampo, Konstanty Korski, and Fabien Gaire. *A comparative study between vision transformers and CNNs in digital pathology*. 2022. arXiv: 2206.00389 [eess.IV]. URL: https://arxiv.org/abs/2206.00389.

[25] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "ImageNet: A large-scale hierarchical image database." In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[26] Li Deng. "The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]." In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142. DOI: 10.1109/MSP.2012.2211477.

[27] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL]. URL: https://arxiv.org/abs/1810.04805.

[28] Thomas G. Dietterich. "Ensemble Methods in Machine Learning." In: *Multiple Classifier Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 1–15. ISBN: 978-3-540-45014-6.

[29] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2021. arXiv: 2010.11929 [cs.CV]. URL: https://arxiv.org/abs/2010.11929.

[30] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. *Activation Functions in Deep Learning: A Comprehensive Survey and Benchmark*. 2022. arXiv: 2109.14545 [cs.LG]. URL: https://arxiv.org/abs/2109.14545.

[31] A. Duncan, N. Nuesken, and L. Szpruch. *On the geometry of Stein variational gradient descent*. 2023. arXiv: 1912.00894 [stat.ML]. URL: https://arxiv.org/abs/1912.00894.

[32] Mark Ebden. *Gaussian Processes: A Quick Introduction*. 2015. arXiv: 1505.02965 [math.ST]. URL: https://arxiv.org/abs/1505.02965.

[33] Omar Elharrouss, Younes Akbari, Noor Almadeed, and Somaya Al-Maadeed. "Backbones-review: Feature extractor networks for deep learning and deep reinforcement learning approaches in computer vision." In: *Computer Science Review* 53 (Aug. 2024), p. 100645. ISSN: 1574-0137. DOI: 10.1016/j.cosrev.2024.100645. URL: http://dx.doi.org/10.1016/j.cosrev.2024.100645.

[34] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. "The Pascal Visual Object Classes (VOC) Challenge." In: *International Journal of Computer Vision* 88.2 (2010), pp. 303–338. DOI: 10.1007/s11263-009-0275-4.

[35] Pedro F. Felzenszwalb, Ross B. Girshick, David McAllester, and Deva Ramanan. "Object Detection with Discriminatively Trained Part-Based Models." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32.9 (2010), pp. 1627–1645. DOI: 10.1109/TPAMI.2009.167.

[36] Yarin Gal. "Uncertainty in Deep Learning." PhD thesis. University of Cambridge, 2016. URL: https://www.cs.ox.ac.uk/people/yarin.gal/website/thesis/thesis.pdf.

[37] Yarin Gal and Zoubin Ghahramani. *Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning*. 2016. arXiv: 1506.02142 [stat.ML]. URL: https://arxiv.org/abs/1506.02142.

[38] Ankush Ganguly and Samuel W. F. Earp. *An Introduction to Variational Inference*. 2021. arXiv: 2108.13083 [cs.LG]. URL: https://arxiv.org/abs/2108.13083.

[39] Zheng Ge, Songtao Liu, Feng Wang, Zeming Li, and Jian Sun. *YOLOX: Exceeding YOLO Series in 2021*. 2021. arXiv: 2107.08430 [cs.CV]. URL: https://arxiv.org/abs/2107.08430.

[40] Pierre Geurts, Damien Ernst, and Louis Wehenkel. "Extremely randomized trees." In: *Machine Learning* 63.1 (2006), pp. 3–42. DOI: 10.1007/s10994-006-6226-1. URL: https://doi.org/10.1007/s10994-006-6226-1.

[41] Ross Girshick. *Fast R-CNN*. 2015. arXiv: 1504.08083 [cs.CV]. URL: https://arxiv.org/abs/1504.08083.

[42] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. *Rich feature hierarchies for accurate object detection and semantic segmentation*. 2014. arXiv: 1311.2524 [cs.CV]. URL: https://arxiv.org/abs/1311.2524.

[43] Ethan Goan and Clinton Fookes. "Bayesian Neural Networks: An Introduction and Survey." In: *Case Studies in Applied Bayesian Data Science*. Springer International Publishing, 2020, pp. 45–87. ISBN: 9783030425531. DOI: 10.1007/978-3-030-42553-1_3. URL: http://dx.doi.org/10.1007/978-3-030-42553-1_3.

[44] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. *Explaining and Harnessing Adversarial Examples*. 2015. arXiv: 1412.6572 [stat.ML]. URL: https://arxiv.org/abs/1412.6572.

[45] Ali Harakeh, Michael Smart, and Steven L. Waslander. *BayesOD: A Bayesian Approach for Uncertainty Estimation in Deep Object Detectors*. 2019. arXiv: 1903.03838 [cs.CV]. URL: https://arxiv.org/abs/1903.03838.

[46] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition." In: *Computer Vision – ECCV 2014*. Springer International Publishing, 2014, pp. 346–361. ISBN: 9783319105789. DOI: 10.1007/978-3-319-10578-9_23. URL: http://dx.doi.org/10.1007/978-3-319-10578-9_23.

[47] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV]. URL: https://arxiv.org/abs/1512.03385.

[48] Yihui He, Chenchen Zhu, Jianren Wang, Marios Savvides, and Xiangyu Zhang. *Bounding Box Regression with Uncertainty for Accurate Object Detection*. 2019. arXiv: 1809.08545 [cs.CV]. URL: https://arxiv.org/abs/1809.08545.

[49] Aral Hekimoglu, Michael Schmidt, and Alvaro Marcos-Ramiro. *Monocular 3D Object Detection with LiDAR Guided Semi Supervised Active Learning*. 2023. arXiv: 2307.08415 [cs.CV]. URL: https://arxiv.org/abs/2307.08415.

[50] Priyanto Hidayatullah, Nurjannah Syakrani, Muhammad Rizqi Sholahuddin, Trisna Gelar, and Refdinal Tubagus. *YOLOv8 to YOLO11: A Comprehensive Architecture In-depth Comparative Review*. 2025. arXiv: 2501.13400 [cs.CV]. URL: https://arxiv.org/abs/2501.13400.

[51] Matt Hoffman, David M. Blei, Chong Wang, and John Paisley. *Stochastic Variational Inference*. 2013. arXiv: 1206.7051 [stat.ML]. URL: https://arxiv.org/abs/1206.7051.

[52] Matthew D. Hoffman and Andrew Gelman. *The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo*. 2011. arXiv: 1111.4246 [stat.CO]. URL: https://arxiv.org/abs/1111.4246.

[53] Eyke Hüllermeier and Willem Waegeman. "Aleatoric and epistemic uncertainty in machine learning: an introduction to concepts and methods." In: *Machine Learning* 110.3 (Mar. 2021), pp. 457–506. ISSN: 1573-0565. DOI: 10.1007/s10994-021-05946-3. URL: http://dx.doi.org/10.1007/s10994-021-05946-3.

[54] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG]. URL: https://arxiv.org/abs/1502.03167.

[55] Pavel Izmailov, Sharad Vikram, Matthew D. Hoffman, and Andrew Gordon Wilson. *What Are Bayesian Neural Network Posteriors Really Like?* 2021. arXiv: 2104.14421 [cs.LG]. URL: https://arxiv.org/abs/2104.14421.

[56] Prakriti Jain. "Image Corruption on Object Detection: Data to Evaluate Uncertainty Estimation." Master's Thesis. MA thesis. Heidelberg, Germany: Heidelberg University, Faculty of Mathematics, Computer Science, Data, and Computer Science, Oct. 30, 2024.

[57] Thomas Jantos, Stephan Weiss, and Jan Steinbrener. *Aleatoric Uncertainty from AI-based 6D Object Pose Predictors for Object-relative State Estimation*. 2025. arXiv: 2509.01583 [cs.RO]. URL: https://arxiv.org/abs/2509.01583.

[58] Glenn Jocher, Jing Qiu, and Ayush Chaurasia. *Ultralytics YOLO*. Version 8.0.0. Jan. 2023. URL: https://github.com/ultralytics/ultralytics.

[59] Michael I. Jordan, Zoubin Ghahramani, Tommi S. Jaakkola, and Lawrence K. Saul. "An Introduction to Variational Methods for Graphical Models." In: *Machine Learning* 37.2 (1999), pp. 183–233.

[60] Laurent Valentin Jospin, Hamid Laga, Farid Boussaid, Wray Buntine, and Mohammed Bennamoun. "Hands-On Bayesian Neural Networks—A Tutorial for Deep Learning Users." In: *IEEE Computational Intelligence Magazine* 17.2 (2022), pp. 29–48. DOI: 10.1109/MCI.2022.3155327.

[61] Alex Kendall and Yarin Gal. *What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?* 2017. arXiv: 1703.04977 [cs.CV]. URL: https://arxiv.org/abs/1703.04977.

[62] Rahima Khanam and Muhammad Hussain. *What is YOLOv5: A deep look into the internal features of the popular object detector*. 2024. arXiv: 2407.20892 [cs.CV]. URL: https://arxiv.org/abs/2407.20892.

[63] Yoon Kim, Carl Denton, Luong Hoang, and Alexander M. Rush. *Structured Attention Networks*. 2017. arXiv: 1702.00887 [cs.CL]. URL: https://arxiv.org/abs/1702.00887.

[64] Florian Kraus and Klaus Dietmayer. "Uncertainty Estimation in One-Stage Object Detection." In: *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, Oct. 2019. DOI: 10.1109/itsc.2019.8917494. URL: http://dx.doi.org/10.1109/ITSC.2019.8917494.

[65] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. *Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles*. 2017. arXiv: 1612.01474 [stat.ML]. URL: https://arxiv.org/abs/1612.01474.

[66] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition." In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[67] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." In: *Nature* 521.7553 (2015), pp. 436–444. DOI: 10.1038/nature14539.

[68] Stefan Lee, Senthil Purushwalkam, Michael Cogswell, David Crandall, and Dhruv Batra. *Why M Heads are Better than One: Training a Diverse Ensemble of Deep Networks*. 2015. arXiv: 1511.06314 [cs.CV]. URL: https://arxiv.org/abs/1511.06314.

[69] Ian Lenz, Honglak Lee, and Ashutosh Saxena. *Deep Learning for Detecting Robotic Grasps*. 2014. arXiv: 1301.3592 [cs.LG]. URL: https://arxiv.org/abs/1301.3592.

[70] Chuyi Li et al. *YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications*. 2022. arXiv: 2209.02976 [cs.CV]. URL: https://arxiv.org/abs/2209.02976.

[71] Xiang Li, Chengqi Lv, Wenhai Wang, Gang Li, Lingfeng Yang, and Jian Yang. "Generalized Focal Loss: Towards Efficient Representation Learning for Dense Object Detection." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45.3 (2023), pp. 3139–3153. DOI: 10.1109/TPAMI.2022.3180392.

[72] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. *Feature Pyramid Networks for Object Detection*. 2017. arXiv: 1612.03144 [cs.CV]. URL: https://arxiv.org/abs/1612.03144.

[73] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. *Focal Loss for Dense Object Detection*. 2018. arXiv: 1708.02002 [cs.CV]. URL: https://arxiv.org/abs/1708.02002.

[74] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. *Microsoft COCO: Common Objects in Context*. 2015. arXiv: 1405.0312 [cs.CV]. URL: https://arxiv.org/abs/1405.0312.

[75] Li Liu, Wanli Ouyang, Xiaogang Wang, Paul Fieguth, Jie Chen, Xinwang Liu, and Matti Pietikäinen. *Deep Learning for Generic Object Detection: A Survey*. 2019. arXiv: 1809.02165 [cs.CV]. URL: https://arxiv.org/abs/1809.02165.

[76] Qiang Liu. *Stein Variational Gradient Descent as Gradient Flow*. 2017. arXiv: 1704.07520 [stat.ML]. URL: https://arxiv.org/abs/1704.07520.

[77] Qiang Liu and Dilin Wang. *Stein Variational Gradient Descent: A General Purpose Bayesian Inference Algorithm*. 2019. arXiv: 1608.04471 [stat.ML]. URL: https://arxiv.org/abs/1608.04471.

[78] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. *Path Aggregation Network for Instance Segmentation*. 2018. arXiv: 1803.01534 [cs.CV]. URL: https://arxiv.org/abs/1803.01534.

[79] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. "SSD: Single Shot MultiBox Detector." In: *Computer Vision – ECCV 2016*. Springer International Publishing, 2016, pp. 21–37. ISBN: 9783319464480. DOI: 10.1007/978-3-319-46448-0_2. URL: http://dx.doi.org/10.1007/978-3-319-46448-0_2.

[80] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. *Swin Transformer: Hierarchical Vision Transformer using Shifted Windows*. 2021. arXiv: 2103.14030 [cs.CV]. URL: https://arxiv.org/abs/2103.14030.

[81] Alexander Mattern, Henrik Gerdes, Dennis Grunert, and Robert H. Schmitt. "A comparison of transformer and CNN-based object detection models for surface defects on Li-Ion Battery Electrodes." In: *Journal of Energy Storage* 105 (2025), p. 114378. ISSN: 2352-152X. DOI: https://doi.org/10.1016/j.est.2024.114378. URL: https://www.sciencedirect.com/science/article/pii/S2352152X24039641.

[82] Jishnu Mukhoti, Andreas Kirsch, Joost van Amersfoort, Philip H. S. Torr, and Yarin Gal. *Deep Deterministic Uncertainty: A Simple Baseline*. 2022. arXiv: 2102.11582 [cs.LG]. URL: https://arxiv.org/abs/2102.11582.

[83] Radford M. Neal. *Probabilistic Inference Using Markov Chain Monte Carlo Methods*. Technical Report CRG-TR-93-1. Toronto, Canada: Department of Computer Science, University of Toronto, Sept. 1993.

[84] Radford M. Neal. *Bayesian Learning for Neural Networks*. 1st ed. Vol. 118. Lecture Notes in Statistics. New York, NY: Springer, 1996, p. 204. ISBN: 978-0-387-94724-2. DOI: 10.1007/978-1-4612-0745-0. URL: https://doi.org/10.1007/978-1-4612-0745-0.

[85] Anh Nguyen, Jason Yosinski, and Jeff Clune. *Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images*. 2015. arXiv: 1412.1897 [cs.CV]. URL: https://arxiv.org/abs/1412.1897.

[86] Keiron O'Shea and Ryan Nash. *An Introduction to Convolutional Neural Networks*. 2015. arXiv: 1511.08458 [cs.NE]. URL: https://arxiv.org/abs/1511.08458.

[87] Adam Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019. arXiv: 1912.01703 [cs.LG]. URL: https://arxiv.org/abs/1912.01703.

[88] L. Peng, H. Wang, and J. Li. "Uncertainty Evaluation of Object Detection Algorithms for Autonomous Vehicles." In: *Automotive Innovation* 4 (August 2021 2021). Received: 04 January 2021, Accepted: 31 May 2021, Published: 30 July 2021, pp. 241–252. DOI: 10.1007/s42154-021-00154-0. URL: https://doi.org/10.1007/s42154-021-00154-0.

[89] David M. W. Powers. *Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation*. 2020. arXiv: 2010.16061 [cs.LG]. URL: https://arxiv.org/abs/2010.16061.

[90] Rajesh Ranganath, Sean Gerrish, and David M. Blei. *Black Box Variational Inference*. 2013. arXiv: 1401.0118 [stat.ML]. URL: https://arxiv.org/abs/1401.0118.

[91] Joseph Redmon. *Darknet: Open Source Neural Networks in C*. https://pjreddie.com/darknet/. 2013–2016.

[92] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: 1506.02640 [cs.CV]. URL: https://arxiv.org/abs/1506.02640.

[93] Joseph Redmon and Ali Farhadi. *YOLO9000: Better, Faster, Stronger*. 2016. arXiv: 1612.08242 [cs.CV]. URL: https://arxiv.org/abs/1612.08242.

[94] Joseph Redmon and Ali Farhadi. *YOLOv3: An Incremental Improvement*. 2018. arXiv: 1804.02767 [cs.CV]. URL: https://arxiv.org/abs/1804.02767.

[95] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2016. arXiv: 1506.01497 [cs.CV]. URL: https://arxiv.org/abs/1506.01497.

[96] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. *Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression*. 2019. arXiv: 1902.09630 [cs.CV]. URL: https://arxiv.org/abs/1902.09630.

[97] Christian P. Robert. *The Metropolis-Hastings algorithm*. 2016. arXiv: 1504.01896 [stat.CO]. URL: https://arxiv.org/abs/1504.01896.

[98] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV]. URL: https://arxiv.org/abs/1505.04597.

[99] Frank Rosenblatt. "The perceptron: A probabilistic model for information storage and organization in the brain." In: *Psychological Review* 65.6 (1958), pp. 386–408. DOI: 10.1037/h0042519.

[100] Max Schwarz, Anton Milan, Arul Selvam Periyasamy, and Sven Behnke. "RGB-D object detection and semantic segmentation for autonomous manipulation in clutter." In: *The International Journal of Robotics Research* 37.4–5 (June 2017), pp. 437–451. ISSN: 1741-3176. DOI: 10.1177/0278364917713117. URL: http://dx.doi.org/10.1177/0278364917713117.

[101] Claude E. Shannon. "A Mathematical Theory of Communication." In: *Bell System Technical Journal* 27.3, 4 (1948), pp. 379–423, 623–656.

[102] Mrinank Sharma, Sebastian Farquhar, Eric Nalisnick, and Tom Rainforth. *Do Bayesian Neural Networks Need To Be Fully Stochastic?* 2023. arXiv: 2211.06291 [cs.LG]. URL: https://arxiv.org/abs/2211.06291.

[103] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. *Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism*. 2020. arXiv: 1909.08053 [cs.CL]. URL: https://arxiv.org/abs/1909.08053.

[104] Petru Soviany and Radu Tudor Ionescu. *Optimizing the Trade-off between Single-Stage and Two-Stage Object Detectors using Image Difficulty Prediction*. 2018. arXiv: 1803.08707 [cs.CV]. URL: https://arxiv.org/abs/1803.08707.

[105] David A. Spade. "Chapter 1 - Markov chain Monte Carlo methods: Theory and practice." In: *Principles and Methods for Data Science*. Ed. by Arni S.R. Srinivasa Rao and C.R. Rao. Vol. 43. Handbook of Statistics. Elsevier, 2020, pp. 1–66. DOI: https://doi.org/10.1016/bs.host.2019.06.001. URL: https://www.sciencedirect.com/science/article/pii/S0169716119300379.

[106] Joshua S. Speagle. *A Conceptual Introduction to Markov Chain Monte Carlo Methods*. 2020. arXiv: 1909.12313 [stat.OT]. URL: https://arxiv.org/abs/1909.12313.

[107] Sophie Steger, Christian Knoll, Bernhard Klein, Holger Fröning, and Franz Pernkopf. *Function Space Diversity for Uncertainty Prediction via Repulsive Last-Layer Ensembles*. 2024. arXiv: 2412.15758 [cs.LG]. URL: https://arxiv.org/abs/2412.15758.

[108] Knut Sveidqvist and Contributors to Mermaid. *Mermaid: Generate diagrams from markdown-like text*. Dec. 2014. URL: https://github.com/mermaid-js/mermaid.

[109] Richard Szeliski. *Computer Vision: Algorithms and Applications*. 2nd. Cham: Springer, 2022, pp. 343–346. ISBN: 978-3030343712. DOI: 10.1007/978-3-030-34372-9. URL: https://szeliski.org/Book/.

[110] Mingxing Tan, Ruoming Pang, and Quoc V. Le. *EfficientDet: Scalable and Efficient Object Detection*. 2020. arXiv: 1911.09070 [cs.CV]. URL: https://arxiv.org/abs/1911.09070.

[111] PyTorch Team. *torch.nn.Dropout — PyTorch documentation*. https://docs.pytorch.org/docs/stable/generated/torch.nn.Dropout.html. Accessed: October 26, 2025. 2024.

[112] PyTorch Team. *torch.nn.Dropout2d — PyTorch documentation*. https://docs.pytorch.org/docs/stable/generated/torch.nn.Dropout2d.html. Accessed: October 26, 2025. 2024.

[113] Juan Terven, Diana-Margarita Córdova-Esparza, and Julio-Alejandro Romero-González. "A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS." In: *Machine Learning and Knowledge Extraction* 5.4 (Nov. 2023), pp. 1680–1716. ISSN: 2504-4990. DOI: 10.3390/make5040083. URL: http://dx.doi.org/10.3390/make5040083.

[114] Ultralytics. *YOLOv8 Performance Metrics — Ultralytics Docs*. https://docs.ultralytics.com/models/yolov8/#performance-metrics. Accessed: October 26, 2025. 2024.

[115] Ultralytics. *Ultralytics: Revolutionierung der Welt der KI*. https://www.ultralytics.com/de. Accessed: 2025-05-02. 2025.

[116] Matias Valdenegro-Toro and Daniel Saromo. *A Deeper Look into Aleatoric and Epistemic Uncertainty Disentanglement*. 2022. arXiv: 2204.09308 [cs.LG]. URL: https://arxiv.org/abs/2204.09308.

[117] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

[118] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*. 2022. arXiv: 2207.02696 [cs.CV]. URL: https://arxiv.org/abs/2207.02696.

[119] Chien-Yao Wang, Hong-Yuan Mark Liao, I-Hau Yeh, Yueh-Hua Wu, Ping-Yang Chen, and Jun-Wei Hsieh. *CSPNet: A New Backbone that can Enhance Learning Capability of CNN*. 2019. arXiv: 1911.11929 [cs.CV]. URL: https://arxiv.org/abs/1911.11929.

[120] Chien-Yao Wang, I-Hau Yeh, and Hong-Yuan Mark Liao. *You Only Learn One Representation: Unified Network for Multiple Tasks*. 2021. arXiv: 2105.04206 [cs.CV]. URL: https://arxiv.org/abs/2105.04206.

[121] Dayong Wang, Aditya Khosla, Rishab Gargeya, Humayun Irshad, and Andrew H. Beck. *Deep Learning for Identifying Metastatic Breast Cancer*. 2016. arXiv: 1606.05718 [q-bio.QM]. URL: https://arxiv.org/abs/1606.05718.

[122] Deng-Bao Wang, Lei Feng, and Min-Ling Zhang. "Rethinking Calibration of Deep Neural Networks: Do Not Be Afraid of Overconfidence." In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan. Vol. 34. Curran Associates, Inc., 2021, pp. 11809–11820. URL: https://openreview.net/pdf?id=NJS8kp15zzH.

[123] Andrew Gordon Wilson and Pavel Izmailov. *Bayesian Deep Learning and a Probabilistic Perspective of Generalization*. 2022. arXiv: 2002.08791 [cs.LG]. URL: https://arxiv.org/abs/2002.08791.

[124] Luhuan Wu and Sinead Williamson. *Posterior Uncertainty Quantification in Neural Networks using Data Augmentation*. 2024. arXiv: 2403.12729 [stat.ML]. URL: https://arxiv.org/abs/2403.12729.

[125] Xianzhe Xu, Yiqi Jiang, Weihua Chen, Yilun Huang, Yuan Zhang, and Xiuyu Sun. *DAMO-YOLO : A Report on Real-Time Object Detection Design*. 2023. arXiv: 2211.15444 [cs.CV]. URL: https://arxiv.org/abs/2211.15444.

[126] G. Yao, S. Zhu, L. Zhang, and M. Qi. "HP-YOLOv8: High-Precision Small Object Detection Algorithm for Remote Sensing Images." In: *Sensors* 24.15 (2024), p. 4858. DOI: 10.3390/s24154858. URL: https://doi.org/10.3390/s24154858.

[127] Muhammad Yaseen. *What is YOLOv8: An In-Depth Exploration of the Internal Features of the Next-Generation Object Detector*. 2024. arXiv: 2408.15857 [cs.CV]. URL: https://arxiv.org/abs/2408.15857.

[128] Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. *Object Detection with Deep Learning: A Review*. 2019. arXiv: 1807.05511 [cs.CV]. URL: https://arxiv.org/abs/1807.05511.

[129] Alon Zolfi, Guy Amit, Amit Baras, Satoru Koda, Ikuya Morikawa, Yuval Elovici, and Asaf Shabtai. *YolOOD: Utilizing Object Detection Concepts for Multi-Label Out-of-Distribution Detection*. 2023. arXiv: 2212.02081 [cs.CV]. URL: https://arxiv.org/abs/2212.02081.

ERKLÄRUNG

Ich versichere, dass ich diese Arbeit selbständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, den 30/11/2025

Maximilian Mielke