Carnegie Mellon University
Software Engineering Institute

A Brief Introduction to the Evaluation of Learned Models for Aerial Object Detection

Eric Heim

May 2022

TECHNICAL REPORT

Artificial Intelligence Division

http://www.sei.cmu.edu

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

This report was prepared for the SEI Administrative Agent AFLCMC/AZS 5 Eglin Street Hanscom AFB, MA 01731-2100

DM22-0470

1 Introduction

Satellites equipped with imaging sensors represent key assets for intelligence, surveillance, and reconnaissance (ISR) missions. However, the shear volume of data that can potentially be gained from tasking these assets can quickly become too much for humans to manually consider in its entirety. It is for this reason that automated systems for analyzing satellite imagery are vital to aid human analysts in the process of extracting actionable information from satellite imagery.

One of the most promising tools for automation is machine learning (ML). ML has shown great success in making accurate inferences from imagery in a number of collection contexts, such as medical imaging [17], and vision for autonomous vehicles [21]. As such, it stands to reason that machine learning models and methods that have been successful in other collection contexts can be used on images taken from above. Indeed, there have been a number of public success stories of using machine learning to make accurate inferences on satellite images. Problem domains ranging from humanitarian and disaster relief [57] to agriculture planning [13] have benefitted from ML-based computer vision systems.

The utility of using machine learning on satellite imagery motivates the development of systems that incorporate machine learned models as core components to analysts workflows. This goal of this document is to provide a small part of the necessary practical guidance to develop such models. To survey the current knowledge on this topic and distill it into practical guidance is a daunting task. In order to provide a more focused guide, we scope this work in a number of ways. We focus on *object detection* - the abstract task of locating the position of objects within an image as well as what category of objects they belong to. We choose this problem for a number of reasons. First, many ISR tasks can be posed as object detection tasks. For example, detecting specific land vehicles, sea vessels, equipment, and people from satellite imagery can all be posed as an object detection tasks. Even if the ISR mission demands more sophisticated reasoning over objects, many of the principles of object detection apply. Second, object detection is one of the more mature fields of modern machine learning in the image domain. As a result, there is a considerable amount of prior work to draw from in order to build accurate object detection models.

We also focus this work on the *evaluation* of object detection models (or object detectors). Namely, we focus on 1) a sampling of characteristics of object detectors that are important to evaluate, 2) a sampling of common metrics for the evaluation of object detectors, 3) how to identify and define evaluation criteria for important use cases, and 4) how to map object detector characteristics and use cases to metrics in order to evaluate object detectors in specific, measurable ways. Because evaluation naturally occurs after data is collected and detectors are learned, we must necessarily touch on details related to relevant to both of these topics as well. Our goal is not to provide a comprehsive treatment of these topics. Rather, we aim to provide basic information as a means to understand some of the unique considerations when designing, learning, and evaluating object detectors, and supplement it with references to allow readers avenues to explore in greater depth.

The remainder of this guide proceeds as follows. In Chapter 2 we formally define the object detection problem, discuss modern approaches for learning object detection models, and provide some practical insights on what the formalisms mean for satellite imagery. In Chapter 3, we discuss practices for evaluation of object detectors, including methodology, metrics, and how to map these to meaningful notions of detector performance. In Chapter 4, we showcase some of these topics in practice by showing the results of an evaluation of object detection models on aerial imagery. Finally, in Chapter 5 we conclude with a summary of practical insights explored in the document, and provide a brief survey in open topics in object detection.

2 Object Detectors for Aerial Imagery

Object detection is a *supervised learning* problem, meaning, one in which a model learns to accept an input and map it to a corresponding output. Formally, the goal of supervised learning is to find a model $f : \mathcal{X} \mapsto \mathcal{Y}$, where $\mathcal{X}$ is a domain of inputs, and $\mathcal{Y}$ is a domain of outputs. More specifically, it is assumed that there exists a distribution P over $\mathcal{X}$ and $\mathcal{Y}$ for which pairs (x, y) can be drawn. An entirely successful model is able to take any input $x \in \mathcal{X}$ and output the corresponding output $y \in \mathcal{Y}$ that is its pair as drawn from P. Commonly, x is called an *instance*, y is called a *label*, and the process of applying f to an instance to get an output (i.e. $f(x)$) is called *inference*. Supervised learning is used in cases where the direct relationship between instances and labels is not known a prioi (equivalently, P is not known a priori). Because the mapping between the two is not known, supervised models are built from data, specifically examples of pairs $(x, y) \sim P$. The process of building a model from data is called *training*, and the data used in the process is called *training data*. Even though the model is trained using a finite amount of data, the model is meant to *generalize*, meaning it is able to infer the correct label when given any instance (i.e. $\forall_{(x,y)\tilde{P}} f(x) \mapsto y$) even if (x, y) is not in the training data. Supervised learning encapsulates a number of machine learning problems, and is given a deeper formal treatment in a number of introductory texts [3, 30, 24]. *Object detection* is a uniquely defined by its input and output domains. In object detection inputs are images and outputs are the *location* and *class* of objects within the image. In the following sections, we describe what this means more specifically. Along the way, we will discuss the role of data in training and inference, details on the practice of training and performing inference with an object detector, and relevant prior work.

2.1 Object Detection Preliminaries

While there are substantial differences between object detection methods, there are many details that are common among almost all modern detectors. Broadly, there are three main design decisions when building an object detector. First, the function f, typically called the *model* or in this case the *detector*, is assumed to take some form with free parameters to be set via the process of training. Choosing the basic form of the model is an important design decision that influences how well a trained model will perform on a task. Second, training must be guided by some quantifiable measure of success for the training task called an *objective function*. Informally, the objective function defines the criteria used by a training algorithm to determine how to adjust a model's parameters to result in the "best" model. Finally, a *training algorithm* takes a model form, an objective, and training data and outputs a trained model. Grounded in these core concepts, the remainder of this section is dedicated to providing details in the practice of training and using object detectors.

2.1.1 Data: Aerial Imagery and Object Labels

Because object detection is defined by the inputs and outputs of the problem, a good starting place for discussion is with data. For object detection performed on aerial satellite imagery we assume that the domain of instances are images, captured within certain spectral bands. Common imaging bands include standard red, blue, and green bands for visual imagery, as well as versions of infrared including near, mid, far, and thermal infrared [60]. In addition, other bands, such as those from hyperspectral imaging sensors [10], can also be used. While we forgo discussions of special considerations for sensing in different spectral bands (preprocessing, transformation, noise correction, etc.), we make the assumption going forward that images are represented by a grid of pixel measurements per band. As such, an instance is defined as $x = \mathbf{x} \in \mathcal{X} = \mathbb{R}^{n \times m \times d}$. Here, an instance $\mathbf{x}$ is a tensor of real numbers that represent

an image that is n pixels long, m pixels wide, and captured at d bands. For instance, if we want to learn an object detector that is able to identify objects in 1080p visual imagery, we can assume that a data instance comes in the form of a $1920 \times 1080 \times 3$ tensor of floating point numbers (see Fig 2.1).

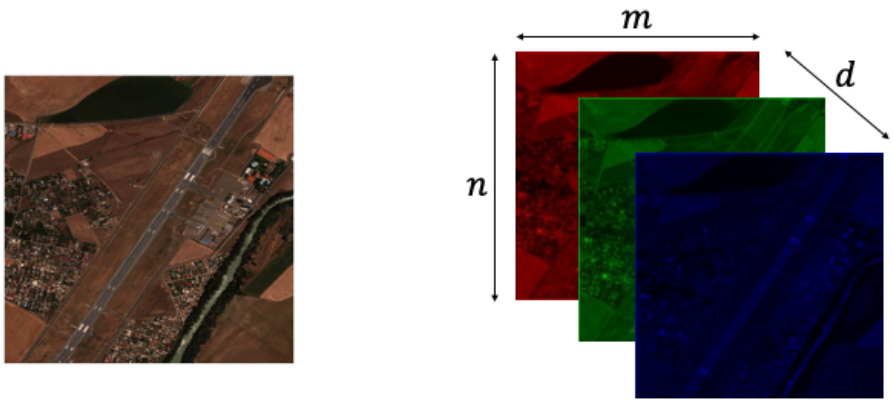

Figure 2.1: Left: Standard visual image (Image Credit: RarePlanes Data Set [61]). Right: Same image, but represented as a $n \times m \times 3$ tensor.

The labels in object detection problems come in the form of a number of *bounding boxes* that describe the location of object in an image and *class labels*, which describe the category of each object. Together, a bounding box and it's class label make a *labeled bounding box*. Formally a single labeled bounding box is a pair defined as $y \in \mathcal{Y} = \{\mathcal{B} \times \mathcal{C}\}$. The bounding box, $b \in \mathcal{B}$, is defined by four coordinates $b = (x_1, y_1, x_2, y_2)$, such that $x_1, x_2 \in [0, n]$, and $y_1, y_2 \in [0, m]$. These coordinate specify top-left and bottom-right pixels of the bounding box. Intuitively, the object that the bounding box is annotating is meant to be fully contained within the "box" designated by these coordinates. Other forms of bounding boxes also can be used, such as those designating the center pixel of an object and the length and height of the bounding box. Bounding boxes can even be oriented, as to incorporate rotation of objects relative to the axes of the image [69]. A class label is defined as $c \in \mathcal{C}$, where $\mathcal{C}$ is a set of *classes* in which the model is trying to categorize objects. For instance, if you wish to detect trucks, boats, and people from aerial imagery, then $\mathcal{C} = \{truck,\ boat,\ person\}$. Note that $\mathcal{C}$, the set of classes, defines what is meant to be detected as objects. Any class of objects not in $\mathcal{C}$ do not have corresponding training data in order for the model to learn how to detect, and thus will not be detected by the model. So, if $\mathcal{C} = \{truck,\ boat,\ person\}$, the model will not be trained to detect other objects in images such as airplanes or buildings, even if they appear in images. This is known as the *closed world assumption*, and is common among supervised learning problems.

Three final notes about data. First, labeled bounding boxes obtained for the purpose of training or evaluation are typically called *ground truth*, as they represent the "true" output in which a detector is meant to mimic. By contrast, labeled bounding boxes obtained via inference from the model being trained or evaluated are often called *predictions*. This is an important distinction as both training and evaluation is dependant on comparing ground truth to predictions. Second, it is important to note that the one-to-many nature of of object detection is somewhat unique among supervised learning problems. Namely, a single image can have many labeled bounding boxes. It is more typical in supervised learning problems that each instance has a fixed number of labels. This is one reason why object detection is a complex problem. Third, there is a practical challenge with obtaining data to train and evaluate an object detector. Sensors need to be tasked to obtain images. Images then often need to be trans-

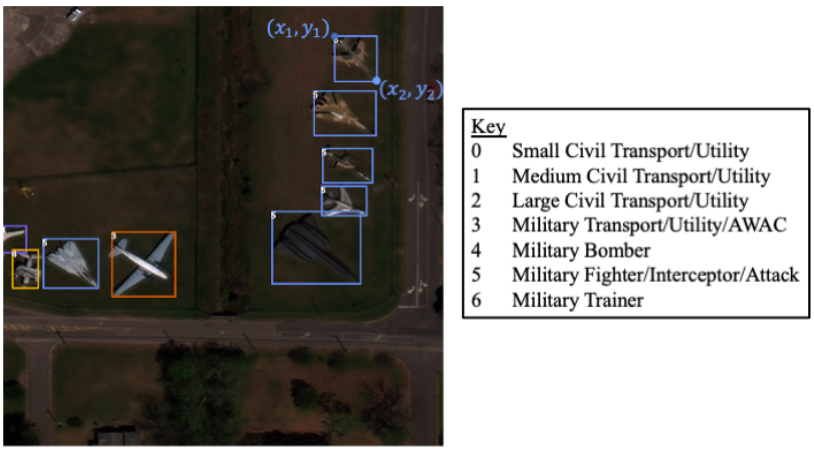

Figure 2.2: An example of an image, annotated with labeled bounding boxes (Image Credit: RarePlanes Data Set [61]). For example, the object in the upper right is labeled as being a member of the "Military fighter/interceptor/attack" class. The location of the object is defined by the upper-left, (x_1, y_1), and bottom-right, (x_2, y_2), coordinates of a bounding box.

mitted from the sensing platform to a specialized computing platform. In some settings this is not a major roadblock. However, obtaining ground truth often is. The most popular way of obtaining labels is to task human annotators to provide them. Considering most benchmark data sets for object detection start at thousands of images, and potentially tens of labeled bounding boxes per image, this can be costly in terms of time, money, and/or human capital, especially if labeling requires humans with specific domain expertise. As such, much of the effort in building production-quality object detectors is in obtaining, curating, preproccessing, and managing data.

2.1.2 Processing and Additional Object Detector Inputs

The quality of training data is perhaps the most influential factor in the success of training a supervised model. However, for complex problems, such as aerial object detection, it is difficult to practically obtain training data that encapsulates all possible scenarios in which a detector is meant to be used on. For this reason another important design decision is the choice of *data augmentation*. Data augmentation refers to the process of transforming training data as a means to expand the variation of instances used during training. Common forms of augmentation for images include randomly cropping, rotating, and/or flipping images before training. These kinds of manual augmentations can be understood as simulations of natural variations that can be present in the data. For example, it is possible that an image could be taken while the satellite is in a different orientation relative to the Earth. Rotating an image can effectively simulate this phenomenon. Another way of understanding augmentations is that they aim to reduce the influence of certain image characteristics, such as rotation, on a detector. Including images rotated randomly forces the detector to learn objects without considering their orientation. Similarly, augmentations such as adjusting blur, discoloration, and brightness simulate sensor noise. Even further, learned data augmentations [14, 76] can be used to alleviate the burden of manually defining augmentations such as those listed above. Finally, domain specific data augmentations exist. For instance, augmentations that remove cloud cover [18] have been developed to denoise both data for training as well as data instances before inference. Many of these techniques significantly improve object detector performance, and should be considered when training a object detector.

To do proper localization, object detectors must predict not only the location of objects in the

image, but also their size and aspect ratios (width, and height). Without given prior knowledge, the learned object detectors are forced to learn size and aspect ratio of objects from data alone, which can often be a difficult task. Fortunately, in many tasks, the size and shape of objects is relatively constrained. In aerial object detection, for example, it is rare that an object would take up the majority of the entire image frame. Similarly, larger aircraft would naturally require bigger bounding boxes than objects such as people. Given that such domain specific knowledge is almost always known to some degree, it makes sense to provide it to the detector. To accomplish this, object detector are often given *anchor boxes* as inputs. Anchor boxes are predefined boxes of different sizes and aspect ratios that can reflect prior knowledge. Anchor boxes can be class specific in cases where the size and aspect ratio can be constrained differently for individual classes, and spatially influenced if it size and aspect ratio is dependent on location within the image. Both object detectors we will discuss later, as well as many other modern ones, utilize anchor boxes by predict bounding box coordinates relative to anchor box coordinates, instead of in terms of absolute coordinates in the image. This way, object detectors have some guiding information as to how to localize objects.

2.1.3 Neural Network Details

Modern object detectors are neural network based. More specifically, most utilize models that can be categorized as Convolutional Neural Networks (CNNs) [1]. CNNs are a series shift-invariant linear transformations, each followed by a nonlinear function, organized into *layers* that feed into each other to form a network. Intuitively, the outputs of each layer of the CNN defines a *feature representation* of an input image. By training the parameters of a CNN (the weights, and sometimes the biases, of the linear transformations) to optimize an objective function, one is effectively learning representations of the input that is most amenable to a learning task. Individual CNNs are defined by their *architecture*, the structure that defines how many transformations are to be performed and how they are organized to feed into one another. Different object detection methods are characterized in part by how they use CNNs within their model, and the architectures of those CNNs. These decisions influence both the ability of the detector to provide accurate predictions, but also the run time of training and inference procedures (e.g. more layers means more parameters, which often means a more expressive model, but also often means longer run time to train and perform inference).

The specific CNN architecture used by object detectors is somewhat independent from other design decisions or even the detection task itself. As such, the network used to perform initial feature extraction in an object detector, called a *backbone network*, has received special attention. Many older models can be improved by simply replacing the backbone network with ones based on advances in neural network based representation learning. For instance, the first object detectors used traditional CNN backbones such as AlexNet [37]. But, architectural advancements such as Residual Networks (ResNets) [31], Feature Pyramid Networks (FPNs) [41], and Visual Transformers (ViTs) [40] have each seen wider usage as they have been introduced. Indeed, many improvements in object detectors over the last decade can be partially attributed to the act of taking older methods and updating their backbone networks. Finally, on a practical note, it is common practice to *pre-train* a backbone network on a large publicly-available data set such as ImageNet [15]. Pretraining is the process of training a neural network on an auxiliary data set and task before training it for a desired task. Intuitively, this allows the backbone to be able to represent imagery at an abstract level, and thus providing a good starting point for more task specific training.

By far the most common training algorithms used to train neural network models are based in *gradient descent*[2]. Gradient descent algorithms are those that take repeated steps in the opposite direction of the gradient of the objective function with respect to the model's free param-

[1] More information on CNNs can be found in Chapter 9 of [25]

[2] See 9.3 of [7] and Chapter 4 of [25] for a more rigorous treatment of gradient descent

eters, and use this information to update the the values of the model parameters. In practice, gradients are computed using small subsets of the training data called *mini-batches*. In mini-batch gradient descent subsets of training data are selected at random to take gradient steps, until *convergence*, which corresponds to some notion of finality to the training procedure, such as failure to improve the objective function value over a number of successive steps. Different gradient descent algorithms are characterized how they use gradient information in iterative steps, in addition to parameters unique to the training algorithm often called *optimization hyperparameters*[3]. Because the complexity of neural network models, the exact relationships between different gradient descent algorithms and common object detection models and objectives is not well understood. As such, the type of gradient descent algorithm used, and it's hyperparameters are often chosen by empirical success. A good starting point for understanding what training algorithms to use to train an object detector is the original works that propose them, as they will likely have achieved a level of empirical success training their model for a detection task.

2.1.4 Interpreting and Post-processing Object Detector Outputs

Ultimately, the vast majority of neural network based object detectors will produce a number of outputs per image, each corresponding to a prediction of the location and class of an object. A single prediction from a detector consists of two parts: A *predicted bounding box* $\hat{b}$, that is a 4 dimensional vector representing the location of the bounding box, and a class *confidence vector* $\hat{\mathbf{c}} \in \mathbb{R}^{|\mathcal{C}|}$. The class confidence vector has an element corresponding to each of the possible object classes. Higher values of elements corresponding to a class indicate that the model is more "confident" that the bounding box contains an object of that class. In practice, elements of the class confidence vectors are typically constrained be in in the range [0,1]. They can further be constrained such that all elements sum to 1. With the additional constraint the class confidence scores can be interpreted as a joint probability of class membership of the object contained in the corresponding bounding box. Without the additional constraint the class confidence scores can be interpreted as independent probabilities of class membership. Since objects are assumed to belong to a single class[4], it is often useful to distill class confidence vectors to a single *predicted class* for the predicted bounding box. Most commonly, the class whose element in the confidence vector is the highest is assumed to be the predicted class. In cases where all class confidences are low, a detector can reject its corresponding bounding box as *background*. In practice, this means defining a *class confidence threshold* that is used during training and inference to determine what is not kept as a prediction. The remaining predicted bounding boxes and a predicted classes represent definitive statements from the detector to be interpreted as "The model predicts the presence objects contained in these bounding boxes that are members of these classes".

It is potentially beneficial to use confidence vector values less coarsely than as a means to reject bounding boxes and determining single class predictions. If an object detector is intended to be deployed in settings where it will inform human decision making, the confidence values provide strictly more information than definitive statements about class membership. For instance, if a detector outputs a single value that is much higher than the others, then a human may be able to more confidently make decisions based on the model's prediction of that class. Conversely, if it outputs many elements with similar values without a single one being substantially higher than the rest, then a human may need to consider that the object could belong to one of numerous classes. It is practically important for the designer of the detector (or the designer of the larger software system in which the detector resides) to understand how to

[3] See Chapter 8 of [25] for more information of variations of gradient descent and their parameterizations

[4] We consider the multi-classification setting in this work as it is by far the most common. If a single object can belong to many object classes, this is known as *multi-label* classification, and has its own considerations and lines of work [46, 56]. Most of the topics discussed throughout this document are still relevant to multi-label classification, but with slight variation.

interpret the outputs of an object detector with respect to how it is going to be used. This includes whether the confidence values are going to be used simply as a means for determining predictions, whether there is some post-processing that must be done to the detector outputs in order to be used by other systems, and/or how to visualize the outputs of the detector in order to satisfy an application need.

While the overwhelming majority of work in object detection assumes predicted bounding boxes and classes to take the aforementioned form, there are variations of common object detectors that are *probabilistic* in nature [19]. These approaches are characterized by their models that produce probability distributions over class and bounding box outputs. These techniques include Bayesian Neural Networks (BNNs) such as Monte Carlo Dropout [20] and Weight Uncertainty Neural Networks [4], Deep Ensembles [39], and direct modeling approaches such as Loss Attenuation [35]. The first two learn detectors that themselves are distributions one can sample from. For instance, you can perform inference using a BNN to get a sample class confidence vector and bounding box from the model. If many samples are taken, then statistical measures can be computed such as means and covariances. Direct modeling on the other hand, attempts to learn means and covariances directly. In this case, inference in direct models will not output samples from a distribution but the means and covariances over class confidence vectors and bounding boxes for a given image. In either case, the means of these models can be interpreted in the same way as the outputs of a non-probabilistic object detector. The practical benefit is in the additional covariance outputs. These covariances can be used to describe uncertainty in predictions. For instance, if a coordinate in a predicted bounding box has high variance, the model can be interpreted being uncertain in the position of that coordinate. Similarly, if a class vector output has high variance, it can be interpreted as the model being uncertain in it's predicted probability of the object being that class. Like with class confidence vectors, uncertainty produced in this way can be used to inform decision making. Humans may be more hesitant to take action based on predictions with less certainty, and may seek alternative methods to gain the information they need to make decisions. Fortunately, almost all common object detectors can be made into probabilistic models, using the techniques referenced above. Sometimes probabilistic variants of non-probabilistic models suffer drawbacks such as increased training or inference time, and minor detection performance degradation. In Chapter 4 we show an empirical evaluation of probabilistic and standard variants of a model for comparison.

Both the post-processing and the evaluation of object detectors require a way of determining how "similar" two bounding boxes are. A common metric for this is *Intersection over Union* (IoU). Let $\mathcal{A}$ be the set of pixels contained in one bounding box, and $\mathcal{B}$ be the pixels contained in a second bounding box. The intersection over union is defined as:

$$IoU\left(\mathcal{A}, \mathcal{B}\right) = \frac{|\mathcal{A} \cap \mathcal{B}|}{|\mathcal{A} \cup \mathcal{B}|} \tag{2.1}$$

Intuitively, the IoU counts the number of pixels in common between the two bounding boxes and divides it by the total number of pixels between the two boxes (not double counting those in common). IoU is 1 when the two bounding boxes are identical, and 0 when they contain no overlap. Generalized IoU (GIoU) [55] has also been proposed as a means to additionally measure how far apart bounding boxes are. For instance, if two bounding boxes have no overlap but are close together, they will have a higher GIoU than ones with no overlap and are far apart.

Often object detectors predict individual bounding boxes without reference to other bounding boxes it has predicted for the same image. For this reason object detectors can output many, seemingly redundant bounding boxes that are all in the same spatial region. This issue is most commonly handled by a post processing step called Non-Maximum Suppression (NMS). Briefly, NMS is an algorithmic procedure that selects a subset of predicted bounding boxes from a group, taking into account 1) The highest class confidence of each prediction and 2)

The IoU between a bounding box and others within a group. NMS will select only the most confident predictions that are most dissimilar to others within the group. Determining thresholds for how dissimilar bounding boxes have to be in order to be selected is a problem specific task. Problems where objects are more close together should have NMS thresholds such that more predictions are selected. As an alternative to the hard thresholding of NMS, Soft-NMS [6] has been proposed. Soft-NMS reduces the confidence of predicted labeled bounding boxes proportional to the IoU it has with others. As a result, no predicted bounding boxes are removed entirely, but their confidence in being any of the object classes can be reduced to the point that the box can be safely considered to not contain an object.

2.2 Two Illustrative Approaches to Object Detection

To further ground the previously introduced concepts, we will briefly discuss two proposed object detectors: Faster R-CNN [54] and RetinaNet [42]. These two methods represent an example of each of the two most prevailing approaches to neural network based object detection. Faster R-CNN is a *two-stage* detector, where a localization model is first used to down-select a dense set of candidate bounding boxes, and then those candidate boxes are classified and refined using a second model. Conversely, RetinaNet is a *single stage* detector, where there is a single model that accepts the dense set of candidate boxes and performs both classification and localization in tandem. Broadly, two-stage detectors achieve better empirical performance in terms of detection accuracy, but single-stage detectors perform inference faster [34]. Much of the research on both single and two-stage detectors aim to close the gap between the two. Also note that while most two-stage detectors are direct descendants of the original two-stage detector, R-CNN, there have been a wide variety of single stage detectors, such as YOLO [50, 51, 52, 5], SSD [45], and EfficientDet [62]. These other detectors were developed by different research groups and have significant differences in methodology. We chose to focus on Faster R-CNN and RetinaNet as illustrative examples that are still in widespread usage today, but note that more recent detectors build off these by making use of advances in machine learning research, such as new neural network architectures and objective functions.

2.2.1 Faster R-CNN

Faster R-CNN is the result of improvements on the Fast R-CNN [22], which in turn is a result of improvements on Regions with Convolutional Features (R-CNN) [23]. Briefly, Faster R-CNN uses a Region Proposal Network (RPN) in the first phase of the model to produce representations of regions that may contain objects from an image. These are passed to a classification network that classifies each proposed region as one of the predefined object classes. See Figure 2.3 for a diagram of the neural network architecture used in Faster R-CNN.

The key improvement of Faster R-CNN over predecessors is the RPN. In previous iterations of R-CNN, object proposals were found using an algorithm called selective search [64], which was decoupled from the classification model entirely. The RPN (Figure 2.4) is a convolutional neural network that shares parameters with the classification network, allowing both models to be trained using the same training procedure. More specifically, an image is passed through a small initial CNN. The initial CNN produces representations for overlapping regions of the image called *sliding windows*. For each sliding window representation and each anchor box, the RPN produces two outputs: 1) An *objectness score* $p_i \in [0, 1]$ predicting whether the anchor box i centered in the window contains an object or not, and 2) *bounding box offsets* t_i predicting the upper-left and bottom-right coordinates of the bounding box, relative to bounding box i. For training, Faster R-CNN uses an objective function of the following form:

$$L\left(\{p_i,\},\{t_i,\}\right) = \frac{1}{N_{cls}} \sum_i L_c ls\left(p_i, p_i^*\right) + \lambda \frac{1}{N_{cls}} \sum_i L_c ls\left(t_i, t_i^*\right) \tag{2.2}$$

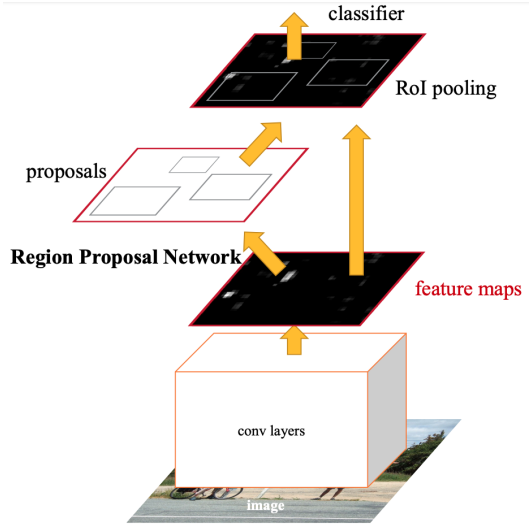

Figure 2.3: Faster R-CNN network diagram (Image Credit: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks [54])

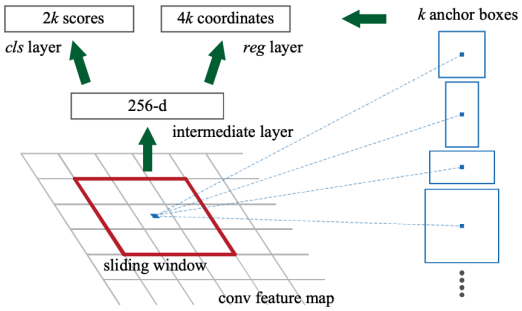

Figure 2.4: Closer look at the Region Proposal Network (Image Credit: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks [54])

Here, the first term measures loss between the predicted objectness score p_i and ground truth objectness label p_i^*. The second term measures loss between the predicted bounding box offset t_i and the ground truth bounding box t_i^*[5]. The fractional scaling factors in front of these terms normalize losses with respect to how many ground truth labeled bounding boxes there are. The λ term is a hyperparameter meant to weigh the importance between objectness and localization in the RPN[6]. When trained using this objective, the RPN is meant to be able to identify the anchor boxes within sliding windows that contain an object, and then refine the anchor box to correctly localize the object. Bounding boxes, called region proposals, with high enough objectness score are passed to the classification model.

The second stage of the model, the classification model, is simply the Fast RCNN classification network. Fast RCNN takes proposed region bounding boxes from the RPN and predicts class labels and coordinate offsets relative to the proposals. This network is trained with an objective function similar to that which is used to train RPN. The main difference is that the Fast RCNN objective contains a multi-classification loss that measures error with respect to the object classes instead of binary objectness labels. Both networks are trained using a 4-Step training algorithm where each network are first trained separately, then they are trained jointly with the parameters common between the network fixed (i.e. they are set to values from previous training steps, and not updated when trained jointly). For more details on the network architectures, objective functions, 4-step training procedure, and other important implementation details see [54, 22, 23].

2.2.2 RetinaNet

RetinaNet is a single stage object detector, its model accepts an image and anchor boxes, and outputs predicted labeled bounding boxes without explicitly identifying candidate bounding boxes before classifying them. The single model much more closely resembles common us-

[5]Note the change of notation from the data discussion in Section 2.1.1. Here t_i^* is derived by matching an anchor and a window to a ground truth bounding b. p_i^* is 1 if the correspondining window and anchor is matched with a ground truth bounding box and 0 otherwise.

[6]The authors of [54] suggest a value of 10 for λ

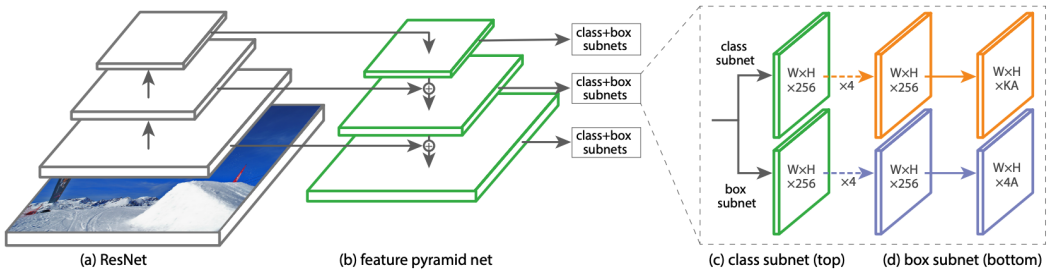

Figure 2.5: RetinaNet network diagram (Image Credit: Focal Loss for Dense Object Detection [42])

age of neural networks for visual tasks, and thus is rather straight forward. However, there are two noteworthy characteristics of RetinaNet. First, they utilize a FPN in their backbone network (See: Figure 2.5), and were one of the first models to do so. The benefit of an FPN over a standard CNN is that is allows for feature extraction and detection at different scales, meaning it more easily can detect objects with different sizes. Second, it uses *Focal Loss* (FL) in their objective for classification, instead of the popular Cross Entropy (CE) loss. In short, FL has been shown to better handle *class imbalance*, when there is considerably more training data for some classes than others. It can be shown that CE that makes training focus on "easy" instances (i.e. ones that are most similar to the majority, such as those of the same class). FL contains a focusing parameter that down-weighs the influence of instances already classified correctly in the objective, and allows for gradients to be more influenced by examples not currently classified correctly. Equipped with these improvements, RetinaNet was one of the first single stage detectors that could compete with two-stage detectors in terms of detection performance.

2.3 Final Notes

To summarize, the process to train an object detector includes:

1. Collecting training data in the form of images (instances), and labeled bounding boxes (labels).
2. Preprocessing the training data according to chosen data augmentation techniques.
3. Defining anchor boxes that represent common sizes and aspect ratios of objects for your domain.
4. Choosing an object detection method including: the general approach (such as Faster R-CNN or RetinaNet), the backbone architecture, and optionally details such as objective functions, training algorithms, and associated hyperparameters if not using those suggested by the general approach.
5. Executing the training algorithm to produce a trained model.

The process to use an object detector to predict bounding boxes on an given image:

1. Preprocessing image (Note: This is likely different than the processing done to training data).
2. Feeding the image to the trained object detector to get predicted labeled bounding boxes.
3. Performing NMS (or Soft-NMS) to remove redundant predicted bounding boxes.

4. Interpreting, post-processing, and/or visualizing the predictions in a way that is tailored to an application setting.

3 Evaluation of Object Detection Models

The obvious question after training a detector is: *How good is it*? While this seems like a simple question, it becomes complex when one considers the complexity of the object detection task and the environment in which it is meant to be deployed. For this reason, it is important to design evaluation procedures that provide test teams with quantifiable data reflecting specific performance characteristics. Equipped with the results of a well-rounded battery of tests, stakeholders can have a clearer understanding of not just the quality of the detector at an abstract level, but an understanding of when and in what ways it is succeeds and fails.

Our discussion of object detector evaluation focuses on two aspects. First, we highlight the importance of *targeted evaluation*. When borrowing object detection techniques from established literature on object detection, it is tempting to also adopt their evaluation procedures. Most object detection works focus on general problems, not the specific settings in which object detectors are practically used. As such, evaluation must focus on the *requirements* of the detector, instead of broad notions of performance typically seen in ML literature. Targeted evaluation is meant to evaluate detectors against requirements and use cases. Second, we focus on evaluation *metrics*, which are the computations used to measure quality of a detector in a specific, quantifiable way. Each metric measures different characteristics, so it is important to understand what specifically they are measuring and how that relates to important requirements. For this report, we focus on performance characteristics associated with the quality of the predictions of detectors. While important, we do not consider performance characteristics such as algorithmic complexity, training/inference run time, and storage requirements of models/data.

3.1 Machine Learning Evaluation Basics

As stated in Chapter 2, the goal of any supervised learning model is to learn from a finite train set, but generalize to the entire domains of instances and labels. To measure generalization performance, one requires data not used during training for the specific purpose for evaluating the ability for the model to generalize. Such data is called *test data*. Formally, training data is a set $\mathcal{D}_{train} = \{(x,y)_1, (x,y)_2, ..., (x,y)_{n_{train}}\}$, and test data is a set $\mathcal{D}_{test} = \{(x,y)_1, (x,y)_2, ..., (x,y)_{n_{test}}\}$, such that $\mathcal{D}_{train} \wedge \mathcal{D}_{test} = \emptyset$. Generally, evaluation is performed by training a model on a train set, and then evaluating it according to chosen metrics on the test set. The full process of training and then testing a model is called an *trial*, and is often repeated under different random assignments of train and test sets. The results of trials are most commonly aggregated using statistical measurements such as sample mean and variance. By performing multiple trials, instead of a single one, performance can be measured without the influence of potential bias present by "unlucky" splits that do not accurately reflect the true distribution being evaluated for. A set of trials where everything except the random selection of train and test set is often called an *experiment*[1].

How one selects train and test data for a trial is an important decision. In the absence of any information about the specific application of a detector, there are two main methods for choosing a test set. In the *holdout* method, each collected data pair is randomly assigned to the test or train set at a particular ratio [2]. Assuming all pairs are unbiased draws from P, evaluating a model on the test set then shows how the model generalizes to samples not seen during training that are still from the assumed distribution of instances and labels. Different

[1] This term is commonly used in the machine learning literature, but perhaps would be better called a *test* in evaluation of models to be deployed in a real-world software system.

[2] popular choices include 70/30, 80/20, and 60/40 train/test splits, but is often dependent on the availability of data

trials are created by performing the evaluation under different random assignments of train and test set at the predefined ratio.

A different approach is *k-fold cross validation*, in which the data is split in k sets of equal size. Then, one set is held out as the test set while the rest are used as the train set. This is repeated k times with a different set held out for test data each time. The result of each of the k evaluations are then aggregated. The difference between the two approaches is that the holdout method may result in a pair being in the test set of multiple trials. In k-fold cross validation, a pair will only be in a test set once. For comparison of these and other validation procedures, see [1].

3.2 Targeted Evaluation

Randomly splitting all collected data for evaluation into test and train sets is often a good first step in understanding the performance of a detector. However, in many cases, knowledge of the domain in which the detector is used motivates certain use cases that should receive special attention in evaluation due to their unique impact on the application. Instances with common characteristics of interest can be isolated in a test set to see how a model performs in those special cases. For instance, a potential challenging use case for an aerial object detector is use in inclement weather. If a model is evaluated on test data randomly chosen from all of P, which could be skewed towards mild weather conditions, the results could similarly be skewed. An experiment where test data exclusively consists of instances of inclement weather is useful to understand performance on that specific case. Such a specific experiment is an example of *targeted evaluation*. Targeted evaluations are defined by their unique data splits as well as their metrics. Data can be chosen based on the label (e.g only on class "car"), metadata over instances (e.g. the inclement weather example from before), human defined characteristics (e.g. a set of instances the detector has done poorly on in the past), or other information that allows you uniquely identify instances for collective evaluation. Similarly, because different metrics measure different performance characteristics of a detector, each metric represents a different target. We will discuss this point more in Section 3.3.

Unfortunately, it is difficult in practice to define targeted experiments before ever fielding a detector. The process of defining the needs or conditions in which a system is required to meet is called *requirements analysis*, and is a developing field for machine learning [66, 49, 2]. Communication between stakeholders and domain experts is key to establish a first set of experiments that covers known important cases. Even if a seemingly complete set of evaluation experiments give confidence to development teams that a detector will perform well in the cases they deem important, it is possible the model will perform poorly in unexpected cases. For this reason, it is important to ensure that a detector is deployed in such a way that unexpected failures can have minimal impact, and data can be collected not only to establish new targeted experiments but also to retrain models to improve performance in failure cases. Further, it is difficult to understand how to balance the results of different targeted evaluations. Do I want a detector that can detect objects better in inclement weather or one that can detect red cars better? One approach is to establish acceptance conditions for the results of all but one experiment [47]. This way, targeted experiments can act as gates for acceptance rather than targets for optimization (e.g. "Our model achieves a score greater than x on metric y in inclement weather, which is greater than the acceptance threshold z."). A single, general experiment can be used to choose between different models that satisfy acceptance conditions (e.g. "Between models that satisfy all acceptance criteria, choose the one that performs the best in the most general sense.").

3.3 Metrics

The choice of metric for an evaluation experiment is key to understand specific performance characteristics of a detector. Because object detection is equal parts localization and classification, metrics for object detection are designed to measure performance and either task or both. The complex relationship among the localization and classification tasks naturally results in complex metrics that require precise understanding in order to build experiments that measure desired characteristics. In this work, we focus on two types on object detection metrics. First, we focus on *detection metrics* that measure some form of correctness of the model in "predicting" a bounding box and class for an object. At their core, detection metrics assume detector outputs are definitive predictions about where and what objects are in an image, and they compare those predictions to ground truth. Alternatively, *uncertainty metrics* interpret model outputs as probabilistic estimates of where and what objects are, and compare those to the probabilistic interpretations of ground truth. Detection metrics are by far more popular because they more directly measure detectors for their desired purpose of predicting the location and class of objects in an image. However, uncertainty metrics measure how well models are able to quantify the uncertainty in those predictions, which enables the use of predictive uncertainty in assessing how much to rely on model predictions. If uncertainty is a consideration for the usage of a detector for a given application, then both kinds of metrics are vital to understanding how well suited a detector is for a task.

3.3.1 Detection Metrics

Detection metrics assume that confidence vectors are used as a means to determine a single predicted class for an object. Let $\hat{c}$ be the class with the highest confidence in a class confidence vector $\hat{\mathbf{c}}$ known as the *predicted class*. The most common detection metrics are build on a common principle: A prediction is "correct" if it both successfully localizes and classifies an object given by ground truth. More specifically, when comparing a predicted labeled bounding box $(\hat{b}, \hat{c})$ and a ground truth labeled bounding box (b, c), the prediction is considered "correct" if 1) $IoU(b, \hat{b}) > \lambda_{IoU}$ and 2) $c = \hat{c}$. If these conditions hold, it is called a *true positive (TP)*. Metrics for detections are based on counting the number of true positive and comparing them to the number of "incorrect" predictions. For that, we require two notions of incorrectness. A *false positive (FP)* is when a prediction does not satisfy both conditions for any ground truth. All ground truth labeled bounding boxes left after removing all true and false positives are called *false negatives (FNs)*. A false positive represents a prediction that does not detect any ground truth. A false negative represents a ground truth for which no prediction detected. The procedure for counting these for an image is as follows:

1. Compute the IoU between all predictions and ground truth for an image.
2. For all pairs of predictions $(\hat{b}, \hat{c})$ and ground truth (b, c) in descending order of their IoU
   (a) If $IoU(b, \hat{b}) > \lambda_{IoU}$ and $c = \hat{c}$
      i. Count it as a TP
      ii. Remove $(\hat{b}, \hat{c})$ and (b, c) from the list of predictions and ground truth.
3. Count all predictions $(\hat{b}, \hat{c})$ not removed as a FP
4. Count all ground truth (b, c) not removed as a FN

Raw values of TPs, FPs, and FNs give a sense of the number of correct and incorrect predictions, as well as how many objects were not detected, but aggregating these into single statistical measures gives relative notions among them. For instance *precision* is defined as:

$$precision = \frac{TP}{TP + FP} \tag{3.1}$$

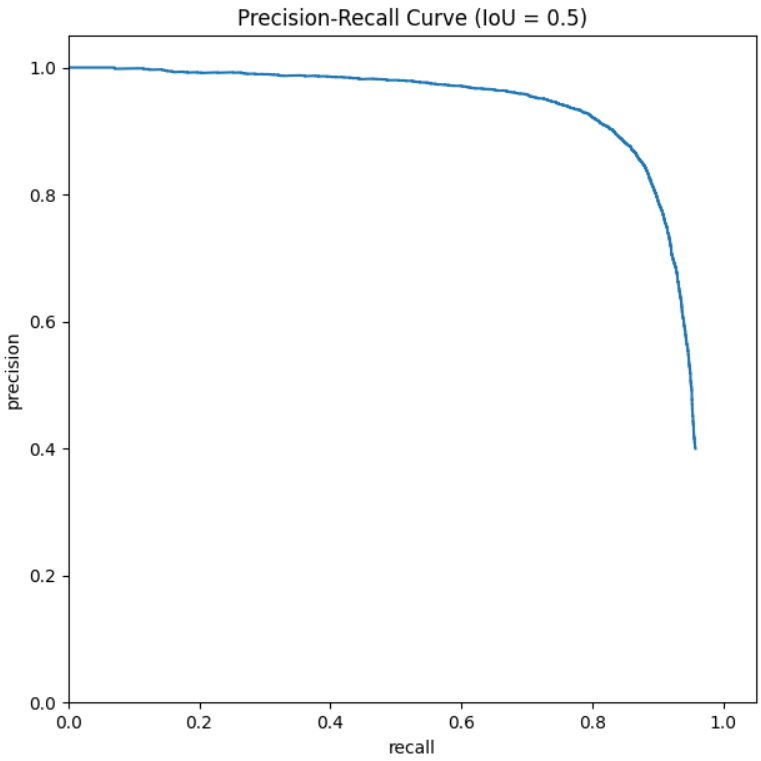

Figure 3.1: Example precision-recall curve of a RetinaNet model trained and evaluated on the RarePlanes data set (IoU threshold at 0.5).

Precision gives a proportion of how often predictions were correct. *Recall* is defined as:

$$recall = \frac{TP}{TP + FN} \tag{3.2}$$

Recall gives a proportion of how many of the ground truth labeled bounding boxes were detected.

Precision and recall, while both useful in determining the detection performance of a detector, are often at odds. Consider a detector that errs on the side of caution when it makes predictions. It only makes predictions when it is very confident that there is an object present and it is certain of the class. Such a detector would likely have high precision (when it predicts, it is most often correct), but low recall (by being cautious, it may miss many of the ground truth labeled bounding boxes). Conversely, consider a detector that attempts to make as many predictions as possible in order to ensure it detects every object. This detector would likely have low precision (many of its predictions are incorrect), but high recall (it detects many of the ground truth labeled bounding boxes simply through prediction volume). Further, a single learned detector can have vastly different precision and recall depending on a chosen class confidence threshold used to determine what predictions are rejected as background.

To understand the relationship precision, recall, and the choice of class confidence threshold for a detector, a common visualization plot called a *precision-recall curve (PR curve)* is often employed. A PR curve is created by sorting the class confidences for all detector outputs and sweeping the confidence threshold from 1 to 0, incrementally increasing the number of predictions made by the detector, and computing the precision and recall each time a new output is included as a prediction. If the more confident predictions are more likely to be correct, the

the model should have high precision and low recall at first. As the threshold is lowered, recall should increase and precision should decrease. The result is a curve like the one shown in Figure 3.1. PR curves are especially useful for three things. First, a PR curve shows how sensitive a detector is to the setting of the class confidence threshold. If precision drops significantly at a point in the middle of the PR curve, then the detector will behave very differently depending on the setting of the confidence threshold. Second, one can use PR curves to find an acceptable tradeoff between precision and recall, commonly at what is known as the "knee" of the curve where precision begins to drop off significantly. For instance, in Figure 3.1, the model is able to achieve around 0.85 recall at around 0.9 precision. In order to achieve 0.95 recall, the model would drop to below 0.5 precision. Finally, requirements analysis can reveal a required amount of precision or recall (e.g. "Our application needs a detector that will fail to detect no more than 20% of objects (0.8 recall)"). A PR curve can reveal the point just at the required precision or recall that maximizes the other. Looking at Figure 3.1, if there is a requirement of 0.8 recall, then the threshold can be set right at the point where recall is 0.8 on the curve, achieving 0.9 precision.

PR curves are a means for vizualizating object detector performance, but are not singular numeric values that can be used to compare detectors. If choosing between two detectors for an application, it can be difficult to look at both of their PR curves and definitely say one is better than the other. For this metrics such as *average precision (AP)*, and *average recall (AR)* are used. Average precision and recall are simply the average of the PR curve with respect to each axis. Taking this thought further, one can aggregate multiple PR curves together to encapsulate other characteristics. For instance, PR curves assume an IoU threshold. In some cases, requirements analysis may reveal an IoU threshold that is acceptable enough for an application (i.e "Our localization needs to be at least *this good* for us to consider it a detection."). In other cases, it may not be obvious. When an IoU threshold cannot be assumed, then a number of PR curves can be drawn, each with a different IoU threshold. All of these PR curves can have their AP or AR computed, and these desperate values can themselves be averaged. Such metrics are known as *mean average precision (mAP)* and *mean average recall (mAR)*. While mAP is attractive as a metric as it encapsulates many different ways of evaluating a detector and makes few assumptions regarding threshold [3], one should take caution in using it as the sole metric for evaluation. By abstracting all notions of threshold of acceptance for IoU, precision, and recall into a single number mAP can hide specific failure modes that do not become apparent until deployment. For example, a detector may be able to achieve higher mAP than another model by making twice the predictions. If such a detector was used in a system where predictions would alert a human user, the user could be overwhelmed by alerts and ignore the system, effectively making it worthless. In short, in the absence of any requirements metrics that abstract assumptions are a good choice. However, once requirements are set, they should be used to determine how to evaluate a detector.

3.3.2 Uncertainty Metrics

Treating outputs of detectors as probability distributions not only allows for interpretations of their outputs as uncertainty in predictions, but also results in a series of metrics meant to measure how accurate predicted probabilities are. We limit our discussion to two kinds of uncertainty metrics. First, *calibration metrics* measure whether probabilistic outputs faithfully capture the relative frequency of events. In object detection, model calibration is measured for the classification and localization parts of the model separately using different metrics. Calibration metrics for classification measure the quality of the probabilities in class confidence vectors. This begs the question: *What is the right way to understand these probabilities?* To first build intuition, consider the case when a detector outputs a prediction with probability (confidence) 0.7. What is 0.7 supposed to mean, exactly? For a fully calibrated detector, this

[3] Indeed, mAP is one of the most prominent metrics used for object detection challenge problems such as Microsoft Common Objects in Context Challenge [43] (though they use others as well)

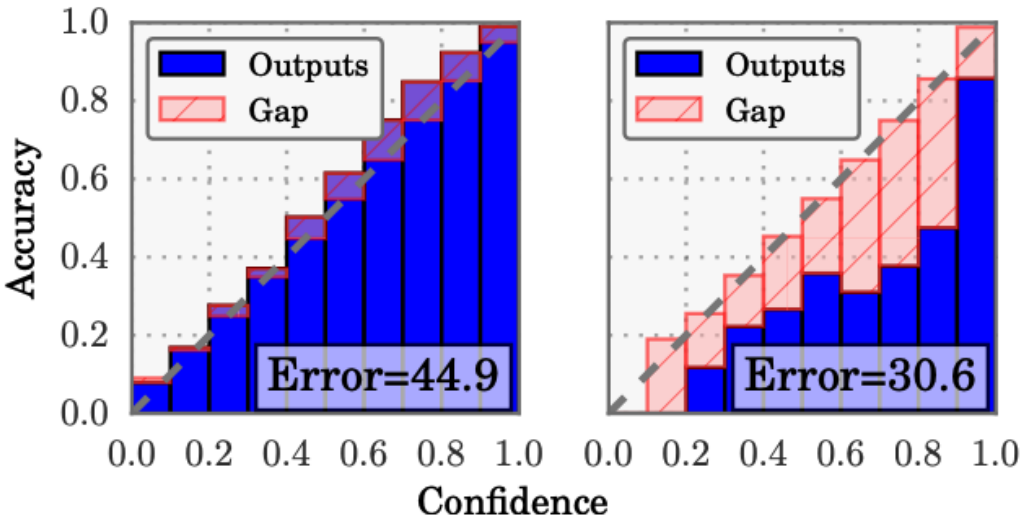

Figure 3.2: Example reliability diagram visualizing classifier calibration. (Image Credit: On Calibration of Modern Neural Networks [27])

would mean that for all instances in which the detector outputs 0.7, the the predicted class matched the ground truth class 70% of the time. Calibration metrics effectively measure the difference between the probabilities output by a classifier and how often those probabilities reflect the ground truth class labels. Note that this is somewhat of an oversimplification, and to give calibration a full treatment would require a considerable amount of space that would detract from the focus of this report. Instead, we will discuss the most basic and well used calibration metric, *Top-1 Expected Calibration Error (ECE)*, at an intuitive level, and leave the following references for those interested in understanding the topic of calibration metrics further [27, 65, 36].

Consider the visualizations in Figure 3.2, known as reliability diagrams. These are created by first taking every instance in a test set and partitioning them into separate bins according to the maximum probability (confidence) that is output by the classifier. Bins are defined by separate intervals over [0,1] as depicted in the x axis. A bar's height is the average of all the probabilities within a bin that is aligned with its interval on the x axis (e.g. the blue bar on the far left of each diagram is for the bin over [0,0.1]). In principle, a for a well-calibrated classifier, a bin's average probability will be close to the proportion of instances in that bin in which the predicted class is the same as the ground truth class label. Borrowing from the above example, a bin with average probability of 0.7 should contain instances in which 70% are classified correctly. The average absolute difference between the average probabilities of each bin and the proportion of correctly classified instances in each bin is the Top-1 ECE. Visually, this error is roughly captured in the reliability diagrams via the red highlighted areas. The reliability diagram on the left represents a fairly well-calibrated classifier, as the bars are close to to the line $x = y$, where the y axis is the proportion of correctly classifier instances (accuracy). The diagram on the right represents a more miscalibrated classifier.

To measure calibration error for the localization component of a detector, localization is treated as a *regression* problem. In short, predicted bounding box coordinates output by a model are treated as individual real numbers to be compared to the real numbers specified by the ground truth bounding box coordinates. Because standard object detectors do not output distributions over bounding box coordinates, they have no obvious probabilistic interpretation and thus cannot be measured for their calibration. Probabilistic object detectors as discussed in Section 2.1.4 on the other hand output distributions over bounding box coordinates and

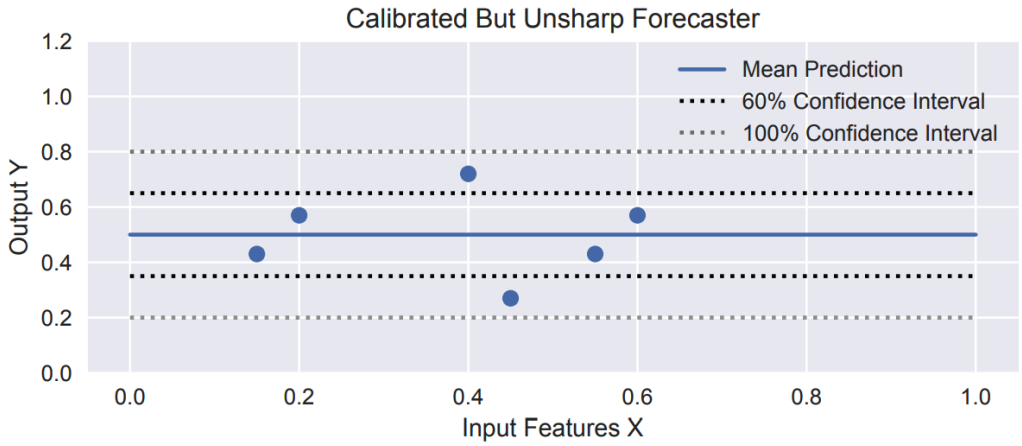

Figure 3.3: Example of a calibrated regressor. (Image Credit: Accurate Uncertainties for Deep Learning Using Calibrated Regression [38])

can be measured for calibration. The only additional requirement for the detector is that its output must have interpretations that produce means and *confidence intervals* for each coordinate. Detectors that output means and covariance matrices, for instance, over bounding boxes satisfy this requirement. You can interpret the confidence intervals of the detector in a similar way as is done with classifier calibration: A calibrated detector would have 70% of the bounding box coordinates within it's 70% confidence interval. As an example, in Figure 3.3 four of the six test points are within the 60% confidence interval around the mean, and all six are within the 100% interval. Thus, this regression model (regressor) is calibrated. Regression ECE is computed similarly to classification ECE. Instead of bins, a number of confidence intervals are defined, and the proportion of instances within the confidence intervals are counted. The average of the squared differences between the intervals and the proportion of instances in each interval is taken as the ECE. For more information on how specifically this metric is computed, see [38].

Calibration metrics provide intuitive ways of understanding the predictive uncertainty of both the localization and classification components of a detector. Analogous to mAP for detection metrics, predictive uncertainty for both localization and classification can be quantified in a single metric called Probabilistic Detection Quality (PDQ) [28]. PDQ is the geometric mean of separate functions that each measure a different aspect of the uncertainty estimates for classification and localization. The classification quality measure is simply the value of the element of the class confidence vector output by a detector that corresponds to the correct class of an object. In this way, PDQ is higher if the detector puts more probability onto the correct class. The localization measure itself is the sum of two components. Both components rely on the fact that probabilistic object detectors output a probability associated with each pixel. Pixels with high probability from a detector indicate more confidence that the pixel is within the ground truth bounding box. The first component of the localization measure scores each pixel in the ground truth bounding box proportionally to the probability assigned to the pixel by the detector. The result is that detectors score higher if they assign higher probability to pixels within the ground truth bounding box. The second component scores each pixel outside of the bounding box inversely proportional to the probability assigned to them. In this way, a detector will score higher if they assign low probability to pixels outside of the bounding box. For a more rigorous treatment of PDQ, see [28].

4 A Case Study: Aerial Object Detection on the RarePlanes

To put a finer point on the topics introduced in the previous chapters, in this chapter we show the results of an evaluation of a number of object detectors trained and evaluated on satellite imagery. For this we use the RarePlanes [61] data set. RarePlanes consists of 253 images that encapsulates 2,142 km^2 spanning 22 countries and all four seasons. The images are tiled into 512x512 squares with 20% overlap to form the instances of the object detection problem. The data set also includes 14,803 labeled bounding boxes on training instances and 3,590 for testing. The seven object classes and the number of labeled bounding boxes per class are shown in table 4.1.

Class Name	Class ID	Train	Test
Small Civil Transport/Utility	0	8,357	1,971
Medium Civil Transport/Utility	1	4,894	1,229
Large Civil Transport/Utility	2	1,174	276
Military Transport/Utility/AWAC	3	205	79
Military Bomber	4	5	1
Military Fighter/Interceptor/Attack	5	153	32
Military Trainer	6	15	2

Table 4.1: Class label breakdown for test and train sets in the RarePlanes data set

A couple important practical notes about the RarePlanes data set. First, there is a heavy class imbalance. There are thousands of examples of Small Civil Transport planes (class 0), but only 6 examples of a Military Bomber (class 5). Traditionally, unless special care is taken, machine learned models will be biased to the classes with more samples, often performing poorly on undersampled classes. The focal loss employed by RetinaNet has been shown to be well-suited for imbalanced classes in object detection, but note that there are a number of other techniques to handle this problem [48]. Also, evaluating on classes 4 and 6 is difficult due to the limited number of examples in the test set. Results reported on 1 or 2 held out examples should be taken with a grain of salt in terms of measuring the ability for a detector to generalize.

We train two models on the RarePlanes train set: a RetinaNet and a Faster R-CNN model; Both were implementations from Detectron2 [68]. For uncertainty evaluation we chose Loss Attenuation [35] and BayesOD [29] as the methods of turning the standard RetinaNet into probabilistic object detectors, and used a implementations from [19]. For real applications of detectors, careful selection of hyperparameters is a key part of model development, and can require a substantial amount of effort [1]. Instead of focusing on selecting hyperpameters to produce the best performing models, we instead aimed to train models that were sufficiently performant to provide a case study in evaluating models. As such, the exact numbers from each model are not meant to reflect the current absolute best performance from the state-of-the-art. Rather, we use the models as a means to discuss their evaluation. Specifically, we aim to provide some examples of how one can define experiments that lead to insightful evaluations. Note though, that without a specific application context for these detectors, we cannot tie experiments to requirements. Nevertheless, we aim to provide multiple different experiments that show how test data and metrics can be chosen to show different performance characteristics of the detectors.

[1] Common practices for finding hyperpameters for ML training procedures include random search, grid search, and successive halving. More sophisticated hyperpameter selection techniques remains an active research topic [71].

Metric	RetinaNet	Faster R-CNN
mAP	0.429	0.421
mAR	0.554	0.504

Table 4.2: Mean average precision/recall of RetinaNet and Faster R-CNN models on RarePlanes test set

4.1 Evaluation of Detection Performance

Without any information about a desired application of this detector, the first step in evaluating our detectors is to treat all data equally, and to perform an evaluation on the most general metrics. Table 4.2 shows the mean average precision and recalls for both detectors[2]. Here, we see that RetinaNet outperforms Faster R-CNN in terms of both metrics, but by a larger margin in mAR. Specifically, this tells that over a wide range of IoU and class confidence thresholds, RetinaNet achieves higher precision and recall. One could conclude that the RetinaNet is simply a better detector, but more specific evaluation can highlight more fine-grained differences in performance.

Metric	RetinaNet	Faster R-CNN
AP@0.50	0.618	0.620
AP@0.75	0.500	0.480

Table 4.3: Average precision at IoU tresholds 0.5 and 0.75 of RetinaNet and Faster R-CNN models on RarePlanes test set

Table 4.3 shows the average precision of the two detectors at IoU thresholds 0.50 and 0.75. We see here that Faster R-CNN has a higher average precision at the lower threshold. If requirements dictate that a 0.5 IoU threshold is sufficient, Faster R-CNN might be the better model, even though it achieves lower mAP and mAR.

Metric	GT BB Size	RetinaNet	Faster R-CNN
mAR	small	0.0.414	0.0.328
mAR	medium	0.557	0.508
mAR	large	0.793	0.801

Table 4.4: Mean average recall of RetinaNet and Faster R-CNN models on RarePlanes test set on ground truth bounding boxes of sizes small, medium, and large

Tables 4.4 and 4.5 show the mAR of the detectors on two different ways of splitting the test set. Table 4.4 shows the performance when the ground truth bounding boxes are separated by size[3]. As we see, both detectors' performance decrease as the size of bounding boxes gets smaller. This intuitively means that detectors are less accurate on smaller planes than bigger ones. Table 4.5 shows the performance when test set instances are separated by the number of objects predicted by the model. The trend here is that when the detectors predict fewer bounding boxes, they achieve lower mAR. Again, this is intuitive as fewer predictions means fewer opportunities to identify ground truth. The main takeaway from these two tables is that by partitioning test sets into data that have common characteristics, experiments can focus on how detectors perform in specific scenarios. These two experiments highlight how the detectors perform with varying ground truth bounding box size, and sparsity in predictions, each of which identifying when the detectors are more likely to succeed.

[2]The IoUs used for both metrics were from 0.5 to 0.95 at 0.5 increments.

[3]"small" bounding boxes are 32 square pixels or less, "medium" are between 32 and 96, and "large" are more than 96.

Metric	Detection Threshold	RetinaNet	Faster R-CNN
mAR	1	0.264	0.254
mAR	10	0.536	0.489
mAR	100	0.554	0.504

Table 4.5: Mean average recall of RetinaNet and Faster R-CNN models on RarePlanes test set on instances with no more than 1, 10, and 100 predicted bounding boxes

Class ID	RetinaNet	Faster R-CNN
0	0.628	0.622
1	0.698	0.691
2	0.678	0.678
3	0.539	0.496
4	0.018	0.090
5	0.322	0.273
6	0.123	0.098

Table 4.6: Average precision (IoU threshold = 0.75) of RetinaNet and Faster R-CNN models on RarePlanes test set on ground truth objects separated by class label

Table 4.6 shows the AP@0.75 of the two detectors in instances separated by their class. First, because classes 4 and 6 have one and two instances in the test set, respectively, these results should be *ignored* as so few examples of these classes does not make for a good indicator of generalization performance. For the classes with the most ground truth labels (0,1,2) the detectors achieve comparable performance. RetinaNet distances itself from Faster R-CNN in objects of classes 3 and 5, which have much fewer examples than 0, 1, or 2. This shows the ability for RetinaNet to learn with imbalanced data better than Faster R-CNN, and thus achieves better overall AP@0.75. Class-specific evaluations are important, as certain classes may have particular significance for an application. For instance, requirements may dictate that military planes are especially important. This evaluation shows that RetinaNet would be a better choice in such a case.

Figure 4.1 shows PR curves for each detector at different IoU thresholds. Expectedly, as the IoU threshold for detection increases, both models have a lower initial precision, and a lower ending recall. Also note the more gradual decrease in precision indicating that high class confidence becomes less of an indicator of a successful detector as IoU increases. This is because a detector's success becomes more influenced by it's localization ability than it's classification ability the higher the IoU thresholds. These also help explain some of the AP numbers in Table 4.3. At a IoU threshold of 0.5, the RetinaNet and Faster R-CNN curves appear very similar with Faster R-CNN seemingly able to maintain higher precision towards the end of the curve. At 0.75, Faster R-CNN's curve has a slightly more drastic decrease towards the beginning, which could explain why it has lower mAP at a IoU threshold of 0.75. Finally, both models perform poorly at IoU threshold of 0.9. This indicates that if a very precise localization is required, further model development work is required to learn a performant model.

Lastly, Figure 4.2 shows examples of an image from RarePlanes with ground truth, with predictions from RetinaNet, and with predictions from Faster R-CNN. The most obvious difference between the two detectors is that RetinaNet produces many more high confidence predictions. This could be due to RetinaNet's use of focal loss which encourages training to focus on undersampled classes. The result is many more predictions of classes with few training instances (classes 4 and 5 in this case). Note though that among the many predictions, almost all objects are correctly detected. This could explain the relatively close mAP between the detectors, but much higher mAR for RetinaNet. If such a behavior is undesirable, a more suit-

able confidence threshold can be found or hyperpameters for focal loss can be adjusted before training. While it makes fewer predictions, Faster R-CNN makes potentially more impactful errors. It identifies road markings as a small civilian transport plane. It misidentifies the bomber as a fighter. The latter could be due to the relatively few bomber labels in the training set.

Class ID	Loss Attenuation	BayesOD
0	0.0280	0.0278
1	0.0087	0.0088
2	0.0072	0.0072
3	0.0118	0.0119
4	0.0137	0.0137
5	0.0132	0.0132
6	0.0130	0.0130

Table 4.7: Top-1 expected calibration error for both Loss Attuated and BayesOD RetinaNet on test set separated by groun truth class.

4.2 Evaluation of Uncertainty Quantification Performance

In this section, we present an evaluation of a RetinaNet with Loss Attenuation (LA) and with changes specified by BayesOD. Before focusing uncertainty quantification performance, we note that we were able to train both detectors with comparable detection performance to the standard RetinaNet from the previous section (0.465/0.467 mAP and 0.570/0.572 mAR for LA/BayesOD). At least in this case, we are able to train a detectors that outputs distributions over it's outputs without sacrificing detection quality. Table 4.7 shows the Top-1 ECEs for both detectors on test data from each class. Both models are very well calibrated. All classes (again, ignoring 4 and 6) have ECE below 0.03, meaning the mean confidence in each bin is no more than 0.06 from the proportion of correct labels in the same bin[4]. Also of note is that both techniques perform roughly the same, though both incurred significantly more error in class 0, the class with most examples. Further investigation at the cause of this is required to fully understand it, but the detectors could be overconfident in predictions of class 0 because they have been trained on more examples of those than other classes. For localization ECE both methods had similarly low values 0.0263/0.0053 (LA/BayesOD), but in this case BayesOD performed significantly better. Finally, both models achieved similar PDQ scores (0.209741/0.211392). In practice, this would be a first pass on investigating these models with respect to their abilities to quantify uncertainty. Many unanswered questions remain. Why are the models most miscalibrated on class 0? Are there other induced classification problems for which classification ECE are of interest[5]? What is different about the probability distributions of LA and BayesOD that results in such a significantly lower localization ECE for BayesOD (more specifically, what is different about their covariances?)? If we look at the individual terms in PDQ, are there bigger differences between the models with respect to different kinds of uncertainty (classification versus localization foreground/background)?

Finally, Figure 4.3 shows another example from the RarePlanes, as well as the outputs from the Loss Attenuated RetinaNet. In the middle figure, one of the predictions from the middle of the image is highlighted. The heatmap is warmer (red) for pixels that the detector is more confident are in the bounding box, and cooler for ones it has little or no confidence (blue). On the right is a bar graph showing the confidence the detector has that the object is of each

[4] In this case, bin error is computed via total variation distance which is half the absolute difference between bin means and proportions.

[5] Inducing classification problems for practical evaluation is discussed in [36]. One of particular interest here is determining the miscalibration between military versus civilian planes.

class. We see that the detector is very confident that the pixels in the middle of the ground truth bounding box are part of the plane, and very quickly become unconfident in the pixels towards the edge. The bar graph shows that the model is also very confident that the object is of class 0, which it is. The bottom figure highlights a plane that is on the edge of the frame with most of the plane actually being out of frame. In this case the detector is less confident overall about the pixels that make up the plane, even at the ground truth bounding boxes center. Indeed, it is common for object detectors to struggle with objects out of frame as the ground truth bounding box has to effectively cut the plane in half in order to stay within bounds of the image. The class confidence, too, is lower for class 0 than in the middle example. This shows an important effect of quantifying uncertainty in detectors: Confidence can be used to better inform those consuming detector outputs. Here, someone can more confidently act on the localization of the middle prediction than on the bottom one.

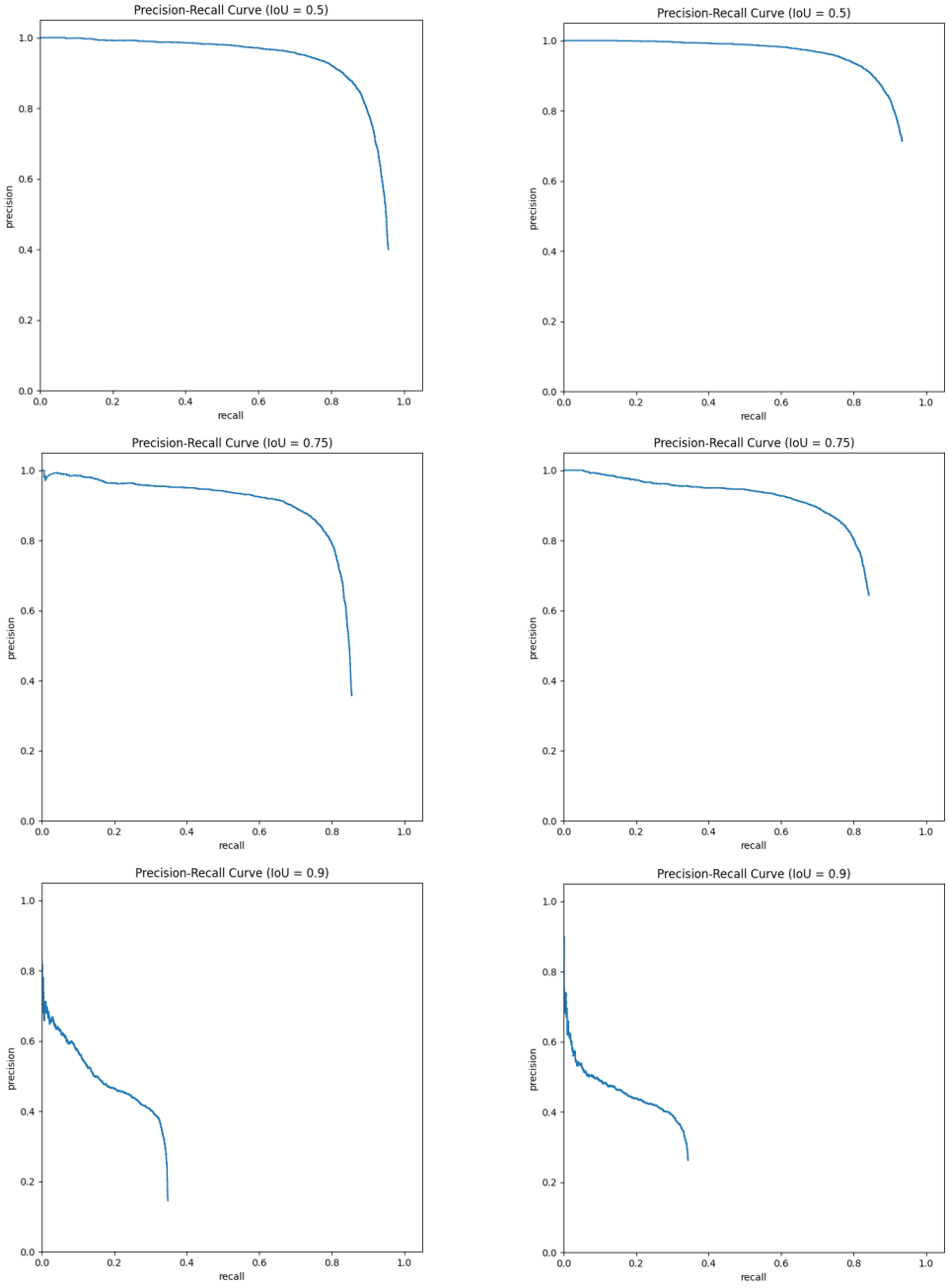

Figure 4.1: Precision-recall curves for RetinaNet and Faster R-CNN models on RarePlanes test set for different IoU thresholds (left column = RetinaNet, right column = Faster R-CNN).

Figure 4.2: Examples from RarePlanes data set. Top image = ground truth labeled bounding boxes. Middle image = outputs from RetinaNet. Bottom image = outputs from Faster R-CNN.

Figure 4.3: Examples from RarePlanes data set. Top image = ground truth labeled bounding boxes. Middle image = uncertainty visualization of Loss Attenuated RetinaNet for middle plane. Bottom image = uncertainty visualization of Loss Attenuated RetinaNet for top plane.

5 Final Words

This document outlined the basics of object detection from aerial imagery with a focus on evaluation. It covers the numerous decisions that go into learning a detector including the role of data, the choices involved in designing and training a detector, and the thresholds used to post-process outputs. It also focuses on targeted evaluation, where experiments are designed to measure the performance of detectors in specific ways in order to better inform stakeholders of how detectors behave in different settings. In short, there are a few main takeaways:

1. Designing an object detector involves a number of decisions that all have significant influence on how a detector will perform. The effects of some of these decisions will not be apparent unless the detector is evaluated in specific ways that highlight them.
2. It is important to understand the requirements of the detector with respect to the context in which it will be deployed. What cases are important? What performance measures are important?
3. While having a singular experiment to optimize the detector on allows for direct comparisons between models, requirements should lead to quantitative thresholds of acceptance of other performance characteristics.
4. Targeted experiments should be a key part of evaluating a detector as means to validate that acceptance thresholds are met.

Beyond these points, this document provides a number of references to understand many of the discussed topics at a deeper technical level. Those looking to gain a working understanding of training and evaluating object detectors should follow these references.

5.1 Open Topics in Object Detection

While object detection as a field is mature enough for practical usage in a number of settings, there exists a number of open research topics that stem from shortcomings in the current state-of-the-art. Most of these are not specific to object detection, but more broadly apply to modern supervised learning approaches. Supervised learning by its very nature is heavily dependent on the availability of high-quality, labeled data for training and evaluation. Without it, models cannot accurately learn the relationships among instances and labels, and models cannot be evaluated to determine their performance. As such, there is a number of lines of work aimed at reducing the reliance on labeled training data. When domain-experts cannot be relied on to provide quality labels, but there exist cheaper, less reliable sources, *weakly-supervised* [12] approaches can be used. When there is an lack of labeled data, but a significant amount of unlabeled data, both may be used in a *semi-supervised* [63] manner. Similarly, unlabeled data by itself can be used in an *unsupervised* [70] manner to learn base models that capture many salient visual features before even training a model on labeled data. In cases where the developers of the model are also in control of the labeling process, and there is a large number of unlabeled instances available, *active learning* [58] can be done. Active learning iteratively trains a model, then inspects the model with respect to the pool of unlabeled instances. Each unlabeled instance is scored according to it's "informativeness" to the model. Instances that are determined to be more informative if they were to be labeled and trained on are chosen and the process begins again. The intuition is that many instances simply do not need to be labeled in order for a model to learned, and if only the informative ones are labeled, many fewer labels are needed.

More broadly, there exists a subfield of machine learning dedicated to transferring knowledge

from a source task to a different target task called *transfer learning*. The pretrained backbone networks typically used in modern object detection are an example of transfer learning, but more direct methods exist [8]. *Domain adaptation* [33] is a subfield of transfer learning where it is assumed that the learning problems are the same (e.g. both the source and target tasks are object detection), but the domains[1] are different. Another subfield of transfer learning is *meta-learning* where the process of learning itself on source tasks is used to more efficiently learn in target tasks [67]. Lastly, there is a growing interest in *general visual models* [16, 73], that can be trained entirely in a task agnostic way but can provide low-dimensional representations of images that can be used for specific tasks without any explicit transfer.

Data-efficiency is a practical problem developers of ML models must overcome. However, there are a number of problems readily apparent to a number of stakeholders besides those building models. One of these is *robustness*, which describes the ability for a model to perform as expected in a variety of deployment contexts. Different subfields of ML robustness research can largely be categorized by their assumptions on P during train and deployment time. The standard assumption is that the distribution in which instances and labels are generated, P, is the same during train time, test time, and when the model is deployed in the real-world. Because P, as well as $\mathcal{X}$ and $\mathcal{Y}$, can be very complex, understanding the robustness of a model even in this simple case is difficult. This is largely the motivation for extensive evaluation, but understanding what to evaluate for is a practical challenge. As such, there is considerable ongoing research in predicting failure modes in ML models [32, 72].

In many real-world applications of ML, the distribution P can shift during deployment. Active research areas studying this problem are defined by how P shifts. If the distribution of instances changes, such as through sensor noise or deploying the model in an environment different than training[2], then it is known as *covariate shift* [59]. If the distribution over labels changes[3], then it is known as *label shift* [44]. Related to both of these is *out-of-distribution (OoD) detection* [53] that aims simply to identify when instances come from a different distribution than the one that generated training data. Finally, *adversarial learning* [26] focuses on the setting where P can be changed during deployment by an adversary for the purpose of degrading model performance.

There are many obvious reasons why robustness is an important property for any ML model, but one of them is trust. Being able to definitively say that a model will not fail in unexpected ways leads stakeholders to have more confidence that the model will act in a reliable manner. However, because ML models and their evaluation is somewhat of a technically deep task, it is not obvious how to convey information about the model that leads to trust for non-technical or semi-technical stakeholders. For this, there has been a recent focus on ML model *interpretability* [74]. Interpretability research focuses on developing techniques that highlight key characteristics of models and the predictions in ways that humans are able to understand. This brings up important questions such as: What form should information take so that humans can understand it? What information is relevant to humans to understand how a model makes predictions? How do we ensure that information provided to humans is an accurate reflection of how complex models make predictions? Most works focus on the two former questions, but there has been recent interest on the latter [75].

Lastly, many object detection pipelines require lots of heavily manual pre and post processing. Both NMS and anchor boxes for instance require domain knowledge to tune properly. Further, many advancements have been due to specialized architectures (such as the RPN of Faster RCNN), or loss functions (such as Focal Loss in RetinaNet). To alleviate some of the

[1] Recall domains are defined by $\mathcal{X}$ and $\mathcal{Y}$, but here it can also mean P can be different between source and target.

[2] Consider training an object detector on aerial images taken in a desert, but deploying it to detect objects in the rain forest

[3] Consider the case that certain vehicles become more popular over time. An aerial object detector may have seen very few of those types of vehicles during training, but will see many more during deployment.

burden on developers, there has been some recent advancements in more general object detection techniques [9, 11]. These techniques pose object detection as direct prediction problems where region proposals are not needed, thus, no explicit prior knowledge is needed to train them. They also tend not to produce many, redundant predictions and do not require NMS. The results reported in both papers show that they can achieve prediction performance on par with the state-of-the-art that requires both of these.

References/Bibliography

URLs are valid as of the publication date of this document.

[1] Sylvain Arlot and Alain Celisse. A survey of cross-validation procedures for model selection. *Statistics surveys*, 4:40–79, 2010.

[2] Alec Banks and Rob Ashmore. Requirements assurance in machine learning. In *SafeAI AAAI*, 2019.

[3] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

[4] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *International conference on machine learning*, pages 1613–1622. PMLR, 2015.

[5] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020.

[6] Navaneeth Bodla, Bharat Singh, Rama Chellappa, and Larry S Davis. Soft-nms–improving object detection with one line of code. In *Proceedings of the IEEE international conference on computer vision*, pages 5561–5569, 2017.

[7] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[8] Xingyuan Bu, Junran Peng, Junjie Yan, Tieniu Tan, and Zhaoxiang Zhang. Gaia: A transfer learning system of object detection that fits your needs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 274–283, 2021.

[9] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer, 2020.

[10] Chein-I Chang. *Hyperspectral imaging: techniques for spectral detection and classification*, volume 1. Springer Science & Business Media, 2003.

[11] Ting Chen, Saurabh Saxena, Lala Li, David J Fleet, and Geoffrey Hinton. Pix2seq: A language modeling framework for object detection. *arXiv preprint arXiv:2109.10852*, 2021.

[12] Gong Cheng, Junyu Yang, Decheng Gao, Lei Guo, and Junwei Han. High-quality proposals for weakly supervised object detection. *IEEE Transactions on Image Processing*, 29:5794–5804, 2020.

[13] Mang Tik Chiu, Xingqian Xu, Yunchao Wei, Zilong Huang, Alexander G Schwing, Robert Brunner, Hrant Khachatrian, Hovnatan Karapetyan, Ivan Dozier, Greg Rose, et al. Agriculture-vision: A large aerial image database for agricultural pattern analysis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2828–2838, 2020.

[14] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 113–123, 2019.

[15] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[16] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2020.

[17] Keith J Dreyer and J Raymond Geis. When machines think: radiology's next frontier. *Radiology*, 285(3):713–718, 2017.

[18] Kenji Enomoto, Ken Sakurada, Weimin Wang, Hiroshi Fukui, Masashi Matsuoka, Ryosuke Nakamura, and Nobuo Kawaguchi. Filmy cloud removal on satellite imagery with multispectral conditional generative adversarial nets. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 48–56, 2017.

[19] Di Feng, Ali Harakeh, Steven L Waslander, and Klaus Dietmayer. A review and comparative study on probabilistic object detection in autonomous driving. *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[20] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016.

[21] Zheng Ge, Songtao Liu, Feng Wang, Zeming Li, and Jian Sun. Yolox: Exceeding yolo series in 2021. *arXiv preprint arXiv:2107.08430*, 2021.

[22] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[23] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[24] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[25] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[26] Ian Goodfellow, Patrick McDaniel, and Nicolas Papernot. Making machine learning robust against adversarial inputs. *Communications of the ACM*, 61(7):56–66, 2018.

[27] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International Conference on Machine Learning*, pages 1321–1330. PMLR, 2017.

[28] David Hall, Feras Dayoub, John Skinner, Haoyang Zhang, Dimity Miller, Peter Corke, Gustavo Carneiro, Anelia Angelova, and Niko Sünderhauf. Probabilistic object detection: Definition and evaluation. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1031–1040, 2020.

[29] Ali Harakeh, Michael Smart, and Steven L Waslander. Bayesod: A bayesian approach for uncertainty estimation in deep object detectors. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 87–93. IEEE, 2020.

[30] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

[31] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[32] Derek Hoiem, Yodsawalai Chodpathumwan, and Qieyun Dai. Diagnosing error in object detectors. In *European conference on computer vision*, pages 340–353. Springer, 2012.

[33] Han-Kai Hsu, Chun-Han Yao, Yi-Hsuan Tsai, Wei-Chih Hung, Hung-Yu Tseng, Maneesh Singh, and Ming-Hsuan Yang. Progressive domain adaptation for object detection. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pages 749–757, 2020.

[34] Licheng Jiao, Fan Zhang, Fang Liu, Shuyuan Yang, Lingling Li, Zhixi Feng, and Rong Qu. A survey of deep learning-based object detection. *IEEE access*, 7:128837–128868, 2019.

[35] Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? *Advances in neural information processing systems*, 30, 2017.

[36] John Kirchenbauer, Jacob R Oaks, and Eric Heim. What is your metric telling you? evaluating classifier calibration under context-specific definitions of reliability. In *International Conference on Learning Representations Workshop on Machine Learning Evaluation*, 2022.

[37] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[38] Volodymyr Kuleshov, Nathan Fenner, and Stefano Ermon. Accurate uncertainties for deep learning using calibrated regression. In *International conference on machine learning*, pages 2796–2804. PMLR, 2018.

[39] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017.

[40] Yanghao Li, Hanzi Mao, Ross Girshick, and Kaiming He. Exploring plain vision transformer backbones for object detection. *arXiv preprint arXiv:2203.16527*, 2022.

[41] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[42] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[43] 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 *European conference on computer vision*, pages 740–755. Springer, 2014.

[44] Zachary Lipton, Yu-Xiang Wang, and Alexander Smola. Detecting and correcting for label shift with black box predictors. In *International conference on machine learning*, pages 3122–3130. PMLR, 2018.

[45] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[46] Weiwei Liu, Haobo Wang, Xiaobo Shen, and Ivor Tsang. The emerging trends of multi-label learning. *IEEE transactions on pattern analysis and machine intelligence*, 2021.

[47] Andrew Ng. Machine learning yearning: Technical strategy for ai engineers in the era of deep learning draft version 0.5. *Harvard Bus. Publishing, Boston, MA, USA, Tech. Rep*, 2016.

[48] Kemal Oksuz, Baris Can Cam, Sinan Kalkan, and Emre Akbas. Imbalance problems in object detection: A review. *IEEE transactions on pattern analysis and machine intelligence*, 43(10):3388–3415, 2020.

[49] Mona Rahimi, Jin LC Guo, Sahar Kokaly, and Marsha Chechik. Toward requirements specification for machine-learned components. In *2019 IEEE 27th International Requirements Engineering Conference Workshops (REW)*, pages 241–244. IEEE, 2019.

[50] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[51] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[52] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[53] Jie Ren, Peter J Liu, Emily Fertig, Jasper Snoek, Ryan Poplin, Mark Depristo, Joshua Dillon, and Balaji Lakshminarayanan. Likelihood ratios for out-of-distribution detection. *Advances in Neural Information Processing Systems*, 32, 2019.

[54] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[55] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. Generalized intersection over union: A metric and a loss for bounding box regression. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 658–666, 2019.

[56] Tal Ridnik, Emanuel Ben-Baruch, Nadav Zamir, Asaf Noy, Itamar Friedman, Matan Protter, and Lihi Zelnik-Manor. Asymmetric loss for multi-label classification. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 82–91, 2021.

[57] David Rolnick, Priya L Donti, Lynn H Kaack, Kelly Kochanski, Alexandre Lacoste, Kris Sankaran, Andrew Slavin Ross, Nikola Milojevic-Dupont, Natasha Jaques, Anna Waldman-Brown, et al. Tackling climate change with machine learning. *ACM Computing Surveys (CSUR)*, 55(2):1–96, 2022.

[58] Soumya Roy, Asim Unmesh, and Vinay P Namboodiri. Deep active learning for object detection. In *BMVC*, page 91, 2018.

[59] Steffen Schneider, Evgenia Rusak, Luisa Eck, Oliver Bringmann, Wieland Brendel, and Matthias Bethge. Improving robustness against common corruptions by covariate shift adaptation. *Advances in Neural Information Processing Systems*, 33:11539–11551, 2020.

[60] Robert A Schowengerdt. *Remote sensing: models and methods for image processing*. Elsevier, 2006.

[61] Jacob Shermeyer, Thomas Hossler, Adam Van Etten, Daniel Hogan, Ryan Lewis, and Daeil Kim. Rareplanes dataset, June 2020.

[62] Mingxing Tan, Ruoming Pang, and Quoc V Le. Efficientdet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10781–10790, 2020.

[63] Peng Tang, Chetan Ramaiah, Yan Wang, Ran Xu, and Caiming Xiong. Proposal learning for semi-supervised object detection. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2291–2301, 2021.

[64] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.

[65] Juozas Vaicenavicius, David Widmann, Carl Andersson, Fredrik Lindsten, Jacob Roll, and Thomas Schön. Evaluating model calibration in classification. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 3459–3467. PMLR, 2019.

[66] Andreas Vogelsang and Markus Borg. Requirements engineering for machine learning: Perspectives from data scientists. In *2019 IEEE 27th International Requirements Engineering Conference Workshops (REW)*, pages 245–251. IEEE, 2019.

[67] Yu-Xiong Wang, Deva Ramanan, and Martial Hebert. Meta-learning to detect rare objects. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9925–9934, 2019.

[68] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. `https://github.com/facebookresearch/detectron2`, 2019.

[69] Gui-Song Xia, Xiang Bai, Jian Ding, Zhen Zhu, Serge Belongie, Jiebo Luo, Mihai Datcu, Marcello Pelillo, and Liangpei Zhang. Dota: A large-scale dataset for object detection in aerial images. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[70] Enze Xie, Jian Ding, Wenhai Wang, Xiaohang Zhan, Hang Xu, Peize Sun, Zhenguo Li, and Ping Luo. Detco: Unsupervised contrastive learning for object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8392–8401, 2021.

[71] Tong Yu and Hong Zhu. Hyper-parameter optimization: A review of algorithms and applications. *arXiv preprint arXiv:2003.05689*, 2020.

[72] Peng Zhang, Jiuling Wang, Ali Farhadi, Martial Hebert, and Devi Parikh. Predicting failures of vision systems. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3566–3573, 2014.

[73] Qinglong Zhang and Yu-Bin Yang. Rest: An efficient transformer for visual recognition. *Advances in Neural Information Processing Systems*, 34, 2021.

[74] Quan-shi Zhang and Song-Chun Zhu. Visual interpretability for deep learning: a survey. *Frontiers of Information Technology & Electronic Engineering*, 19(1):27–39, 2018.

[75] Yilun Zhou, Marco Tulio Ribeiro, and Julie Shah. Exsum: From local explanations to model understanding. *arXiv preprint arXiv:2205.00130*, 2022.

[76] Barret Zoph, Ekin D Cubuk, Golnaz Ghiasi, Tsung-Yi Lin, Jonathon Shlens, and Quoc V Le. Learning data augmentation strategies for object detection. In *European conference on computer vision*, pages 566–583. Springer, 2020.