TU WIEN Informatics

Increasing Trustworthiness of Edge AI by Adding Uncertainty Estimation to Object Detection

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Daniel Hofstätter, BSc
Matrikelnummer 11807885

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof.in Mag.a rer.soc.oec. Dr.in rer.soc.oec. Ivona Brandić

Wien, 13. Oktober 2025

Daniel Hofstätter	Ivona Brandić

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

TU WIEN Informatics

Increasing Trustworthiness of Edge AI by Adding Uncertainty Estimation to Object Detection

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Daniel Hofstätter, BSc
Registration Number 11807885

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof.in Mag.a rer.soc.oec. Dr.in rer.soc.oec. Ivona Brandić

Vienna, October 13, 2025

Daniel Hofstätter	Ivona Brandić

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Daniel Hofstätter, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, haben ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT- Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 13. Oktober 2025

Daniel Hofstätter

Danksagung

Ich möchte mich bei Professorin Ivona Brandić für ihre Betreuung und Unterstützung bedanken, die es mir ermöglicht hat, verschiedene Forschungsgebiete frei zu erkunden. Mein Dank gilt auch meinen Laborkolleginnen und Kollegen und unseren interessanten Mittagsgesprächen. Mein Dank gilt auch unserem Burrito-Dealer, der mir jede Woche neue Kraft gibt. Ein besonderer Dank gilt all meinen wunderbaren Freunden, mit denen ich viele schöne Erinnerungen geschaffen habe und die es mir ermöglicht haben, das Leben auch in stressigen Zeiten zu genießen. Abschließend möchte ich Dr. Dimity Miller danken, die mir eine große Inspiration für meine Arbeit war und mir bei Fragen zu den Themen freundliche Unterstützung bot.

Diese Arbeit ist von der *Internet Foundation Austria* im Rahmen des *netidee Scholarship Calls #18* [1] ausgezeichnet und gefördert worden.

[1] https://www.netidee.at/increasing-trustworthiness-edge-ai-adding-uncertainty-estimation-object-detection

Acknowledgements

I would like to thank Professor Ivona Brandić for supervising me and supporting me all this time, allowing me to explore various research domains freely. Thanks also go out to my lab colleagues and our interesting lunch discussions. I would also like to acknowledge our local burrito dealer for replenishing my power to move on every week. Special thanks go to all my amazing friends, with whom I have made some great memories and allowed me to enjoy life even in stressful times. Lastly, I would like to express my gratitude to Dr. Dimity Miller, who served as a great inspiration for my work and offered me kind assistance when I needed clarification on the topics.

This thesis is awarded and funded by the *Internet Foundation Austria* through the *netidee Scholarship Call #18* [2]

[2] https://www.netidee.at/increasing-trustworthiness-edge-ai-adding-uncertainty-estimation-object-detection

Kurzfassung

Objekterkennung ist heutzutage eine stark nachgefragte Computer-Vision-Aufgabe und wird auch in sicherheitskritischen Bereichen wie autonomem Fahren, Fußgängererkennung und medizinischer Bildanalyse eingesetzt. Modelle ignorieren jedoch häufig die Unsicherheit ihrer Vorhersagen, d. h. sie können zu selbstsichere, falsche Vorhersagen liefern. Um vertrauenswürdige Modelle zu erstellen, muss diese Unsicherheit erfasst und sorgfältig behandelt werden. Kein Modell ist perfekt, und es kann immer Situationen geben, in denen es besser ist, überhaupt keine Vorhersage zu treffen, als eine bestmögliche Schätzung abzugeben.

Daher untersuchen wir in dieser Arbeit, welche Ansätze es Objekterkennungsnetzwerken ermöglichen, Schätzungen ihrer Unsicherheit zu generieren. Da diese Schätzmethoden in der Regel nicht kostenlos sind, muss ihr Rechenaufwand sorgfältig berücksichtigt werden. Wir konzentrieren uns auf Edge-KI, einen Bereich, in dem Echtzeitvorhersagen entscheidend sind, beispielsweise für das automatisierte Verkehrsmanagement in einer Smart City. Hier müssen Informationen mit der begrenzt verfügbaren Hardware direkt an der Quelle verarbeitet werden.

In unserer Arbeit haben wir die Literatur nach vielversprechenden Ansätzen durchsucht, verglichen und notwendige Anpassungen vorgenommen, um modernste Objekterkennung mit Unsicherheitsschätzung auf Edge-Geräten zu ermöglichen. Insgesamt haben wir vier verschiedene Ansätze zur Unsicherheitsschätzung implementiert und evaluiert. Diese umfassen eine einfache Basislinie, Ensemble-Netzwerke, probabilistische Näherungen und Lernen basierend auf der Evidenztheorie. Training und Evaluierung werden anhand von sechs verschiedenen Datensätzen zum autonomen Fahren durchgeführt, wodurch ein Domänenverschiebungsszenario zur Bewertung der Qualität der Unsicherheitsschätzung entsteht. Unsere Ergebnisse zeigen signifikante Leistungsunterschiede zwischen den Ansätzen und zeigen, dass komplexe Ansätze zur Unsicherheitsschätzung in vielen Fällen eine einfache Basislinie nicht übertreffen.

Abstract

Object detection is nowadays a highly demanded computer vision task, being applied also in safety-critical domains, such as autonomous driving, pedestrian detection, and medical image analysis. Yet, models very often remain ignorant of the uncertainty in their predictions, i.e., they may produce overconfident false predictions. To build trustworthy models, this uncertainty must be captured and handled carefully. No model is perfect, and there may always be situations in which it is better to make no prediction at all, rather than give a best-effort guess.

Therefore, in this thesis, we will explore what approaches enable object detection networks to generate estimates of their uncertainty. As these estimation methods typically do not come for free, their computational overhead has to be considered carefully. We focus on Edge AI, a domain where real-time predictions are crucial, such as for automated traffic management in a smart city. Here, information needs to be processed directly at the source with the limited hardware available.

In our work, we searched and compared the literature for promising approaches and made necessary modifications to them, enabling state-of-the-art object detection with uncertainty estimation on the Edge. In total, we implemented and evaluated four different approaches on uncertainty estimation, covering a simple baseline, ensemble networks, probabilistic approximations, and learning based on the theory of evidence. Training and evaluation are conducted across six different autonomous driving datasets, creating a domain-shift scenario for evaluating the quality of uncertainty estimation. We selected eight different metrics to evaluate the performance of our models in both object detection and uncertainty estimation. Our results show significant differences in performance between the approaches and that complex uncertainty estimation approaches may, in many cases, not outperform a simple baseline.

Contents

CHAPTER 1

Introduction and Motivation

In this chapter, we will cover the basic concepts necessary to understand object detection and why reasoning over the uncertainty of its predictions is important, especially for safety-critical, real-time applications on the edge of our networks. Building on that, the structure and aim of this thesis will be described at the end of the chapter.

1.1 What is Object Detection

Object Detection is a common computer vision task concerned with locating instances of specific classes (e.g., persons, cars) in images [ZCS+23], i.e., answering the questions "Where are objects in the image?" and "What category of objects are they?". It is considered a multi-task problem, as it simultaneously tackles the localization and classification of objects [CLZT24]. Nowadays, this problem is also prevalent in the domains of artificial intelligence (AI) and machine learning (ML), given their immense capabilities. The challenging aspect of this problem is that there is no predefined number of objects that may appear in an image, and the objects may come at arbitrary scales. An example of object detection is shown in Figure 1.1, where several people walking by a public space are being detected. The detector takes as input the image in Figure 1.1a and outputs the image in Figure 1.1b, where each detection is accompanied by a rectangular bounding box. On top of each bounding box, we can see the detected class label (e.g., *person*) and the confidence score (in the range $[0, 1]$), indicating how likely the detector thinks the prediction is correct. As exemplified in the figures, pedestrian detection is a common use case for object detection, with further usage being found in autonomous driving, robotic applications, medical image analysis, and many more [ZCS+23, CLZT24].

(a) Original Oxford Town Centre image. (b) Object detection applied.

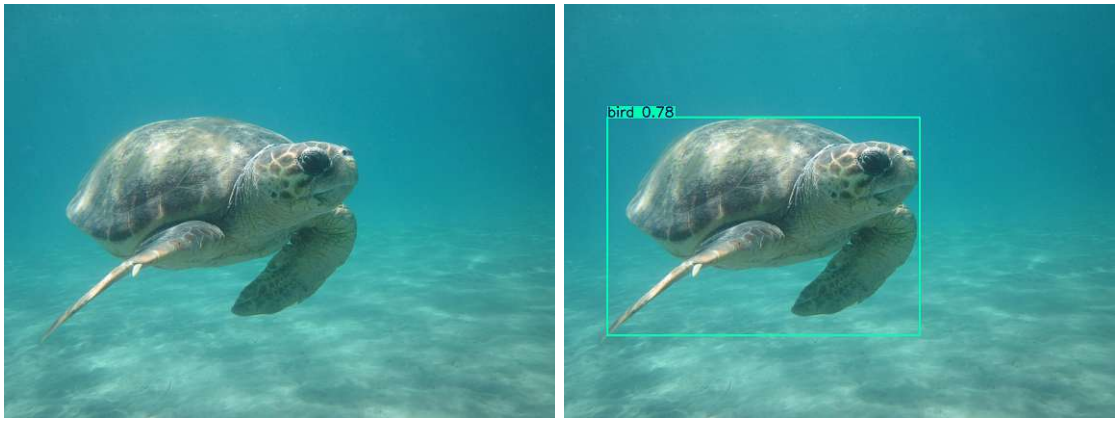
Figure 1.1: Prediction example of the YOLO11n [JQ24] object detector on a sample image from the Oxford Town Centre dataset [BR11].

1.2 Why Uncertainty Estimation

The deep neural networks (DNNs) commonly used for object detection struggle with trustworthiness in their predictions, as they are often not able to provide reliable uncertainty estimates, are overconfident, and are sensitive to shifts in the input data domain [GTA+23]. Therefore, just because a network generates an output, even if it is intended to be a probability score, it may not accurately represent the certainty of the prediction.

We give an example of what can happen when the detector becomes overconfident in Figure 1.2. On the left in Figure 1.2a, we take an image from a turtle underwater and pass it through a pre-trained detection model to produce Figure 1.2b. Clearly, the output here is wrong; a turtle is not a bird. But why did this happen? The used detection model YOLO11n [JQ24] has been pre-trained on the MS COCO [LMB+14] dataset, containing thousands of images for 80 different object class labels. However, despite the vastness of the dataset, turtles underwater are not represented in it, and so the detector cannot know what it is seeing in the underwater image. Still, given an input image, an object detection model will produce an output, regardless of whether it was trained to do so or not. The goal here is to determine when the model is being overwhelmed, i.e., when it is uncertain, so that it can be indicated when a prediction can be trusted and when not.

To obtain uncertainty estimates along with a regular prediction, typically, further effort is required, meaning additional computation is often necessary [Mil21]. Uncertainty estimation can be applied to various machine learning tasks, including classification and object detection. Furthermore, obtaining the uncertainty for model outputs not only improves the trustworthiness of predictions but also enables further downstream machine learning tasks, such as active learning and reinforcement learning [GTA+23]. In this work, we will explore various uncertainty estimation methods, each with its own requirements for adapting the model network and the training procedure.



(a) Original turtle image. (b) Overconfident false detection.

Figure 1.2: YOLO11n [JQ24] prediction on an image from a turtle dataset [Sha24]. The detector has not been trained to recognize turtles and mistakenly identifies this instance as a bird.

1.3 Running AI on the Edge

In many cases, AI applications must provide results in real-time (with low latency) and often operate under data privacy constraints. The defining paradigm here is called Edge AI [GGH+24]. For example, autonomous vehicles must process vision data of their surroundings without significant delay to make timely safety-critical decisions, and health data of people may be too sensitive to be transferred over certain networks. For a safety-critical and latency-sensitive Edge AI use case, *InTraSafEd5G* [LMP+21] can serve as an example. Here, an object detection edge device with a camera was mounted on a traffic light in the city of Vienna to notify drivers approaching the intersection about pedestrians crossing the street, which may appear in the driver's blind spot. This process, from capturing the image of the traffic scene to running it through the Edge AI model to sending it to the driver's smartphone, is heavily constrained. Not only must everything occur within latency targets (e.g., 100ms) on computationally low-power hardware (e.g., a Raspberry Pi), but it also must be processed locally to meet the privacy constraints of traffic participants. Such systems greatly benefit from uncertainty estimation, as it can help further strengthen trust in automated AI systems, which are emerging in various domains of our everyday lives.

1.4 Problem Statement and Aim of this Thesis

To summarize, object detection has already been widely studied and applied. As a machine learning task, the training of detection models heavily relies on massive datasets and large investments of time and computational power. However, as all these resources are always limited, an AI can typically not be perfect in its task. Therefore, mistakes

in predictions will still happen. Uncertainty estimation, although it cannot provide the correct answer, can help mitigate overconfident false predictions by making the boundaries of the model's knowledge clearer. While uncertainty estimation for object detection is already a challenging problem on its own, imposing real-time requirements and limitations on the computational power of the hardware highlights a further, still little-explored subdomain of research.

Therefore, in this thesis, we will study the combined aspects of object detection with uncertainty estimation on the Edge. Three main research questions are formulated as follows:

1. *How can state-of-the-art uncertainty estimation techniques be effectively implemented for real-time object detection on the Edge?*

   Here, we aim to filter literature into a subset of approaches that are promising candidates for an edge use case. The goal is not to invent a new methodology, but rather to find suitable combinations of existing approaches.

2. *What are the trade-offs regarding object detection quality and computational efficiency for additional uncertainty estimation?*

   We want to quantify the impact of additional uncertainty estimation. How high is the penalty in computational time, and are the modifications affecting the performance of the base model? We start with a well-rounded baseline in detection quality and prefer not to drop significantly below that.

3. *What recommendations can be made to enhance the trustworthiness of detection models at the edge?*

   This last question ties together the answers from the previous ones. Based on our results, we will determine which uncertainty estimation technique is best suited for practical settings. We further discuss the additional challenges that arise, such as network modifications and model retraining, and how they might influence the choice of using uncertainty estimation.

1.5 Methodology

Prior to this work, during the proposal stage of the thesis, a research gap was identified and formalized into the aforementioned research questions. Going from there, the process of this thesis can be structured into:

1. **Literature Review**: We start off with an extensive literature review on the topics of object detection, uncertainty estimation, and Edge AI applications. The main search tool was Google Scholar, and Zotero served as a literature management system. Following the guidelines of Kitchenham and Brereton [KB13], a literature selection approach inspired by forward and backward snowballing has been applied to identify

key literature references, i.e., searching citations further into the future and past from existing works. The goal is to narrow down the state of real-time uncertainty estimation and object detection and identify possible contributions for resource-constrained edge environments. Several larger surveys, such as [GTA+23, CLZT24, ZCS+23, VV24], provide well-structured sources for background information on the entire domain. The scope of the literature will be further refined through experiments, such as those involving hyperparameter selection. A broad overview of the literature will be covered in the following Chapter 2, which serves as a basis for design decisions throughout this thesis.

2. **Model and Method Combinations**: The first step was to find a suitable object detection model, which in our case is the YOLO11 [JQ24] real-time capable detection model. This provides a fundamental basis for all experimental modifications. Furthermore, we research how uncertainty is defined and how it can be measured. We select uncertainty estimation methods based on their computational efficiency and make implementation choices to further minimize their runtime overhead. A key aspect here is to make modifications only in the necessary parts of the object detector, thereby maintaining its base detection quality as far as possible and introducing additional processing only when necessary. Our focus is on classification uncertainty for object detection, leaving the localization aspect of objects unchanged. We narrowed down our selection of uncertainty estimation methods and integrated them into our chosen base detection model to evaluate four different versions. A deeper explanation can be found later in Chapter 3, with a brief summary here as:

   a) **Baseline** [JQ24]: We take the base model without modifications and only re-interpret its output as a proxy uncertainty estimation approach.

   b) **Ensemble** [VT19]: We replicate network components and compare their disagreement on the same input.

   c) **MC Dropout** [YKPC24]: Enabling uncertainty estimation by sampling a probabilistic network multiple times.

   d) **EDL MEH** [PCK+23]: Evidence learning allows for efficient uncertainty estimation in this approach through direct prediction.

   By employing several different methods, we can effectively address our first research question, namely, verifying the feasibility of uncertainty estimation in our Edge AI setting.

3. **Dataset and Use Case Selection**:

   Uncertainty may arise when a model encounters novel input [GTA+23]. Since training data is finite and the open world is full of previously unseen inputs, such situations are unavoidable. We aim to replicate a similar scenario in our evaluation framework. We chose a domain-shift scenario [GTA+23], where our model is trained on one data domain and then evaluated on a different domain. Autonomous driving

additionally fits the edge environment, as a risk-sensitive and real-time application of object detection. Therefore, we selected several autonomous driving datasets and constructed a domain-shift scenario based on them. We achieve this by first training the models on the datasets of:

a) **Cityscapes** [COR+16]: Featuring traffic scenes in central Europe.
b) **KITTI** [GLU12]: Driving sences from a German city.

These two datasets primarily depict clear skies and good driving conditions. We now introduce a domain shift by evaluating the model on datasets with more diverse scenery and various weather conditions as well. There are four validation datasets withheld from training and used exclusively to evaluate the model:

a) **Foggy Cityscapes** [SDVG18]: Artificial fog overlay on images simulates a different domain based on changing weather conditions.
b) **RainCityscapes** [PMBN20]: Rain drops and streaks are layed over the image.
c) **BDD100K** [YCW+20]: Diverse weather conditions in a large-scale real-world dataset.
d) **nuImages** [CBL+20]: Another large dataset, featuring multiple camera angles.

Noteworthy, we aim to preserve the pre-trained weights of our detection models to maintain their generalization capability. Therefore, a label processing is required to bring the labels of all datasets to a homogeneous format that the detector already understands. More information will be presented later in Chapter 4.

4. **Evaluation Scope**: In our evaluation setup, we train a model on one training dataset first, before testing it on all of the validation datasets. Consequently, for our four model approaches, two training sets, and four validation sets, this results in an extensive grid of $4 \times 2 \times 4$ evaluations. Furthermore, we select a diverse set of metrics. For general object detection performance, we measure:

   a) **Precision** [OD08]: Good when few false positive detections are made.
   b) **Recall** [OD08]: Good if all objects are found, regardless of precision.
   c) **Mean Average Precision** [EVGW+10]: The area under the precision-recall curve as a balance of those two metrics.
   d) **Framerate**: Measures the processing speed in images per second that the model can handle.

   Additionally, our models generate a classification uncertainty value for each detection. We selected multiple suitable metrics from the literature:

   a) **Minimum Uncertainty Error** [MDMS19]: Given an uncertainty output, how well can it be used to distinguish correct from incorrect detections?

b) **Area Under the Receiver Operating Characteristic** [DG06]: How do true positive rate and false positive rate behave together?

c) **False Positive Rate at 95% True Positive Rate** [DWGL22]: Our recall is at 95%, what is the false positive rate then?

d) **Excess Area Under the Risk-Coverage Curve** [GUEY19]: Reasoning about the calibration of uncertainty estimates.

All those metrics will be explained in more detail in Chapter 5. With this large set of metrics, we can address the second research question regarding trade-offs in estimation techniques by comparing performance across metrics and models.

5. **Training and Evaluation Runs:** A systematic search was conducted using Ray Tune [LLN+18] to identify optimal hyperparameters for each uncertainty estimation approach. The final parameter sets were selected by analyzing trade-offs between uncertainty and detection qualities, as well as their impact on runtime. Training was performed using transfer learning from MS COCO [LMB+14] pretrained weights. Only parts of the modified detection heads were unfrozen from parameter updates, allowing training to focus on our implemented approaches. The details about hyperparameters and search spaces are listed in Section 5.2. A large-scale evaluation across various datasets and models was conducted to generate results data for analysis. In Chapter 6, the results will be presented through visualizations, such as scatter plots and bar plots, that offer high-level insights, accompanied by additional detailed values in some tables.

6. **Discussion and Interpretation:** Lastly, the results will be critically evaluated, and the research questions will be answered accordingly (see Chapter 7). In addition to an overall conclusion, a summary of the major limitations in Chapter 8 will reveal the remaining weaknesses of this work, while also providing a starting point for future research.

The code used to produce results in this thesis is public on GitHub [1] to allow for reproducibility. In the following chapter, we will delve further into the theory of the main science subdomains of this work, providing a more detailed overview that contrasts with the high-level overview presented in this chapter.

[1] https://github.com/Sokadyn/yolo-edge-uncertainty

CHAPTER 2

Background

The fields of object detection and uncertainty estimation are very diverse and expand in various directions. Here, we will explore some of the many methodologies, compare their trade-offs and use cases, and ultimately provide the choices of approaches considered for evaluation within the scope of this work.

2.1 Object Detection

First, we examine the definition of object detection and explain how it differs from other computer vision tasks. We then examine the research development and the various branches established over the past decade. After narrowing down the detection framework suitable for addressing the problem statement in this thesis, a brief overview will explain the components of the network, with some parts subject to modification later in this work.

2.1.1 Definition of Object Detection

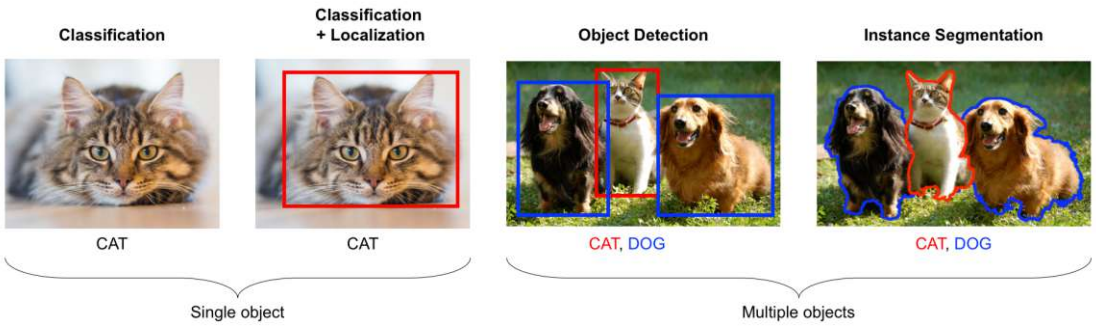


Figure 2.1: Common computer vision tasks. Illustration from [Sha18].

Computer vision encompasses many sub-tasks, with the desired level of understanding of what is shown in an image differing with each task. Some of them are visualized in Figure 2.1. Image-level object classification assigns class labels for the entire image, while localization additionally tries to find a bounding box that tightly encloses the found object [LOW+20]. The class labels come from a finite list of predefined instance categories (e.g, *cat*, *dog*, *person*, *car*, etc.). Localization can be seen as a regression task of predicting continuous values that define bounding boxes [LOW+20]. These boxes are usually defined by several parameters measured in the pixel space of the image, e.g., by the center, width, and height dimensions within the image pixel grid [RDGF16]. Object detection further generalizes this idea of classification and localization to multiple instances within an image [LOW+20]. Lastly, instance segmentation takes it one step further and estimates the object instance class label of each pixel in the image individually [LOW+20]. The concerned computer vision task addressed in this thesis is object detection, with the focus on 2D images in the RGB color space.

2.1.2 History of Object Detection

Object detection has seen steep progress over the last decades. According to Zou et al. [ZCS+23], a brief roadmap can be summarized as follows:

- **Hand-crafted Detectors**: At the beginning of the 21^{st} century, computational resources were limited and object detection was dominated by detectors with hand-crafted features, like the Viola-Jones detectors for detecting human faces, or the Histogram of Oriented Gradients (HOG) detector focusing on pedestrian detection. Although these detectors had a remarkable performance at the time, they soon reached their limits.

- **Two-Stage Detectors**: With the advancements in computation and the increasingly popular Deep Neural Network (DNN) models, Convolutional Neural Networks (CNNs) took over image processing. Starting in 2014, the most accurate object detectors were defined by the two-stage paradigm, where, in the first stage, region proposals were generated for the entire input image, and in the second stage, these proposed regions were checked for the presence of objects. One representative model that can be named here is the Faster R-CNN detector [RHGS15]. Despite their accuracy in detection, these types of detectors still struggle with real-time performance due to the inherent bottleneck of the two-stage paradigm.

- **One-Stage Detectors**: To solve the computational performance limitations of two-stage detectors, the tasks of region proposals and identifying objects were unified into a single forward pass through the DNN. This network was primarily a CNN building a Feature Pyramid Network, where each layer was responsible for extracting features from the image at varying granularities, from very fine grids of features in the shallow layers to increasingly coarse grids in the deeper layers. Building on these extracted features, the detection head of the network attempts to

locate objects simultaneously at each grid cell of the feature maps. Two prominent models here are the Single-Shot Multibox Detector [LAE+16] and the You Only Look Once [RDGF16] detector, where the latter has evolved into a series of model versions, ongoing to this day. Especially for real-time detection, one-stage detectors have become the focus of research.

Current state-of-the-art real-time object detectors include YOLO11 [JQ24], released in 2024 as a CNN-based one-stage detector, and YOLOv12 [TYD25] as a transformer-based alternative. Newest YOLO models come in different scale variants of the networks, indicated by a suffix after the name (*n* for *nano* until *x* for *extra-large*) [JQ24]. When building the model network, layer depths and sizes are automatically adjusted to the scaling, where larger models provide higher precision through an increased number of parameters, at the cost of slower processing speeds. Nano-models (e.g., YOLO11n) go the other direction, trading a certain amount of precision for the highest image processing speeds. Although transformer-based detectors have been dominating the MS COCO detection benchmark [ZCS+23], a recent study has suggested that for real-time applications, YOLO11n is the most suitable due to having the lowest inference times with similar detection performance to comparable models when comparing YOLO versions 8 to 12 [SMC+24]. Therefore, YOLO11n will be the choice for evaluation models in this thesis. From here on, we will be discussing only deep learning models for object detection. We are concerned with two different phases of using a model: (i) the training phase, where model weights are being optimized on a training dataset through passing data forward through the network and then propagating the calculated loss back, and (ii) testing time (also inference time), where model weights are frozen and a test dataset (or a single image) gets passed through the network in order to obtain object detection results, which can also be validated against ground truth if available.

2.1.3 The original YOLO Detection Framework

The original YOLO detector was proposed by Redmon et al. [RDGF16] in 2016. Figure 2.2 shows a basic overview of how YOLO works. The detector first passes the input image through a series of convolutional layers of the neural network. The output is then an $S \times S$ grid (e.g., 7×7), reducing the higher-resolution input pixel space to a lower scale [RDGF16]. This output is also referred to as a feature map [SSRRR24], as it extracts condensed image features from the original high-detailed input, enabling more efficient processing. The defining concept of one-stage detectors is to detect all objects simultaneously [ZCS+23]. Therefore, each grid cell now individually predicts bounding boxes and class probabilities, resulting in a multitude of potential detections across the entire image. Each bounding box comes at a 5-tuple of (x, y, w, h) plus a confidence score in the original YOLO detector [RDGF16]. Here, x and y represent the object center coordinates, and w and h are the predicted width and height of the object, respectively. The fifth value represents confidence, indicating the likelihood of an object being present in that specific box. Moreover, each grid cell predicts B bounding boxes simultaneously,

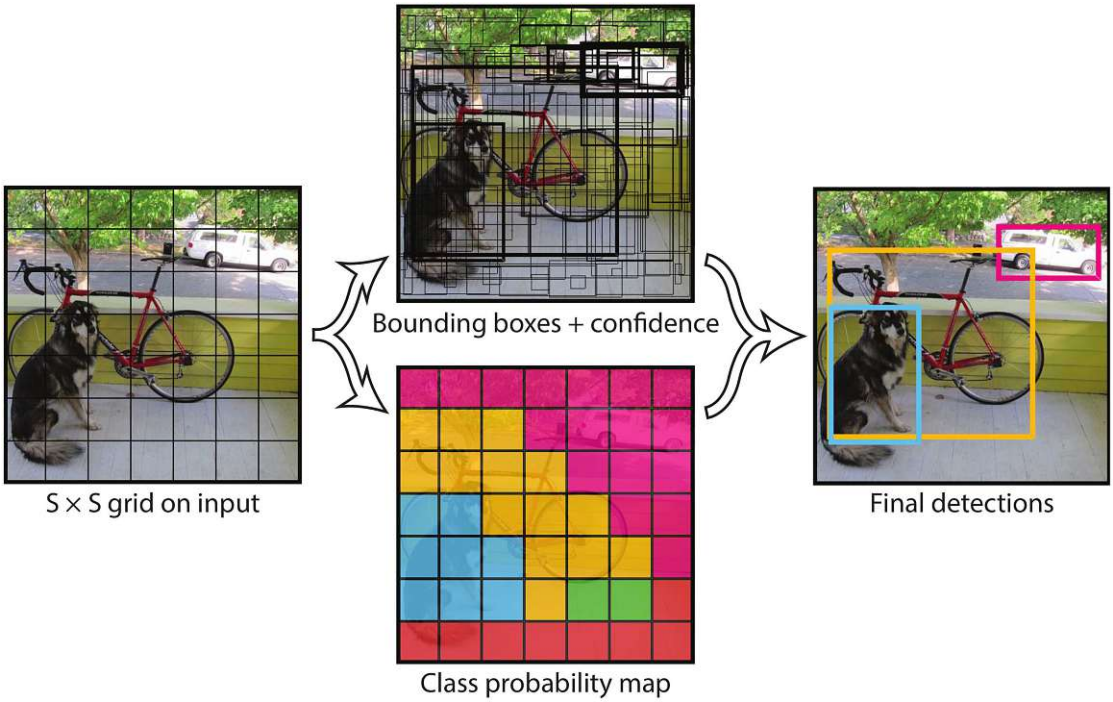


Figure 2.2: First version of the YOLO detector. Illustration taken from the original work of [RDGF16].

each capable of taking on a different shape during prediction, thereby further increasing the detection rate. Finally, there is a vector of classification probability scores C per grid cell that represents the likelihood of each class being detected. For example, in the PASCAL VOC dataset [EVGW+10, EW11], there are 20 different classes of objects to be detected, resulting in a vector C with the same number of probability scores. Counting all output values, this results in a total of $S \times S \times (B * 5 + C)$ detections for the entire feature map [RDGF16]. As illustrated in the top center of Figure 2.2, having this many detections results in a large number of bounding boxes across the entire image. That is why many object detection frameworks employ non-maximum suppression (NMS) as a post-processing step [VV24], keeping only the most confident predictions as final detections while filtering out redundant, overlapping ones with lower classification scores. Throughout the thesis, we will be referring to the process of passing an input sample through the entire network as inference. Additional steps that accompany inference include pre-processing (e.g., scaling of the input image) and post-processing of the outputs, such as NMS for object detection.

2.1.4 Advancements of YOLO

One of the latest versions, YOLO11 [JQ24] (sometimes also referred to as YOLOv11), follows the footsteps of the original YOLO [RDGF16] as a CNN-based one-stage object

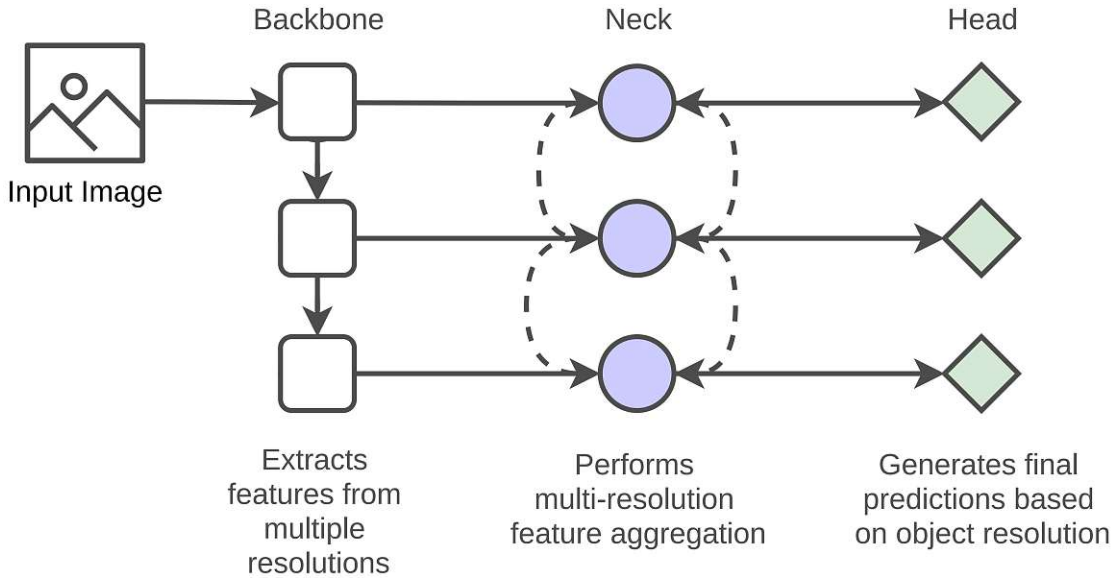


Figure 2.3: Typical YOLO architecture. Illustration from Kateb et al. [KMH+21].

detector, but has undergone numerous architectural updates over the last decade. Starting with YOLOv3 [RF18], the networks became more modular by splitting them into the backbone, neck, and head parts [VV24]. Figure 2.3 illustrates a high-level overview of typical YOLO object detection networks. The backbone, primarily a CNN, takes in an input image and serves as a feature extractor at multiple scales, enabling detection of objects ranging from very small to large, spanning the entire image. Commonly, the backbone stems from a standalone image classification network, which, instead of classifying the whole image, now forwards raw features to further stages in the object detector [SSRRR24, KMH+21]. The neck then takes extracted features from the backbone and aggregates them to further enhance semantic information gain using a Feature Pyramid Network (FPN) [VV24]. Lastly, detection heads are connected to the outputs of the neck part, producing the final detections for objects at various resolutions. Compared to its origins, YOLO11 outputs only one detection per grid cell, where each detection comes with $4 + C$ values, four values for bounding box localization, and C classification scores for each class label available. Recent YOLO versions also come with interchangeable heads, allowing for the handling of various computer vision tasks beyond object detection, such as classification, segmentation, and pose estimation [SSRRR24, VV24, JQ24].

2.2 Uncertainty Estimation

DNNs are now widely used in various safety-critical fields, including medical analysis and autonomous vehicles [GTA+23, WSLL25]. Still, their lack of expressiveness, transparency, and inherent inability to provide uncertainty estimates make them vulnerable to shifts in the input data domain the network is seeing, as well as adversarial attacks [GTA+23].

Therefore, estimating the uncertainty of predictions is a step towards more trustworthy models. However, it is essential to understand the sources and categories of uncertainty.

2.2.1 Types of Uncertainty

Concerning machine learning, it is common to distinguish between two main types of uncertainty, (i) *aleatoric* and (ii) *epistemic* uncertainty for model outputs [SBD+14]. These two types contribute to the total uncertainty of a prediction generated by a model, also known as predictive uncertainty [GTA+23]. Aleatoric uncertainty (also known as data uncertainty) can be seen as a statistical component and randomness in the input, and epistemic uncertainty (also called model uncertainty) as the systematic uncertainty in the model itself [KG17, GTA+23]. Senge et al. [SBD+14] provide simple real-world examples for explaining both uncertainty types. They state that aleatoric uncertainty can be related to a coin flip, which is an inherently random process, and no amount of additional information will enable us to predict with certainty which outcome a flip will have. In contrast, epistemic uncertainty can be present when a doctor makes a medical diagnosis of a disease with limited knowledge about it. Therefore, the doctor may want to consult other experts or seek additional knowledge in the literature to reduce the uncertainty in the diagnosis. In deep learning, aleatoric uncertainty represents the noise inherent in the sensor generating the input data (e.g., camera), the captured resolution, or uncertainty in the labeling of the training data itself, whereas epistemic uncertainty indicates insecurities in the choice of model architecture, parameters, and training routine [WSLL25]. Furthermore, when encountering novel out-of-distribution data, which is unlike the data encountered during model training, epistemic uncertainty is high, but aleatoric uncertainty is not affected [KG17]. Consequently, epistemic uncertainty can arise when a model is being overwhelmed and truly does not know what prediction to make. Moreover, an important distinction is that model uncertainty can be reduced by using more training data, while data uncertainty persists even after seeing an infinite number of data samples [KG17].

The datasets used to train a model may stem from various domains of data. This can be exemplified by comparing a dataset featuring undersea images with turtles [Sha24] to common objects in our everyday lives [LMB+14]. Not only is the environment setting entirely different, but even the object classes appearing in both datasets are mutually exclusive. This provides a challenge for detectors trained on one data domain, but then applied to the other. One research field related to this problem is domain adaptation, which concerns adapting models to new data domains [OSVP24]. Looking at the total predictive uncertainty from another perspective, further categorizations of predictive uncertainty can be distinguished regarding the domains of data the model encounters during and after training, as discussed by Gawlikowski et al. [GTA+23]:

- **In-domain Uncertainty**: Here, the data used to train the model and data inputs after training come from the same domain. Uncertainty may still arise due to the

limited availability of training data and a non-optimal choice of model architecture and parameters, indicating that both data and model uncertainties are present.

- **Domain-shift Uncertainty**: Further divergence between train data and data the model is being tested on represents a shift in the data domain. Some examples here are occlusions of objects and additional noise in the input image. The model can still apply its in-domain knowledge to some extent and make predictions on the new domain. Uncertainties here can be reduced by covering more of the shifted domain in the training data, although this may not be feasible due to the vastness of different possible domains.

- **Out-of-domain Uncertainty**: This uncertainty appears when the model faces truly unknown data not encountered before during training. For instance, asking the detectors what they see on an image with a turtle, while the class label for a turtle is not even available among its designed outputs. These kinds of samples are also referred to as out-of-distribution (OOD). The in-domain knowledge gained so far is no longer of help to the model. Consequently, the model should no longer provide a prediction, as it cannot be trusted.

While data uncertainty can capture only in-domain uncertainty, meaning that the uncertainty from the noise of the data (label ambiguities, occlusions, image noise), model uncertainty captures all in-domain, domain-shift, and out-of-domain uncertainty [GTA+23]. Within the scope of this thesis, we will focus on domain-shift uncertainty due to its high relevance in many practical applications, such as autonomous driving. Examples of domain shift scenarios include changes in weather conditions from clear to foggy, or the use of synthetic rendered images compared to real-world images [OSVP24]. Furthermore, in this work, we will focus on capturing the full predictive uncertainty while exploring various methods to achieve this.

2.2.2 Uncertainty Estimation Techniques

Various approaches have been proposed to estimate the uncertainty of DNNs. Gawlikowski et al. [GTA+23] broadly categorize these techniques into four groups: (i) Bayesian inference methods, (ii) ensemble methods, (iii) test-time augmentation approaches, and (iv) single deterministic models that incorporate uncertainty. Each category has distinct characteristics and trade-offs, coming with both advantages and limitations. Moreover, the concrete formulation of the technique varies for the underlying machine learning task. Regarding object detection, the regression of bounding box parameters and the classification of the underlying objects within those boxes are of concern. This thesis will exclusively tackle the classification aspect, i.e., estimating uncertainty for the class label of every object.

The following description of uncertainty types is mostly based on the detailed explanation found in [GTA+23]:

- **Bayesian Inference**: Common DNNs have static model weights for their neurons and behave deterministically, i.e., the same input results in the same output every time. However, Bayesian neural networks (BNNs) model these network parameters not as fixed values, but as stochastic random variables. Given a dataset and an initial prior assumption of the model weights, using Bayes' rule, a posterior distribution over the model weights is estimated. As a result, the model does not use a single set of weights; instead, it samples an individual set of weights from a probability distribution every time an input is processed. Since processing an exact Bayesian inference proves to be intractable in practice, Monte Carlo (MC) Dropout [GG16] serves as an approximation. Dropout [SHK+14] is a regularization method for neural networks for reducing the risk of overfitting. During training, Dropout randomly drops some connections in the neural network to force the input to take a different path through the model, thereby helping the model generalize better. For MC Dropout, this process is also employed at test time, where the same input is passed through the network multiple times, generating several non-deterministic outputs. This process is also categorized as variational inference. Uncertainty estimates can be retrieved by examining the distribution of predicted values among the output samples, where a high spread indicates high uncertainty. This approach allows for capturing the full predictive uncertainty, including data and model uncertainty.
- **Ensemble Methods**: Similar to the Bayesian approach, ensemble methods [LPB17] can provide uncertainty estimates by reasoning over non-equal outputs for the same input. However, instead of sampling from a single stochastic BNN, ensembles contain multiple individual deterministic DNNs. To enable variability in the outputs, the ensemble members must be different, sometimes in architecture, but varying only in model weights is also sufficient. A simple way to ensure diversity after training each model is to introduce a stochastic element into the training routine, such as random weight initialization or randomly applied data augmentation. Uncertainty intuitively represents the level of disagreement among the ensemble members for the same input. Both aleatoric and epistemic uncertainty can be estimated.
- **Test-time Augmentation**: Data augmentation during training of a network is a common practice for further strengthening the model's robustness. Basic augmentations for images include, for example, brightness adjustments, image scaling, or image flipping [KKM21]. Using the same augmentation strategy at both training and test times allows us to artificially generate multiple data samples of the same original input and pass them through a deterministic network. The core idea is similar to the Bayesian and ensemble approaches, where a diverse set of outputs serves as the basis for uncertainty estimation. However, a major difference of this approach is that it focuses on data uncertainty and does not capture model uncertainty as well.
- **Single Deterministic Models**: These types of models only need one forward pass to estimate uncertainty and always produce the same output for the same

input. This is made possible by directly predicting a higher-order distribution over the desired output and utilizing the predicted parameters to calculate uncertainty estimates. One branch of research that falls into this category of uncertainty estimation is Evidential Deep Learning (EDL), where the model is tasked with learning and predicting the parameters of a higher-order distribution based on evidence. The model gains evidence during training and uses it at test time to check how much evidence the new input can be assigned to. If the model issues low evidence, then uncertainty is high, because the model recognized an irregular input that it cannot handle. EDL has been defined for classification [SKK18] and regression [ASSR20]. For the classification task, the target output is a categorical distribution of all available class labels, indicating the likelihood of each one being correct. In EDL for classification, a Dirichlet distribution serves as a higher-order prior distribution, i.e., a distribution over categorical distributions. Consequently, both epistemic and aleatoric uncertainty can be calculated from the properties of the higher-order distribution in a post-processing step. In the case of regression, a Normal-Inverse Gamma distribution serves as an appropriate prior to directly model uncertainty estimates.

Comparing those four categories, it stands out that all methods, except single deterministic models, come with inherent processing overhead due to the requirement of generating multiple samples. In a comparison of deep ensembles and MC Dropout conducted by Gustafsson et al. [GDS20] up to 16 samples were generated for uncertainty estimation. Although providing uncertainty estimation, the authors noted that the real-time capability is limited under these conditions, as inference time scales linearly with the sample number. Test-time augmentation suffers from similar inference overhead drawbacks, as it requires processing an input multiple times within the network. Furthermore, it is the only method listed here that does not provide the full picture of predictive uncertainty, missing out on model uncertainty. For real-time applications, single deterministic models are a suitable choice, as they come only with minimal computational overhead for post-processing uncertainty, eliminating the need for multiple model forward passes. However, especially EDL networks prove to be very sensitive to model architecture, training routine, and training dataset, potentially failing to provide well-calibrated uncertainty estimates [GTA+23].

So far, we have discussed uncertainty estimation in general, with examples for regression and classification. The multi-task nature of object detection makes the concrete implementation of uncertainty estimation more challenging. Not only are both tasks of classification and regression present, but also typical one-stage object detectors tackle those tasks thousands of times across an image at once, given their multitude of feature map grids cells [DAT23]. In the scope of this thesis, we focus on the classification uncertainty of detected objects and compare different approaches, while keeping real-time applicability for Edge AI applications in mind. In the following sections, we will discuss the suitable methods and their implementation. An abstract representation of the chosen approaches we have discussed so far is given in Figure 2.4.

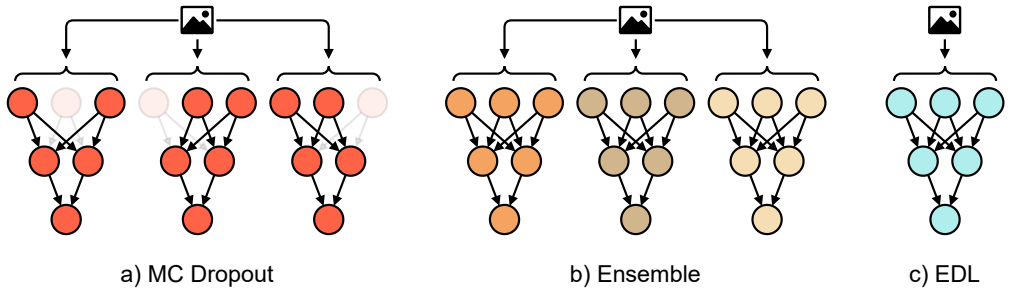


Figure 2.4: Chosen uncertainty estimation networks: a) Monte Carlo (MC) Dropout as a Bayesian approach, b) Deep ensembles of networks, c) Evidential Deep Learning (EDL) as a single deterministic network. Illustration re-drawn with inspiration from [GTA+23, AS21].

2.3 Related Work

In this section, we provide a review of related works based on the categorization of approaches discussed thus far. Furthermore, we will highlight where they differ from our work and where concepts are adopted for our implementation.

2.3.1 Stochastic Approaches

Srivastava et al. [SHK+14] introduced Dropout, a method capable of reducing overfitting in neural networks. Later, [GG16] adopted this approach and used it to approximate Bayesian models, laying the foundation for stochastic sampling-based uncertainty estimation at inference time. Consequently, regarding Bayesian models, MC Dropout stands out as a popular and widely used method, and will therefore also be considered for evaluation in this thesis, in a more recent variation.

Miller et al. [MNDS17, MNDS18, MSZ+19, MDMS19, Mil21] have done lots of foundational work in porting MC Dropout to object detection, while focusing on the problem of open-set object detection. Still, in their research, they noted the computational overhead of sampling-based uncertainty estimation methods. To enable MC Dropout, dropout layers must be inserted into the network. A common choice is to apply dropout before the detection head and leave the prior network's parts (backbone and neck) deterministic [Mil21]. This enables caching of the model layers before the dropout layer, and only requires sampling a fraction of the network (head) multiple times, a practice already employed in the work of [ZWX+24]. We will also take notes here, as these are highly influential decisions on inference time. Therefore, the design of network layer caching and applying Dropout only in the later layer will be chosen as representative for Bayesian uncertainty estimation in this thesis.

Further refinements of the Dropout technique have been proposed in DropBlock [GLL18], a structured Dropout technique designed for CNNs. Unlike standard Dropout, which randomly removes individual activations, DropBlock drops contiguous spatial regions

within the feature maps. Since feature maps in convolutional networks such as YOLO typically have the shape (W, H, C) [DAT23], with W and H denoting width and height and C the number of channels, DropBlock eliminates whole rectangular sub-regions across $(W \times H)$, thereby encouraging the network to learn robust features. When using this DropBlock variant at test time, this approach can also be referred to as Monte Carlo (MC) DropBlock, as it has been previously employed in the work of [YKPC24]. Consequently, we will choose this Dropout variation as one of our experimental approaches, in combination with the layer caching strategy discussed earlier.

Zhao et al. [ZWX+24] add DropBlock to the YOLOv5 model for enhancing safety-critical object detection scenarios, such as autonomous driving. They also take real-time applicability into account by caching static network predictions before the multiple forward passes required for the Monte Carlo method. While caching indeed provides a significant performance gain over regular multiple forward passes, their evaluation suggests that the inference time is still twice that of the unmodified single forward pass model. Depending on the available hardware and requirements for the frame rate for processing input images, this may or may not be sufficient. In any case, their work demonstrates that MC Dropout (DropBlock) remains a viable option. Our implementation will follow a similar approach. However, we will be using the much more recent YOLO11 model, conducting a hyperparameter search, employing a more fine-grained layer freezing strategy, but instead only focus on classification and not localization optimization as well.

2.3.2 Ensemble Networks

Deep Ensembles, although less common than MC Dropout, have also seen usage in uncertainty estimation for object detection [Mil20, MSZ+19]. To make them real-time capable, instead of ensembling entire networks, we can build ensemble members with multiple detection heads, while leaving the remainder of the network common to all ensemble heads. This approach is inspired by the concept of Deep Sub-Ensembles [VT19], and will also be considered in the evaluation of this thesis for object detection.

2.3.3 Evidential Deep Learning Methods

EDL methods have demonstrated that sample-free deterministic uncertainty estimation is possible. An early implementation of EDL for object detection can be found in the thesis of [Ros19], where EDL-YOLO was created by applying EDL for classification [SKK18] to the YOLOv2 [RF17] detector. Still, EDL-YOLO struggled to match the precision of pre-trained baseline networks without uncertainty quantification. Park et al. [PCK+23] also noted similar problems and proposed splitting the evidential learning aspect into a separate part of the network, known as the Model Evidence Head (MEH). Consequently, the base object detection network produces classification scores as originally designed, and the MEH focuses solely on predicting evidence, which is then used to estimate uncertainty. The only requirement here is to have one more neural network output to predict per object, in addition to bounding box parameters and classification scores.

In our evaluation, the EDL MEH implementation will represent the category of single deterministic models. We will not be further exploring test-time augmentation methods, as they do not capture the full picture of uncertainty.

2.3.4 Other Works and Approaches

Here, we will mention a few other works from the domain of uncertainty estimation that did not make it into the evaluation scope of this thesis. Most of the time, their uncertainty estimation is targeted towards a different use case, or the computation overhead remains either unclear or too high. Still, some inspirational ideas can be observed.

The work of [VOP23] explores Online Source-Free Domain Adaption (Online-SFDA) for object detection. By utilizing a transformer-based memory module, the model is adapted from the source data domain to the target domain in an online manner. Additionally, a mean-teacher framework reduces the impact of noisy pseudo-labels, helping the model generalize better in the target domain. Regarding our concerned edge use case, the approach of [VOP23] suffers from heightened computational latency due to its reliance on a two-stage detection model. It is unclear how their method would perform when tailored to a more efficient one-stage detector.

Fully Test-time adaptation [RT24] is a strategy that seeks to improve object detection models after deployment continually. At test time, the model is updated on each input before making the prediction, with the limitations that neither the source training set nor the distribution or labels of the encountered target dataset are available. Therefore, there is a strong reliance on pseudo-labels to improve the models. To increase the quality of pseudo-labels, Ruan and Tang [RT24] propose a new Intersection over Union (IoU) filter that selects only the most confident labels based on consistent predictions on the same image over multiple iterations. This approach utilizes uncertainty estimation to filter out noisy pseudo-labels and exhibits some similarities to the method pursued in this thesis. However, [RT24] first uses a slower two-stage object detection network and then relies on multiple iterations (forward passes) on the same input to utilize their filtering strategy. Therefore, their framework, without major modifications, is not suitable for our edge scenario.

Wang et al. [WZZF24] tackle the challenge of adopting Evidential Deep Learning (EDL) to object detection by implementing customized loss functions and annealing training strategies. Their proposed solution balances uncertainties of foreground and background objects to enable efficient and effective Open Set Object Detection (OSOD), where unknown detections get labeled as such, even during test time. The use of EDL facilitates fast inferences; however, its reliance on a two-stage object detection base network still leaves untapped potential for improved performance. It is unclear how their approach would perform in a fast one-stage detector. It is worth considering for potential modification and re-evaluation, but this will be beyond the scope of this thesis.

A different approach to active learning for object detection is presented in [CKZC24], where the authors propose an uncertainty-guided self-training process to facilitate Un-

supervised Domain Adaptation (UDA) of the model through training on predicted and filtered pseudo-labels, eliminating the need for manual, human-processed labels in the adaptation process. They further implement memory banks for storing intermediate object class representations, one for each source and target domain, to compensate for imbalanced numbers of samples among classes. Unfortunately, their reliance on Monte Carlo (MC) Dropout and a two-stage object detector implies slow inference times, and therefore, it is not suitable for computationally efficient real-time detection. Still, an adaptation of this strategy, where the detector and uncertainty estimation are replaced with faster alternatives, is worth considering. Their automated domain adaptation strategy is related to the use case of this thesis, with the difference being that their work adapts the detector to the new domain, while our focus is on evaluating the domain shift itself for newly appearing uncertainties in predictions.

In [HJW+24], direct uncertainty estimation through Gaussian mixture models (GMMs) is implemented to guide an active learning process. This happens in two stages. First, the model checks for unfamiliar detections of known classes. Second, novel detections of unknown classes are identified. Consequently, the model learns to detect new classes iteratively and continuously expands the network for each new class that it encounters. The authors note that this approach struggles with high computational complexity on large datasets and the fine-grained inclusion of novel classes. While they do showcase an interesting uncertainty estimation process through a GMM, their computational feasibility for the Edge remains unclear. Furthermore, their target of open-world detection is beyond the scope of this thesis.

Miller et al. [MSMD22] proposed a real-time capable uncertainty estimation approach, GMM-Det, leveraging the phenomenon that object classes tend to form clusters in the logit space of detection networks. Their work focused on open-set error reduction, aiming to handle new class categories that the detector encounters but has no prior knowledge of, without making false predictions. In our work, we do not deal with unknown class categories, as we are dealing with the related use case of domain shift scenarios. Although we have not included GMM-Det in our comparison, we note that this approach also looks promising, and adapting it into our evaluation scenario might provide an interesting future direction.

The next chapter will take all the insights gathered so far and discuss practical implementations for the chosen approaches.

CHAPTER 3

Model Design

Object detection models are typically designed not to include dedicated uncertainty estimation. In this chapter, we will discuss in detail how to equip a state-of-the-art object detection model with the necessary tools to output uncertainty estimates. The foundation for all experiments will be the recent one-stage detector YOLO11 [JQ24], specifically the nano-scale version YOLO11n, to ensure the highest compatibility with resource-constrained devices and enable real-time capable Edge AI.

3.1 Network Overview

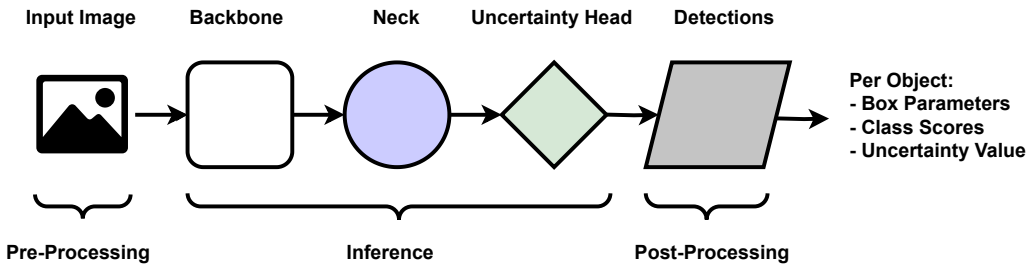


Figure 3.1: Uncertainty network overview. The head of the network is modified to also estimate uncertainty per object. Illustration re-drawn after the design in [KMH+21].

The architectural overview and processing flow of our models is given in Figure 3.1. At the beginning, there is a pre-processing step required to fit the image into the network, e.g., scaling (down) the image to the network's dimensions. In our case, we use the default configuration of the model to accept images with a width and height of 640 pixels and three color channels. Next is the inference stage, which encompasses the entire forward pass of data through the network. For our experimental models, we leave the backbone

and neck architecture unchanged and focus our modifications on the network head, where we design an individual uncertainty head for each evaluated estimation method. The output of the uncertainty head contains countless raw detections, where an individual detection D of our object detection model is defined as the tuple

$$D = (\mathbf{b}, \mathbf{c}, u), \tag{3.1}$$

where

- $\mathbf{b} = (x, y, w, h)$ with $(x, y, w, h) \in [0, 1]^4$ denoting the bounding box parameters in YOLO format [DAT23],
- $\mathbf{c} \in [0, 1]^K$ the class probability vector for $K = 80$ MS COCO [LMB+14] labels, and
- $u \in \mathbb{R}_{\geq 0}$ a scalar uncertainty estimate associated with the detection.

For YOLO11n with an input size of $640 \times 640 \times 3$, the network produces a total of 8,400 raw detections. Each detection is therefore represented by

$$\dim(D) = 4 + 80 + 1 = 85 \tag{3.2}$$

values.

These raw detections are then filtered in the post-processing step (the last part of Figure 3.1) using non-maximum suppression (NMS) [HBS17] to eliminate redundant boxes. We maintain the NMS step unchanged and use the classification scores for ranking overlapping detections, while the uncertainty estimates are simply passed through.

3.2 Network Design

All tested models are designed to produce one uncertainty value per detection. Here, we briefly revisit the uncertainty head (see Figure 3.1) and provide a detailed explanation for each model of how it handles the classification scores and uncertainty calculation. Bounding box generation will remain the same for all approaches, as designed in the original work [JQ24]. In our experiments, we compare four different networks:

1. **Basline** [JQ24]: YOLO11 baseline with without changes to the network architecture.
2. **Ensemble** [VT19]: Multiple classification heads enable a sampling-based approach.
3. **MC Dropout** [YKPC24]: A stochastic classification head will be sampled multiple times.
4. **EDL MEH** [YKPC24]: An additional head predicts evidence for the seen input.

3.2.1 Baseline

The YOLO11n network represents the baseline, which comes pretrained on the MS COCO dataset by Ultralytics [JQ24]. Since the original YOLO does not include uncertainty estimation, we adopted a simple approach of deriving uncertainty proxies directly from the classification output.

Classification

The baseline YOLO11n employs the default detection head without any architectural modifications. As designed in the original framework [JQ24], for each detection, the classification branch produces a vector of raw logits

$$\mathbf{z} = (z_1, \ldots, z_K) \in \mathbb{R}^K,$$

where K denotes the number of classes ($K = 80$ for MS COCO [LMB+14]). Applying the sigmoid activation function $\sigma(.)$ element-wise yields the class probability vector

$$\mathbf{c} = \sigma(\mathbf{z}) = (\sigma(z_1), \ldots, \sigma(z_K)), \qquad \mathbf{c} \in [0,1]^K, \tag{3.3}$$

where $c_k = \sigma(z_k)$ represents the predicted probability for class k.

Uncertainty Estimation

Following Gawlikowski et al. [GTA+23], a simple certainty measure is the maximum class probability $\max_k c_k$. Accordingly, we view the per-object uncertainty as its complement:

$$u = 1 - \max_{k \in \{1,\ldots,K\}} c_k. \tag{3.4}$$

where c_k denotes the probability assigned to class k. Intuitively, detections with low maximum confidence are considered more uncertain. Certain detections have high peak classification scores for at least one class.

Hyperparameters

Deep Learning models have numerous hyperparameters that influence the outcome of training, such as the learning rate or batch size. In this work, we focus on hyperparameters that optimize uncertainty estimation. Since the baseline ties uncertainty to classification scores, there is no additional parameter to optimize on. Therefore, we use the recommended parameters from the benchmark study in [MPS+25] or the default parameters from [JQ24] as our baseline, as detailed in Section 5.2.

Loss Function

For the default YOLO11 detector, the loss is divided into three main components [JQ24]:

- **Localization:** The bounding box regression branch tries to maximize the overlap between the predicted and ground-truth bounding boxes for precise object localization [TCERG23].
- **Distribution Focal Loss (DFL):** To refine the bounding box regression, YOLO11 also applies the Distribution Focal Loss [LLW+23].
- **Classification:** For each predicted box, class probabilities are generated using a sigmoid activation function, to allow prediction of multiple labels with individual probabilities [TCERG23]. Optimization happens through binary cross-entropy (BCE) with logits $\mathbf{z}$ before sigmoid activation in YOLO11 [JQ24].

In our training, we only optimize on the classification part of the loss, as this is the basis for our uncertainty estimates as well.

3.2.2 Ensemble

In the ensemble variant, the classification branch of the detection head is replicated n_{ens} times, with each branch connected to the same feature maps from the neck, inspired by Deep Sub-Ensembles [VT19]. After the heads, samples from all ensemble members are gathered for uncertainty estimation and aggregated into the final detections. To encourage diverse weights within the ensemble, a dropout mechanism is applied before all members and activated during training. Specifically, we employ a layer based on DropBlock [GLL18], a structured dropout technique designed for CNNs. In our work, we use a block size of 7, i.e., dropping 7×7 patches from feature maps with a probability $p_{\text{drop}}^{\text{ens}}$. A block size of 7 has also been identified as the recommended choice in [YKPC24]. To further increase diversity during training, we consider only one randomly chosen ensemble head for loss calculation during each batch.

Classification

In the ensemble variant, our detection head contains n_{ens} parallel classification heads. This produces a set of logit vectors

$$S = \{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(n_{\text{ens}})}\}, \qquad \mathbf{z}^{(i)} \in \mathbb{R}^K, \tag{3.5}$$

where $\mathbf{z}^{(i)}$ denotes the logits output of ensemble member i and K is the number of classes. Applying the sigmoid activation $\sigma(.)$ element-wise yields the corresponding class probability vectors

$$\mathbf{c}^{(i)} = \sigma(\mathbf{z}^{(i)}), \qquad \mathbf{c}^{(i)} \in [0,1]^K, \tag{3.6}$$

and the aggregated predictive distribution is then obtained as the mean across all ensemble members, as

$$\bar{\mathbf{c}} = \frac{1}{n_{\text{ens}}} \sum_{i=1}^{n_{\text{ens}}} \mathbf{c}^{(i)}, \tag{3.7}$$

which will be the representative classification score for each box.

Uncertainty Estimation

We follow common practice [MDMS19, PCK+23, MNDS18] of estimating classification uncertainty from multiple probability distributions obtained via a softmax activation. The softmax maps the logits to a proper categorical distribution over classes, i.e., all entries are nonnegative and their sum is one, which builds a basis for entropy-based uncertainty estimation [DCB+24].

Therefore, for each ensemble member, we compute

$$\mathbf{q}^{(i)} = \text{softmax}(\mathbf{z}^{(i)}), \tag{3.8}$$

and average across all members to obtain

$$\bar{\mathbf{q}} = \frac{1}{n_{\text{ens}}} \sum_{i=1}^{n_{\text{ens}}} \mathbf{q}^{(i)}. \tag{3.9}$$

The averaged distribution $\bar{\mathbf{q}}$ serves as the basis for uncertainty estimation. Following [MNDS18], we compute the entropy $H(\cdot)$ of $\bar{\mathbf{q}}$ as the total predictive uncertainty per detection:

$$U_{\text{total}} = H(\bar{\mathbf{q}}) = -\sum_{k=1}^{K} \bar{q}_k \log \bar{q}_k. \tag{3.10}$$

Breaking down total uncertainty as the sum of aleatoric and epistemic, the epistemic part can be separated as $U_{\text{epis}} = U_{\text{total}} - U_{\text{alea}}$, after retrieving aleatoric uncertainty U_{alea} [PCK+23, DCB+24]. In our case, we report U_{total} per object as the representative uncertainty value, as is the practice in prior works [MNDS18, MDMS19].

Hyperparameters

The ensemble variant can be configured by:

- DropBlock with dropout probability $p_{\text{drop}}^{\text{ens}}$ is applied before each replicated head and active during training to encourage diversity in weights. A higher dropout rate introduces more randomness into the training, but also makes it more challenging for the model to learn.
- Number of ensemble members n_{ens} controlling the number of generated samples. More samples allow for more accurate uncertainty estimation, but also increase the inference time of the model.

Loss Function

As our ensemble head modifications mainly concern replicating the existing classification head, and the added dropout layer does not require training, the loss calculation remains mostly unchanged from the baseline (see Subsection 3.2.1). During training, we select one ensemble member to use its logits $\mathbf{z}$ to optimize the BCE loss, which from there operates in the same manner as the YOLO11 baseline.

3.2.3 MC Dropout

Similar to the ensemble head, a DropBlock layer is placed before the classification layers of the head. The key difference is that there is only one classification head, but it is sampled n_{mcd} times during inference in order to produce a set of classification samples. This is achieved by keeping the dropout layer active at inference/testing time, as described in [GG16]. We will use the DropBlock variant of Dropout again, creating a model similar to MC DropBlock [YKPC24].

Classification

Formally, the formulation is identical to the ensemble case (Equations 3.5-3.7), with the difference that $\mathbf{z}^{(i)}$ corresponds to a stochastic forward pass of the same head, and the number of samples is determined by n_{mcd} instead of n_{ens}. Thus, for the classification of a detection, the aggregated score $\bar{\mathbf{c}}$ is computed as the mean over n_{mcd} sampled predictions, analogous to Equation 3.7.

Uncertainty Estimation

Uncertainty estimation for MC Dropout follows directly from the ensemble definitions in Equations 3.8-3.10, with n_{ens} replaced by n_{mcd}. Again, U_{total} represents the estimated uncertainty per box, based on the entropy of the produced samples.

Hyperparameters

The MC Dropout variant is similar to Ensemble, as it configures the Dropout rate and sample number:

- Dropout probability $p_{\text{drop}}^{\text{mcd}}$ applied before the single classification head, which is active during training and kept active at inference time for sampling.
- Number of stochastic forward passes n_{mcd} determining the sampling budget for uncertainty estimation.

Loss Function

For the MC Dropout, we take $\mathbf{z}$ as a single sample from the classification part of the head for BCE loss calculation. During training, there is no need to sample the head multiple times, as this is only done at testing time for uncertainty calculation. Therefore, the loss calculation is the same as in the Baseline.

3.2.4 EDL MEH

We extend the detection head with an additional output branch, referred to as the Model Evidence Head (MEH) [PCK+23]. This branch predicts a single non-negative evidence score λ for each object, which, together with the class probabilities $\mathbf{c}$, is used to

parameterize a Dirichlet distribution over class probabilities. The Dirichlet distribution serves as a prior to the typical categorical distribution of the softmax function, commonly used for classification [SKK18].

Classification

One key design point of EDL MEH is to leave the original classification outputs untouched, thereby maintaining the performance of the underlying detector, while equipping them with additional evidence prediction [PCK+23]. Consequently, classification happens in the same way as described for the baseline in Subsection 3.2.1.

Uncertainty Estimation

The distribution Dirichlet($\boldsymbol{\alpha}$), which builds the basis for uncertainty estimation of object classification in this approach, is defined by its parameters $\boldsymbol{\alpha}$ of length K classes [SKK18]. In the approach here based on EDL MEH, the uncertainty head outputs a scalar evidence λ per detector. Given the classification scores $\mathbf{c}$ as defined in Equation 3.3, we form the Dirichlet parameters following the process described in [PCK+23] as

$$\boldsymbol{\alpha}_{\text{MEH}} = \lambda \cdot \mathbf{c}, \tag{3.11}$$

where larger λ indicates stronger evidence in the predicted class probabilities.

However, unlike Park et al. [PCK+23], who explicitly sample from Dirichlet($\boldsymbol{\alpha}_{\text{MEH}}$) in post-processing after the network to obtain a set of categorical predictions, we adopt the direct uncertainty estimation formulation of Sensoy et al. [SKK18]. In this formulation, the final Dirichlet parameters are the evidence scores per class ($\boldsymbol{\alpha}_{\text{MEH}}$ in our case) shifted by one,

$$\boldsymbol{\alpha} = \boldsymbol{\alpha}_{\text{MEH}} + 1, \tag{3.12}$$

resulting in a distribution Dirichlet($\boldsymbol{\alpha}$) defined over the class probability simplex. The overall predictive uncertainty u per object, also called vacuity caused by lack of evidence [ZYCX25], is then estimated analytically following [SKK18] as

$$u = \frac{K}{S}, \qquad S = \sum_{i=1}^{K} \alpha_i, \tag{3.13}$$

where S denotes the Dirichlet strength. A higher strength S corresponds to lower uncertainty u. With this approach to uncertainty estimation, we have combined the advantages of [PCK+23], which proposes decoupling evidence prediction and classification for EDL, and then followed the original EDL uncertainty estimation approach [SKK18] to reduce additional computation required for sampling the Dirichlet distribution, as in [PCK+23].

Hyperparameters

The EDL MEH predicts evidence λ, which can be influenced by:

- Evidence supervision weight w_{edl} scaling the MEH loss relative to the standard classification BCE term, balancing classification loss and evidence loss.
- The activation function $\sigma_{\lambda}(\cdot)$ of the lambda output at the final layer of the neural network. This parameter can influence how eager the network is to predict evidence. For example, an exponential activation may produce stronger outputs than a linear activation.

CHAPTER 4

Datasets

In this chapter, we provide an overview of the datasets used and discuss the differences between training and evaluation. We adopt a domain-shift evaluation setup, where we train on one dataset and evaluate on different datasets to measure generalization under data distribution shift. For example, some datasets introduce reduced visibility and weather changes (e.g., fog, rain), creating a challenging new domain for evaluating the quality of uncertainty estimation. To evaluate multiple datasets together, they must be in the same labeling format. Moreover, the number of class labels is typically a fixed output format of the object detection network, so the format it was trained on will also be used for evaluation.

4.1 Overview and Coversion

In this work, models are initialized from MS-COCO-pretrained YOLO11n weights (`yolo11n.pt` [JQ24]) and then fine-tuned on datasets featuring driving domains. This allows us to utilize the existing weights to their fullest extent and minimize network modifications. The following subsections provide an overview of all the relevant datasets and the necessary label mapping required to port them from their original format to our detector format.

4.1.1 MS COCO

The Microsoft COCO [LMB+14] dataset is a large-scale object detection benchmark with everyday scenes containing multiple objects per image across 80 categories. It provides diverse contexts, object scales, and occlusion patterns, and with over 200,000 labeled images, it offers an excellent basis for training general-purpose detectors. An example image is given in Figure 4.1. Table 4.1 showcases all the class labels present in this dataset. All the other datasets will be adapted to this schema (also referred to as

COCO80). In practice, we focus primarily on the first eight classes, as they are present in most autonomous driving datasets.

Figure 4.1: Everyday scene as an Example from MS COCO [LMB+14]).

4.1.2 Cityscapes

The Cityscapes [COR+16] dataset contains street scenes from European cities (mostly in Germany), captured from a vehicle-mounted camera in urban environments. An example is given in Figure 4.2. The dataset features diverse traffic scenes, including cars, pedestrians, bicycles, and urban infrastructure, under mostly clear weather conditions. In Table 4.2 we see the mapping from the original labeling to the MS COCO 80 class labels (see Table 4.1). Here, there is only one conflict in the second row, where the original Cityscapes label *rider* has no counterpart in MS COCO, so it is mapped to *person* instead. The label rider refers to individuals riding two-wheeled vehicles, such as bicycles. Otherwise, labels can be transitioned without any conflicts.

ID	Label	ID	Label	ID	Label	ID	Label
0	person	1	bicycle	2	car	3	motorcycle
4	airplane	5	bus	6	train	7	truck
8	boat	9	traffic light	10	fire hydrant	11	stop sign
12	parking meter	13	bench	14	bird	15	cat
16	dog	17	horse	18	sheep	19	cow
20	elephant	21	bear	22	zebra	23	giraffe
24	backpack	25	umbrella	26	handbag	27	tie
28	suitcase	29	frisbee	30	skis	31	snowboard
32	sports ball	33	kite	34	baseball bat	35	baseball glove
36	skateboard	37	surfboard	38	tennis racket	39	bottle
40	wine glass	41	cup	42	fork	43	knife
44	spoon	45	bowl	46	banana	47	apple
48	sandwich	49	orange	50	broccoli	51	carrot
52	hot dog	53	pizza	54	donut	55	cake
56	chair	57	couch	58	potted plant	59	bed
60	dining table	61	toilet	62	tv	63	laptop
64	mouse	65	remote	66	keyboard	67	cell phone
68	microwave	69	oven	70	toaster	71	sink
72	refrigerator	73	book	74	clock	75	vase
76	scissors	77	teddy bear	78	hair drier	79	toothbrush

Table 4.1: MS COCO [LMB+14] 80 class IDs (0-based) with labels, shown as four (ID, Label) column pairs per row.

4.1.3 Foggy Cityscapes

Foggy Cityscapes [SDVG18] is derived from Cityscapes by adding synthetic fog to simulate difficult weather conditions, as shown in Figure 4.3. The reduced visibility increases the difficulty of detection, especially for distant objects. Since Foggy Cityscapes only contains modifications to the images, the ground truth labels for each image stay the same; therefore, the label mapping is also consistent with Cityscapes in Table 4.2.

4.1.4 RainCityscapes

RainCityscapes [HFZH19] augments Cityscapes with synthetic rain effects (rain streaks), modeling reduced visibility and motion streak artifacts typical in rainy conditions. Figure 4.4 shows these effects, compared to the original in Figure 4.2. Just as with Foggy Cityscapes, only the images are modified in this dataset, and label conversion from MS COCO happens as in the original dataset (see Table 4.2).

Figure 4.2: Example from Cityscapes [COR+16]. Urban daytime scene with multiple road users and clear visibility.

Original	→	COCO80
person	→	person
rider	→	person
car	→	car
motorcycle	→	motorcycle
bicycle	→	bicycle
bus	→	bus
truck	→	truck
train	→	train

Table 4.2: Cityscapes to COCO80 label mapping (used for Cityscapes and its weather variants).

4.1.5 KITTI

Figure 4.5 showcases the KITTI [GLU12] dataset, which consists of real-world driving scenes from Karlsruhe, Germany, captured from a front-facing camera. It includes a variety of urban and suburban scenes with differing traffic densities and viewpoints. One major difference is that, compared to Cityscapes, KITTI has a higher count of annotated images, but lacks the diversity of multiple cities. The label mapping in Table 4.3 shows that KITTI has a finer distinction between types of persons in the scenery, distinguishing the labels *Pedestrian*, *Cyclist*, and *Person_sitting*. However, since our pre-trained detector only recognizes the coarser category *person*, those original KITTI labels will be mapped to this category. Similar case for *Car* and *Van* being mapped to just *car*.

Figure 4.3: Foggy Cityscapes [SDVG18] example. Synthetic fog reduces the contrast and partially hides distant objects.

Original	→	COCO80
Car	→	car
Pedestrian	→	person
Van	→	car
Cyclist	→	person
Truck	→	truck
Misc	→	—
Tram	→	train
Person_sitting	→	person

Table 4.3: KITTI to COCO80 label mapping. “Misc” is not mapped and is dropped.

4.1.6 BDD100K

BDD100K [YCW+20] (see Figure 4.6) covers diverse US driving scenery across cities and highways with significant variation in time-of-day, weather, and scene composition. It encompasses day, dusk, night, and adverse weather conditions, providing various domains for challenging object detection. This dataset contains a vast number of images (100,000), significantly larger than KITTI (approximately 15,000) and Cityscapes (approximately 5,000). Label conversion (see Table 4.4 here is straightforward, as most labels find their exact match from BDD100K to COCO80, with the exception *traffic sign* not being present in the target mapping.

Figure 4.4: Example from the RainCityscapes [HFZH19] dataset. Rain streaks pose challenges to object detection and calibration.

Figure 4.5: The KITTI [GLU12] dataset features a front-facing view showing pedestrians, vehicles, and roadside structures.

4.1.7 nuImages

nuImages (from the nuScenes ecosystem [CBL+20]) contains images from multiple cameras mounted on autonomous vehicles in various urban settings, as exemplified in Figure 4.7. It is another large-scale dataset, similar to BDD100K, with over 93,000 images suitable for 2D object detection. Regarding label adaptation in Table 4.5, detailed pedestrian and vehicle labels are matched to the more general MS COCO labels. Many labels also remain unmatched, like other objects in the scene, such as traffic cones, being removed from the annotations of any concerning image.

Figure 4.6: Image from BDD100K [YCW+20], a large-scale driving dataset.

Original	→	COCO80
person	→	person
car	→	car
bus	→	bus
truck	→	truck
traffic light	→	traffic light
train	→	train
rider	→	person
bike	→	bicycle
motor	→	motorcycle
traffic sign	→	—

Table 4.4: How BDD100K labels are ported to the COCO80 labels. *traffic sign* is not mapped and therefore dropped.

4.2 Dataset Usage

In our evaluation setup, we will utilize the Cityscapes and KITTI datasets to train a set of detection models. Both datasets exhibit relatively consistent weather conditions, providing a suitable starting point for our domain-shift scenario. For model validation after training, we use Foggy Cityscapes, RainCityscapes, BDD100K, and nuImages, as those datasets feature highly diverse scenes, including different weather conditions. Therefore, those datasets provide the desired shift in the data domain for our use case.

Figure 4.7: Example from nuImages [CBL+20], a multi-camera dataset with varied viewpoints and dense traffic scenes.

All datasets come in multiple splits, each with a mutually exclusive subset of images from the entire dataset, allowing the model to be trained on one split and then evaluated on the other unseen split. Consequently, for training, we use the training split of each dataset, and for validation, we use the corresponding validation splits. A special case is hyperparameter tuning, a set of small training sessions conducted before the actual training, to determine the best hyperparameter configuration for our models. Here, we selected the train split of Cityscapes for training and the train split of KITTI for validation, while keeping the validation splits untouched.

Original	→	COCO80
human.pedestrian.adult	→	person
human.pedestrian.child	→	person
human.pedestrian.construction_worker	→	person
human.pedestrian.personal_mobility	→	person
human.pedestrian.police_officer	→	person
human.pedestrian.stroller	→	person
human.pedestrian.wheelchair	→	person
vehicle.bicycle	→	bicycle
vehicle.car	→	car
vehicle.motorcycle	→	motorcycle
vehicle.bus.bendy	→	bus
vehicle.bus.rigid	→	bus
vehicle.truck	→	truck
vehicle.emergency.ambulance	→	car
vehicle.emergency.police	→	car
animal	→	—
movable_object.barrier	→	—
movable_object.debris	→	—
movable_object.pushable_pullable	→	—
movable_object.trafficcone	→	—
static_object.bicycle_rack	→	—
vehicle.construction	→	—
vehicle.ego	→	—
vehicle.trailer	→	—

Table 4.5: nuImages labels to COCO80 conversion. Several entries with "—" cannot be mapped and are therefore removed in the converted version.

CHAPTER 5

Evaluation

This chapter will tie together the previous chapters on model design and dataset overview, and explain the process of evaluation, from finding suitable hyperparameters to training and finally validating all models. Before experiments can begin, metrics must be defined, which enable the comparison of the different models. These metrics assess both detection accuracy and the quality of uncertainty estimation.

5.1 Metrics

Here, we summarize and briefly explain the evaluated metrics, which include a set of standard metrics for object detection and an additional set specifically chosen to evaluate uncertainty estimation. We follow the practice of [DLXS20] and annotate each metric with ↑/↓ to tell at first glance if *higher is better*/*lower is better* applies, respectively.

5.1.1 Object Detection Metrics

The following standard detection metrics are employed to measure both the localization and classification quality of detections. Additionally, we measure the inference times of the models in order to check their real-time capability.

Intersection over Union (IoU) ↑

The Intersection over Union (IoU) measures the overlap between predicted and ground-truth bounding boxes [CLZT24]:

$$\text{IoU} = \frac{|\mathbf{b}_p \cap \mathbf{b}_g|}{|\mathbf{b}_p \cup \mathbf{b}_g|},$$

where $\mathbf{b}_p$ and $\mathbf{b}_g$ denote the predicted and ground-truth bounding boxes, respectively. Higher IoU indicates better localization accuracy. During evaluation, we compare our

predicted boxes to the ground truth box labels from our dataset. The ground truth contains the accurate box location and the correct class label. Given an IoU threshold τ (e.g., $\tau = 0.5$), a detection is counted in [EVGW+10] as:

- **True Positive (TP)**: Matched to exactly one ground-truth box with IoU $\geq \tau$ and correct class.
- **False Positive (FP)**: Prediction that fails to match (IoU$< \tau$ or duplicate on an already matched GT).
- **False Negative (FN)**: Ground-truth object with no matching prediction.

In this work, a localization threshold of $\tau = 0.5$ will be used for all metrics. We will not report direct IoU values in our evaluation further, as they will be incorporated into almost all other metrics anyway (except FPS).

Precision (P) ↑

The Precision metric quantifies the proportion of correctly predicted positive detections [OD08, CLZT24]:

$$P = \frac{TP}{TP + FP},$$

where TP and FP denote true and false positives. Precision rewards the detector for only making predictions that it is very confident in, reducing FPs in contrast to TPs.

Recall (R) ↑

The model's ability to detect all ground-truth objects is measured as the Recall [OD08, CLZT24]:

$$R = \frac{TP}{TP + FN},$$

with FN being false negatives, respectively. This metric rewards the detectors for finding as many TPs as possible, reducing missed detections (FNs). However, as false detections (FPs) do not influence the recall, the detector may want to make as many detections as possible, even if they have low precision.

Mean Average Precision (mAP) ↑

In practice, we need to strike a balance between making precise detections and identifying all objects present in the image. Therefore, the mean Average Precision (mAP) summarizes the trade-off between precision and recall. Average Precision (AP) is computed per class by integrating the precision–recall curve (i.e., calculating the area under the curve) [EVGW+10, CLZT24]:

$$\text{AP} = \int_0^1 P(R)\, dR,$$

and the mean Average Precision (mAP) is obtained by averaging the AP across all object classes:

$$\text{mAP} = \frac{1}{K} \sum_{k=1}^{K} \text{AP}_k,$$

for number of classes K.

Traditionally, the PASCAL VOC [EVGW+10] benchmark reports results using an IoU threshold of $\tau = 0.5$, denoted as mAP_{50}. The COCO [LMB+14] evaluation protocol later extended this definition to a range of IoU thresholds $\tau \in [0.5, 0.95]$ in increments of 0.05, resulting in the more comprehensive metric $\text{mAP}_{50\text{-}95}$. The overall mAP is obtained by averaging the class-wise AP_k across all object classes and IoU thresholds. The area under the precision–recall curve is computed by interpolating precision values at various recall levels, following standard practice in object detection benchmarks [EVGW+10, LMB+14]. The mAP_{50} will be reported primarily in our evaluation, as other metrics use that threshold $\tau = 0.5$ as well distinguish correct from incorrect detections.

Frames Per Second (FPS) ↑

In our evaluation setup, we validate our model on large datasets. Per dataset, we report the average processing time per image in milliseconds (ms), which is the sum of pre-processing, inference, and post-processing times (see Figure 3.1). Time per image is then converted to frames per second as

$$\text{FPS} = \frac{1000}{\text{ms/image}},$$

where we report the average per dataset evaluation.

5.1.2 Uncertainty Metrics

The following metrics take into account the uncertainty outputs of our detector, in addition to the classification and localization outputs of regular object detection.

Minimum Uncertainty Error (mUE) ↓

Our models are designed to report an uncertainty value per object. This value can be utilized in a selective prediction scenario [DLXS20], where predictions are either accepted or rejected based on a threshold. As a representative metric here, the Uncertainty Error (UE) quantifies how effectively an uncertainty measure can accept correct detections and reject incorrect ones, for a given uncertainty threshold δ [MDMS19]:

- The proportion of *correct detections* $\mathbb{D}_c$ (i.e., TPs) that are incorrectly rejected (i.e., with uncertainty above δ), and
- The proportion of *incorrect detections* $\mathbb{D}_i$ (i.e., non-TP detections) that are incorrectly accepted (i.e., with uncertainty below or equal to δ).

Specifically, for estimated uncertainty $U(.)$, the uncertainty error at threshold δ is expressed as [MDMS19]:

$$UE(\delta) = 0.5\,\frac{|U(\mathbb{D}_c) > \delta|}{|\mathbb{D}_c|} + 0.5\,\frac{|U(\mathbb{D}_i) \leq \delta|}{|\mathbb{D}_i|},$$

being the average of incorrectly rejected detections and incorrectly accepted detections. Noteworthy is that correct and incorrect detections are counted as a whole across all classes, so it differs from metrics like mAP, which are first computed per class and then macro-averaged. Miller et al. [MDMS19] further define the minimum uncertainty error (mUE) as the optimal uncertainty threshold δ that minimizes the uncertainty error. In our evaluation, for each model and dataset, this threshold may take on individual values.

Area Under the Receiver Operating Characteristic (AUROC) ↑

Additionally, we evaluate how well uncertainty distinguishes correct from incorrect detections using receiver operating characteristic (ROC) curves. While the precision-recall (PR) curve plots precision against recall, the closely related ROC curve compares the true positive rate (which is recall) against the false positive rate [DG06]. We again follow the example of Miller et al. [MDMS19], where for the uncertainty estimation function $U(.)$ and each considered uncertainty threshold δ, the true positive rate (TPR) and false positive rate (FPR) are computed as

$$\mathrm{TPR}(\delta) = \frac{|\,U(\mathbb{D}_c) \leq \delta\,|}{|\mathbb{D}_c|}, \qquad \mathrm{FPR}(\delta) = \frac{|\,U(\mathbb{D}_i) \leq \delta\,|}{|\mathbb{D}_i|},$$

with correct detection $\mathbb{D}_c$ and incorrect detections incorrect detections $\mathbb{D}_i$ being defined as in the Subsection 5.1.2 for mUE. The AUROC is then defined as the area under the curve, when measuring TPR and FPR jointly across various thresholds [DG06], in our case, the uncertainty threshold δ. Intuitively speaking, this metric provides a probability that a randomly selected correct detection has lower uncertainty than a randomly chosen incorrect detection [MDMS19]. The baseline is represented by an AUROC of 50%, which can be achieved by random guessing alone [WFD+23], and a higher value is preferable.

False Positive Rate at 95% True Positive Rate (FPR95) ↓

Following works [WFD+23, DWGL22], we report the FPR95 metric, which describes the FPR value when the TPR is at 95%. In other words, for achieving a true positive rate of 95%, what false positive rate do we have to tolerate, where lower is better. Since the ROC curve already plots TPR (y-axis) against FPR (x-axis) [WFD+23], retrieving the FPR95 value is possible by directly reading the FPR value at the 95% mark on the TPR axis of the ROC curve.

Excess Area Under Risk-Coverage Curve (E-AURC) ↓

The metric E-AURC [GUEY19] quantifies how effectively a model ranks detections by their uncertainty in a selective prediction setting. It characterizes the trade-off between

coverage (the fraction of accepted detections) and *risk* (the error rate among accepted detections). Lower values indicate that uncertain detections are rejected earlier, resulting in a more reliable ranking.

Formally, the metric is defined in [GUEY19] as

$$\text{E-AURC} = \text{AURC} - \text{AURC}^*,$$

where AURC denotes the area under the empirical risk–coverage curve, and AURC* represents the optimal (oracle) area obtained by perfectly ranking all correct detections before incorrect ones. To obtain the RC curve, detections are ordered by their uncertainty scores, and the coverage-risk trade-off is evaluated by varying an uncertainty threshold δ, accepting each detection D with uncertainty $U(D) \leq \delta$ and rejecting those above. An E-AURC value of 0 corresponds to a perfectly ordered uncertainty ranking [GUEY19].

5.2 Training and Validation Setup

Our training setup consists of two stages. First, we evaluate suitable hyperparameters for our implemented model modifications, then we do final training with fixed parameters.

We found that 100 epochs of transfer learning from the MS COCO pre-trained base models were sufficient for our analysis. An optimizer comparison conducted by [MPS+25] revealed that for YOLO11 and the stochastic gradient descent (SGD) optimizer, high validation performance can already be achieved within even fewer epochs than 100. Preliminary results in our setup confirmed that training gains are only marginal with higher epochs. Furthermore, in our detection model, most layers and even parts of the final detection head are frozen, so the smaller number of trainable parameters also demands fewer training cycles. Lastly, in our evaluation, we do not aim for the top absolute detection scores; rather, we want to make a relative comparison between the approaches and to determine if uncertainty estimation is beneficial compared to a similarly trained baseline.

All methods start off with pre-trained YOLO11n weights [JQ24] and are trained with the following parameters:

- **Epochs**: 100. Guided by benchmarks found in [MPS+25].
- **Batch Size**: 16. Also the preferred choice in [MPS+25].
- **Optimizer**: SGD with a learning rate of 0.001 and momentum of 0.9. The benchmark study of [MPS+25] also recommended this setting for YOLO11.
- **Image size**: $640 \times 640 \times 3$ pixels. Higher image sizes trade in detection speed for detection quality. We stay with the default value, as also tested in [MPS+25].

- **Layer Freezing**: All layers are frozen, except for the head of the network producing classification and uncertainty scores. We optimize only the necessary parts for our network modifications, aiming to retain as much of the generalization capability gained from pre-trained weights as possible.

- **Loss weights**: We only consider classification losses and the uncertainty loss of EDL MEH, setting losses from other sources, like bounding box localization, to 0. The loss weight for EDL MEH is subject to hyperparameter optimization and will be defined later on.

- **Data augmentation**: Various image augmentations during training, like scaling, translation, and hue changes, as used by default in [JQ24].

- **Other Hyperparameters**: For each uncertainty estimation approach, we select additional hyperparameters from our defined search space in the following Subsection. Further (default) hyperparameters for training (e.g., floating point precision) are taken from the Ultralytics framework [JQ24] unchanged.

5.2.1 Hyperparameter Search

The hyperparameter optimization was performed using Ray Tune [LLN+18], a scalable approach that enables the computation of many search trials in parallel.

The training dataset was the train split of Cityscapes, and for validation, the train split of KITTI was used. For each model, a search of 100 iterations was conducted. An iteration (or trial) in the Ray Tune framework represents a complete training session for a specified number of epochs, utilizing a sample of hyperparameters drawn from the defined search space. We take over most parameters as defined earlier in this Section, with the main difference being that each model is trained for 50 epochs instead of 100. We select a lower number of epochs, as benchmarks in [MPS+25] showed that our optimizer settings tend to converge early, which also allows us to test more parameter combinations in shorter trial runs within the same resource budget. Additionally, a ten-epoch grace period is provided for early stopping, should no improvement be observed during a tuning trial. In our search, we optimize for the mUE value, and in the final selection of hyperparameters, we consider a trade-off between mUE, mAP, and impact on inference time, as a suitable fit for our final chosen parameters.

For each of the three uncertainty estimation models selected, we optimize two hyperparameters.

Ensemble For the ensemble-based detection head, we tuned the number of parallel heads and the internal dropout rate:

- Number of heads $n_{\text{ens}} \in \{3, 5, 7, 9\}$. We use the five ensemble members from a prior work [MSZ+19] as a starting point and build a search space around them.

- Dropout rate $p_{\text{ens}} \in [0.0, 0.5]$. The dropout rate for the DropBlock layer has been tested up to a value of 0.7 in [YKPC24], but lower values (e.g., 0.1) showed better performance. We adjust our range accordingly and only test up to 0.5.

MC Dropout The MC-Dropout configuration explored different dropout regularization intensities and stochastic inference iterations:

- Number of forward passes $n_{\text{mc}} \in \{5, 10, 15, 20\}$. A prior work [MDMS19] used 20 forward passes. However, they did not focus on inference time, which scales with the number of forward passes, which is why we will use this value as an upper bound for optimization. MC DropBlock has also been tested with 8 forward passes [ZYX24], showing that fewer forward passes are feasible.
- Dropout rate $p_{\text{mc}} \in [0.0, 0.5]$. We pick the same search range as for Ensemble, based on initial insights from [YKPC24].

EDL MEH For the evidential model, we tuned both the activation function $\sigma_\lambda(\cdot)$ producing the evidence λ and the evidential loss weighting w_{edl}:

- Evidence activation function $\sigma_\lambda(\cdot) \in \{\text{Exponential}, \text{ReLU}, \text{Softplus}, \text{AGLU}\}$. ReLU and softplus can commonly be used as EDL activation functions [HSRW20], and experiments with exponential activation for EDL have also been conducted already [PPSM25]. Additionally, we test the AGLU [ADNL24] activation. We clamp all activations to non-negative values, in order to generate valid evidence according to [SKK18]. Additionally, we clamp maximum values to 1,000 to prevent exploding activation values, especially those resulting from the exponential function.
- EDL loss weight $w_{\text{edl}} \in [0.1, 5.0]$. This loss weight determines the impact of the EDL MEH loss compared to the classification loss, where a higher value makes training focus more on predicting evidence for classification scores. The default value here is 1.0 as the original work did not introduce this parameter [PCK+23], i.e., classification loss and evidence loss are treated equally. Here, we test a range of smaller and larger values to determine their impact.

5.2.2 Validation

Validation is the last step of evaluating our models. We use four different datasets: RainCityscapes, Foggy Cityscapes, BDD100K, and nuImages. Compared to the train datasets, these datasets simulate the domain-shift use case to test our uncertainty estimates. For all of our implemented and trained models, we validate on each dataset and compare the metrics defined in Section 5.1. The results will be visualized and compared in the following chapter. The hardware used for training and validation is an AMD EPYC 9354P 32-Core Processor with 500GB of RAM and an NVIDIA A100 GPU. Python 3.10.12 was used as the programming language.

CHAPTER 6

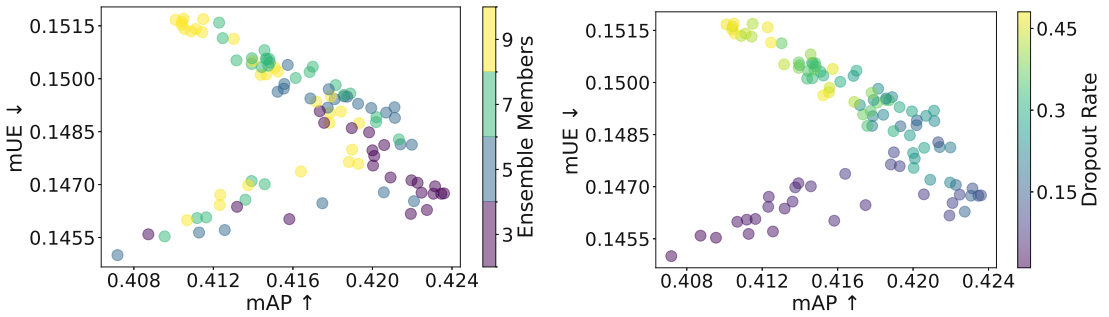
Results

This chapter presents the results, beginning with insights from the hyperparameter tuning and then showing the outcome of a comprehensive validation of the trained models. Lastly, there will be some additional results on the detection speed of the models.

6.1 Hyperparameter Analysis

In this section, we provide a brief visualization of the results from the hyperparameter tuning and justify our parameter choices for the subsequent training and evaluation.

6.1.1 Ensemble



(a) Number of ensemble members n_{ens}.

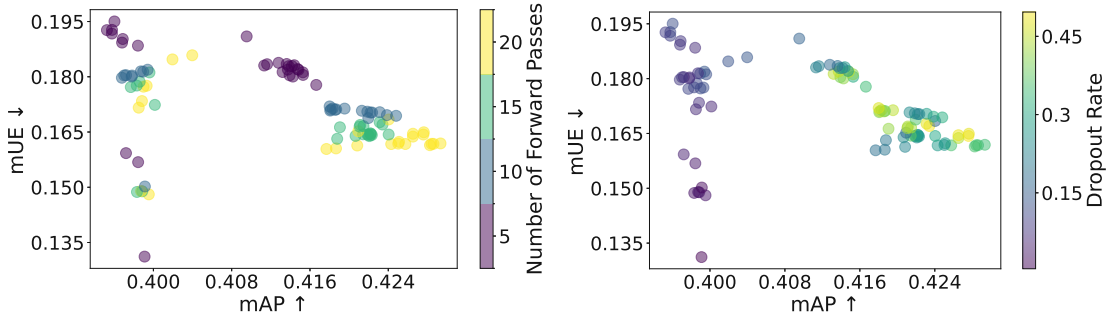
(b) Ensemble Dropout rate p_{ens}.

Figure 6.1: Hyperparameter analysis of the Ensemble uncertainty head. The bottom right represents the best performance spot in the plots. The Ensemble approach prefers lower ensemble member counts and a medium dropout rate.

In Figure 6.1, we can see the results of tuning hyperparameters for the Ensemble uncertainty head. The mAP score on the x-axis represents the overall quality of object

detection (higher is better), and the mUE on the y-axis represents the error in uncertainty estimation (lower is better). Consequently, the bottom right in the plot represents the best case, while the top left represents the worst case. Each dot in the scatter plots represents an outcome of a trial run during the parameter search. For the search space, the parameter name is depicted on the color bars next to the plots. For the number of ensemble members in our detection head, Figure 6.1a shows that a lower number of ensemble members (purple) can already produce good results, especially for mAP, while higher numbers (yellow) show worse performance. The analysis of the dropout rate in Figure 6.1b highlights that, in general, slightly lower dropout rates (blue) are preferable, and too high rates (yellow) reduce the overall performance across both metrics. Therefore, we choose $n_{\text{ens}} = 3$ ensemble heads, and $p_{\text{ens}} = 0.13$ as dropout rate for our ensemble head training.

6.1.2 MC Dropout



(a) Number of MC Dropout forward passes n_{mc}.

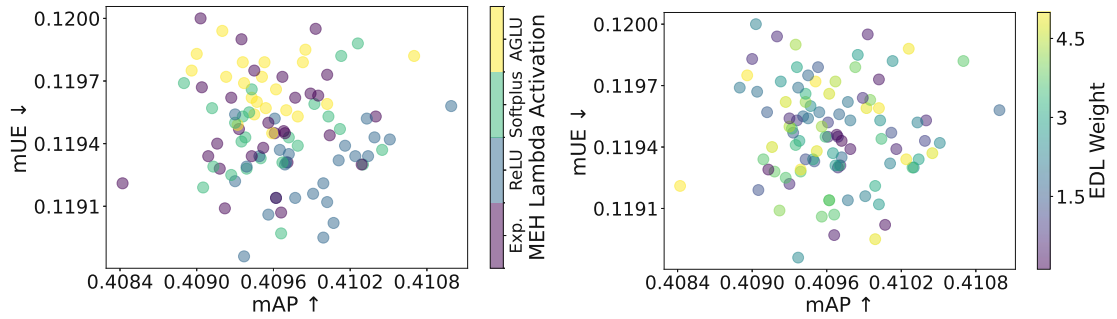
(b) MC Dropout rate p_{mc}.

Figure 6.2: Hyperparameters of the MC Dropout uncertainty head. Higher numbers of forward passes are preferred for both mUE and mAP. However, this increases inference time cost significantly.

Looking at the MC Dropout hyperparameters in Figure 6.2, the plot in Figure 6.2a shows a clear tendency that a higher number of forward passes produces better mAP and mUE numbers. However, by the clusters forming on the right of the plot, we can see the difference is sometimes only marginal. We opt for $n_{\text{mc}} = 15$ (green in Figure 6.2a) as a middle ground for good detection performance, thereby limiting the computational overhead associated with higher counts of forward passes. The corresponding dropout rate in Figure 6.2b tends to favor medium rates of dropout, where we selected $p_{\text{mc}} = 0.34$ as the final choice, also reported by some of the better performing trial runs of the tuning process.

6.1.3 EDL MEH

Lastly, we look at the EDL MEH approach in Figure 6.3. The trial runs for various activation functions of the evidence output in Figure 6.3a show a separation in mUE,



(a) Evidence Activation function $\sigma_\lambda(\cdot)$.

(b) Weight of the EDL loss w_{edl}.

Figure 6.3: Tuning the EDL MEH head for activation function and loss weight during training. This approach favors the ReLU activation, but is rather indifferent to the loss weight.

where ReLU and Softplus have the lowest error. The exponential activation (Exp.) and AGLU have overall higher mUE values, indicating worse performance in evidence generation. Based on those insights, we pick the $\sigma_\lambda(\cdot) = \text{ReLU}$ as the chosen activation. Moving on to Figure 6.3b, there is no clear tendency or pattern showing for the influence of the EDL weight. Consequently, we choose $w_{\text{edl}} = 2.5$ as a balanced value from the search space.

6.2 Model Validation

This section presents the results for models trained on the Cityscapes and KITTI datasets, which were then validated on the Foggy Cityscapes, RainCityscapes, BDD100K, and nuImages datasets. The evaluated metrics include mAP, precision, recall, and FPS for object detection performance, as well as mUE, AUROC, FPR95, and E-AURC for uncertainty estimation performance.

In Figures 6.4-6.7, we can see grids of barplots, where each row represents one metric, and each column represents one validation dataset, with the mean of all datasets in the last column. Each barplot shows the performance of the four models: Baseline, Ensemble, MC Dropout, and EDL MEH. The color coding highlights the best-performing (green) and worst-performing (red) models for each metric and dataset. The accompanying values for the shown barplots are listed in Tables 6.1 and 6.2.

6.2.1 Cityscapes Results

In this subsection, we present the results for models trained on the Cityscapes dataset, and then validated on the Foggy Cityscapes, RainCityscapes, BDD100K, and nuImages datasets.

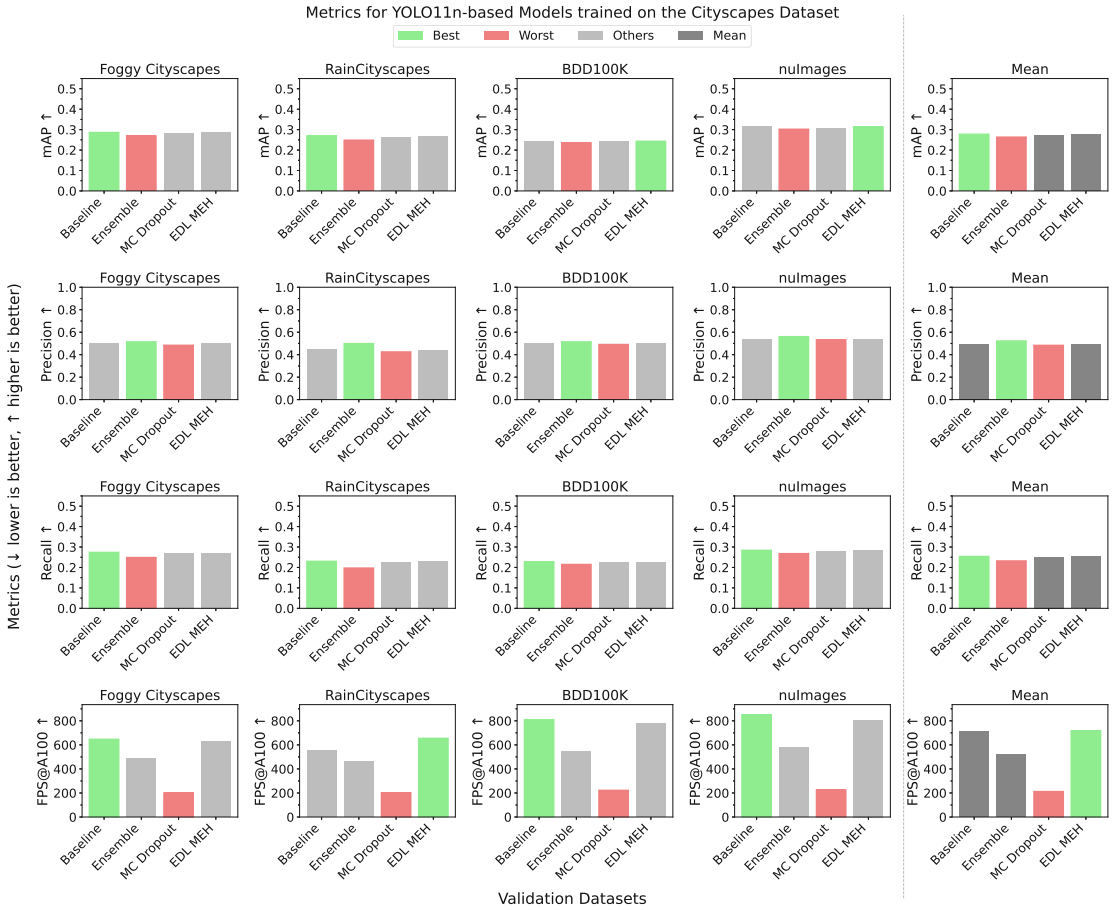


Figure 6.4: Validation performance grid for models trained on the Cityscapes dataset and validated on Foggy Cityscapes, RainCityscapes, BDD100K, and nuImages datasets. The metrics displayed are mAP, Precision, Recall, and FPS, which indicate the overall detection quality and computational cost.

In Figure 6.4, we show object detection quality based on common metrics, as well as the images per second (FPS) that each model (GPU-accelerated) can process. We will briefly summarize the average values across the four validation datasets in the rightmost barplots of the Figure. The mean Average Precision (mAP) scores show that models generally perform similarly on each validation dataset, with Ensemble having the lowest mAP of 0.264 on average, while the other three models are very close to each other (Baseline: 0.278, MC Dropout: 0.273, EDL MEH: 0.278). Precision is highest for the Ensemble model (0.524), with other approaches lagging behind noticeably. Recall shows the opposite trend to precision, with Ensemble at the lowest. Overall, these metrics indicate that the uncertainty estimation models generally maintain close proximity to the Baseline model in terms of detection quality, which is a desirable outcome. The last metric in Figure 6.4 is FPS, where the EDL MEH model achieves the highest speed of

718.4 FPS on average, followed closely by the Baseline at 715.1 FPS, Ensemble at 517.7 FPS, and MC Dropout at 213.4 FPS. The differences here are notable and demonstrate that uncertainty estimation incurs a considerable computational cost, with the most significant impact on the MC Dropout variant. Further detailed results for each validation dataset can be found in Table 6.1.

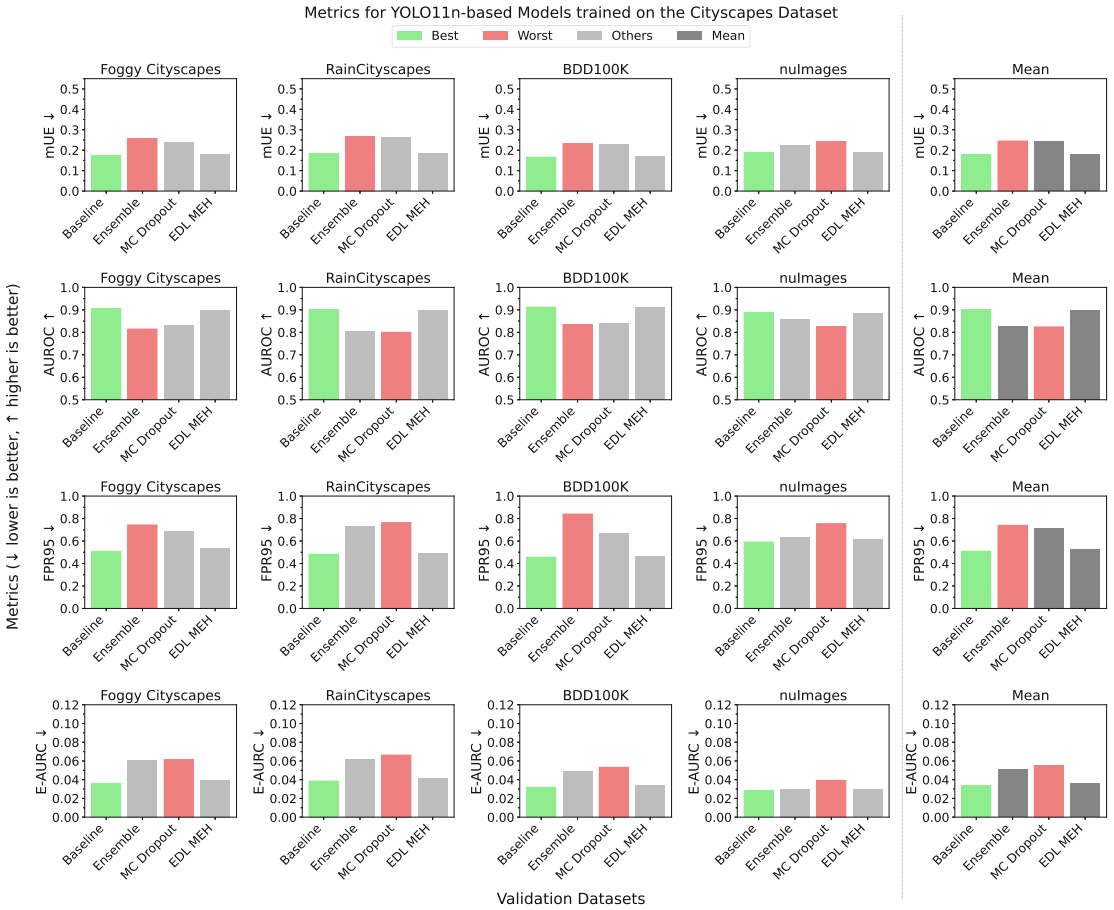


Figure 6.5: Validation performance grid for models trained on the Cityscapes dataset and validated on Foggy Cityscapes, RainCityscapes, BDD100K, and nuImages datasets. The metrics mUE, AUROC, FPR95, and E-AURC show the uncertainty estimation performance of each model.

Moving on to the uncertainty metrics in Figure 6.5 (detailed values again in Table 6.1), we observe more pronounced differences between the models. The minimum uncertainty error (mUE) is lowest for the Baseline at 0.178, indicating well-calibrated uncertainty estimates, followed closely by the EDL MEH variant at 0.180, MC Dropout at 0.244, and Ensemble at 0.245. The Area Under the Receiver Operating Characteristic curve (AUROC), measuring performance in distinguishing between correct and incorrect predictions based on uncertainty, shows the same pattern, where the Baseline comes out on top, not much

over the evidence-based EDL MEH, but a greater distance to the worst performing Ensemble and MC Dropout variants. The False Positive Rate at a 95% True Positive Rate (FPR95) again favors the uncertainty approximation method of the Baseline, with little difference from EDL MEH. Ensemble and MC Dropout again both have the overall worst performance compared to other models. Finally, the Expected Area Under the Risk-Coverage curve (E-AURC) is also lowest for the Baseline at 0.034, demonstrating its effectiveness in balancing risk and coverage based on uncertainty estimates. The EDL MEH model is next, at 0.036, followed by MC Dropout at 0.055 and Ensemble at 0.051. Similarly, the Baseline takes the lead, while only EDL MEH can keep up, and other approaches fall behind significantly

These results suggest that the Baseline is a well-calibrated model that provides the best uncertainty estimation performance among the evaluated methods. However, it is often only marginally better than the EDL MEH model. Ensemble and MC Dropout lag behind in most uncertainty metrics, particularly in terms of E-AURC, indicating that their uncertainty estimates are less well-calibrated. The trends of the best-performing models also stay relatively consistent across all individual validation datasets. The FPS metric experiences the most significant changes, likely due to the varying image sizes of each dataset requiring different pre-processing times.

Metric	Model	Foggy C.	RainC.	BDD100K	nuImages	Mean
mAP ↑	Baseline	**0.287**	**0.270**	0.244	0.314	**0.278**
	Ensemble	0.269	0.249	0.235	0.303	0.264
	MC Dropout	0.279	0.263	0.242	0.307	0.273
	EDL MEH	0.286	0.268	**0.244**	**0.314**	0.278
Precision ↑	Baseline	0.504	0.445	0.501	0.538	0.497
	Ensemble	**0.516**	**0.500**	**0.516**	**0.562**	**0.524**
	MC Dropout	0.485	0.426	0.492	0.535	0.485
	EDL MEH	0.507	0.437	0.503	0.542	0.497
Recall ↑	Baseline	**0.275**	**0.231**	**0.229**	**0.285**	**0.255**
	Ensemble	0.250	0.198	0.216	0.269	0.233
	MC Dropout	0.270	0.225	0.226	0.280	0.250
	EDL MEH	0.271	0.228	0.228	0.284	0.253
FPS ↑	Baseline	**648.4**	551.7	**810.3**	**849.8**	715.1
	Ensemble	483.8	459.9	546.8	580.4	517.7
	MC Dropout	201.3	203.2	223.6	225.7	213.4
	EDL MEH	629.8	**656.1**	780.6	807.2	**718.4**
mUE ↓	Baseline	**0.173**	**0.183**	**0.166**	**0.189**	**0.178**
	Ensemble	0.256	0.267	0.231	0.224	0.245
	MC Dropout	0.239	0.264	0.230	0.243	0.244
	EDL MEH	0.179	0.185	0.168	0.190	0.180
AUROC ↑	Baseline	**0.905**	**0.901**	**0.912**	**0.887**	**0.901**
	Ensemble	0.814	0.803	0.834	0.858	0.827
	MC Dropout	0.831	0.800	0.840	0.825	0.824
	EDL MEH	0.898	0.897	0.909	0.882	0.897
FPR95 ↓	Baseline	**0.508**	**0.483**	**0.456**	**0.591**	**0.509**
	Ensemble	0.743	0.737	0.840	0.640	0.740
	MC Dropout	0.690	0.764	0.668	0.755	0.719
	EDL MEH	0.538	0.489	0.469	0.620	0.529
E-AURC ↓	Baseline	**0.036**	**0.039**	**0.032**	**0.028**	**0.034**
	Ensemble	0.061	0.062	0.049	0.030	0.051
	MC Dropout	0.061	0.066	0.053	0.039	0.055
	EDL MEH	0.040	0.041	0.034	0.030	0.036

Table 6.1: Validation performance results for models trained on the Cityscapes dataset. Arrows indicate optimization direction (↑ higher is better, ↓ lower is better). Best values are shown in **bold**. Columns are validation datasets; the last column (after |) is the Mean across datasets. Abbrev.: Foggy C.=Foggy Cityscapes, RainC.=RainCityscapes.

In summary, the best uncertainty estimators are the Baseline and EDL MEH, which, interestingly, also have the lowest inference time overhead (highest FPS). MC Dropout and Ensemble, although they have slower runtime performance, do not provide an improvement over the Baseline in most cases. Most notably, MC Dropout achieves only 30% of the FPS compared to the Baseline, highlighting the significant inference time overhead of this sampling approach.

6.2.2 KITTI Results

In a similar fashion to the previous Subsection 6.2.1, here we present the results for models trained on the KITTI dataset, and then validated on the Foggy Cityscapes, RainCityscapes, BDD100K, and nuImages datasets. Detailed values for the shown barplots are listed in Table 6.2.

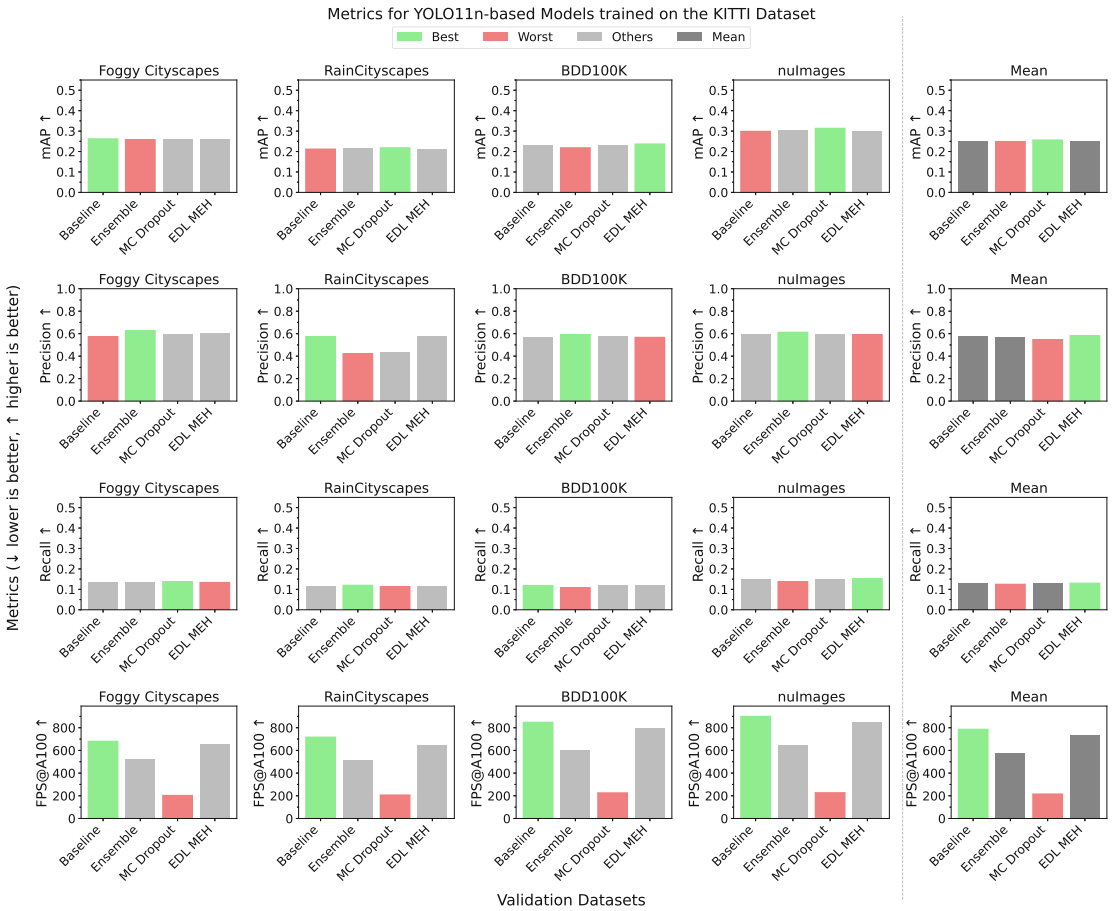


Figure 6.6: Validation performance grid for models trained on the KITTI dataset and validated on Foggy Cityscapes, RainCityscapes, BDD100K, and nuImages datasets. The metrics displayed are mAP, Precision, Recall, and FPS, which indicate the overall detection quality and computational cost.

Examining the general metrics for object detection performance in Figure 6.6, we observe similar performance among the Baseline, MC Dropout, and EDL MEH models, particularly in terms of mAP, precision, and recall. MC Dropout comes out on top in mAP slightly before the other models, while EDL MEH advances slightly in precision and recall. Specifically, the MC Dropout model achieves the highest mAP of 0.257 on average, followed closely by the Baseline at 0.252 and the EDL MEH at 0.252, while the Ensemble lags a little behind at 0.249. Precision is highest for the EDL MEH model at 0.583, followed by the Baseline at 0.577. In contrast, MC Dropout and Ensemble are significantly lower at 0.548 and 0.563, respectively. Recall values are very similar across all models, with the Ensemble model being slightly behind. The FPS metric shows the best performance for the Baseline model, with an average of 787.2 FPS, followed by EDL MEH at 734.5 FPS, Ensemble at 572.6 FPS, and MC Dropout at 216.4 FPS. These results overall show the same trends as in Subsection 6.2.1, with the additional observation that the models generally have higher precision across the validation datasets when trained on KITTI, but lower recall compared to models trained on Cityscapes. This showcases the general impact on the training dataset, regardless of the evaluated model.

Metrics focusing on uncertainty estimation performance in Figure 6.7 show that Baseline and EDL MEH again provide the best uncertainty estimation performance among the evaluated methods. The Baseline model achieves the lowest mUE at 0.172, followed by the EDL MEH at 0.177, with Ensemble and Dropout having higher values of 0.266 and 0.270, respectively. The AUROC metric follows the same trend. FPR95 shows a change where Ensemble is the worst performer by far, with the highest value of 0.837. MC Dropout retrieves an FPR95 of 0.718, which is still significantly higher than the Baseline (0.580) and EDL MEH (0.611) approaches. E-AURC exhibits the typical pattern, with Baseline and EDL MEH outperforming the other approaches, and MC Dropout having the worst value, approximately 77% higher than the Baseline. These results reinforce the findings from Subsection 6.2.1, highlighting the strong performance of the Baseline and, as a close second, the EDL-based approach, while Ensemble and MC Dropout struggle to justify their computational overhead.

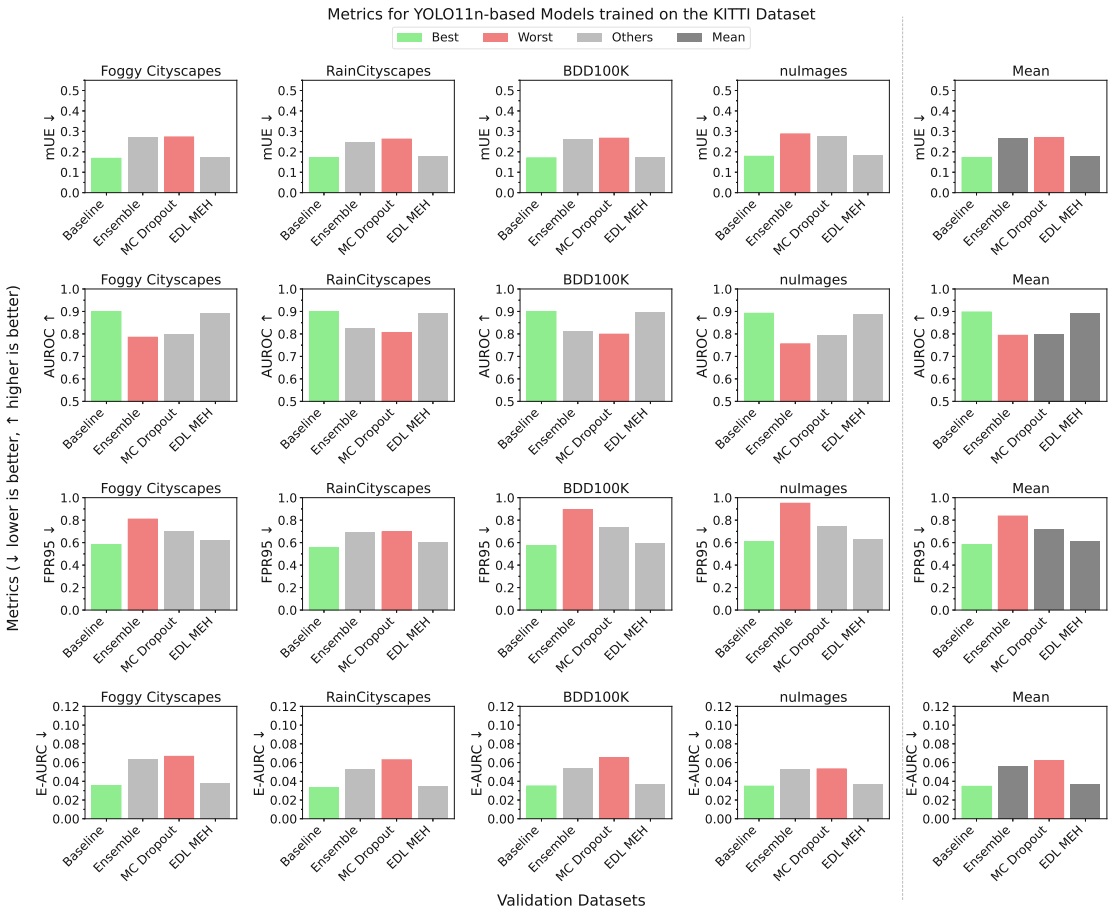


Figure 6.7: Validation performance grid for models trained on the KITTI dataset and validated on Foggy Cityscapes, RainCityscapes, BDD100K, and nuImages datasets. The metrics mUE, AUROC, FPR95, and E-AURC show the uncertainty estimation performance of each model.

Metric	Model	Foggy C.	RainC.	BDD100K	nuImages	Mean
mAP ↑	Baseline	**0.262**	0.212	0.233	0.299	0.252
	Ensemble	0.259	0.215	0.219	0.305	0.249
	MC Dropout	0.262	**0.219**	0.233	**0.314**	**0.257**
	EDL MEH	0.259	0.212	**0.237**	0.300	0.252
Precision ↑	Baseline	0.574	**0.573**	0.568	0.593	0.577
	Ensemble	**0.626**	0.420	**0.592**	**0.613**	0.563
	MC Dropout	0.591	0.434	0.571	0.596	0.548
	EDL MEH	0.601	0.572	0.568	0.590	**0.583**
Recall ↑	Baseline	0.134	0.117	**0.119**	0.151	0.130
	Ensemble	0.134	**0.120**	0.108	0.138	0.125
	MC Dropout	**0.137**	0.114	0.119	0.150	0.130
	EDL MEH	0.134	0.117	0.119	**0.152**	**0.130**
FPS ↑	Baseline	**681.3**	**717.6**	**849.1**	**900.7**	**787.2**
	Ensemble	522.5	516.0	606.2	645.9	572.6
	MC Dropout	203.7	207.5	226.5	227.9	216.4
	EDL MEH	653.5	643.4	794.6	846.5	734.5
mUE ↓	Baseline	**0.168**	**0.172**	**0.171**	**0.178**	**0.172**
	Ensemble	0.271	0.246	0.260	0.288	0.266
	MC Dropout	0.273	0.262	0.267	0.278	0.270
	EDL MEH	0.174	0.179	0.174	0.182	0.177
AUROC ↑	Baseline	**0.899**	**0.900**	**0.900**	**0.892**	**0.898**
	Ensemble	0.785	0.824	0.811	0.756	0.794
	MC Dropout	0.800	0.806	0.799	0.795	0.800
	EDL MEH	0.892	0.891	0.895	0.887	0.891
FPR95 ↓	Baseline	**0.583**	**0.557**	**0.573**	**0.607**	**0.580**
	Ensemble	0.809	0.693	0.894	0.950	0.837
	MC Dropout	0.697	0.695	0.736	0.745	0.718
	EDL MEH	0.619	0.604	0.597	0.624	0.611
E-AURC ↓	Baseline	**0.036**	**0.033**	**0.035**	**0.035**	**0.035**
	Ensemble	0.064	0.053	0.054	0.053	0.056
	MC Dropout	0.067	0.063	0.065	0.053	0.062
	EDL MEH	0.038	0.035	0.037	0.036	0.037

Table 6.2: Validation performance results for models trained on the KITTI dataset. Arrows indicate optimization direction (↑ higher is better, ↓ lower is better). Best values are shown in **bold**. Columns are validation datasets; the last column (after |) is the Mean across datasets. Abbrev.: Foggy C.=Foggy Cityscapes, RainC.=RainCityscapes.

6.2.3 Inference Time Analysis

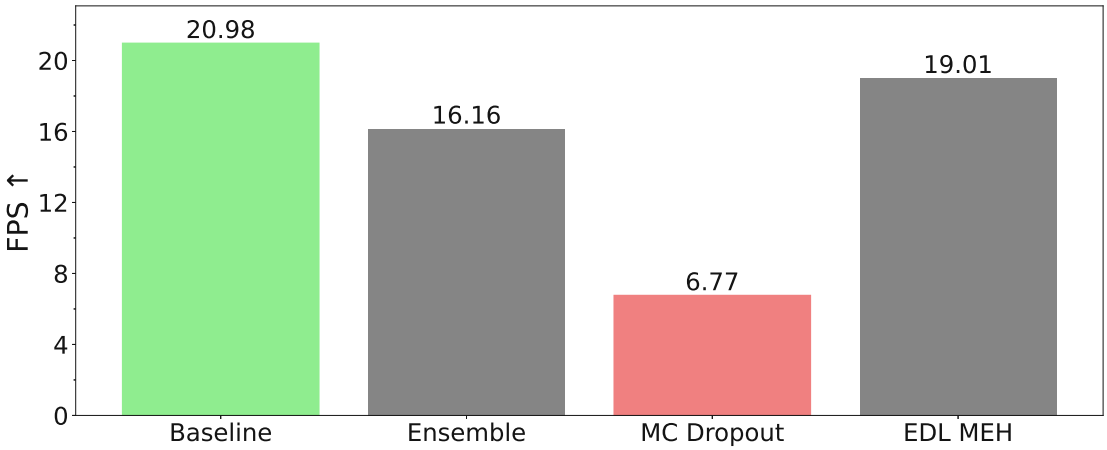


Figure 6.8: Inference time analysis showing FPS on CPU.

Training and Validation on large datasets have been done with the help of GPU acceleration. In this subsection, we export models to the runtime-optimized ONNX format [dev21] and perform an inference time evaluation on the CPU, comparing it to the GPU results from the previous section. Running the models on the CPU provides more realistic insights into how they would perform on resource-limited edge devices. In Figure 6.8, we can see the average FPS of each model when benchmarked on the RainCityscapes dataset. The Baseline with the fewest modifications is again the fastest, with 20.98 FPS, while the EDL MEH model is just slightly behind with 19.01 FPS. The Ensemble implementation drops noticeably by almost five FPS compared to the Baseline. However, the largest penalty in frame rate is observed with the MC Dropout approach, which has only 32% of the FPS of the Baseline. These results demonstrate that the inference time overhead resulting from uncertainty estimation can be significant. Assuming a typical real-time operation speed of 30 FPS [YKPC24], the Baseline and also the EDL MEH are close to achieving this. However, the MC Dropout approach cannot keep up, and depending on the underlying device, may not be fit for real-time inference.

CHAPTER 7

Discussion

7.1 Summary of Results

We began by analyzing the hyperparameters and confirmed a general assumption that increasing the number of forward passes in MC Dropout yields better uncertainty estimates. Surprisingly, our hyperparameter search revealed a significantly weaker dependence on the number of heads for the Ensemble approach, specifically the number of generated samples. As a result, by choosing only the three ensemble heads suggested by the tuning, the Ensemble head is far more computationally efficient than the MC Dropout head with 15 forward passes/samples. We did not find much optimization potential for EDL MEH and mostly stuck with the default configuration. One change we made to the EDL MEH proposed by [PCK+23] was to utilize the faster uncertainty calculation method from the original work of [SKK18], which enabled our EDL MEH head to achieve a similar detection speed to the Baseline.

We evaluated our approaches across eight metrics, focusing on detection quality, inference speed, and uncertainty estimation. In most cases, the Baseline came out on top, showcasing that a well-calibrated detection model does not necessarily benefit from additional uncertainty estimation; it may even hinder its performance. Although EDL MEH behaved very similarly to the baseline in terms of detection performance, uncertainty estimation, and FPS. The Ensemble and MC Dropout approaches were mostly on the same performance level metric-wise; however, given the significantly lower inference time of Ensemble in our implementation, it is the preferred choice when compared to MC Dropout.

7.2 Answering the Research Questions

1. *How can state-of-the-art uncertainty estimation techniques be effectively implemented for real-time object detection on the Edge?*

   Ensemble networks and stochastic (MC Dropout) networks are a foundation for uncertainty estimation approaches. Yet, their computational overhead has always been a bottleneck. In our implementation, we reduced this computational overhead by restricting uncertainty estimation only to a subpart of the entire network. This enables partial caching of the network to reduce redundant computations. Still, the demand for multiple samples remains, preventing the base model from reaching its detection speed, especially for MC Dropout. Therefore, we conducted a hyperparameter search with a focus on uncertainty estimation to select only the necessary number of samples while minimizing uncertainty error. For the evidence learning model, the EDL MEH approach from recent years provided an efficient starting point, from which we searched for further improvements in tuning the activation function and loss weighting of the evidence component. While the hyperparameter search indicates that there is no necessary need for optimization beyond their default values, we found room for improvement in uncertainty estimation in post-processing by switching to a less demanding version of the underlying estimation calculation from related literature. To conclude, implementing efficient uncertainty estimation requires combining knowledge from multiple foundational works in the literature and optimizing hyperparameters that have a high influence on runtime, as well as uncertainty estimation.

2. *What are the trade-offs regarding object detection quality and computational efficiency for additional uncertainty estimation?*

   The counterintuitive results from our work are that the simpler the approach, the better it performs. Ensemble and especially MC Dropout require significant additional computation without yielding better metrics. Judging by our results and evaluation setup, those two approaches are essentially trading something for nothing in return. The EDL MEH network demonstrated better performance with almost no inference overhead. However, while EDL MEH does not put additional demands on real-time detection, it also does not improve object detection or uncertainty estimation performance beyond the Baseline. To summarize, the trade-off discovered here is that the more intrusive the uncertainty estimation approach is in the base model network, the worse it performs. This can be exemplified by Ensemble and MC Dropout directly affecting the original network layers, while EDL MEH only branches off from what is already there. However, one downside of EDL MEH is that it introduces a new loss component, which affects the original training routine to a greater extent than Ensemble and MC Dropout.

3. *What recommendations can be made to enhance the trustworthiness of detection models at the edge?*

Judging by the evaluated approaches, we find the EDL MEH variant based on evidence learning is the best-performing method for true uncertainty estimates according to the literature. However, as the Baseline, with a simple proxy for uncertainty (complemented by classification scores), demonstrates such strong performance across all metrics and datasets, choosing a regular pre-trained state-of-the-art model is a valid choice, even in situations that could benefit from uncertainty estimation, such as our domain-shift scenario. However, as implemented in our benchmark, we cannot recommend Ensemble and MC Dropout as real-time uncertainty estimators, as they either do not provide better uncertainty estimates compared to the Baseline, or they suffer from framerate drops to one-third of the original.

CHAPTER 8

Conclusion

8.1 Summary

To conclude, our goal was to implement uncertainty estimation for real-time object detection on Edge devices. Our design choices were targeted at achieving fast inference speed, which is why we based our implementation on the slim YOLOv11n model. We researched various uncertainty estimation methods and developed four distinct versions of them. The Baseline represents a simple approximation of uncertainty estimation without any computational overhead. Ensemble and MC Dropout represent the classic uncertainty estimation methods in general, but given their inherent runtime overhead, we had to make some tweaks to our code to keep their runtime within bounds. By focusing our modifications solely on the head of the detector, we kept modifications to the network to a minimum and were even able to load pre-trained weights into all models. The EDL MEH approach represents a newer addition, which shows almost no overhead in inference time. Still, overall, our simply defined Baseline without any modifications to the network, remained on top, giving the other approaches a hard time justifying their usage. Although the EDL-based approach did not outperform the Baseline in many cases, it shows promise as a candidate for further optimization, as the gap to the Baseline is very narrow.

8.2 Limitations

A major limitation in this work is the implementation of the models and the evaluation framework. Neural networks come with numerous defining hyperparameters, and although we have analyzed the impact of some of them, a huge number remain untested. In other words, the performance we showed as results may not be the best the described approach could do. Yet, given the numerous options, trade-offs must be made, and parameters are set according to the literature or left as default in the original work.

Furthermore, the proposed models have not been tested on a wide range of hardware yet, which is particularly important for accurately estimating processing time. In our work, we provided FPS numbers for both CPU- and GPU-based evaluations to provide some contrast in the evaluation setup. We also note that further work is needed to modify our approaches to allow export to arbitrary formats and hardware. In our setup, we tested the original PyTorch-based implementation on the GPU and the ONNX export on the CPU to provide more diverse inference time measurements.

Lastly, the chosen uncertainty estimation approaches may not be adequately suited for the discussed use case. The domain-shift scenario evaluated here may put more focus on data uncertainty rather than model uncertainty. Therefore, the provided uncertainty estimates (except for the Baseline approximation) may not be fully utilized, as their key strength lies in also retrieving epistemic uncertainty. In the domain-shift scenario, we mostly change the scenery (different weather/location), but do not introduce new classes to detect. For example, a use case beyond domain shift would be open-set detection, where newly introduced classes may challenge the model with higher values of uncertainty than simply seeing the same classes in a different setting (domain shift). This setting may be more highly affected by uncertainty estimation, especially epistemic uncertainty [MSMD22].

8.3 Future Work

A potential extension of this work includes more uncertainty estimation approaches in the evaluation framework. The approaches discussed here mainly concern the most well-known ones, but many more are available to try [GTA+23]. One strategy worth investigating is how to optimize the use of the network architecture and effectively utilize the given outputs. For example, Miller et al. [MDMS19] utilize the many redundant bounding boxes of sampling-based object detection to first cluster them into observations, and then estimate uncertainty. Other transformer-based EDL approaches have emerged recently [PPSM25], potentially offering alternative possibilities for uncertainty estimation compared to classical CNN-based models. Using Gaussian Mixture Models (GMMs) has also shown success in the work of [MSMD22] on a different use case for detecting open-set errors, which could be adapted to fit our evaluation setup.

Overview of Generative AI Tools Used

The use of AI tools affects both the written thesis and the code used to produce results. ChatGPT-5 has been utilized in both cases, primarily for checking, debugging, generating Python code, and supporting TeX writing, particularly in the formula sections and for tables. Further support on the code side was provided by GitHub Copilot and Codex CLI. Thesis writing was augmented by Grammarly sentence rephrasing and grammar corrections. Generated content is primarily present in the code framework; however, all generated code blocks have been proofread and, in many cases, significantly refactored. The paragraphs in this thesis are handwritten and corrected using Grammarly suggestions, with AI auto-generation primarily employed for formula generation. These formulas have been verified and modified to align with the source literature. The only majorly copied paragraphs in this thesis are the German versions of the Abstract, Acknowledgments, and this paragraph, which are translations of the handwritten English counterparts using Google Translate.

Übersicht verwendeter Hilfsmittel

Der Einsatz von KI-Tools wirkt sich sowohl auf die schriftliche Arbeit als auch auf den zur Ergebnisgenerierung verwendeten Code aus. ChatGPT-5 wurde in beiden Fällen eingesetzt, hauptsächlich zur Überprüfung, Fehlerbehebung, Generierung von Python-Code und zur Unterstützung des TeX-Schreibens, insbesondere in den Formelabschnitten und bei den Tabellen. Weitere Unterstützung auf der Codeseite wurde durch GitHub Copilot und Codex CLI bereitgestellt. Das Schreiben der Arbeit wurde durch Satzumformulierungen und Grammatikkorrekturen von Grammarly ergänzt. Der generierte Inhalt ist hauptsächlich im Code-Framework vorhanden; alle generierten Codeblöcke wurden jedoch Korrektur gelesen und in vielen Fällen erheblich überarbeitet. Die Absätze dieser Arbeit sind handschriftlich verfasst und mit Grammarly-Vorschlägen korrigiert, wobei die automatische KI-Generierung hauptsächlich zur Formelgenerierung eingesetzt wurde. Diese Formeln wurden überprüft und an die Quellenliteratur angepasst. Die einzigen großteils kopierten Absätze in dieser Arbeit sind die deutschen Versionen der Kurzfassung, der Danksagung und dieses Paragraphen, die Übersetzungen der handschriftlichen englischen Gegenstücke mit Google Translate sind.

List of Figures

List of Tables

Bibliography

[ADNL24] Konstantinos Panagiotis Alexandridis, Jiankang Deng, Anh Nguyen, and Shan Luo. Adaptive parametric activation. In *European conference on computer vision*, pages 455–476. Springer, 2024.

[AS21] Alexander Amini and Ava Soleimany. MIT 6.S191: Introduction to deep learning lecture 7 evidential deep learning, 2021.

[ASSR20] Alexander Amini, Wilko Schwarting, Ava Soleimany, and Daniela Rus. Deep evidential regression. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in neural information processing systems*, volume 33, pages 14927–14937. Curran Associates, Inc., 2020.

[BR11] Ben Benfold and Ian Reid. Stable multi-target tracking in real-time surveillance video. In *CVPR 2011*, pages 3457–3464, Colorado Springs, CO, USA, June 2011. IEEE.

[CBL+20] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuScenes: a multimodal dataset for autonomous driving. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, June 2020.

[CKZC24] Minjie Cai, Jianaresi Kezierbieke, Xionghu Zhong, and Hao Chen. Uncertainty-Aware and Class-Balanced Domain Adaptation for Object Detection in Driving Scenes. *IEEE Transactions on Intelligent Transportation Systems*, 25(11):15977–15990, November 2024.

[CLZT24] Wei Chen, Jinjin Luo, Fan Zhang, and Zijian Tian. A review of object detection: Datasets, performance evaluation, architecture, applications and current trends. *Multimedia Tools and Applications*, 83(24):65603–65661, July 2024.

[COR+16] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele.

The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, June 2016.

[DAT23] Tausif Diwan, G. Anirudh, and Jitendra V. Tembhurne. Object detection using YOLO: challenges, architectural successors, datasets and applications. *Multimedia Tools and Applications*, 82(6):9243–9275, March 2023.

[DCB+24] Ruxiao Duan, Brian Caffo, Harrison X. Bai, Haris I. Sair, and Craig Jones. Evidential Uncertainty Quantification: A Variance-Based Perspective. In *2024 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 2121–2130, Waikoloa, HI, USA, January 2024. IEEE.

[dev21] ONNX Runtime developers. ONNX runtime, 2021.

[DG06] Jesse Davis and Mark Goadrich. The relationship between Precision-Recall and ROC curves. In *Proceedings of the 23rd international conference on Machine learning*, ICML '06, pages 233–240, New York, NY, USA, June 2006. Association for Computing Machinery.

[DLXS20] Yukun Ding, Jinglan Liu, Jinjun Xiong, and Yiyu Shi. Revisiting the evaluation of uncertainty estimation and its application to explore model complexity-uncertainty trade-off. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (CVPR) workshops*, June 2020.

[DWGL22] Xuefeng Du, Xin Wang, Gabriel Gozum, and Yixuan Li. Unknown-Aware Object Detection: Learning What You Don't Know from Videos in the Wild. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13668–13678, New Orleans, LA, USA, June 2022. IEEE.

[EVGW+10] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010. Publisher: Springer.

[EW11] Mark Everingham and John Winn. The pascal visual object classes challenge 2012 (voc2012) development kit. *Pattern Analysis, Statistical Modelling and Computational Learning, Tech. Rep*, 8:5, 2011.

[GDS20] Fredrik K. Gustafsson, Martin Danelljan, and Thomas B. Schon. Evaluating Scalable Bayesian Deep Learning Methods for Robust Computer Vision. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1289–1298, Seattle, WA, USA, June 2020. IEEE.

[GG16] Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 1050–1059. PMLR, June 2016. ISSN: 1938-7228.

[GGH+24] Sukhpal Singh Gill, Muhammed Golec, Jianmin Hu, Minxian Xu, Junhui Du, Huaming Wu, Guneet Kaur Walia, Subramaniam Subramanian Murugesan, Babar Ali, Mohit Kumar, Kejiang Ye, Prabal Verma, Surendra Kumar, Felix Cuadrado, and Steve Uhlig. Edge AI: A Taxonomy, Systematic Review and Future Directions. *Cluster Computing*, 28(1):18, October 2024.

[GLL18] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. DropBlock: A regularization method for convolutional networks. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[GLU12] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? The KITTI vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361, June 2012. ISSN: 1063-6919.

[GTA+23] Jakob Gawlikowski, Cedrique Rovile Njieutcheu Tassi, Mohsin Ali, Jongseok Lee, Matthias Humt, Jianxiang Feng, Anna Kruspe, Rudolph Triebel, Peter Jung, Ribana Roscher, Muhammad Shahzad, Wen Yang, Richard Bamler, and Xiao Xiang Zhu. A survey of uncertainty in deep neural networks. *Artificial Intelligence Review*, 56(1):1513–1589, October 2023.

[GUEY19] Yonatan Geifman, Guy Uziel, and Ran El-Yaniv. Bias-Reduced Uncertainty Estimation for Deep Neural Classifiers, April 2019. arXiv:1805.08206 [cs].

[HBS17] Jan Hosang, Rodrigo Benenson, and Bernt Schiele. Learning Non-maximum Suppression. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6469–6477, Honolulu, HI, July 2017. IEEE.

[HFZH19] Xiaowei Hu, Chi-Wing Fu, Lei Zhu, and Pheng-Ann Heng. Depth-Attentional Features for Single-Image Rain Removal. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8014–8023, June 2019. ISSN: 2575-7075.

[HJW+24] Qinghua Hu, Luona Ji, Yu Wang, Shuai Zhao, and Zhibin Lin. Uncertainty-driven active developmental learning. *Pattern Recognition*, 151:110384, July 2024.

[HSRW20] Maximilian Henne, Adrian Schwaiger, Karsten Roscher, and Gereon Weiss. Benchmarking uncertainty estimation methods for deep learning with safety-related metrics. In *SafeAI@AAAI*, volume 2560 of *CEUR workshop proceedings*, pages 83–90. CEUR-WS.org, 2020.

[JQ24] Glenn Jocher and Jing Qiu. Ultralytics YOLO11, 2024. tex.orcid: 0000-0001-5950-6979, 0000-0002-7603-6750, 0000-0003-3783-7069.

[KB13] Barbara Kitchenham and Pearl Brereton. A systematic review of systematic review process research in software engineering. *Information and Software Technology*, 55(12):2049–2075, December 2013.

[KG17] Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in neural information processing systems*, volume 30. Curran Associates, Inc., 2017.

[KKM21] Parvinder Kaur, Baljit Singh Khehra, and Er. Bhupinder Singh Mavi. Data Augmentation for Object Detection: A Review. In *2021 IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 537–543, August 2021. ISSN: 1558-3899.

[KMH+21] Faris A. Kateb, Muhammad Mostafa Monowar, Md Abdul Hamid, Abu Quwsar Ohi, and Muhammad Firoz Mridha. FruitDet: Attentive Feature Aggregation for Real-Time Fruit Detection in Orchards. *Agronomy*, 11(12):2440, December 2021. Publisher: Multidisciplinary Digital Publishing Institute.

[LAE+16] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single Shot MultiBox Detector. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, volume 9905, pages 21–37. Springer International Publishing, Cham, 2016. Series Title: Lecture Notes in Computer Science.

[LLN+18] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. Tune: a research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018.

[LLW+23] Xiang Li, Chengqi Lv, Wenhai Wang, Gang Li, Lingfeng Yang, and Jian Yang. Generalized Focal Loss: Towards Efficient Representation Learning for Dense Object Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(3):3139–3153, March 2023. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[LMB+14] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common objects in context. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer vision – ECCV 2014*, pages 740–755, Cham, 2014. Springer International Publishing.

[LMP+21] Ivan Lujic, Vincenzo De Maio, Klaus Pollhammer, Ivan Bodrozic, Josip Lasic, and Ivona Brandic. Increasing Traffic Safety with Real-Time Edge Analytics and 5G. In *Proceedings of the 4th International Workshop on Edge Systems, Analytics and Networking*, pages 19–24, Online United Kingdom, April 2021. ACM.

[LOW+20] Li Liu, Wanli Ouyang, Xiaogang Wang, Paul Fieguth, Jie Chen, Xinwang Liu, and Matti Pietikäinen. Deep Learning for Generic Object Detection: A Survey. *International Journal of Computer Vision*, 128(2):261–318, February 2020.

[LPB17] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *NIPS*, pages 6402–6413, 2017.

[MDMS19] Dimity Miller, Feras Dayoub, Michael Milford, and Niko Sunderhauf. Evaluating Merging Strategies for Sampling-based Uncertainty Techniques in Object Detection. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 2348–2354, Montreal, QC, Canada, May 2019. IEEE.

[Mil20] Dimity Miller. Probabilistic Object Detection with an Ensemble of Experts. In Adrien Bartoli and Andrea Fusiello, editors, *Computer Vision – ECCV 2020 Workshops*, volume 12540, pages 46–55. Springer International Publishing, Cham, 2020. Series Title: Lecture Notes in Computer Science.

[Mil21] Dimity Miller. *Epistemic uncertainty estimation for object detection in open-set conditions*. PhD, Queensland University of Technology, 2021.

[MNDS17] Dimity Miller, Lachlan Nicholson, Feras Dayoub, and Niko Sünderhauf. Dropout variational inference improves object detection in open-set conditions. In *Bayesian deep learning workshop at the internation conference on neural information processing systems*, 2017.

[MNDS18] Dimity Miller, Lachlan Nicholson, Feras Dayoub, and Niko Sunderhauf. Dropout Sampling for Robust Object Detection in Open-Set Conditions. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3243–3249, Brisbane, QLD, May 2018. IEEE.

[MPS+25] André Magalhães Moraes, Luiz Felipe Pugliese, Rafael Francisco dos Santos, Giovani Bernardes Vitor, Rodrigo Aparecido da Silva Braga, and Fernanda Rodrigues da Silva. Effectiveness of YOLO Architectures in Tree Detection: Impact of Hyperparameter Tuning and SGD, Adam, and AdamW Optimizers. *Standards*, 5(1):9, March 2025. Publisher: Multidisciplinary Digital Publishing Institute.

[MSMD22] Dimity Miller, Niko Sunderhauf, Michael Milford, and Feras Dayoub. Uncertainty for Identifying Open-Set Errors in Visual Object Detection. *IEEE Robotics and Automation Letters*, 7(1):215–222, January 2022.

[MSZ+19] Dimity Miller, Niko Sünderhauf, Haoyang Zhang, David Hall, and Feras Dayoub. Benchmarking sampling-based probabilistic object detectors. In *CVPR workshops*, pages 42–45. Computer Vision Foundation / IEEE, 2019.

[OD08] David L. Olson and Dursun Delen. *Advanced Data Mining Techniques*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[OSVP24] Poojan Oza, Vishwanath A. Sindagi, Vibashan Vs, and Vishal M. Patel. Unsupervised Domain Adaptation of Object Detectors: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(6):4018–4040, June 2024.

[PCK+23] Younghyun Park, Wonjeong Choi, Soyeong Kim, Dong-Jun Han, and Jaekyun Moon. Active learning for object detection with evidential deep learning and hierarchical uncertainty aggregation. In *The eleventh international conference on learning representations*, 2023.

[PMBN20] Horia Porav, Valentina-Nicoleta Musat, Tom Bruls, and Paul Newman. Rainy screens: Collecting rainy datasets, indoors, March 2020. arXiv:2003.04742 [cs].

[PPSM25] Tejas Pandey, Nick Pears, William A P Smith, and John Alexander McDermid. E-DETR: Evidential deep learning for end-to-end uncertainty estimation in object detection, 2025.

[RDGF16] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *CVPR*, pages 779–788. IEEE Computer Society, 2016.

[RF17] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. In *CVPR*, pages 6517–6525. IEEE Computer Society, 2017.

[RF18] Joseph Redmon and Ali Farhadi. YOLOv3: An Incremental Improvement, April 2018. arXiv:1804.02767 [cs].

[RHGS15] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[Ros19] Harry Rose. *Evidential deep learning for uncertainty estimation in object detection*. MSc Dissertation, University College London, Department of Computer Science, London, United Kingdom, September 2019.

[RT24] Xiaoqian Ruan and Wei Tang. Fully Test-time Adaptation for Object Detection. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1038–1047, Seattle, WA, USA, June 2024. IEEE.

[SBD+14] Robin Senge, Stefan Bösner, Krzysztof Dembczyński, Jörg Haasenritter, Oliver Hirsch, Norbert Donner-Banzhoff, and Eyke Hüllermeier. Reliable classification: Learning classifiers that distinguish aleatoric and epistemic uncertainty. *Information Sciences*, 255:16–29, January 2014.

[SDVG18] Christos Sakaridis, Dengxin Dai, and Luc Van Gool. Semantic Foggy Scene Understanding with Synthetic Data. *International Journal of Computer Vision*, 126(9):973–992, September 2018.

[Sha18] Rajalingappaa Shanmugamani. *Deep learning for computer vision: Expert techniques to train advanced neural networks using TensorFlow and keras*. Packt Publishing Ltd, 2018.

[Sha24] Mohit Sharma. turtle dataset Dataset, October 2024. Type: Open Source Dataset.

[SHK+14] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[SKK18] Murat Sensoy, Lance Kaplan, and Melih Kandemir. Evidential deep learning to quantify classification uncertainty. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in neural information processing systems*, volume 31. Curran Associates, Inc., 2018.

[SMC+24] Ranjan Sapkota, Zhichao Meng, Martin Churuvija, Xiaoqiang Du, Zenghong Ma, and Manoj Karkee. Comprehensive Performance Evaluation of YOLO11, YOLOv10, YOLOv9 and YOLOv8 on Detecting and Counting Fruitlet in Complex Orchard Environments. *Qeios*, October 2024.

[SSRRR24] Mupparaju Sohan, Thotakura Sai Ram, and Ch. Venkata Rami Reddy. A Review on YOLOv8 and Its Advancements. In I. Jeena Jacob, Selwyn Piramuthu, and Przemyslaw Falkowski-Gilski, editors, *Data Intelligence and Cognitive Informatics*, pages 529–545, Singapore, 2024. Springer Nature.

[TCERG23] 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. *Machine Learning and Knowledge Extraction*, 5(4):1680–1716, December 2023. Number: 4 Publisher: Multidisciplinary Digital Publishing Institute.

[TYD25] Yunjie Tian, Qixiang Ye, and David Doermann. YOLOv12: Attention-Centric Real-Time Object Detectors, February 2025. arXiv:2502.12524 [cs].

[VOP23] Vibashan VS, Poojan Oza, and Vishal M. Patel. Towards online domain adaptive object detection. In *WACV*, pages 478–488. IEEE, 2023.

[VT19] Matias Valdenegro-Toro. Deep Sub-Ensembles for Fast Uncertainty Estimation in Image Classification, November 2019. arXiv:1910.08168 [cs].

[VV24] Ajantha Vijayakumar and Subramaniyaswamy Vairavasundaram. YOLO-based Object Detection Models: A Review and its Applications. *Multimedia Tools and Applications*, 83(35):83535–83574, October 2024.

[WFD+23] Samuel Wilson, Tobias Fischer, Feras Dayoub, Dimity Miller, and Niko Sünderhauf. SAFE: Sensitivity-Aware Features for Out-of-Distribution Object Detection. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 23508–23519, Paris, France, October 2023. IEEE.

[WSLL25] Ke Wang, Chongqiang Shen, Xingcan Li, and Jianbo Lu. Uncertainty Quantification for Safe and Reliable Autonomous Vehicles: A Review of Methods and Applications. *IEEE Transactions on Intelligent Transportation Systems*, 26(3):2880–2896, March 2025.

[WZZF24] Ruofan Wang, Rui-Wei Zhao, Xiaobo Zhang, and Rui Feng. Towards Evidential and Class Separable Open Set Object Detection. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(6):5572–5580, March 2024.

[YCW+20] Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan, and Trevor Darrell. BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning, April 2020. arXiv:1805.04687 [cs].

[YKPC24] Sai Harsha Yelleni, Deepshikha Kumari, Srijith P.K., and Krishna Mohan C. Monte Carlo DropBlock for modeling uncertainty in object detection. *Pattern Recognition*, 146:110003, February 2024.

[ZCS+23] Zhengxia Zou, Keyan Chen, Zhenwei Shi, Yuhong Guo, and Jieping Ye. Object Detection in 20 Years: A Survey. *Proceedings of the IEEE*, 111(3):257–276, March 2023.

[ZWX+24] Rui Zhao, Kui Wang, Yang Xiao, Fei Gao, and Zhenhai Gao. Leveraging Monte Carlo Dropout for Uncertainty Quantification in Real-Time Object Detection of Autonomous Vehicles. *IEEE Access*, 12:33384–33399, 2024.

[ZYCX25] Yuxian Zhou, Xiaodong Yue, Yufei Chen, and Shaorong Xie. Deep Evidential Active Learning with Uncertainty-Aware Determinantal Point Process. In Apostolos Antonacopoulos, Subhasis Chaudhuri, Rama Chellappa, Cheng-Lin Liu, Saumik Bhattacharya, and Umapada Pal, editors, *Pattern Recognition*, pages 17–32, Cham, 2025. Springer Nature Switzerland.

[ZYX24] Yu Zhu, Qiang Yang, and Li Xu. Active learning enabled low-cost cell image segmentation using bounding box annotation. *CoRR*, abs/2405.01701, 2024.