On the Identification of Uncertainty Regions

Peter Orlovskiy

School of Computer Science
The University of Auckland

Supervisor: Joerg Wicker

A dissertation submitted in partial fulfillment of the requirements for the degree of Master of Data Science, The University of Auckland, 2025.

Abstract

Neural networks are often not well-calibrated and are overconfident in their predictions so uncertainty quantification methods become critical for developing reliable machine learning models, as decision boundaries learnt from limited training data are at best, approximations of the true boundaries. However, network retraining is often untenable and class labels for test data not available, motivating the need for post-hoc uncertainty-based identification of uncertainty regions; regions where predictions are unreliable and often incorrect such as adversarial regions or regions outside the domain of the training data.

The current work reviews the current uncertainty literature, incorporating concepts of applicability domains, and conformal prediction to develop a novel, simple-but-effective methodology, for isolating regions of uncertainty. The background, motivation, and intuition behind the proposed method is discussed in depth and the effectiveness examined, with practical uses presented in three case studies with five datasets. The proposed method was found to be beneficial in improving model performance, adversarial detection, and misclassification prediction. The findings are discussed and directions for future work are presented.

Contents

Chapter 1

Introduction

Uncertainty quantification in model development, whether it involves a statistical or machine learning model, is critical; the data on which the model is developed on are always limited. This means that the decision function a model learns is based on incomplete information, and it is crucial to be able to quantify and understand the uncertainty surrounding a prediction. For example, in statistical models, it is common to report point estimates with corresponding prediction bounds (with an associated confidence score; i.e., 95%), rather than the point estimates alone [1].

The point of supervised machine learning, in practice, is to be able to develop a model that "learns" a decision function. This model can then be given inputs and, through the learnt decision function, the model is able to make a prediction about what the corresponding labels should be. However, evaluating the quality of the predictions in reality can be difficult since one often does not have the corresponding targets for the test set; if targets were always available, we would not need machine learning models. Therefore, being able to quantify uncertainty becomes essential, as it provides model practitioners the ability to understand the reliability of their predictions and to determine whether the outputs that the model is producing may not be applicable, i.e., the model is operating outside of the domain it has been trained on.

For supervised classification tasks, machine learning models typically output a probability distribution over the classes, which can be used to model the uncertainty in predictions [2]. For a motivating example, suppose we have two examples in a cats vs. dogs prediction task. On the first one, the model predicts the image to contain a cat with a 90% probability and a dog with 10% probability. For the second, suppose this is 55% and 45%. For both examples, the model would predicts "cat" since the highest probability is associated with the cat label. However, it is obvious that in the first example the model is significantly more confident with lower uncertainty in its prediction than in the second example.

A related problem is that a cat and dog model, when given an image of a shark, will still predict a cat or a dog, although obviously, this is incorrect. Likewise, if a self-driving car model was trained on videos during sunny days and clear nights, it may not respond as well to rarer conditions such as fog or smog. These problems occur when the test feature distribution changes and therefore it is important that a machine learning model is able to signal that it is unable to make an accurate prediction. Unfortunately, most of the time, the models are not even designed for this to be possible. However, with an uncertainty quantification mechanism in place, the natural extension to this is selective classification which directly ties into predictive uncertainty. If the test data falls outside of the domain it was trained on, or if the prediction reliability is low, or there is difficulty in deciding on the prediction, the model should be able to abstain from making the prediction. However, this is easier said than done, since adding an abstaining mechanism (such

as an ”I don’t know” class) complicates the model further, requires the model to be retrained, and comes with its own questions regarding the uncertainty pertaining to the new architecture.

Another complicating factor is the temptation to call the softmax-normalised model outputs a probability distribution, because even though some models such as logistic regression, naive Bayes etc. are probabilistic by design, neural networks are just optimising a score function and do not inherently try to learn a probability distribution. This means the “probabilities” one obtains from these models is not guaranteed to reflect true confidence or uncertainty [4] and, in fact, the models are almost always overconfident [5]. This is because neural networks, by optimising the loss function, encourage assigning higher values to the correct class, but normalising and learning anything about the distribution of the labels is not part of the process; i.e., model’s predicted probabilities do not reflect the true likelihood of outcomes.

Figure 1.1: Incorrectly classified stop sign with high-confidence. Reprinted from [3]

Because of this, whenever a machine learning model makes a prediction, it is important to critically assess its reliability. Important questions include: does the model have sufficient capacity to make this prediction? Is the prediction within the domain of data the model has seen, or is it an out-of-distribution case where the model may be prone to error? How reliable is the prediction, and what metrics or uncertainty estimates can help quantify this reliability? Finally, is the model’s reported confidence well-calibrated, or is it overconfident in cases where uncertainty is actually high?

A framework in cheminformatics named the applicability domain helps to alleviate some of these questions by formalising the prediction process into three steps [6]. *Applicability* determines whether the model is applicable to the input. For example, images of sharks are not applicable for a cat vs. dog predicting model. *Reliability* determines whether an input is within a reliable feature space. For example by using the local density of surrounding points. Finally, *decidability* assesses whether the model is able to make a confident decision between competing classes. In the cat vs. dog example above, the decidability for cat is higher with a 90% probability than for a 55% probability. Each of these steps is done sequentially, with the predictive example being outside of the *applicability domain* if any of the steps is disappointed as the model is operating outside of the domain it has been trained on. However, unreliable examples can be found in regions either outside of the model’s domain or near regions with class overlap [2], which are two distinct

problems, meaning simple uncertainty quantification is more difficult than it seems.

Some interesting examples of models operating outside of their applicability domains are found in adversarial examples. The work in [7] showed that machine learning models are susceptible to adversarial attacks - (typically) visually-identical constructed inputs on which models mis-perform; three examples are shown in Fig. 3.2. Although these attacks are constructed by researchers with the intent to fool the models, it is a crucial security risk, never mind that it is possible that such adversarial examples exist in the "wild". For example, previous research found that stop signs can be categorised as speed signs if real (small) stickers are placed on them [3]. This can be seen in Figure 1.1. However, stickers are not always stuck on stop signs by researchers, but possibly by individuals for their own entertainment. The consequences of model failures such as this have the potential to be disastrous. However, while adversarial examples are synthetically-generated examples that are able to fool models, more realistic examples can occur in test sets because the learnt model is an imperfect approximation of the true decision boundaries which was trained on incomplete and noisy data. These test time examples can occur outside of the model's applicability domain and therefore it is important to develop mechanisms where pre-trained networks can be enhanced with uncertainty measures to enable them to reject making a prediction without changing the underlying classification model.

Chapter 2

Related Work

2.1 Uncertainty

Uncertainty in machine learning can be roughly broken down into two parts. *Epistemic* uncertainty comes from having incomplete information about which model or architecture is the best fit for the data we have on hand, and *aleatoric* uncertainty is uncertainty coming from variation that we are unable to account for, such as image bluriness or rotation [8]. Both types of uncertainty are important to detect. For example, out-of-distribution (OOD) detection aims to detect examples with high epistemic uncertainty while avoiding confounding from high aleatoric uncertainty examples [9]. Unfortunately, recent work has found that modern methods fail to separate the two types although well-trained networks may still send OOD examples into low-confidence (high uncertainty) regions [10]. It may also be the case that, in practice, good outcomes can be achieved without separating the two types of uncertainty.

For example, the work in [11] proved that for ReLU-activation networks with a softmax layer, one can always find examples far outside of the training domain with arbitrarily high confidence. The authors proposed two approaches to increase a model's uncertainty in these far away regions. The first added noisy examples outside of the training domain and the second generated adversarial examples (also outside the domain) maximising the classifier's confidence. The authors then optimised on these and the original training samples finding substantial improvements (increases) in uncertainty on OOD points, and detection of adversarial examples, all while maintaining the original test performance, implying that while separating the two types of uncertainty may be difficult, identifying and optimising OOD regions to have low uncertainty can be beneficial for certain tasks. Unfortunately, this approach also requires model retraining which may not always be feasible.

A related work in [5], realising that machine learning models are often overconfident making uncertainty estimates generally unreliable, proposed a novel post-hoc method not requiring model retraining or extra data. The process involved adding a Dirichlet meta-model to a trained classifier over multiple intermediate layers with the goal of meta-learning a model's uncertainty. This led to promising results on various tasks such as out of domain detection, misclassification detection, and transfer learning, implying that post-hoc methods for uncertainty quantification are possible. However, while approaching the problem through uncertainty quantification is one perspective, another view considers regions of uncertainty where models are not reliable.

2.2 Applicability Domain

Originating in cheminformatics, the concept of an applicability domain (AD) defines a framework for determining regions in which a model's predictions may be considered reliable. The valid region(s) are termed the AD. Although approaches to defining the process of finding the AD vary [12], the prominent work in [6] describes a simple three-step criterion to determine which points are outside of the AD based on their applicability, reliability and decidability. The purpose of this is to reject predictions for chemicals or molecules that fall outside the AD, since a model's predictions on these instances are almost certainly unreliable. However, applications of the AD concept in general machine learning remain limited altkough there have been many related works examining OOD detection [13], novelty detection [14], and outlier detection [15]. All of these are analogous problems dealt by the AD, since they all deal with points in regions where the model should not have high certainty. However, these different tasks indicate that understanding a model's uncertainty is a complex process, requiring tailored techniques for each distinct problem.

The work in [2] extended the AD concept, distinguishing between uncertainty through the features space (novelty detection) and the expected reliability of predictions (confidence estimation). *Novelty detection* considers a point as novel when it is outside the domain of the data; this point is not necessarily wrong or incorrect, the model has just not seen similar points before. In contrast, *confidence estimation* aims to use the class labels to estimate prediction confidence [16], since points located in regions with class-overlap should have higher uncertainty [2]. However, class-overlap has been found to be a smaller cause of failure than large domain extrapolations [17]. OOD data is also typically located closer to the decision boundaries than in-distribution data [18]. This is perhaps because well-trained models have an approximately-optimal decision boundary structure which allows them to send OOD examples into low-confidence regions around the boundaries [17]. Deeper networks have also been found to have a filtering effect which are able to discriminate between relevant and irrelevant directions, separating in-distribution and OOD examples, with this effect being stronger for deeper networks (those with more layers) [17]. This illustrates the need for robust methods to isolate high uncertainty regions around the decision boundaries (areas of class-overlap). Unfortunately, while the decision boundaries of some models are well understood (such as support vector machines), decision boundaries of deep neural networks are not well understood at all [19], although approximating closeness to these boundaries is essential for isolating regions of high uncertainty.

2.3 Adversarial Regions

One way to understand decision boundaries is through adversarial examples [19]. The intuition is that since adversarial attacks seek to find a minimal perturbation (causing the model to misclassify) and the perturbed image looks almost identical to humans, the adversarial attack is only successful if it crosses the decision boundary. This means the boundary is close in space to the

original image and the goal of an attack is to find it. Since misclassification occurs for this point but the input is visually identical (or almost) to the original, the location where the adversarial example lies is an adversarial region for the model since it is possible to generate infinitely-many adversarial examples within a small hyper-rectangle surrounding a clean image [20]. Since these regions can essentially be defined by the points located inside them, they should, intuitively, also constitute regions of uncertainty since the model misclassified the points defining them. However, this is not the case, with models often having very high confidence (which is inversely proportional to a model's uncertainty) on adversarial examples [21], which can make uncertainty estimates unreliable [5]. Therefore, the intuition that high confidence should be equivalent to low uncertainty, and vice versa, is not accurate. It then becomes important to consider both uncertain regions and adversarial regions simultaneously although literature comparing the two is lacking, perhaps due to a lack of widely accepted definitions. In fact, the only mention of an adversarial region is in [22], where they define an adversarial region as the space between the learnt boundary and the true boundary. However, since we never know the true boundary, except in simulated tasks, it is impossible to concretely isolate these regions and we must resort to approximations. The view in this work is that adversarial regions are where models make mistakes and thus should be treated as regions of uncertainty, although perhaps identified using different techniques, to normal uncertainty regions. Adversarial regions are just as relevant as uncertainty regions since adversarial examples can be thought of as artificial versions of "natural" (test-time) adversarial examples, which have been shown to occur in the real world [23].

Previous work in [24] developed a detector used to define adversarial subspaces, finding that these subspaces exist across all layers in a network, and that a specially-trained classifier was able to distinguish between normal and adversarial points on five adversarial attacks. However, the authors only examined the use of this detector on generated adversarial examples and did not consider whether the detector could identify points of uncertainty. Likewise, the detector utilises distances measures which have been found to be outperformed by confidence-based measures [2]. However, other work has explored predictive entropy and mutual information for adversarial detection, finding promising results with the latter on MNIST and a cat vs. dog classification task [21]. This motivates the further exploration of using uncertainty measures for adversarial detection.

Another related work in [25] examined the effect of adversarial training [26] on adversarial robustness. Adversarial training involves training a model on original and adversarial examples simultaneously. While this work focuses adversarial robustness and not identifying regions of uncertainty, the authors examined the behaviour of logit-gaps between standard and adversarially-trained networks which indicate the proximity to decision boundaries. The work examined the logit-distributions of adversarially-trained networks, finding that adversarial training lowers the maximum logit value and shrinks the training logit gaps. This means the distribution of the gaps becomes less spread, with lower mean, a stronger positive skew, leading to less confident predictions and more effort by the model to learn the true underlying features. The authors proposed this

happens due to the regularisation that implicitly takes place during adversarial training, with models learning to avoid being extremely confident by shrinking the high-confidence margins around the decision boundaries. Intuitively, this behaviour occurs since adversarial examples cross the boundary but remain close to it, thus shrinking the high-confidence margin lowers the confidence on the adversarial examples. However, for already-trained models, approximating the closeness to the decision boundaries can provide the adversarial and uncertain regions necessary for identifying these points, although other work has shown that specially-crafted attacks designed to bypass uncertainty-based detection measures can still be effective at fooling the models [27].

2.4 Abstention

The next practical step after isolating uncertainty regions is to reject (or abstain from making) predictions which occur in these regions. This is because examples found in uncertain regions are themselves bound to be uncertain. Much of previous research focuses on rejection through confidence which can be estimated using many techniques [2]. For example, statistical methods such as posterior probabilities or naive Bayesian classifier [28], machine learning techniques such as k-nearest neighbours, decision trees, random forests, support vector machines, or neural networks [2]. Perhaps the simplest measure to use, however, is to take the highest predicted probability as a model's confidence. However, often-times model's confidence does not correspond with the true likelihood of correctness so confidence calibration techniques have been developed to align the predicted probabilities with the true likelihood of correctness [4, 29]. With a model's confidence, abstention becomes possible. The work in [30] categorises abstention into confidence-based methods [31], selective classification with a predictor and selector [32], or predictor and rejector [33], and score-based methods [34]. The works in [35, 36] review existing literature on techniques to enable models with a reject option when a model's confidence is lacking or it falls outside of the AD. However, while many techniques have been developed, the problem with all these approaches is that many require class labels which may not be available and therefore determining regions of uncertainty based on the feature space or predicted class probabilities becomes important.

A related, but distinct to the above, work in [37] is an extension of conformal prediction [38] where a classification pipeline is enhanced to have a reject option. The process involves generating nonconformity scores (how strange an example is compared to the training data) on a calibration set, determining a threshold based on a desired error rate, and rejecting examples with low confidence. This technique has strong theoretical guarantees, bounding the probability of making incorrect predictions on accepted instances, allowing for reliable statistically-guaranteed control over uncertainty and risk in classification tasks. This method is also model and dataset agnostic, allowing it to be used for almost any problem with minimal restrictions. However, this method only typically considers the maximum predicted probability and does not consider the whole class distribution, or proximity the decision boundaries and may not perform well when a model's confidence is not calibrated.

Chapter 3

Methodology

3.1 Research Aims

The primary research aim for this work is in isolating uncertainty regions for supervised classification tasks across various datasets. The formal research question is as follows:

- Can post-hoc uncertainty quantification measures be used to isolate regions of uncertainty?

The proposed methodology is evaluated using three case studies in Chapters 5, 6, and 7, which report the adequacy of the proposed methodology on three distinct problems:

1. Improving model performance
2. Detecting adversarial examples
3. Misclassification prediction

This work serves as a first step towards establishing a model evaluation metric through the concept of an applicability domain by increasing prediction reliability and interpretability of machine learning models in practice.

3.2 Formal Definitions

Formally, for features $\mathcal{X}$ and labels $\mathbf{y}$, there is an input/feature vector $\mathcal{X}_i \in \mathbb{R}^k$, where k is the dimensionality (i.e. number of pixels in an image), for which there is a corresponding label y_i for $i \in 1, .., n$, where n is the number of examples in the dataset. A machine learning model, f, outputs logits $f(X_i) = \mathbf{z}$, trying to minimise the loss $\ell(\mathbf{z}, y_i)$. Note that the dimensionality of the logit vector $\mathbf{z}$ is C, the number of classes. The logits $\mathbf{z}$ can be normalised to [0, 1] using the softmax function:

$$\text{Softmax}(z_i) = p_i = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}}, \quad i = 1, \ldots, C \tag{3.1}$$

Applying softmax to all elements of $\mathbf{z}$ produces the predicted probabilities vector $\mathbf{p}$. A common classification loss function to use is the cross-entropy loss which is defined as:

$$\text{Cross Entropy} = -\sum_{i=1}^{C} \mathbf{1}\{i = y\} \log(p_i) \tag{3.2}$$

where $\mathbf{1}\{\cdot\}$ is the indicator function, y is the true label for the example, and p_i is the predicted probabilities. Note that this is the general, multi-class no-reduction cross-entropy loss and, since there is no reduction, this loss outputs a value for every example. In practice during model training, there are multiple examples (the size of the first dimension of a batch) and the "mean" reduction is used, where Eq. 3.2 is calculated for all examples in the batch and then averaged. This "loss" is then minimised by the chosen optimizer. To make a prediction $\hat{y}_i$, typically the index of the maximum value of $f(x_i)$ taken as the predicted class. Note that since the logit to probability mapping (Eq. 3.1) is order-preserving, there is no difference between whether the maximum value is found from the logits or the predicted probabilities.

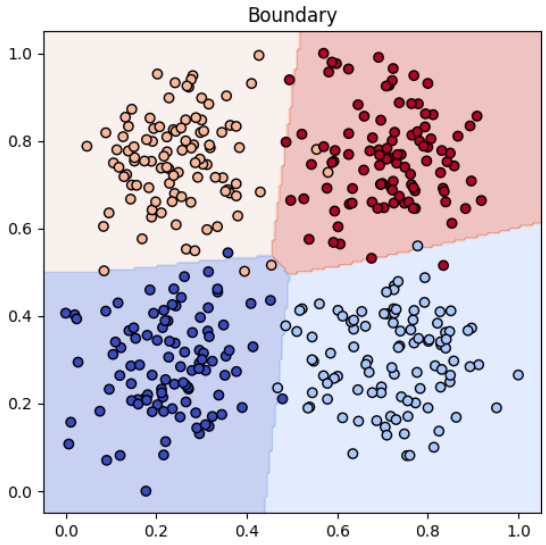

Figure 3.1: Chessboard dataset with four classes.

3.3 Uncertainty

The predicted probabilities p_i can be thought of as a joint probability distribution across the classes which represents a model's certainty about its predictions. For example if class 2 has the highest predicted probability, then the model believes that, according to its learnt decision function, that class 2 is the most likely class for this example. However, there are many ways of quantifying the uncertainty from these predicted probabilities.

3.3.1 Entropy

Entropy measures the average "surprise" in the predicted distribution. Formally, entropy is defined as:

$$\text{Entropy} = -\sum_{i=1}^{C} p_i \log(p_i) \tag{3.3}$$

where C is the number of classes and $p_i = \text{Softmax}(z_i)$ is the predicted probability of class i, where z_i are the logits. Note that like 3.2, this formulation is for a single example - there is no reduction.

Entropy values are high when the distribution is flat ($p_i \approx p_j, \forall i, j$) and low when there is a single class which has a much higher predicted probability than the rest. This directly corresponds to a model's uncertainty since a model is uncertain when all p_i are similar (the model has difficulty choosing between one or thinks that multiple classes are likely), and certain when one probability is much higher than the rest (indicating no indecision between classes).

Consider the simulated four-class dataset shown in Figure 3.1. A model was trained on this dataset and the decision boundaries are shown. Since there are four classes, there are four class regions and five distinct decision boundaries with two points where three boundaries meet. A point on a decision boundary separating two classes will have two predicted probabilities that are equal. For example, a point on the yellow/red boundary will have $p_{yellow} = p_{red}$ and similar for the other boundaries.

For the areas where three boundaries meet, say yellow/red/blue, there will be $p_{yellow} = p_{red} = p_{dark\ blue}$. This means that Eq. 3.3 will be higher at that point and the red/dark blue/light blue points than anywhere else, since these points, and their surrounding regions, represent the highest uncertainty where the model is unable to differentiate between any of these classes. On the flip side, the clusters inside each region will have a high predicted probability for the corresponding class and low for probabilities for the others, representing areas of low entropy.

Therefore, entropy is able to quantify a model's uncertainty with high values corresponding to high uncertainty. The advantage of using the entropy defined in Eq. 3.3 is that we do not need the true class labels; entropy can be computed just from the logits z_i. This is convenient for test sets for which no true class labels exist and one is only able to calculate the entropy.

3.3.2 Information Content

Another information-theory approach to quantifying uncertainty is the self-information (also known as the surprise or the pointwise information content), of the model's predicted class; referred to as the information content for simplicity from now on. Given the predicted probabilities, $\mathbf{p}$, the information content is defined as,

$$\text{Information Content} = -\log(\max_i p_i) \tag{3.4}$$

and shows how surprised a model is in its own prediction. If the predicted class probability is high, $\max_i p_i \approx 1$, then the information content is close to 0, because the model is very confident in its prediction. However, if the predicted probability is very low, $\max_i p_i \approx 0$, then the information content will be a large number, indicating that the model is uncertain. Realistically, $\max_i p_i$ is minimised when all classes have the same predicted probability, i.e., $p_i = \frac{1}{C}, \forall i \in C$, i.e., when the point is at the intersection of all decision boundaries, or on the boundary if $C = 2$.

However, because Eq. 3.4 only considers the maximum class, it is unable to differentiate between when the model is on the decision boundary or not. This will be discussed further in section 3.5.3. Note that since the information content approach only relies on the model's maximum

probability, like entropy, it does not require the target labels for computation.

3.3.3 Probability Gaps

Another alternative to entropy and information gain, is the logit or probability gaps. Logit gaps are the difference between the largest and second-largest logit [25], $z_1 - z_2$, where z_1 is the largest logit and z_2 is the second-largest logit. Similarly, probability gaps are the difference between the largest and second largest predicted probability, $p_1 - p_2$. Finally, there are the normalised probability gaps considered in this work:

$$\text{Probability gap} = \frac{p_1 - p_2}{p_1} \tag{3.5}$$

where p_1 and p_2 are the highest and second-highest predicted probabilities, which ensure that the probability gap is bounded: gap $\in [0, 1]$. If the difference between the two highest probabilities is low, then the probability gap is low, indicating proximity to the decision boundary. The point where the gap is minimised is when a point is on a decision boundary and the two highest probabilities are equal $p_1 = p_2$. On the flip side, when the probability gap is large, the largest probability is much larger than all others, indicating that the example is inside confident regions because the model is confident in its classification.

Considering the example in Fig. 3.1, probability gaps are ideal for determining proximity to decision boundaries but, unlike entropy, probability gaps are unable to distinguish between a point on the yellow/red boundary if a point is at the intersection of the red/yellow/dark blue boundaries.

3.4 Adversarial Attacks

In general, an adversarial example $\mathbf{x}' = \mathbf{x} + \boldsymbol{\eta}$ is an altered version of an input $\mathbf{x}$, with adversarial noise $\boldsymbol{\eta}$, generated by an adversarial attack g, given to a machine learning model f, which causes the model to perform worse. How the model "performs worse" depends on the task; typically, adversarial literature has focused on classification and while it is possible (and sometimes trivial) to adapt these adversarial attacks for regression tasks, the results are typically less interesting. This is because in classification, an adversarial example is successful if $f(\mathbf{x} + \boldsymbol{\eta}) \neq f(\mathbf{x}) = y$, where y is the true label. However, in regression tasks, y is a continuous value, so we may have higher mean squared error (MSE) with the adversarial example, but it's, perhaps, less obvious that the performance is much worse.

Typically, adversarial examples can be categorised into two main buckets, poisoning and evasion [39]. The former "poisons" the training data to compromise normal model function. This means that given unaltered, normal examples at test time, the model will either perform worse, have an implanted back-doors or trojans etc. The importance of mitigating such attacks can not be understated. On the other hand, evasion attacks, do not compromise normal model function,

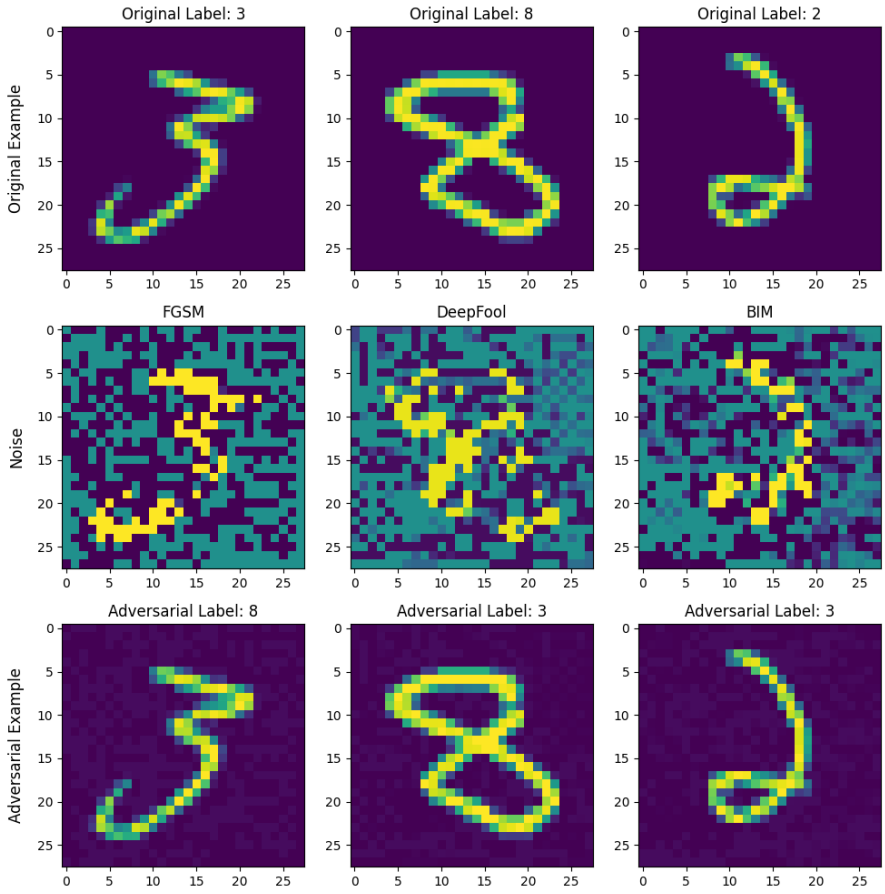

Figure 3.2: Adversarial examples on the MNIST dataset. The top row shows the original examples, the titles contain the original labels, the second row shows the (exaggerated) noise, η, and the third row shows the adversarial examples. Note that the model originally classified these original examples correctly

but focus on generating adversarial examples which make a classifier misclassify. These can be anything from specially crafted noise [40, 26] to adversarial patches [41] and can be very effective.

Adversarial attacks can also be targeted (see [40]) or untargeted (see [26]). An untargeted attack is one such that $f(\mathbf{x}) = y \neq f(\mathbf{x} + \boldsymbol{\eta})$, and a targeted attack is when we have a target class t and an example is adversarial if $f(\mathbf{x}) = y$ and $f(\mathbf{x} + \boldsymbol{\eta}) = t$. This work focuses on the latter since we are not interested in the attacks themselves here, but are more interested in the regions that these attacks are found.

Trivial examples of inputs causing a model to misclassify are not hard to come up with. For

example, if $f(\mathbf{x})$ outputs "dog", then creating $\mathbf{x}' = \mathbf{0}$, i.e. setting all pixels to black, will make any classifier "misclassify". However, examples like this are not interesting because even a human would then "misclassify" them. Thus, the examples we are actually interested in are those where a human would have no problem making a correct classification. Because of this, adversarial literature has typically focused on adversarial attacks imperceptible to the human eye.

Formally, we define an adversarial attack as a procedure (could be an algorithm or another model), g, such that $g(\mathbf{x}) = \boldsymbol{\eta}$, producing the input perturbation $\boldsymbol{\eta}$, which produces an adversarial example:

$$\mathbf{x}' = \mathbf{x} + \boldsymbol{\eta} \tag{3.6}$$

such that $f(\mathbf{x}') \neq f(\mathbf{x})$, for classifier f. To make such perturbations indistinguishable for humans, mathematically, the perturbations are constrained to be within a magnitude ϵ and are typically constrained using the L1 norm:

$$||\mathbf{x}' - \mathbf{x}||_1 = \sum_{i=1}^{k} |x_i - x_i'| < \epsilon \tag{3.7}$$

the L2 norm:

$$||\mathbf{x}' - \mathbf{x}||_2 = \sum_{i=1}^{k} (x_i - x_i')^2 < \epsilon \tag{3.8}$$

the L_∞ norm:

$$||\mathbf{x}' - \mathbf{x}||_\infty = \max_i |x_i - x_i'| < \epsilon \tag{3.9}$$

for some small ϵ. Adversarial attacks are ideal in uncertainty-related works since attacks aim to move examples to high uncertainty areas, cross a decision boundary, and achieve misclassification. Since a decision boundary is where the logits of a model are equal, this means that adversarial attacks generally push examples towards a higher loss. Looking at Equations 3.2 and 3.3, we see that the cross entropy function is very similar to the entropy, except that the cross entropy function takes only the correct class' log-probability and the entropy takes a predicted-probability weighted sum of the log-probabilities. Eq. 3.2 is small when the predicted probability for the correct class is close to 1 and Eq. 3.3 is small when one class dominates the logits. This means if cross entropy is the loss used, there is an inherent relationship between the loss and the entropy function. By this rational, the location of adversarial examples in the feature space indicates regions where the loss and entropy are both high, and due to the proximity of the decision boundary, the model's predictions should not be confident.

3.4.1 FGSM

The Fast Gradient Sign Method (FGSM) was introduced by Goodfellow in [26]. The attack takes the model parameters $\boldsymbol{\theta}$, the input $\mathbf{x}$, and the target y and computes the gradient of the loss W.R.T. the input $\mathbf{x}$ to obtain an adversarial perturbation:

$$\boldsymbol{\eta} = \epsilon \times \text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \mathbf{x}, y)) \tag{3.10}$$

Where ϵ is some small value. This perturbation is then added to the input to obtain the adversarial example $\mathbf{x}' = \mathbf{x} + \boldsymbol{\eta}$. Note that $\mathbf{x}'$ is typically clipped so that its feature values do not exceed the boundaries of the data; i.e., clipped to be within [0, 1] for [0, 1]-scaled images.

3.4.2 BIM

The Basic Iterative Method (BIM) was introduced in [42] and extends the attack in section 3.4.1 with another parameter α to make the attack iterative. The number of iterations is typically computed as $T = \epsilon/\alpha$. The attack begins with initialising the adversarial example to $\mathbf{x}'_0 = \mathbf{x}$, and then iteratively computes:

$$\mathbf{x}'_{t+1} = \text{clip}\left(\mathbf{x}'_t + \alpha \times \text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \mathbf{x}'_t, y))\right) \tag{3.11}$$

Where clip is same function described above, ensuring that the example remains inside the valid input space.

3.4.3 DeepFool

DeepFool is an untargeted iterative attack originally introduced in [43]. Unlike FGSM and BIM which use a fixed ϵ, DeepFool iteratively computes the smallest perturbation required to reach the decision boundary.

In each iteration, DeepFool linearises the classifier around the current point x and computes the orthogonal projection onto the nearest decision boundary of the linearised model. Given a classifier f, we define $f_j(x)$ as the j-th element of $f(x)$ for $j \in 1, ..., C$ classes, where $\hat{y} = \arg\max_j f_j(x)$. For each iteration i, the attack initialises $\mathbf{x}'_0 = \mathbf{x}$ and finds the class l whose decision boundary is closest to the current point in the linearised space using:

$$l = \arg\min_{j \neq \hat{y}} \frac{|f_{\hat{y}}(\mathbf{x}'_i) - f_j(\mathbf{x}'_i)|}{||\nabla_{\mathbf{x}} f_{\hat{y}}(\mathbf{x}'_i) - \nabla_{\mathbf{x}} f_j(\mathbf{x}'_i)||_2} \tag{3.12}$$

It computes the minimal perturbation:

$$\boldsymbol{\eta}_i = \frac{|f_l(\mathbf{x}'_i) - f_{\hat{y}}(\mathbf{x}'_i)|}{||\nabla_{\mathbf{x}} f_l(\mathbf{x}'_i) - \nabla_{\mathbf{x}} f_{\hat{y}}(\mathbf{x}'_i)||_2^2} \left(\nabla_{\mathbf{x}} f_l(\mathbf{x}'_i) - \nabla_{\mathbf{x}} f_{\hat{y}}(\mathbf{x}'_i)\right) \tag{3.13}$$

The adversarial example is then updated as $\mathbf{x}'_{i+1} = \mathbf{x}'_i + \boldsymbol{\eta}_i$. The algorithm continues until misclassification is achieved, i.e., when $f(\mathbf{x}'_N) \neq f(\mathbf{x}) = y$ for some iteration N. The final adversarial perturbation is then:

$$\boldsymbol{\eta} = \sum_{i=1}^{N} \boldsymbol{\eta}_i = \mathbf{x}'_N - \mathbf{x} \tag{3.14}$$

Unlike FGSM and BIM DeepFool is able to dynamically determine the required perturbation size for each example, often resulting in smaller perturbations. Note that the equations above are presented using the L2-norm.

3.5 Motivation

The aim of this section is to provide the motivation behind the methodology and explaining the rationale behind the design decisions. This section focuses only on the simulated chessboard/checkerboard dataset described in Section 3.7.2 and the model used is described ion Section 3.7.2. This is a simulated dataset, where the training and testing datasets come from the exact same distribution. The dataset is relatively hard for a model to learn due to the diagonal decision boundary lines the classifier is required to learn, because the classifier is forced to learn non-linear boundaries.

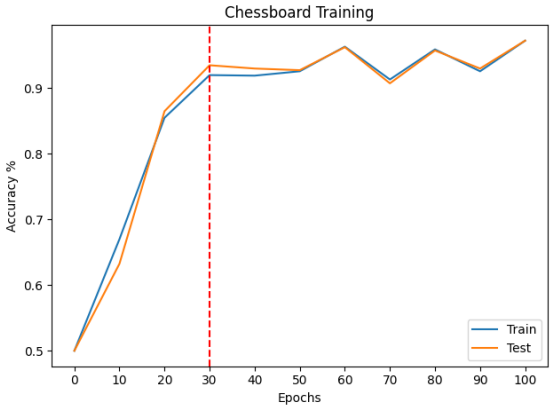

Figure 3.3: Training and test accuracy by epoch trained on the chessboard dataset.

3.5.1 Decision Boundaries

Figure 3.3 shows the training and test accuracies by epoch trained on the chessboard dataset. The vertical horizontal line, at epoch 30, shows the first occurrence of where the training starts to plateau. After this point, the model's accuracy on both the training and test sets oscillates slightly, although roughly remains constant. It is important to note that the model's test performance is expected to be almost identical for the training and test sets due to the construction of this dataset.

Figure 3.4 shows the decision boundaries learned for every 10th epoch, the same epochs as in Figure 3.3. The title for each facet shows the epochs trained and the corresponding test accuracy. Figure 3.4 shows that effective decision boundaries start being learned at epoch 30, which continues up to and including epoch 50 and then starts overfitting, which is consistent with the vertical

line in Figure 3.3. Here, it is easy to tell that the model is overfitting because instead of learning diagonal decision boundaries, the model starts wrapping some clusters inside their own decision boundary; this behaviour is most evident for epoch 100. However, with just the training/test accuracy, seen in Figure 3.3, understanding the shape of the boundary is much harder.

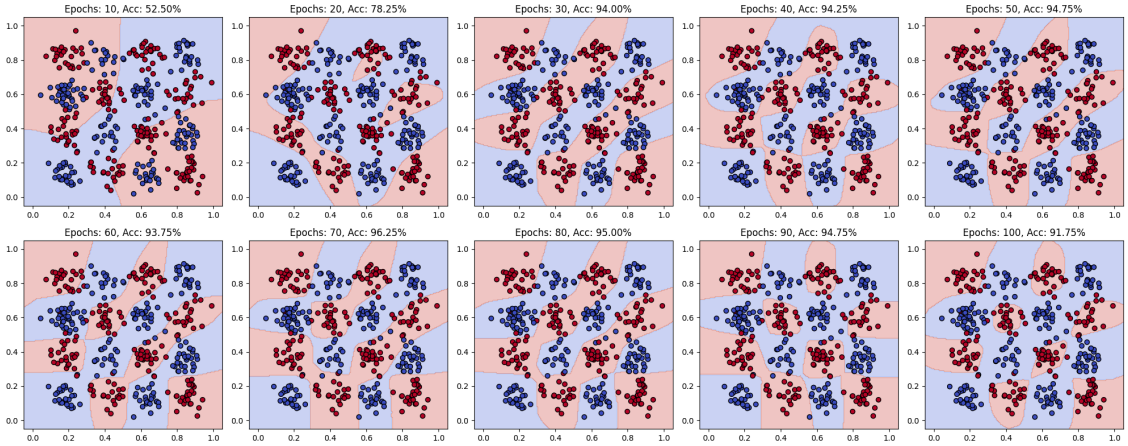

Figure 3.4: Decision boundaries for every 10 epochs trained. Note that the test points are plotted for visibility.

A point on the decision boundary, naturally, represents the most uncertainty as the model does not know which class to assign the prediction to. For points that are very close but not quite on the boundary, the model's uncertainty should still be high, since the decision boundary learnt by the model is never perfect; data is noisy, the data-generating process may be faulty, and the model is just an approximation of the true underlying relationships. In the areas around a decision boundary, even a slight change to the input features may cause the class label to be flipped. Although this simulated example is in two dimensions, higher-dimensional spaces have the same properties, albeit with much more complicated decision boundaries that are impossible to visualise.

3.5.2 Loss

Machine learning models learn by minimising the loss (e.g. cross-entropy loss in Eq. 3.2), which is able to act as a functional proxy for the decision boundaries. Typically this loss is reduced (averaged most-commonly) and then minimised, however, without this reduction, loss is calculated per point and therefore we can calculate a loss landscape across the range of the training data. Figure 3.5 shows this loss landscape as the model trains. Note that Figure 3.5 is equivalent to 3.4 except that the contours are showing loss.

Figure 3.5 shows that the loss for most points starts off being high everywhere and, through

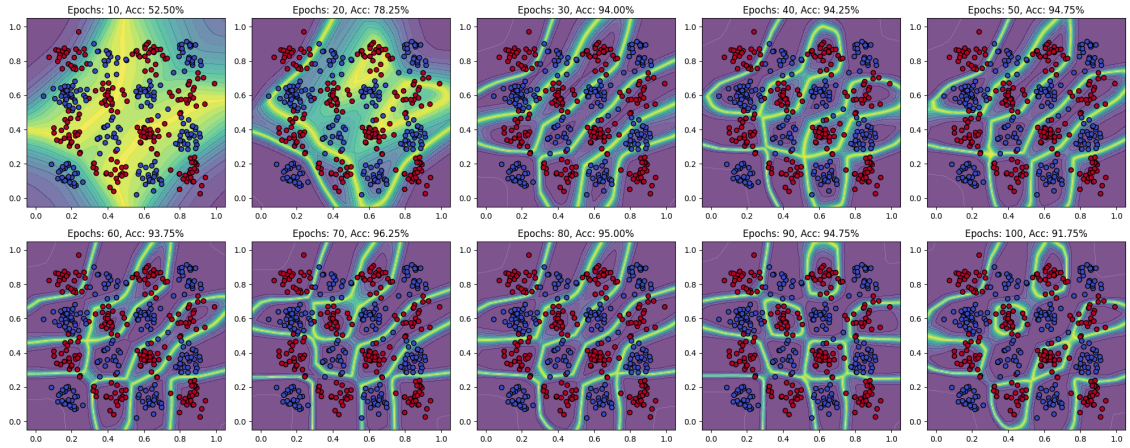

Figure 3.5: Loss landscape for every 10 epochs trained. Note that the test points are plotted for visibility.

training, decreases on average, although the loss remains relatively high in the regions where the decision boundaries are. Note that no matter how overfit the model gets (epochs > 30), the location around the decision boundary have high loss. This means that if the loss is high, the model is uncertain, defining the decision boundaries. This makes intuitive sense - if an example is at or near the boundary, then it is not likely that the model's decision boundaries, while approximating the true boundaries, are imperfect and this example could be one or the other class. However, as discussed in Section 3.3.1, the problem is that to calculate the loss, the target labels are required, and one must resort to approximations.

3.5.3 Loss Approximations

Since models are trained to minimise the loss, the decision boundaries are inherently learnt using the loss function. However, since the loss requires the true class labels to compute, we can approximate the loss using the three methods described in Section 3.3.

Figure 3.6 shows the loss, entropy, information content, and probability gaps for the **four**-class classification problem illustrated in Fig. 3.1, except with a model (architecture in section 3.7.2) trained intentionally for 10 epochs for more visual clarity. This example illustrates that entropy, information content, and probability gaps are all able to approximate the loss and decision boundaries with reasonable effectiveness, since they maintain similar characteristics. For example, the entropy and information content are high at the boundaries (probability gaps are inverted; i.e., equal to 0 at the boundary) and generally follow similar contours to the loss plot. However, there are some subtle but important differences. The running four-class example was constructed to

highlight these differences.

Given a point, if the two highest predicted probabilities for this point are the same, then the point lies on the boundary between two of the four classes. Referring to Fig. 3.1, this occurs on the dark blue and orange, the orange and red, the red and light blue, and the light blue and dark blue boundaries. If the highest three predicted probabilities are the same, the point lies at the intersection of the decision boundaries of three classes, for example where the orange, red, and dark blue classes meet. Finally, if all of the predicted probabilities are the same, then the point lies at the intersection of all four of decision boundaries. Note that although the model which learnt the Fig. 3.1 boundaries does not have a point like this, this is still relevant, since this can occur in other models and other datasets with a large number of classes. In particular, this is important to mention since entropy (Eq. 3.3), information content (Eq. 3.4), and probability gaps (Eq. 3.5) differ in how they approach these scenarios. Note that in the discussion below, logits and predicted probabilities are sometimes used interchangeable, but in a way logical way.

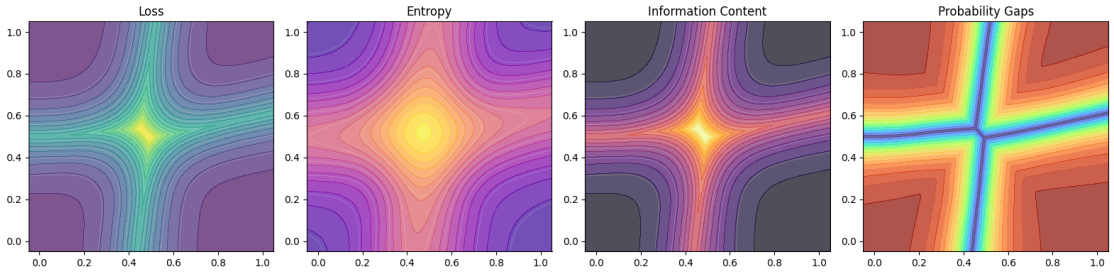

Figure 3.6: Loss, entropy, information content, probability gaps. A continuation from the Fig. 3.1 example.

Entropy measures the ”flatness” of the predicted-probability distribution - if the predicted probabilities are all the same, then entropy is maximised. However, if the two highest predicted probabilities are the same, say for classes a and b, then entropy can be increased if any of the other logits corresponding to, say, class c are increased. Therefore, increasing c’s logit translates to moving along the decision boundary between class a and b in a specific direction; towards the intersection of the boundaries of classes a, b, and c, where all logits (and predicted probabilities) will become equal. Therefore higher entropy values do not necessarily correspond to decision boundaries when there is more than two classes. If there are 1000 classes, as in ImageNet [44, 45], then high entropies are in the areas close to where a large number of decision boundaries meet, rather than areas between two decision boundaries. At the very least, entropy will be higher in the former than in the latter. Figure 3.6 directly demonstrates this phenomenon, where the highest entropy values are located in a circle-like area around the intersection of all boundaries. We also see lower, but still prominent entropy values, following the decision boundaries in a ”t” shape.

Note that if we remain on the boundary, but move away from the center (where multiple boundaries meet) entropy decreases even though the model should still remain uncertain. Therefore, while entropy is a good measure for understanding where very uncertain areas are, entropy values can be diluted when there are many classes or where points are far away from all but two classes.

On the other hand, the information content (Eq. 3.4) relies only on the maximum predicted probability, ignoring all others. This means the information content for $\mathbf{p} = [0.4, 0.4, 0.15, 0.05]$ and $\mathbf{p} = [0.4, 0.25, 0.25, 0.1]$ is identical. This is problematic, because in the former, the input point lies on the decision boundary (where there should be more uncertainty) than in the latter, where the point lies within the boundary of class a and is equidistance from classes b and c. Information content cannot distinguish between these two cases. In fact, we see this behaviour from the shape of the information content landscape in Figure 3.6, where (following one of the purple contours) we can move further away from decision boundaries while having the same information content. However, it is also important to note that the information content appears to be the closest approximation to the cross-entropy loss, as they have very similar landscapes. Nevertheless, because information content is unable to distinguish between being on the boundary and not, it is natural to consider the probability gaps (briefly discussed in 3.3.1), since they do not exhibit these problems.

As Eq. 3.5 shows, probability gaps are the difference between the highest and second highest predicted probability. This means they directly illustrate when a point is on a decision boundary; the probability gap is 0. This behaviour can directly be seen in Figure 3.6, since the probability gap landscape follows the decision boundaries, is not smooth, and has sharp corners, no matter how far a point is from a decision boundary. However, consider the two points with predicted probabilities $p_1 = [0.4, 0.4, 0.15, 0.05]$ and $p_2 = [0.25, 0.25, 0.25, 0.25]$. Both of these have probability gaps of 0, but the first is at the intersection of two decision boundaries and the second is at the intersection of four decision boundaries, and probability gaps are unable to differentiate between these two cases. However, clearly, there should be more uncertainty for the second point since for the first, the model thinks it may be one of two classes, but for the second, the model doesn't know at all.

Overall, all three discussed methods have strengths and limitations, although no single one alone is perfect. However, it possible to use one, two, or all three of these with minimal modifications to the following algorithms.

3.6 Method

The main method in this work involves identifying regions of uncertainty to specify which predictions will be uncertain or not. Assuming the data is independent and identically distributed (IID), choosing $\alpha = 0.05$ means making the assumption that the most extreme 5% of data in the calibration set is uncertain according to a chosen uncertainty measure g. Whether this value is appropriate depends on the task requirements, as there are inherent trade-offs by setting lower and higher values of α seen in Chapter 5. As always, it is important for the practitioner to test

whether a certain α value is appropriate. Some examples for how to choose an appropriate value are discussed in Chapter 5. Nevertheless, the general algorithm for identifying uncertain points is as follows:

Algorithm 1 Identify Uncertain Points

Input: classifier f, calibration loader, test loader, uncertainty measure g, quantile level α
Output: boolean vector of whether a point is uncertain or not
$\lambda \leftarrow$ compute_uncertainty_threshold$(f, \text{Calibration Loader}, g, \alpha)$
Initialise uncertain_points $\leftarrow$
for X in test loader **do**
    $p \leftarrow \text{SoftMax}(f(X))$
    $u \leftarrow g(p)$
    append $\mathbf{1}\{u \geq \lambda\}$ to uncertain_points
end for
uncertain_points = concatenate(uncertain_points)
return uncertain_points

where g is either entropy (Eq. 3.3), information content (Eq. 3.4), or probability gaps (Eq. 3.5), and $\mathbf{1}\{\cdot\}$ is the indicator function. Note that if g is the probability gap measure, then replace $\geq$ with $\leq$ in Algorithm 1, since low probability gaps indicate high uncertainty. α is a user-specified value such as 0.05, which is equivalent to selecting a threshold based on the 95th-percentile of data uncertainties. This α value is used to calculate the uncertainty threshold λ required in Algorithm 2. The algorithm for computing the uncertainty threshold is as follows:

Algorithm 2 Compute Uncertainty Threshold

Input: classifier f, calibration loader, uncertainty measure g, quantile level α
Output: uncertainty threshold λ
Initialise uncertainties $\leftarrow$
for X in calibration loader **do**
    $p \leftarrow \text{SoftMax}(f(X))$
    $u \leftarrow g(p)$
    append u to uncertainties
end for
$\lambda \leftarrow \text{quantile}(\text{uncertainties}, 1-\alpha, \text{“higher”})$
return λ

We can also, optionally, consider a stricter uncertainty criterion, where a point is uncertain if it entropy-uncertain, information-content-uncertain, or probability-gap-uncertain. This way, we can make use of the benefits of any of the approaches, since they may have different behaviour

depending on the task - as discussed in Section 3.5.3. In this combined approach, we determine which points are uncertain using Algorithm 3:

Algorithm 3 Identify Uncertain Points - Combined

Input: classifier f, test loader, uncertainty measure g, quantile level α
Output: boolean vector of whether a point is uncertain or not
$\lambda_1 \leftarrow$ compute_uncertainty_threshold(f, Calibration Loader, entropy, α)
$\lambda_2 \leftarrow$ compute_uncertainty_threshold(f, Calibration Loader, information_content, α)
$\lambda_3 \leftarrow$ compute_uncertainty_threshold(f, Calibration Loader, probability_gap, α)
Initialise uncertain_points $\leftarrow$
for X in test loader **do**
    $p \leftarrow$ SoftMax($f(X)$)
    $u_1 \leftarrow$ entropy(p)
    $u_2 \leftarrow$ information_content(p)
    $u_3 \leftarrow$ probability_gap(p)
    $u \leftarrow (u_1 \geq \lambda_1) \vee (u_2 \geq \lambda_2) \vee (u_3 \leq \lambda_3)$
    append u to uncertain_points
end for
uncertain_points = concatenate(uncertain_points)
return uncertain_points

As mentioned, Algorithm 3 is a stricter version of Algorithm 1, and is more suitable in tasks/industries where it is required that uncertain points are definitely labelled as uncertain (minimise false-negatives) at the cost of more "certain" points labelled as uncertain (higher false-positive), such as healthcare. Being the key mechanism in setting the cut-off threshold, α plays a crucial role here. Its impact is discussed further in Section 4.1 and the practical implications in Chapter 5.

It is important to note that the Algorithm 3 can be trivially modified to be less strict, if the task requirements allow, by changing the logical $\vee$ to logical $\wedge$. In this case, the algorithm will only categorise points as highly uncertain if all three of the measures flagged that point as highly uncertain. In a sense, this is actually a stricter criterion for high uncertainty as less points will be flagged, although this depends on the task at hand and the agreement between the three measures. The implications of this are reported in Section 4.1.

Also, we note that since the chosen uncertainty measures essentially measure uncertainty by approximating the loss to some respect and the shape of the boundaries strongly depends on the model architecture, dataset, the initial random initialisation, training time, number of points, this method is unable to measure any uncertainty in the data itself, since the uncertainty measures in Section 3.3 rely on the model and consequently any changes to the mentioned factors will lead to different values of uncertainty for the same test point. However, this is not considered a problem in the current work, as we are interested in determining which test test points live in the highly

uncertain regions, given the model.

Finally, the proposed methodology requires minimal computational demands in the order $O(nC)$ where n is the number of examples in the calibration and test sets and C is the number of classes. The memory requirements are $O(1)$ since the proposed method only requires the storage of the chosen uncertainty thresholds. The method is easy to implement post-hoc and is able to be implemented without using any outside packages, although it is much easier with PyTorch or NumPy etc.

3.7 Experimental Setup

This work uses Scikit-Learn [46], PyTorch [47], and the datasets described below. Since this work is interested in exploring model quality rather than tuning hyperparameters, it is deemed that typical model selection strategies such as cross-validation or a separate validation set are unnecessary, since all experiments make use only of the training set.

All models are trained on the training set and adversarial examples generated from the training set only. The models nor attacks see the test set directly nor indirectly. However, models are evaluated for their performance using the test set and whether a model is overfit is determined by the test set.

The results in Chapter 4 were obtained using an Apple MacBook Pro with an M4 Max GPU. The code can be found on GitHub (all model implementations can be found under `src/models.py`).

For most results in this work, pre-trained models are used so the test set is split into a calibration and test splits. For trained models, the used dataset is split into training, calibration, and testing splits using Scikit-Learn [46]. For performance metrics, such as those in the case studies, the data is split only into training and testing splits; a model is trained on the training data and evaluated on the test data. This is done to give an example of how the model would perform *without* the current methodology, i.e., if normal training procedures were followed. Please note that unless otherwise specified, all datasets are scaled using [0, 1]-scaling (also called Min-Max scaling). The batch size used is always 128.

3.7.1 Chessboard

The chessboard dataset is a simulated rectangular two-dimensional "board" of blocks/clusters. The dataset is created by calculating the centre of each block, generating points from Gaussian distribution and scaling the blocks to an appropriate size. Note that, due to the datasets construction, post-creating scaling is not required since the dataset is already scaled. Simulation examples are shown in Figure 3.7. The `make_chessboard` function can be found in this work's repository in `src/datasets.py`.

This simulated dataset is interesting to explore as, even with a small datset size, this dataset is difficult for classifiers to learn effective decision boundaries without over-fitting. In Figure 3.7, the first two plots clearly require straight diagonal boundaries, however, it is much easier for classifiers

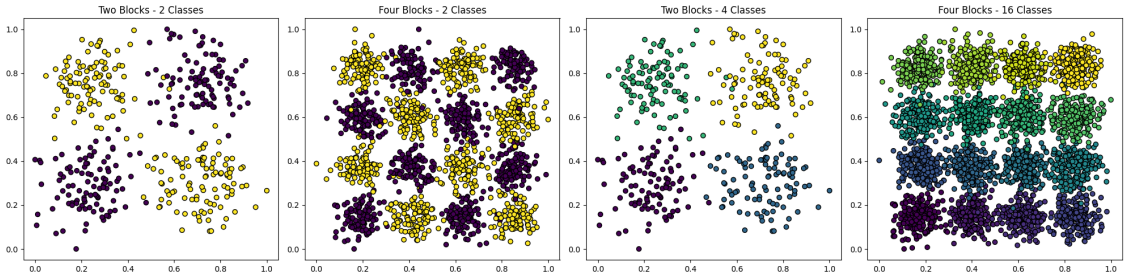

Figure 3.7: Three examples of simulated chessboard data.

to learn boundaries that "wrap around" the clusters even if the model is underfit. For the third plot, there are four classes, just like in Figure 3.1, which is actually an easier classification task than the previous two plots, since a model just needs to learn a linear "t" shape. Similarly, the last plot shows the dataset with sixteen distinct classes. This structure is also easier for a model to learn than the second plot due to its highly linear nature; a model just needs to learn a grid. The fourth dataset is used to illustrate the subtleties discussed in Section 3.5.3. This is done in Section 4.2.

It is also important to note that the clusters, while generally being reasonably distinct, often have some overlap between the classes, making a model's learning more difficult. Often, it is not possible to achieve 100% training accuracy due to this reason, although classifiers, provided they have enough capacity, can get reasonably close. However, the benefit with all of these datasets is that they are easily visualised, we can plot the decision boundaries and various contour plots.

Unless otherwise specified, results are reported on the 4x4 chessboard with two classes, although the 16-class version of the 4x4 dataset is also used. If this is the case, it is clearly stated. It is also important to note that the training, calibration, and test splits are stratified by class, to ensure an even class distribution.

For the 16-block dataset seen in the middle plot in Figure 3.7, the following architecture is used. There are four linear layers with weights of size 2x1024, 1024x512, 512x256, 256xC, where $C = 2$ if there are two classes or $C = 16$ for the 16-class alternative. Following each layer, a ReLU activation is used. All models were trained using the cross-entropy loss (Eq. 3.2), with the Adam optimizer (with default parameters) and a learning rate of 0.001. However, the length of time a model is trained varies depending on the task.

For the two-class middle plot in Figure 3.7, the model was trained for 30 epochs since learning diagonal decision boundaries is harder than learning vertical and horizontal ones. In contrast, for the 16-class version of the same dataset (where every cluster has its own class), the model was trained for 10 epochs.

3.7.2 SpamBase

The SpamBase dataset [48] is a real-world spam classification task. There are 4,140 examples in total, with 1,630 spam and 2,510 non-spam. There are 57 numeric features and all are engineered variables. Some examples of the features are the character frequencies of certain symbols (such as !, $, and #), words, or longest running sequence of capital letters etc. See [48] for a full description. Note that *no preprocessing or data exploration* was done in this work except splitting the data and scaling it, as described above.

The SpamBase model architecture is as follows. There were three linear layers with weight tensors of size 57x128, 128x32, and 32x2. The linear layers were separated by ReLU and the cross entropy loss was used. Adam was used as an optimizer (with all default parameters) and a learning rate of 0.01. The model was trained for 30 epochs.

3.7.3 MNIST

MNIST is a famous dataset of 70,000 grayscale examples of handwritten digits. Containing digits from 0 to 9 with each image being of size 28×28, width and height respectively. The MNIST dataset is popular because of its simplicity, balanced class distribution, and ease of training. It is quite trivial to obtain a test accuracy of 98%+ on MNIST. Since the images are grayscale, there is no colour and consequently there is only one colour dimension. The dataset is pre-split into 60,000 training points and 10,000 test points. Some examples can be seen in the top row of Fig. 3.2. No pre-processing was done except convert the PIL images to tensors for use with PyTorch. MNIST images are already normalised with pixel values within [0, 1].

The MNIST model was inspired by [49], and has two convolution layers. The first convolution layer takes 1 channel and outputs 6. The second takes 6 and outputs 16. Both convolution layers had a kernel size of 5, stride of 1, and no padding. Each convolution layer is followed by 2D batch normalisation [50], ReLU [51], and 2D max-pooling with kernel size of 2, stride 2, and no padding. The output of this is then flattened and passed through two fully connected layers with weights 256x120 and 120x84, respectively, which are separated by 1D batch normalisation followed by ReLU. Finally, the output is passed through a fully connected layer with weights of size 84x10. The model trained for 3 epochs with cross-entropy loss, Adam optimizer, with learning rate of 0.001.

3.7.4 CIFAR-10

The CIFAR-10 dataset [52] is an RGB-coloured dataset composed of 32x32 images. There are 50,000 images in the training set and 10,000 in the test set. For this work, we split the test dataset into two to obtain a validation and test set, both containing 5,000 images.

For the CIFAR-10 dataset, we use a pre-trained ResNet-56 model with weights and architecture provided by [53]. We do not discuss the architecture here as this is beyond the scope of this work.

3.7.5 ImageNet

ImageNet is a large and famous dataset [44, 45]. The dataset used in this work is the 2012 dataset with 1 million training, 50,000 validation, and 50,000 test images with 1,000 classes. The images are in colour and are of size 224x224.

This specific version was used as pre-trained model weights for a ResNet-50 [54] are available in PyTorch, and therefore no model training was required. Thus, the training data was not used, only the validation and test data.

This work used two different models. The first is a pre-trained ResNet-50 described in [54], loaded with the default weights from TorchVision [55]. For the full architecture, see [54]. The second model uses the `yolov8n-cls` model [56] with weights retrieved from [57]. Discussing the architectures here is beyond the scope of this work and we refer the reader to the provided citations.

Chapter 4

Results

This section discusses the results and findings of the method defined in Section 3.6. Note that due to there being no inherent ground-truth for uncertain points, the effectiveness of these methods is best demonstrated in the following Case Studies in Chapters 5, 6, and 7.

4.1 Quantile Method - Binary

Figure 4.1 shows the decision boundaries for the two-class 4x4 chessboard discussed above (for details see 3.7.2). The entropy threshold λ was computed using Algorithm 2 and the uncertain points identified using Algorithm 1.

Note that the alphas and computed thresholds are in each plot's title and that the landscape colours here are irrelevant except to distinguish between the two sides of a decision boundary. The uncertainty measure here is entropy.

We observe that, as α is increased, more points are coloured red and the corresponding α-level threshold decreases. This is not surprising, since a higher α will consider more points as having higher entropy and higher uncertainty and therefore the acceptance threshold (acceptance meaning whether a point is considered as having high uncertainty) decreases.

We also observe that points around the decision boundaries are selected indicating that this method works for identifying points that live in uncertain regions; i.e., near the boundaries. However, the position of the boundaries themselves can make some points appear more uncertain. For example, for $\alpha = 0.3$, consider the third cluster in the second row (from the top). This cluster should be blue, since the region underneath it is blue. Note that the colour of the points here in-

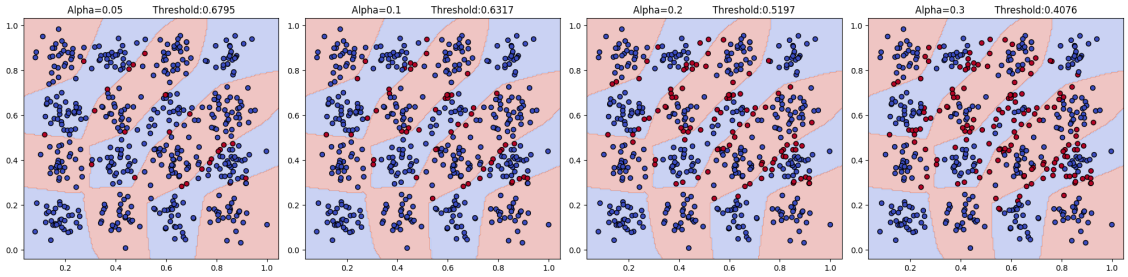

Figure 4.1: Decision boundaries with test points coloured by low uncertainty (blue) or high uncertainty (red) for various alphas and corresponding thresholds. The uncertainty measure is entropy.

Table 4.1: Pairwise agreement matrix between the three uncertainty measures for the binary case. Each value represents the fraction of points where both methods agree on which points have high uncertainty or low uncertainty.

	Entropy	**Information Content**	**Probability Gaps**
Entropy	-	-	-
Information Content	1.0	-	-
Probability Gaps	1.0	1.0	-

dicates high uncertainty (red) or low uncertainty (blue) and **not** the class labels. Nevertheless, we see that this cluster is squashed between the two decision boundaries on either side. Because the decision boundaries are so close, this cluster has higher uncertainty than the other clusters, which is indicated by it having proportionally more red points than the other clusters. It is possible that due to the training, calibration, and test splits, this cluster, by chance, was represented less during training or had lower variance than the other clusters, and consequently was considered less important, with the learnt boundaries squashing it slightly. Regardless, this is a flaw in the model design and training process, which the proposed method highlights well since points in this region have higher uncertainty than in others according to entropy. However, while not shown here for space reasons, similar figures to Figure 4.1 can be recreated using the two other uncertainty measures such as information content and probability gaps. The question is, since all measures are measuring uncertainty, just in different ways, by how much do these measures agree with each other?

Table 4.1 shows the agreement matrix between the three measures. For example, if using entropy we get that the point is highly uncertain, do information content and probability gap measures also calculate that the example is highly uncertain? We observe that all methods agree with each other exactly. This is because the results provided so far are for a two-class classification problem using entropy as the uncertainty measure. The peculiar result of this is that all of the measures agree with each other. It is not shown here (for space reasons) but setting α to any value in [0, 1] results in the exact same agreement matrix; i.e., all measures agree with each other exactly, regardless of the α. This is because to appreciate the differences between these methods, we must go to the multi-class case.

4.2 Quantile Method - Multi-Class

This example follows the same 16-class dataset shown in the rightmost plot in Figure 3.7, except with a single $\alpha = 0.1$. As can be seen in Table 4.2, while the methods mostly agree, they no longer perfectly agree, since the uncertainty measures are now picking up on the various subtleties discussed in Section 3.5.3. However, we do observe that information content and probability gaps agree on significantly more points than with entropy. A visual demonstration is presented below.

Table 4.2: Pairwise agreement matrix between the three uncertainty measures for the multi-class case. Each value represents the fraction of points where both methods agree on which points have high uncertainty or low uncertainty.

	Entropy	**Information Content**	**Probability Gaps**
Entropy	-	-	-
Information Content	0.9725	-	-
Probability Gaps	0.9675	0.9950	-

Figure 4.2 shows five plots generated with $\alpha = 0.1$ for the 16-class chessboard dataset. The leftmost plot shows the test data and the boundaries learnt from the training data. This serves as the reference point for the decision boundaries, although some boundaries are not clearly visible due to the large number of classes. The middle three plots show the entropy, information content, and probability gap contours, with only the points identified by each respective measure shown. Finally, the rightmost plot shows the test data coloured red if a point is identified by all three measures, blue if the point is identified by at least one, but not all three, measure, and transparent otherwise.

Starting with the rightmost plot in Figure 4.2, we see the points which have high uncertainty. The blue points correspond to the disagreement seen in Table 4.2 and, for this example, correspond mostly to the disagreement between entropy and the other two measures. This is due to entropy measuring the flatness of the class distribution. At the intersection of multiple boundaries, the predicted probabilities will be almost identical which corresponds to high entropy. In the entropy plot, we can see this effect directly since the areas around the intersections of the decision boundaries have the highest entropy values; higher than the entropy for the boundaries separating just two classes. In fact, information content, albeit to a smaller extent, is able to measure the same since the locations of multiple boundary intersections also have higher information content values than the two-class boundaries. However, information content generally has smaller values with more extreme peaks around the decision boundaries, whereas entropy has wider areas with higher values, indicating that it considers wider areas as regions of uncertainty. This is clearly seen with the two blue points at the top left of the rightmost plot in Figure 4.2. These points are close to the intersection of multiple boundaries, and therefore *should* be highly uncertain although only entropy flags them as uncertain. Note that for this example, probability gaps have almost perfect agreement with information content, so its effects are not discussed here.

Information content and probability gaps do disagree and this is illustrated in Table 4.3, which shows the agreement matrix for the pre-trained ResNet-50 model on the ImageNet dataset ($\alpha = 0.1$), and while, unfortunately it is impossible to visualise the decision boundaries for this model, we are still able to gain some intuition in this 1000-class case. We observe that the information content and probability gaps here have an agreement of just over 90%, which most likely is due to the demonstrated behaviour of information content in Section 3.5.3. We also see that entropy has

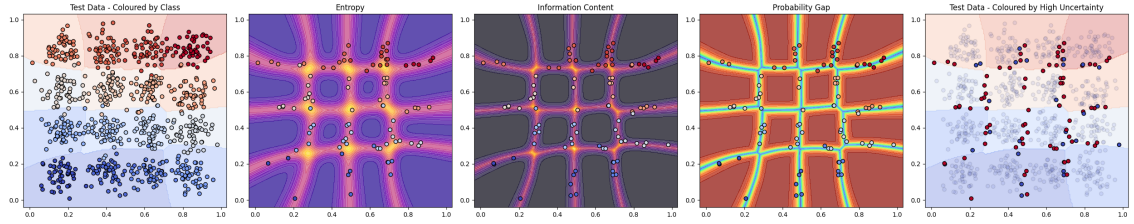

Figure 4.2: Left to right: 1. The dataset. 2. High uncertainty points identified by entropy $\geq$ entropy threshold. 3. High uncertainty points identified by information content $\geq$ information content threshold. 4. High uncertainty points identified by probability gap $\leq$ probability gap threshold. 5. Test points identified as uncertain by all measures are coloured red, points identified by at most two measures are coloured blue. All other points (transparent) have low uncertainty.

the highest disagreement with probability gaps, most likely because entropy and probability gaps do not inherently measure the same thing; entropy measures how peaked the predicted probability distribution is, while probability gaps essentially measure the proximity to the decision boundary. This means that the model may be uncertain about some test examples because they may be near multiple decision boundaries (the model is uncertain which class this is) whereas probability gaps fail to consider these points as uncertain as probability gaps are only interested in the distance of the a point from *a* decision boundary, paying no heed to how many boundaries are in close proximity.

Table 4.3: Pairwise agreement matrix between the three uncertainty measures ImageNet. Each value represents the fraction of points where both methods agree on which points have high uncertainty or low uncertainty.

	Entropy	**Information Content**	**Probability Gaps**
Entropy	-	-	-
Information Content	0.9339	-	-
Probability Gaps	0.8592	0.9004	-

Nevertheless, there is still reasonably high agreement between these three methods, and while each have their downsides, all are useful measures of uncertainty in their own right. How these can be used further, is discussed next.

Chapter 5

Case Study 1: Model Performance

A straightforward application of the proposed method is to abstain from prediction when points are highly uncertain. The idea being that points that are highly uncertain are likely to have higher misclassification rates than points with low uncertainty. The approach involves using Algorithm 2 to find the threshold λ on the calibration set. Then, following Algorithm 1, if a point has an uncertainty measure more extreme than λ, we abstain from making the prediction or classify the point as "unknown" or similar. This is analogous to abstaining from prediction when a model's confidence in the most likely class is only 10%.

The results of this approach for all uncertainty measures is reported in Table 5.1 for $\alpha = 0.1$. The results presented show the accuracy on the non-uncertain points followed by the accuracy on the uncertain points in parentheses. Note that the last two columns correspond to using the method with the logical OR and logical AND, respectively, as discussed in Section 3.6. The former considers a point as highly uncertain if any of the three uncertainty measures flagged it and the latter only if all three flagged the point. The maximum row-wise accuracies are highlighted in bold and the table is split by binary and multi-class tasks.

Table 5.1 demonstrates that abstaining from predictions on the identified uncertain points leads to an accuracy improvement across all datasets and across all models, by at least a few percentage points. As discussed in Section 4.1, all methods have 100% agreement for binary tasks, and all methods outperform the baseline performance. For multi-class tasks, we observe that the best-performing method, for all datasets and models, was the logical OR combination of all three uncertain-point identification methods. This implies that all three methods are able to meaning-

Table 5.1: Model accuracy after abstention with $\alpha = 0.1$ on low(high)-uncertainty points. The highest performance is bolded.

	Baseline	**Entropy**	**IC**	**PG**	**At Least One**	**All Three**
Binary Classification						
Chessboard	85.62%	**90.14%**(34.62%)				
SpamBase	93.14%	**96.16%**(64.22%)				
Multi-Class Classification						
Chessboard (16 class)	92.68%	96.50%(56.41%)	96.50%(56.41%)	96.73%(58.14%)	**97.24%**(58.33%)	96.01%(55.88%)
CIFAR10 (ResNet-56)	94.24%	97.69%(60.56%)	97.66%(60.94%)	97.68%(61.23%)	**97.76**%(61.60%)	97.60%(60.13%)
ImageNet (ResNet-50)	78.85%	81.78%(52.28%)	83.73%(34.07%)	83.70%(33.92%)	**86.17%**(45.62%)	80.40%(23.80%)
ImageNet (YOLOv8n-cls)	66.08%	70.92%(22.35%)	71.36%(17.51%)	70.50%(24.42%)	**74.83%**(24.76%)	67.64%(13.01%)

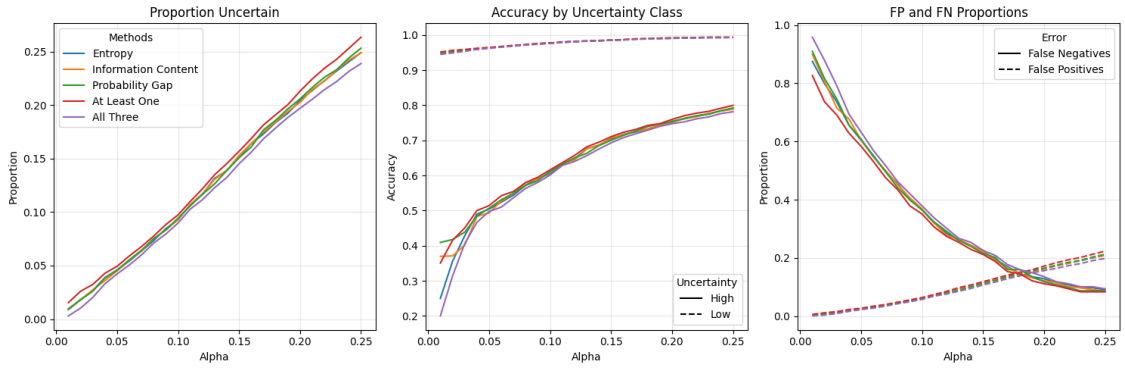

Figure 5.1: Left to right: 1. Proportion of examples identified as uncertain. 2. Accuracy by method and uncertainty classification. 3. False positive and false negative rates. All by α and method. Results are from the ResNet-56 model on the CIFAR-10 dataset.

fully contribute to improving performance and the combination of all three is superior to using any one of them individually. We also observe that abstaining from prediction when all methods agree may be too strict, as the performance improvement for the ImageNet dataset is modest for both models. However, this warrants further exploration. Nevertheless, all methods show promising results with the highest percentage change being for the pre-trained YOLO model on the ImageNet dataset with a $\sim 9\%$ jump in accuracy, followed by an $\sim 8\%$ accuracy improvement for the ResNet-50 model. However, these performance figures depend on the α-level, with higher values leading to more points being added to the high-uncertainty category. This is demonstrated on the CIFAR-10 dataset in Figure 5.1 for $\alpha \in (0, 0.25]$.

The leftmost plot of Figure 5.1 shows, no matter what method is used, the proportion of points identified as uncertain is almost perfectly correlated with α. This is unsurprising, since α is the quantile used to find the threshold λ, assuming the calibration and test data are IID, it is expected that the test data should have a similar distribution of uncertainty no matter which measure is used. This means that if $\alpha = 0.2$, then roughly 20% of examples are excluded. However, also unsurprisingly, we observe that the proportion selected is highest for the for the logical OR method (At Least One) and lowest for the logical AND method (All Three).

The middle plot in Figure 5.1 shows the accuracy obtained for all methods by α, with accuracy calculated separately for the low-uncertainty and high-uncertainty points. For the low-uncertainty points, the accuracy increases as α increases as higher α leads to more points being flagged as uncertain. Counterintuitively, the same effect is seen for the high-uncertainty points. This is because the identification process initially successfully identifies the poorly-performing points, but as the number of false positive points (correct prediction but identified as uncertain) increases, the accuracy "degrades" and increases.

The rightmost plot of Figure 5.1 show the false positive and false negative rates for all methods. The former is the proportion of correct predictions labelled as highly-uncertain and the latter is the proportion of incorrect predictions labelled as having low-uncertainty. As α increases, the number of false positives increases, although the proportion always remains below the level of α. This is because as the uncertain regions increase in size with α, more points are flagged as uncertain. Likewise, for the false negatives the number of missed points decreases. We observe that these behaviours are consistent across all methods, the results of which are almost identical. From these data, it is possible to determine an α level which is appropriate for the task at hand. For example, we can minimise the sum of both proportions, or minimise the number of false positives with the constraint that the false negative rate must be less than 5%, and so on. It is important to note that since the performance of this model is roughly 94% (Table 5.1), there are not many misclassified points which inflates the false negative proportions. For example, to identify 60% of the incorrectly-classified points it is enough to set $\alpha = 0.1$ (CIFAR-10). However, we leave the further exploration of α-tuning for future work.

Chapter 6

Case Study 2: Adversarial Detection

Another application of the proposed method is the ability to abstain from prediction on adversarial examples. Since the goal of adversarial examples is to find a minimal perturbation, adversarial examples will typically move to areas of high uncertainty and aim to cross the decision boundary. Since an example aims to be minimal and must resemble the original image, it is unable to venture deep into the low-uncertainty region of another class. This means the example remains in high-uncertainty regions and therefore using the proposed method it is possible to detect them and abstain from prediction. Note that if adversarial examples are mixed in with non-adversarial data, the proposed method is not designed for identifying which are adversarial and which are not.

Figure 6.1 shows the proportion of adversarial examples identified for the three attacks described in Section 3.4, generated with $\epsilon = 0.03$. We observe that as α increases, the proportion of attacks identified increases regardless of the attack. In fact, for all methods we see that, with $\alpha \geq 0.04$, the identification rates across all attacks are quite high, indicating that most (equivalent to the corresponding proportion detected) adversarial examples are found in regions of high uncertainty. This indicates that the proposed methods successfully identify regions of high uncertainty, although the effectiveness varies depending on the method used.

In general, we see that information content is the most effective uncertainty measure for adversarial identification, followed by entropy, and finally by probability gaps. Information content achieves 100% identification at $\alpha = 0.6$ and entropy at $\alpha = 0.1$. The method does not identify all adversarial examples using the probability gap method, although the identification rates reach 80% at $\alpha \geq 0.6$. This result most likely depends on the perturbation size, ϵ, which corresponds to the allowed deviation of an adversarial example from the original non-perturbed example. This means that a larger ϵ leads to a larger movement in the feature space, leading to stronger adversarial

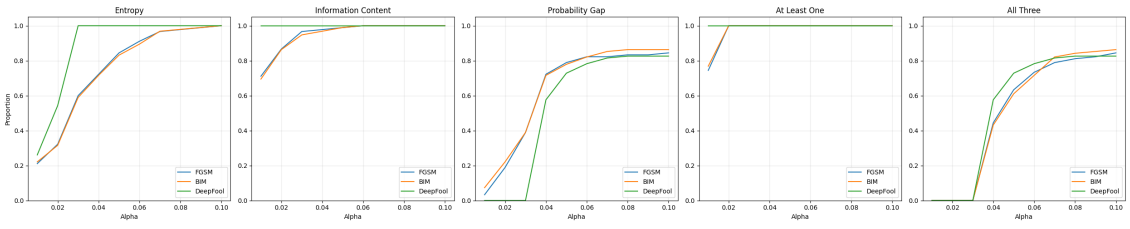

Figure 6.1: The proportion of adversarial examples identified by identification measure and adversarial attack on CIFAR-10 for varying levels of α.

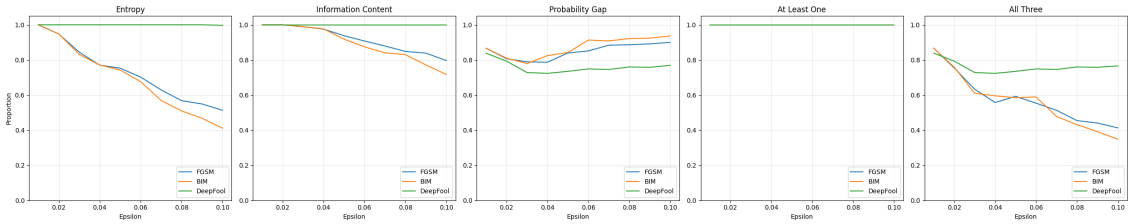

Figure 6.2: The proportion of adversarial examples identified by identification measure and adversarial attack on CIFAR-10 for varying levels of ϵ.

examples.

Figure 6.2 shows the same proportion of identified adversarial examples to Figure 6.1, except with ϵ now varied. We see that as ϵ increases, the identification rates for entropy and information content reduce (except for DeepFool, which remains at 100%), while for probability gaps the identification remains roughly constant. For these examples to be mis-identified, this means that their values of entropy and information content decrease as ϵ increases. However, since the probability gap measure remains roughly constant, this means the identified examples remain (roughly) as close to the decision boundaries, no matter the level of ϵ. This means that as the (FGSM and BIM) attacks deviate further from their original location in space, they move away from locations of high uncertainty (implied by the entropy/information content results), but continue to roughly follow the decision boundary (implied by the probability gaps).

Considering both Figures 6.1 and 6.2, we see that although entropy and information content have better adversarial identification for $\epsilon = 0.03$, their performance eventually degrades (for FGSM and BIM) whereas probability gaps remain at reasonably good performance. This motivates further the use of all three measures in combination, where we see the logical OR method in Figure 6.2 having perfect identification regardless of ϵ. Likewise, in Figure 6.1, this method also shows the best performance across all levels of α, where even for $\alpha = 0.1$, almost 80% of adversarial examples are detected. For $\alpha \geq 0.2$, the identification rate rises to 100% for all attacks. These results strongly motivate the use of Algorithm 3 for identifying adversarial attacks since this method is fast and easy to implement. An important question to ask is regarding the trade-off in using the logical OR method, since it, inherently, selects more points than the other methods. However, the answer to this can be found in the left plot in Figure 5.1 in Chapter 5, which showed that the number of examples the logical OR method selects is only marginally higher than each method alone. Thus, the sacrifice is considered well worth it.

Another motivation for the use of the logical OR is that, in Figure 6.1 for a set α, entropy and information content are better able to identify DeepFool attacks, whereas probability gaps underperform on DeepFool more than the other attacks. This, in itself, illustrates the differences between how DeepFool and the other attacks work, where DeepFool takes adversarial examples closer to

regions where there is more ambiguity (more boundary intersections) but further past the boundary, whereas the FGSM and BIM take the example closer to the closest decision boundary, rather than regions of higher multi-class uncertainty. However, the inherent strengths and weaknesses in each of the uncertainty measures motivates the use of them simultaneously.

Chapter 7

Case Study 3: Misclassification Prediction

Algorithm 1 uses a threshold λ, computed using Algorithm 2, which is computed from the calibration set. However, a natural extension to this is to train a machine learning model to learn to classify misclassification from the model uncertainties instead of through an α-level threshold as done in Chapter 5. This process is illustrated in Algorithm 4.

Algorithm 4 Get Uncertain Points

Input: trained classifier f, new classifier h, calibration data (X, y), test data (X', y')
Output: uncertain_points
calibration_logits $\leftarrow f(X)$
uncertainty_features $\leftarrow$ get_uncertainty_features(calibration_logits)
calibration_predictions $\leftarrow$ argmax(calibration_logits)
misclassification_labels $\leftarrow$ not_equal(calibration_predictions, y)
$h \leftarrow$ train(h, uncertainty_features, misclassification_labels)
test_logits $\leftarrow f(X')$
test_uncertainty_features $\leftarrow$ get_uncertainty_features(test_logits)
uncertain_points $\leftarrow$ predict(h, test_uncertainty_features)
return uncertain_points

where the function for calculating the uncertainty features is given in Function 1. Note that Algorithm 4 returns an array of 1s or 0s, corresponding to predictions of whether the classifier f will misclassify the test points. The benefits of this approach is that besides requiring a calibration set, the computational costs are minimal and this procedure is also done in an post-hoc manner. Likewise, there is no constraint on the new classifier h, which can be as simple as a decision tree or as complicated as a neural network.

Function 1 Get Uncertainty Features

Input: logits z
Output: uncertainty_features
$p \leftarrow$ Softmax(z)
entropy $\leftarrow$ compute_entropy(p)
information_content $\leftarrow$ compute_information_content(p)
probability_gaps $\leftarrow$ compute_probability_gaps(p)
uncertainty_features $\leftarrow$ stack([entropy, information_content, probability_gaps])
return uncertainty_features

Table 7.1: Performance results of misclassification prediction of a meta-learner h.

	Accuracy	**Recall**	**Precision**	**# Positive (Negative)**
Binary Classification				
Chessboard	66.56%	61.60%	56.62%	123 (197)
SpamBase	72.28%	79.41%	15.04%	68 (795)
Multi-Class Classification				
Chessboard (16 class)	82.20%	76.67%	25.84%	40 (370)
CIFAR10 (ResNet-56)	84.52%	83.68%	24.9%	287 (4,713)
ImageNet (ResNet-50)	79.21%	75.57%	50.58%	5,420 (19,580)
ImageNet (YOLOv8n-cls)	76.48%	76.36%	62.55%	8,541 (16,459)

This case study, uses `BalancedRandomForestClassifier` from Imbalanced-Learn [58] for the classifier h. No hyper-parameter tuning is done and all default parameters are used. The performance results for h are computed based on whether the uncertain points match the misclassification of f on the test set. The obtained results are reported in Table 7.1 which shows the accuracy, recall, precision, and the number of positive (misclassified) and negative (correctly classified) instances, for different models f. One problem with this approach is that for the high-performing models (ResNet-56 on CIFAR-10 and SpamBase), imbalanced training data is produced for h as these models have low misclassification rates on the calibration set.

Since h's goal is to predict whether points will be misclassified by the trained classifier f, the most important metrics are recall, and precision, due to dataset imbalance. This is especially important for the high-performing models such as ResNet-56 (on CIFAR-10) and SpamBase, which have low misclassification rates on the calibration set and therefore produce imbalanced training data for h. From Table 7.1, we see that the meta-learner h identifies most misclassified points due to relatively high recall for most classifiers f. However, there are a few instances with very low

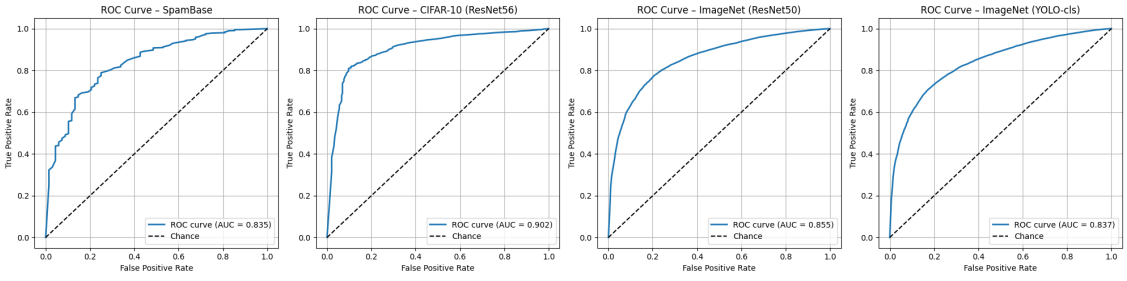

Figure 7.1: ROC-AUC Curves for four models: SpamBase, ResNet56, Resnet50, and YOLOv8n-cls.

precision indicating that a large number of points are flagged as uncertain (will be misclassified) although they are not. These results highlight the severe training data imbalance since h performs better on a more-balanced dataset from a model such as YOLOv8n-cls.

Finally, Figure 7.1 shows the ROC-AUC curves for h trained on the four models separately. We see that the AUC is generally very good (80-90%), particularly for ResNet-56 (second plot) which indicates that it is possible to achieve much better results than those found in Table 7.1 by optimising the classification threshold. The strong AUC scores support the proposition that it is possible to learn to predict model's uncertainty using entropy, information content, and probability gaps. Since the point of this case study is to provide a proof of concept, we leave the further exploration of this to future work which may consider hyper-parameter tuning, using more advanced meta-learning classifiers for h, or data augmentation with adversarial examples and so on.

Chapter 8

Conclusions and Future Work

The main contributions of this work include a novel post-hoc methodology to isolate regions of uncertainty with demonstrated practical applications in three case studies. This work aimed to provide another method for the *reliability* section of the AD framework with the main benefits of the proposed approach being the extremely low computational and memory requirements, as well as ease of implementation.

Overall, case study 1 demonstrated the effectiveness of the proposed method by showing that models performed significantly worse on the points inside the isolated regions of uncertainty and their performance was improved for the low-uncertainty points. Case study 2 demonstrated that adversarial examples are significantly more likely to lie in the identified uncertainty regions and that the proposed methods correctly identified them as uncertain, with the differences between the proposed methods and impacts of parameters discussed in detail. Case study 3 demonstrated a natural extension to the proposed methods, showing that it was possible to train a meta classifier on uncertainty features to predict misclassification with promising AUROC scores, finding that this process was more effective for models with lower accuracy, as these models provide more data for the meta learner to learn from. Overall, these case studies showed that the proposed method was effective at isolating regions of uncertainty through the properties of points within, and outside of, those regions. Therefore, as the performance of the proposed method is established, we present directions for future work next.

The current work only considered three adversarial attacks, but more powerful and diverse attacks exist, for example the Carlini & Wagner attack in [40] etc. Therefore, future work may consider exploring more attacks to determine the effectiveness of the proposed method across a broader and more varied range of adversarial examples. It is also possible to explore ways to improve the meta learner from Chapter 7 or improve on the data imbalance since this would make the meta learner a powerful tool for predicting when a model's performance will be severely impacted. Likewise, the current work did not look into out-of-distribution detection generally besides the adversarial examples, which future work may consider. It will be also interesting to explore the impact of calibration set size on the method's performance, since it is likely that a smaller calibration set will suffice. However, it is possible that the calibration set can be avoided if the same results can be demonstrated with uncertainty thresholds obtained from the training set. This may bias the method, but for practical purposes it may be sufficient especially when a calibration set is impossible to obtain.

The current work aimed to assess a model's predictive ability in performing reliable predictions, however, the proposed method makes no assertions about a model's quality. Therefore future work may focus on establishing a metric to evaluate a model's quality based on its predictive un-

certainty. This may be possible since a model's uncertainty is high across the domain of the data when the model is under-fit. In contrast, when the model is over-fit, the uncertainty landscape becomes more jagged, with a model becoming very confident within the domain of the data, with very narrow uncertainty regions around the decision boundaries.

An interesting extension is to explore the multi-label image detection problem. Models such as YOLOv8 produce a probability for each class, indicating the likelihood a class is present in the image. A cut-off threshold is then used to indicate whether the class exists in the image or not as in logistic regression. This is essentially a binary classification problem for many classes and therefore uncertainty metrics can be calculated by class across a calibration set to find the per-class uncertainty thresholds. Classes may then be considered as having high uncertainty if their corresponding uncertainty measure is above this threshold.

Overall, the current work provides a computationally-low and easy to implement procedure for isolating uncertainty regions where a model's predictions are likely to be erroneous. The benefits of this include abstention of uncertain points, helping make machine learning models more reliable and trustworthy as well as abstaining on points which may be adversarial. For example, the proposed methodology may help in abstaining from predictions on examples similar to the one shown in Figure 1.1 where the control can be given back to the driver. Promising results in Chapter 7 have shown that these uncertainty regions can also be learnt by a meta-learner, helping practitioners understand the capabilities of their models, particularly in difficult domains where high performance is almost impossible. We hope that this work serves as a foundation for future research in isolating regions of uncertainty and helps contribute to deployment of safe, trustworthy, and reliable machine learning systems.

Acknowledgements

First of all I would like to thank my partner Ashleigh for her support and patience throughout the many hours of testing and writing that went into this dissertation. I am deeply grateful Joerg Wicker for his guidance, encouragement, and consistent support. My sincere thanks to Tobias O. for providing invaluable ideas, feedback, and proofreading, particularly during the more challenging stages of this work.

I would also like to thank Andy and Matt for granting me the NIWA/Fisheries NZ scholarship, as well as Arnaud, Alan, Milan, and James for their support throughout this process. Finally, I with to express my gratitude to the countless friends and colleagues who contributed their time, discussions, and working ideas towards this dissertation. I am very grateful to you all. Thank you.

References

[1] Mark Elkins. Research note: Interpreting confidence intervals. *Journal of Physiotherapy*, 70(4):319–323, 2024. ISSN 1836-9553. doi: https://doi.org/10.1016/j.jphys.2024.08.010. URL https://www.sciencedirect.com/science/article/pii/S1836955324000869.

[2] Miriam Mathea, Waldemar Klingspohn, and Knut Baumann. Chemoinformatic classification methods and their applicability domain. *Molecular Informatics*, 35(5):160–180, 2016. ISSN 1868-1743. doi: 10.1002/minf.201501019. URL https://doi.org/10.1002/minf.201501019.

[3] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain, 2019.

[4] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks, 2017. URL https://arxiv.org/abs/1706.04599.

[5] Maohao Shen, Yuheng Bu, Prasanna Sattigeri, Soumya Ghosh, Subhro Das, and Gregory Wornell. Post-hoc uncertainty learning using a dirichlet meta-model, 2022. URL https://arxiv.org/abs/2212.07359.

[6] Thierry Hanser, Chris Barber, Jean-François Marchaland, and Stéphane Werner. Applicability domain: towards a more formal definition. *SAR and QSAR in Environmental Research*, 27(11):893–909, 2016. doi: 10.1080/1062936X.2016.1250229.

[7] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks, 2014. URL https://arxiv.org/abs/1312.6199.

[8] Umang Bhatt, Javier Antorán, Yunfeng Zhang, Q. Vera Liao, Prasanna Sattigeri, Riccardo Fogliato, Gabrielle Gauthier Melançon, Ranganath Krishnan, Jason Stanley, Omesh Tickoo, Lama Nachman, Rumi Chunara, Madhulika Srikumar, Adrian Weller, and Alice Xiang. Uncertainty as a form of transparency: Measuring, communicating, and using uncertainty, 2021. URL https://arxiv.org/abs/2011.07586.

[9] Jishnu Mukhoti, Andreas Kirsch, Joost van Amersfoort, Philip H. S. Torr, and Yarin Gal. Deep deterministic uncertainty: A simple baseline, 2022. URL https://arxiv.org/abs/2102.11582.

[10] Bálint Mucsányi, Michael Kirchhof, and Seong Joon Oh. Benchmarking uncertainty disentanglement: Specialized uncertainties for specialized tasks, 2024. URL https://arxiv.org/abs/2402.19460.

[11] Matthias Hein, Maksym Andriushchenko, and Julian Bitterwolf. Why relu networks yield high-confidence predictions far away from the training data and how to mitigate the problem, 2019. URL https://arxiv.org/abs/1812.05720.

[12] Irene Luque Ruiz and Miguel Ángel Gómez-Nieto. Study of the applicability domain of the qsar classification models by means of the rivality and modelability indexes. *Molecules*, 23(11):2756, October 2018. ISSN 1420-3049. doi: 10.3390/molecules23112756. URL https://www.mdpi.com/1420-3049/23/11/2756. Academic Editor: Kok Hwa Lim.

[13] Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017. URL https://openreview.net/forum?id=Hkg4TI9xl.

[14] Stephen Marsland. Novelty detection in learning systems. *Neural Computing Surveys*, 3(2): 157–195, 2003. doi: 10.1162/109434603321337526. URL https://homepages.ecs.vuw.ac.nz/~marslast/PUBS/NCS.pdf.

[15] Mahjabeen Tahir, Azizol Abdullah, Nur Izura Udzir, and Khairul Azhar Kasmiran. A systematic review of machine learning and deep learning techniques for anomaly detection in data mining. *International Journal of Computers and Applications*, 47(2):169–187, 2025. doi: 10.1080/1206212X.2025.2449999. URL https://doi.org/10.1080/1206212X.2025.2449999.

[16] Sander Vanderlooy, Laurens van der Maaten, and Ida Sprinkhuizen-Kuyper. Enhancing the performance of dynamic scripting in computer games. In *Advances in Artificial Intelligence*, volume 4571 of *Lecture Notes in Computer Science*, pages 310–323. Springer, 2007. doi: 10.1007/978-3-540-73575-5_32.

[17] Tim Pearce, Alexandra Brintrup, and Jun Zhu. Understanding softmax confidence and uncertainty, 2021. URL https://arxiv.org/abs/2106.04972.

[18] Litian Liu and Yao Qin. Fast decision boundary based out-of-distribution detector, 2024. URL https://arxiv.org/abs/2312.11536.

[19] David Mickisch, Felix Assion, Florens Greßner, Wiebke Günther, and Mariele Motta. Understanding the decision boundary of deep neural networks: An empirical study, 2020. URL https://arxiv.org/abs/2002.01810.

[20] Juan Shu, Bowei Xi, and Charles Kamhoua. Understanding adversarial examples through deep neural network's response surface and uncertainty regions, 2021. URL https://arxiv.org/abs/2107.00003.

[21] Lewis Smith and Yarin Gal. Understanding measures of uncertainty for adversarial example detection, 2018. URL https://arxiv.org/abs/1803.08533.

[22] Patrick McDaniel, Nicolas Papernot, and Z. Berkay Celik. Machine learning in adversarial settings. *IEEE Security & Privacy*, 14(3):68–72, 2016. ISSN 1540-7993. doi: 10.1109/MSP.2016.51. URL https://ieeexplore.ieee.org/document/7473051. Editors: Sean W. Smith and William Enck.

[23] Antonio Pedraza, Oscar Deniz, and Gloria Bueno. Really natural adversarial examples. *International Journal of Machine Learning and Cybernetics*, 13:1065–1077, 2022. doi: 10.1007/s13042-021-01435-0. URL https://doi.org/10.1007/s13042-021-01435-0. Issue date: April 2022.

[24] Xingjun Ma, Bo Li, Yisen Wang, Sarah M. Erfani, Sudanthi N. R. Wijewickrema, Grant Schoenebeck, Dawn Song, Michael E. Houle, and James Bailey. Characterizing adversarial subspaces using local intrinsic dimensionality. In *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, 2018. URL https://openreview.net/forum?id=B1gJ1L2aW. arXiv preprint arXiv:1801.02613.

[25] Landan Seguin, Anthony Ndirango, Neeli Mishra, SueYeon Chung, and Tyler Lee. Understanding the logit distributions of adversarially-trained deep neural networks, 2021. URL https://arxiv.org/abs/2108.12001.

[26] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2015. URL https://arxiv.org/abs/1412.6572.

[27] Emanuele Ledda, Daniele Angioni, Giorgio Piras, Giorgio Fumera, Battista Biggio, and Fabio Roli. Adversarial attacks against uncertainty quantification, 2023. URL https://arxiv.org/abs/2309.10586.

[28] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Wiley-Interscience, New York, NY, 2nd edition, 2001. ISBN 978-0471056690.

[29] Telmo Silva Filho, Hao Song, Miquel Perello-Nieto, Raul Santos-Rodriguez, Meelis Kull, and Peter Flach. Classifier calibration: A survey on how to assess and improve predicted class probabilities. *Machine Learning*, 112(9):3211–3260, 2023. doi: 10.1007/s10994-023-06336-7. Open Access.

[30] Anqi Mao, Mehryar Mohri, and Yutao Zhong. Predictor-rejector multi-class abstention: Theoretical analysis and algorithms, 2024. URL https://arxiv.org/abs/2310.14772.

[31] Chenri Ni, Nontawat Charoenphakdee, Junya Honda, and Masashi Sugiyama. On the calibration of multiclass classification with rejection, 2019. URL https://arxiv.org/abs/1901.10655.

[32] Ran El-Yaniv and Yair Wiener. On the foundations of noise-free selective classification. *Journal of Machine Learning Research*, 11(53):1605–1641, 2010. URL http://jmlr.org/papers/v11/el-yaniv10a.html.

[33] Christopher Mohri, Daniel Andor, Eunsol Choi, and Michael Collins. Learning to reject with a fixed predictor: Application to decontextualization, 2023. URL https://arxiv.org/abs/2301.09044.

[34] Anqi Mao, Mehryar Mohri, and Yutao Zhong. Theoretically grounded loss functions and algorithms for score-based multi-class abstention, 2024. URL https://arxiv.org/abs/2310.14770.

[35] Xu-Yao Zhang, Cheng-Lin Liu, Guo-Sen Xie, Xiuli Li, and Tao Mei. A survey on learning to reject. *Proceedings of the IEEE*, 111(2):146–183, 2023. doi: 10.1109/JPROC.2023.3238024.

[36] Yonatan Geifman and Ran El-Yaniv. Selective classification for deep neural networks. In *Advances in Neural Information Processing Systems*, volume 30, pages 4885–4894, 2017. doi: 10.5555/3295222.3295241. URL https://arxiv.org/abs/1705.08500.

[37] Håkan Linusson, Ulf Johansson, Henrik Boström, and Tuwe Löfström. Classification with reject option using conformal prediction. In *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, volume 10937 of *Lecture Notes in Computer Science*, pages 94–105. Springer, 2018. doi: 10.1007/978-3-319-93037-7_10. URL https://www.diva-portal.org/smash/record.jsf?pid=diva2:1242267.

[38] Vladimir Vovk, Alexander Gammerman, and Glenn Shafer. *Algorithmic Learning in a Random World*. Statistics for Engineering and Information Science. Springer, New York, NY, 2005. ISBN 978-0-387-25797-0. doi: 10.1007/b138568.

[39] Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84:317–331, December 2018. ISSN 0031-3203. doi: 10.1016/j.patcog.2018.07.023. URL http://dx.doi.org/10.1016/j.patcog.2018.07.023.

[40] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks, 2017. URL https://arxiv.org/abs/1608.04644.

[41] Tom B. Brown, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. Adversarial patch. *arXiv preprint arXiv:1712.09665*, 2017. URL https://arxiv.org/abs/1712.09665.

[42] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *CoRR*, abs/1607.02533, 2016. URL https://arxiv.org/abs/1607.02533.

[43] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks, 2016. URL https://arxiv.org/abs/1511.04599.

[44] 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.

[45] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

[46] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[47] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, pages 8024–8035, 2019. URL https://papers.nips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.

[48] Mark Hopkins, Erik Reeber, George Forman, and Jaap Suermondt. Spambase. UCI Machine Learning Repository, 1999. URL https://doi.org/10.24432/C53G6X.

[49] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[50] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015. URL https://arxiv.org/abs/1502.03167.

[51] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 315–323, 2011. URL http://proceedings.mlr.press/v15/glorot11a.html.

[52] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. Technical Report.

[53] Yaofo Chen. pytorch-cifar-models: Pretrained cifar-10 and cifar-100 models for pytorch, 2020. URL https://github.com/chenyaofo/pytorch-cifar-models.

[54] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. URL https://arxiv.org/abs/1512.03385.

[55] PyTorch Core Team. torchvision: Datasets, Transforms and Models for Computer Vision in PyTorch, 2016. URL https://github.com/pytorch/vision.

[56] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. Ultralytics yolov8, 2023. URL https://github.com/ultralytics/ultralytics.

[57] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. Yolo by ultralytics, 2023. URL https://github.com/ultralytics/ultralytics.

[58] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5, 2017. URL http://jmlr.org/papers/v18/16-365.html.