Technische Universität München
TUM School of Computation, Information and Technology

Monte Carlo Averaging for Uncertainty Estimation in Neural Networks

Cedrique Rovile Njieutcheu Tassi

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technische Universität München zur Erlangung eines
Doktors der Ingenieurwissenschaften (Dr. Ing.)
genehmigten Dissertation.

Die Dissertation wurde am 01.02.2023 bei der Technische Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 03.07.2024 angenommen.

I confirm that this thesis is my own work and I have documented all sources and material used.

Munich, 01.02.2023 Cedrique Rovile Njieutcheu Tassi

Acknowledgments

This thesis would not have been possible without the contribution of a number of people whom I will try to thank here.

First, I would like to express my gratitude to **Prof. Dr. Stefan Leutenegger** for giving me the opportunity to conduct my PhD thesis at the German Aerospace Center (DLR), Institute of Optical Sensor Systems, Department of Real-time Data Processing. His invaluable help and guidance were vital for the completion of this thesis.

I would also like to thank **Prof. Dr. Rudolph Triebel** for his invaluable supervision and constructive suggestions. His guidance has always left me with much space for exploring new directions.

In addition, I would like to thank all my colleagues from the Department of Real-time Data Processing, especially **Dr. Patrick Irmisch** and **Maik Wischow**, who shared with me the burden of a tiring but rewarding work of conducting a PhD thesis.

Moreover, I would like to thank my colleagues **Auliya Unnisa Fitri** and **Jakob Gawlikowski** from the Institute of Data Science for the meetings wherein we discussed our ideas and results.

Furthermore, I wish to thank the anonymous reviewers of all publications who were involved in this thesis for their insightful comments and suggestions.

I would also like to thank my family, especially my aunt (**Djeukam Bernadette**), my grandmother (**Djomgoue Marie**), my brother (**Nimeni Tassi Eric**), and my sisters (**Yameze Tassi Christelle**, **Djeumeni Tassi Irene**, and **Siewandji Tassi Adrienne Sandra**), for their unconditional and unending love, understanding, and support.

Finally, I would like to thank my late grandfather (**Njieutcheu Bernard**), late father (**Tassi Tondomnou Pierre**), late mother (**Njieutcheu Tchagam Emeraude Charlie**), and late brothers (**Ngueukam Tassi Franck** and **Kameni Tassi Pignore Xavier**), to whom I dedicate this thesis. Without them, this day would not have been possible.

Abstract

Although neural networks have been used for pattern classification for decades, convolutional neural networks (CNNs) have become increasingly important over the past several years. In particular, CNNs are utilized in automated scenarios for traffic sign recognition and disease classification. However, they still suffer from overfitting and lack of robustness to undesired inputs. Hence, they can generate overconfident false predictions (FPs), which can be dangerous and costly, especially when used in safety- and/or mission-critical applications. Here, overconfident FPs can (1) cause collisions in robotic applications, (2) prompt false treatments in medical applications, or (3) increase costs in financial applications. These significant consequences limit the use of CNNs in the aforementioned fields even though their technological potential is of great interest. To overcome these limitations and encourage the widespread use of CNNs in safety- and/or mission-critical applications, we aim to prevent FPs by improving the separability between true predictions (TPs) and FPs. To achieve this, we will force the degree of confidence (measuring uncertainty) to be high for TPs and low for FPs. This is based on the hypothesis that if the confidence is high for TPs and low for FPs, both TPs and FPs will be well-separated using a threshold. Therefore, the research questions are as follows:

(1) Which method forces the degree of confidence to be high for TPs and low for FPs?

(2) Under what circumstances does the method work?

(3) At what cost does the method help to maintain a low confidence for FPs and a high confidence for TPs?

To address the first question, we develop a method called Monte Carlo averaging (MCA) and compare it to related methods, such as baseline (single CNN), Monte Carlo dropout (MCD), ensemble of CNNs, and mixture of Monte Carlo dropout (MMCD). To answer the second question, we gauge the performance of the developed and related methods on four datasets with different difficulties. In addition, we gauge the performance of the developed and related methods on different CNNs to assess their performance on different architectures. Further, we investigate the impact of applying logit instead of probability averaging on the developed and related methods, as well as the impact of reducing the strength of regularization during training. To address the third question, we evaluate the ability of the developed and related methods to separate TPs and FPs and examine the classification accuracy, calibration error, and inference time.

Experimental results show improvements in the developed MCA and the state-of-the-art MMCD compared to the other related methods (baseline, MCD, and ensemble of CNNs). Specifically, similar to MMCD, the developed MCA can preserve the accuracy of the underlying ensemble, which may increase the baseline accuracy. The baseline accuracy could only be preserved by MCD. Both MMCD and MCA improve the separability of TPs and FPs at the cost of increasing the calibration error and inference time. However, applying logit instead of probability averaging in MCA and related methods or reducing the strength of regularization decreases the calibration error at the cost of negatively impacting the separability of TPs and FPs. Hence, there is a tradeoff between improving the calibration and improving the separability of TPs and FPs. Although the performance of all methods heavily relies on the dataset and/or architecture, MCD and MMCD are more sensitive to the dataset and/or architecture.

Overall, we developed MCA to force the degree of confidence to be high for TPs and low for FPs in order to improve the separability of TPs and FPs. Compared to the state-of-the-art MMCD, the developed MCA is more than four times faster, has the same purpose and underlying principle, and shows similar or sometimes better performance. Therefore, we suggest utilizing MCA instead of MMCD for applications that require separability of TPs and FPs and where the computational budget is limited. MCA may also be advantageous for other fields of machine learning, such as active or reinforcement learning, where uncertainty is required. Moreover, MCA is preferable in the field of explainable artificial intelligence, which explores the role of uncertainty to explain predictions and increase the social acceptance of CNN-based decision-making systems. Finally, MCA opens new perspectives to fuse features of ensemble members.

Keywords: Machine learning, Deep learning, Classification, Convolutional neural network, Ensemble, Bayesian neural network, Monte Carlo dropout, Mixture of Monte Carlo dropout, Confidence calibration, Uncertainty quantification, Uncertainty estimation, Separating true predictions and false predictions, Regularization strength, Logit averaging

Kurzfassung

Obwohl neuronale Netze seit Jahrzehnten zur Musterklassifikation verwendet werden, hat CNNs in den letzten Jahren immer mehr an Bedeutung gewonnen. Insbesondere werden CNNs in automatisierten Szenarien zur Verkehrszeichenerkennung und Krankheitsklassifizierung eingesetzt. Sie leiden jedoch immer noch unter Overfitting und mangelnder Robustheit gegenüber unerwünschten Eingaben. Daher können sie overconfident FPs erzeugen, was gefährlich und kostspielig sein kann, insbesondere wenn sie in sicherheits- und/oder missionskritischen Anwendungen eingesetzt werden. Hier kann overconfident FPs (1) Kollisionen in Roboteranwendungen verursachen, (2) falsche Behandlungen in medizinischen Anwendungen auslösen, oder (3) Gewinn in Finanzanwendungen vermindern. Diese erheblichen Konsequenzen schränken die Verwendung von CNNs in den vorgenannten Bereichen ein, obwohl ihr technologisches Potenzial von großem Interesse ist. Um diese Einschränkungen zu überwinden und den weit verbreiteten Einsatz von CNNs in sicherheits- und/oder missionskritischen Anwendungen zu fördern, wollen wir FPs verhindern, indem wir die Trennbarkeit zwischen TPs und FPs verbessern. Um dies zu erreichen, wollen wir die Konfidenz (welche die Unsicherheit misst) erzwingen, für TPs hoch und für FPs niedrig zu sein. Dies basiert auf der Hypothese, dass TPs und FPs durch einen Schwellenwert gut getrennt werden können, wenn die Konfidenz für TPs hoch und für FPs niedrig ist. Die Forschungsfragen lauten daher wie folgt:

(1) Welche Methode brauchen wir, um eine hohe Konfidenz für TPs und niedrige Konfidenz für FPs zu erzwingen?

(2) Unter welchen Umständen funktioniert die vorgeschlagene Methode?

(3) Zu welchem Preis trägt die vorgeschlagene Methode dazu bei, eine hohe Konfidenz für TPs und eine niedrige Konfidenz für FPs aufrechtzuerhalten?

Um die erste Forschungsfrage zu beantworten, entwickeln wir eine Methode namens MCA und vergleichen sie mit verwandten Methoden wie baseline (single CNN), MCD, ensemble of CNNs, und MMCD. Um die zweite Forschungsfrage zu beantworten, evaluieren wir die Performance von MCA und verwandten Methoden an vier Datensätzen mit unterschiedlichen Schwierigkeiten. Darüber hinaus evaluieren wir MCA und verwandten Methoden auf verschiedenen CNNs, um ihre Performance auf verschiedenen Architekturen zu bewerten. Außerdem bewerten wir die Auswirkung der Anwendung von Logit anstelle

Probabilitäten in MCA und verwandten Methoden sowie die Auswirkung der Verringerung der Regularisierungsstärke. Um die dritte Forschungsfrage anzugehen, bewerten wir die Fähigkeit von MCA und verwandten Methoden, TPs und FPs zu trennen, und analysieren die Klassifikationsgenauigkeit, der Kalibrierungsfehler, und die Inferenzzeit.

Experimentelle Ergebnisse zeigen eine Verbesserung des entwickelten MCA und des State-of-the-Art MMCD gegenüber verwandten Methoden wie Baseline, MCD, und Ensemble von CNNs. Insbesondere kann MCA, ähnlich wie MMCD, die Klassifikationsgenauigkeit des zugrunde liegenden Ensembles bewahren, welches die Klassifikationsgenauigkeit von Baseline erhöhen kann, die von MCD nur bewahrt werden kann. Sowohl MMCD als auch MCA verbessern die Trennbarkeit von TPs und FPs auf Kosten einer Erhöhung des Kalibrierungsfehlers und der Inferenzzeit. Die Anwendung von Logit anstelle Probabilitäten in MCA und verwandten Methoden oder die Verringerung der Regularisierungsstärke vermindert jedoch den Kalibrierungsfehler auf Kosten der Beeinträchtigung der Trennbarkeit zwischen TPs und FPs. Daher gibt es einen Kompromiss zwischen der Verbesserung der Kalibrierung und der Verbesserung der Trennbarkeit zwischen TPs und FPs. Obwohl die Performance aller Methoden stark von dem Datensatz und/oder der Architektur abhängt, sind MCD und MMCD empfindlicher gegenüber dem Datensatz und/oder der Architektur.

Zusammengefasst haben wir MCA entwickelt, um eine hohe Konfidenz für TPs und eine niedrige Konfidenz für FPs aufrechtzuerhalten und die Trennbarkeit von TPs und FPs zu verbessern. Im Vergleich zum State-of-the-Art MMCD, ist das entwickelte MCA mehr als viermal schneller, hat den gleichen Zweck und das gleiche zugrunde liegende Prinzip, und zeigt eine ähnliche oder manchmal bessere Performance. Daher empfehlen wir die Verwendung von MCA anstelle von MMCD für Anwendungen, die eine Trennbarkeit zwischen TPs und FPs erfordern und bei denen das Rechenbudget begrenzt ist. MCA kann auch für andere Bereiche des maschinellen Lernens von Vorteil sein, wie z. B. Aktive oder Reinforcement Learning, wo Unsicherheit erforderlich ist. Darüber hinaus ist MCA vorzuziehen im Bereich der erklärbaren Maschinenlernen, die die Rolle von Unsicherheit untersucht, um Vorhersagen zu erklären und die soziale Akzeptanz von CNN-basierten Entscheidungssystemen zu erhöhen. Schließlich eröffnet MCA neue Perspektiven, um Merkmale von Ensemblemitgliedern zu fusionieren.

Publications derived from this thesis

[70] Cedrique Rovile Njieutcheu Tassi. "Bayesian convolutional neural network: Robustly quantify uncertainty for misclassifications detection". In: *Pattern Recognition and Artificial Intelligence: Third Mediterranean Conference, MedPRAI 2019, Istanbul, Turkey, December 22–23, 2019, Proceedings 3*. Springer. 2020, pp. 118–132.

[99] Jakob Gawlikowski, Cedrique Rovile Njieutcheu Tassi, Mohsin Ali, Jongseok Lee, Matthias Humt, Jianxiang Feng, Anna Kruspe, Rudolph Triebel, Peter Jung, Ribana Roscher, et al. "A survey of uncertainty in deep neural networks". In: *Artificial Intelligence Review* 56.Suppl 1 (2023), pp. 1513–1589.

[101] Cedrique Rovile Njieutcheu Tassi, Anko Börner, and Rudolph Triebel. "Monte Carlo averaging for uncertainty estimation in neural networks". In: *Journal of Physics: Conference Series*. Vol. 2506. 1. IOP Publishing. 2023, p. 012004.

[102] Cedrique Rovile Njieutcheu Tassi, Jakob Gawlikowski, Auliya Unnisa Fitri, and Rudolph Triebel. "The impact of averaging logits over probabilities on ensembles of neural networks." In: *AISafety@ IJCAI*. 2022.

[103] Cedrique Rovile Njieutcheu Tassi, Anko Börner, and Rudolph Triebel. "Regularization Strength Impact on Neural Network Ensembles". In: *Proceedings of the 2022 5th International Conference on Algorithms, Computing and Artificial Intelligence*. 2022, pp. 1–9.

Nomenclature

If not stated otherwise, the following symbols have the following meanings:

Models

f	A trained neural network
f_0	An untrained but initialized neural network
f_m	The neural network for the m^{th} ensemble member
$f_{Discriminator_m}$	The discriminator for the m^{th} ensemble member
$f_{Discriminator}$	The discriminator of a neural network based classifier
$f_{FeatureExtractor_m}$	The feature extractor for the m^{th} ensemble member
$f_{FeatureExtractor}$	The feature extractor of a neural network based classifier
$f_{FeatureSampling}$	A feature sampling model parameterized by random variables drawn from known distributions
g	A sensing model, for example, a camera
h	An annotator model, for example, a human
hn	A Bayesian hypernetwork

Hyperparameters

β	The number of datapoints or samples in minibatches
γ	Learning rate
ν	Momentum
B	The number of equallyspaced bins
C	The number of channel of an input image

F	The number of convolution kernels or feature maps, also referred to as the dimension of features
H	The height of an input image
K	The number of possible output classes
M	The number of ensemble members
N	The number of data points or samples within a dataset
S	The number of stochastic forward passes in MCD or MMCD
W	The width of an input image

Other symbols

α	A sampling mask including random variables drawn from a Gaussian distribution
β	A sampling mask including random variables drawn from a Bernoulli distribution
ϵ	A random noise
$\hat{x}$	Output of a feature extractor, also referred to as feature vector
$\hat{x}^a$	A feature vector obtained after application of the feature averaging operation
$\hat{x}^s$	A feature vector obtained after application of the feature sampling operation
$\hat{x}^{a_{m_n}}$	The n^{th} average feature vector of the m^{th} ensemble member
$\hat{x}^{m_s}$	A feature vector sampled at the s^{th} forward pass by the m^{th} ensemble member
$\hat{x}^m$	The feature vector for the m^{th} ensemble member
$\hat{x}_i^m$	The i^{th} element of the feature vector $\hat{x}^m$
$\hat{y}$	Output of a neural network, also referred to as the predicted label for x
λ	The inverse temperature constant
$\lvert\hat{x}\rvert$	The cardinality or number of elements of $\hat{x}$
μ	Mean
∇L	The gradient vector
$\overline{z}$	The average logit vector

σ	Standard deviation
θ	Neural network weights, also referred to as parameters
a	A feature or neuron output
d	A small location of the input image of size $I \times J$
e	Sensing errors
k	A logit vector of size K, also referred to as the input to softmax
L	The cross-entropy loss
l	A convolution kernel of size $I \times J$
r	A pooling region
u	The predictive uncertainty
x	Output of a sensing model, also referred to as the input to a neural network
x_0	Input to a sensing model, also referred to as the measurements of an object in the operating environment
y	Output of the annotator model, also referred to as the ground-truth label for x
z	A logit vector, also referred to as an input to the softmax function
z^m	The logit vector for the m^{th} ensemble member

Spaces

X	The input space
Y	The output or label space

Sets

$\mathbb{R}$	The set of real numbers
b_τ	The set of indices of evaluation samples whose confidences fall into the interval $I_\tau =]\frac{\tau-1}{B}, \frac{\tau}{B}]$ with $\tau \in [1, B]$
D_{train}	The training dataset
U^K	The set of standard unit vectors of $\mathbb{R}^K$

Probabilities

$\mathcal{N}()$	A normal probability distribution
$\overline{p}()$	The average discrete probability vector
$KL()$	The Kullback-Leibler divergence
$p()$	A discrete probability vector of size K, also referred to as the softmax output
$p^m()$	The discrete probability vector for the m^{th} ensemble member
$p^s()$	The discrete probability vector for the s^{th} forward pass
$p^{a_{m_n}}()$	The discrete probability vector for the n^{th} feature averaging operation of the m^{th} ensemble member
$p^{m_s}()$	The discrete probability vector for the s^{th} forward pass of the m^{th} ensemble member
p_k	The probability that x belongs to class $k \in [1, K]$
$p_{\overline{z}}$	The discrete probability vector obtained from the average logit vector $\overline{z}$

Contents

1. Introduction

1.1. Background

Artificial intelligence in general and machine learning in particular, plays an increasingly important role in many scientific areas. Predominantly, neural networks have exhibited unprecedented success in recent years. The basic idea behind neural networks was postulated decades ago based on a mathematical modeling of the biological brain through the interconnection of artificial neurons (computing units) that form a structure consisting of a great number of layers (each with a great number of units). The learning process of neural networks essentially consists of a chain of gradient calculations, with the help of parameters that minimize a given error function. Owing to the emergence of large datasets, increasing computational power, and tremendous advances in deep learning, modern architectures such as CNNs rapidly gain ground in recent years and have now emerged as a leading technology for solving complex data analysis problems. CNNs have propelled the significant progress of various fields such as natural language processing, handwriting recognition, text classification, visual instance retrieval, and image processing, just to name a few. Specifically, CNNs are utilized in scene recognition, face recognition and verification, human pose estimation, vehicle search and reidentification, off-road obstacle avoidance, traffic sign recognition, and speech recognition. Therefore, sooner or later, CNNs will highly impact our daily lives.

1.2. Problem statement

Despite the great successes and ongoing advances in deep learning, deep neural networks such as CNNs still have complications such as overfitting and lack of robustness.

Overfitting Given a training and test dataset, a deep neural network f is said to overfit the training dataset if there exists another deep neural network f' such that f' has more error than f on the training dataset, but f' has less error than f on the test dataset. Simply put, overfitting occurs when a deep neural network error on the test dataset is higher than its error on the training dataset. Bejani et al. [1] discussed and summarized several factors that cause overfitting. The two main factors are as follows: the deep neural network may learn noise inherent in the training instead of the underlying pattern in the data and the deep neural network may memorize the training data owing to the large capacity (number of parameters) of the deep neural network or the lack of training data. In addition, Bejani et al. [1] discussed and summarized existing methods to avoid overfitting in deep learning. One of the methods is *early stopping [2, 3]*, which reduces the training time and prevents the deep neural networks from memorizing training data through continuous update of weights and biases. Since deep neural networks with large weights exhibit a high level of overfitting, several *weight norm penalties* [4, 5] have been proposed to force the deep neural networks learn smaller weights. To further curb overfitting, standard and advance *data augmentation* techniques have been proposed to increase the size of the training data [6, 7, 8]. In addition, *dropout [9]* and variants [10, 11], such as randomly removing some neurons (connections, channels, blocks, or data) at training, force a deep neural network to become sensitive to individual neuron (connection, channel, block, or data point) and to therefore, generalize better on unseen data. *Batch normalization [12, 13]* is another method that counters overfitting by augmenting (via scaling and shifting) internal feature representations using data statistics extracted from training batches. Despite the encouraging progresses in addressing overfitting, deep neural networks still exhibit certain levels of overfitting [14] resulting in generalization errors (e.g., test classification error) and false predictions (FPs). For instance, assume that a CNN was trained and achieved a classification accuracy of 99.99% on the test data; however, it has a test error of 0.01% that indicates FPs. Moreover, overfitting causes overconfident [15] and miscalibrated [16] predictions, which means that deep neural networks can make overconfident FPs. Empirically, miscalibration occurs when the average confidence on the test dataset does not match the classification accuracy. In this context, we say that a deep neural network is *overconfident* when the average confidence is greater than the classification accuracy and *underconfident* when the average confidence is lower than the classification accuracy.

Lack of robustness A deep neural network is said to *lack robustness* when it cannot process undesired inputs in an acceptable manner. Particularly, in the presence of an undesired input, the deep neural network makes FPs instead to remain silent (e.g., "I don't know") or seek human guidance. The violation of the principle of empirical risk minimization primarily causes the lack of robustness. According to Vapnik [17], the empirical risk minimization principle states that by minimizing the training error, a deep neural network will generalize to previously unseen data, under the condition that novel data points and labels are drawn from the same distribution as the training data. Nevertheless, deployed deep neural networks often encounter *out-of-domain (OOD)*[1] and *domain-shift*[2] examples that are different from the training data of deep neural networks. The mismatch between the input and training data distribution can result in violations of the empirical risk minimization principle, thereby leading to FPs. For instance, Hendrycks and Dietterich [18] empirically showed that CNNs change predictions when corruptions such as blur and noise (non-affine transformation) are applied on the input. The inability of the training data to capture all possible representations (present in the real-world) of a given pattern resulted in mismatches between the input and training data distribution. For instance, noise that is inherent in the input data owing to variable illumination, camera angle, clutter, occlusion, or other physical phenomena (e.g., changes in temperature, vibration, or mechanical stress) cannot be completely represented in the training data. In addition, the inability of a deep neural network to learn all possible representations present in a given training data caused the mismatches. This is justified by the fact that the training error is usually nonzero. Moreover, many works such as that of Goodfellow et al. [19] that aimed to design *adversarial*[3] examples have experimentally showed that deep neural networks can react in an unexpected and incorrect manner to slight perturbations of their inputs. Furthermore, Nguyen et al. [20] empirically showed that deep neural networks can be easily fooled, because they can classify many unrecognizable objects with high confidence as members of a well-known class. Improving the robustness of deep neural networks against perturbed or unknown examples has become an important research topic in machine learning. Some works [21, 22] augmented the training data with adversarial examples. However, learning augmented examples may result in overfitting [23] and may not provide robustness to unlearned examples. Other works [24, 25, 26] evaluated measures of the predictive uncertainty for detecting adversarial examples. According to Hendrycks and Gimpel [27] and further acknowledged by Liang et al. [28], deep neural networks tend to assign higher softmax scores to in-distribution than distribution-shift and out-of-distribution examples. Consequently, they evaluated softmax scores to determine whether an input is misclassified or from a distribution different from the training one. Despite the encouraging efforts in building robust deep neural networks, modern classifiers such as CNNs are still prone to misclassifications and cannot completely process distribution-shift and

[1]**Out-of-domain (or out-of-distribution) examples** are data with scenarios never seen by a deep neural network at training. These examples always exist because a training dataset can never capture all scenarios present in the real world.

[2]**Domain-shift (or distribution-shift) examples** are training data affected by a set of perturbations such as changes in camera lens and lighting conditions, occlusions, clutter, adversarial perturbations, random noise, geometric transformations, and other physical phenomena such as changes in temperature or mechanical stress.

[3]**Adversarial examples** are perturbed inputs that have been intentionally slightly modified by human attackers to fool a model. The perturbations are usually imperceptible to a human observer.

out-of-distribution examples in an acceptable manner.

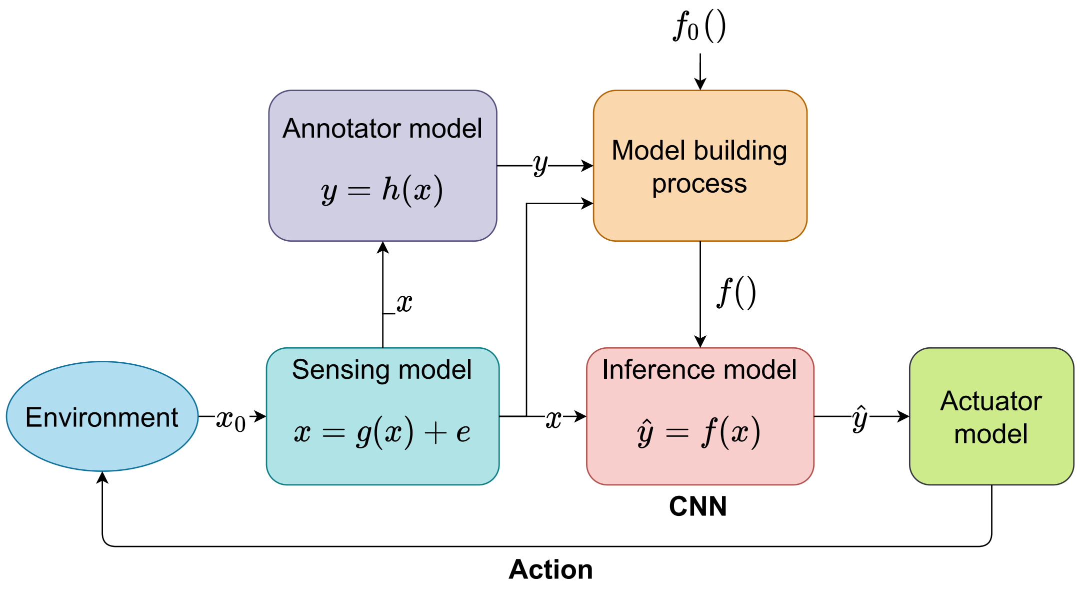


Figure 1.: A CNN-based classifier as inference model and part of the decision-making unit of a safety- and/or mission-critical system. The CNN processes an image x generated by a camera representing the sensing model to produce a signal $\hat{y}$ for the actuator, that is, the CNN specifies what action the actuator must take in the operating environment. Herein, overconfident FPs made by the CNN will result in false actions by the actuator, which can possibly damage the operating environment. Overconfident FPs occur owing to changes in the environment inherent in x_0, sensing errors e inherent in x, or errors in the annotation or model building process inherent in $f()$

1.3. Motivation for uncertainty estimation

The abovementioned complications such as overfitting and lack of robustness can be costly and dangerous, especially when deep neural networks are part of the decision-making unit of safety- and/or mission-critical applications. Figure 1 presents a CNN-based classifier producing actuator signals. False prediction or actuator signal will result in false action in the environment, which leads to significant consequences such as collisions in robot applications, potential false treatments in medical applications, or increased costs in financial applications. These three cases are further discussed as follows:

Collisions in robotic applications Over the past decades, the adoption of deep learning in robotic applications can be attributed to its major advances. According to

Pierson and Gashler [29], specific applications are in collision prediction [30, 31], door recognition for robot navigation [32], pedestrian detection [33], and prediction of traffic maneuvers [34]. In all these applications, failure in recognition or prediction can cause collisions. These collisions can damage the robot, which results in severe economic losses, and also damage the operating environment. For instance, if a pedestrian prediction system in a self-driving car fails to predict the presence of a human, it can result in collision and potential loss of human life.

False treatments in medical applications In recent years, deep learning has become the standard for medical diagnosis, especially in diagnostic imaging. Specifically, it can be utilized in the diagnosis of eye diseases such as diabetic retinopathy [35], diagnosis of skin cancer [36], and recognition of lymph node metastasis of breast cancer [37]. In all these applications, false recognition or diagnosis can potentially lead to no or false treatments. For instance, the absence of diabetic retinopathy in the scanned images of the eyes can erroneously result in preventing the doctor to administer the necessary treatments to a patient. On the contrary, a false recognition of the presence of diabetic retinopathy can encourage a doctor to administer unnecessary treatments to a patient. An unnecessary treatment is not only costly for a patient but also exposes a patient to potentially years of physical discomfort and pain and to all kinds of risk associated with the treatment. False treatments not only have negative consequences on patients but also diminish the trust of (potential) patients to medical practices, which can cause economic losses to the healthcare industry.

Increased costs in financial applications In recent years, deep learning has gained popularity in financial and banking services owing to the proliferation of financial technology. Specifically, according to Huang, Chai, and Cho [38] and Ozbayoglu, Gudelek, and Sezer [39], deep learning has been utilized in banking and credit risk prediction [40], exchange rate prediction [41], and financial market prediction [42]. In all these applications, false predictions can lead to severe financial losses. For instance, imagine that a credit risk prediction system attributes a good score to a potential client who is in reality ineligible for credit. Consequently, the client can obtain a credit that will cause an interruption of cash flows to the bank later on because the client cannot repay the loan or meet contractual obligations. This can result in increased costs for collection or even potential collapse of the financial institution if the number of clients with false credit score is large.

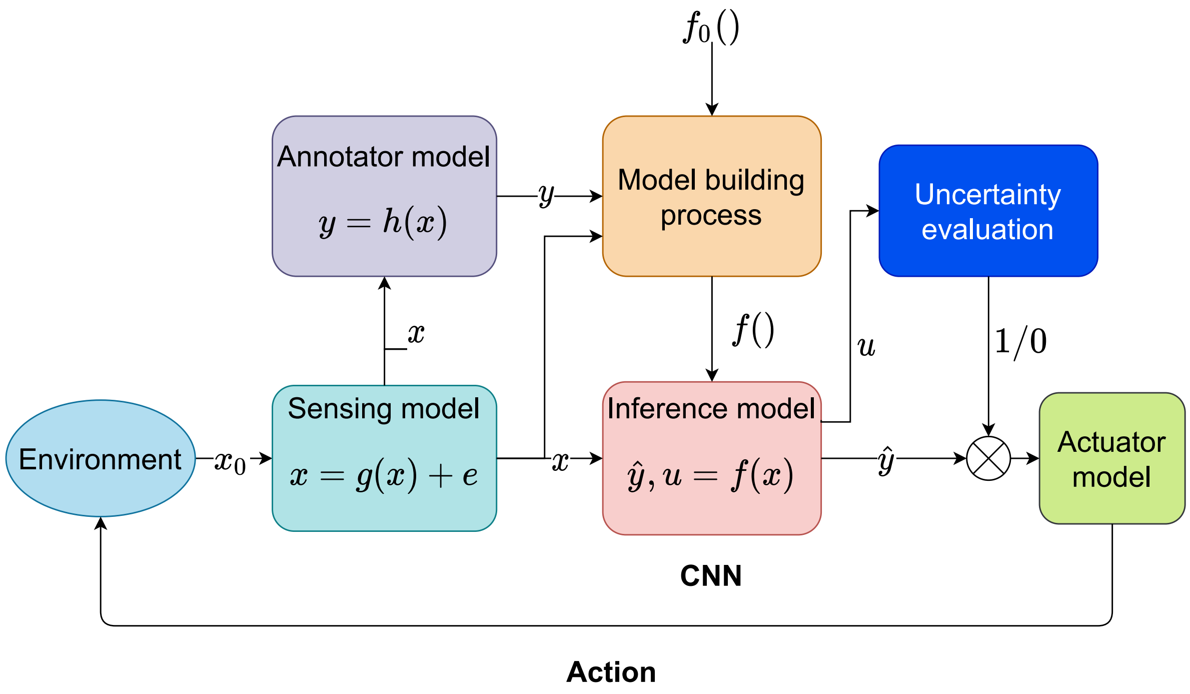


Figure 2.: A CNN-based classifier as inference model and part of the decision-making unit of a safety- and/or mission-critical system. To avoid FPs, we want to estimate and evaluate the uncertainty u inherent to the output $\hat{y}$ of the CNN. The uncertainty evaluation consists of thresholding u, for example, the predictive confidence, to produce signals "1" for true (certain) predictions and "0" for false (uncertain) predictions

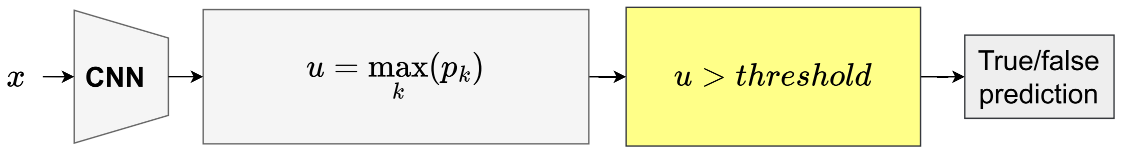


Figure 3.: Overview of the process of thresholding the predictive uncertainty. Here, uncertainty u is measured using the predictive confidence or the softmax score. We evaluate uncertainty by comparing the predictive confidence to the threshold value

1.4. Research objective

The dramatic consequences of FPs hamper the wider adoption of deep neural networks in safety- and/or mission-critical applications. Therefore, preventing FPs is vital to avoid those significant repercussions, such as road accidents, financial loss, or false treatments, and encourage the widespread adoption of deep neural networks. To achieve this objective, we estimated and evaluated the uncertainty associated with predictions, as shown in Figure 2. Particularly, we evaluated the predictive uncertainty to separate TPs and FPs. As shown in Figure 3, evaluating the predictive uncertainty required thresholding it at test time. Herein, a binary classifier (detector) labels predictions as either TP or FP. As shown in Figure 4, we want:

Low uncertainty on TPs, which occurs when a CNN-based classifier assigns an input to the true underlying ground truth class. We refer to the set of all possible inputs that were correctly classified by a CNN-based classifier as *correctly classified in-domain examples*. We want the predictive uncertainty to be low on correctly classified in-domain examples.

High uncertainty on FPs, which occurs when a CNN-based classifier cannot generalize to in-domain examples (misclassified in-domain examples), when an input was perturbed with affine or non-affine transformations (misclassified domain-shift examples), or when an input is drawn from a distribution far from the training distribution (misclassified OOD examples). We refer to the set of all possible inputs that were falsely classified by a CNN-based classifier as *misclassified examples*. We want the predictive uncertainty to be high on misclassified examples.

1.5. Research questions

In this thesis, our main goal is to improve the separability of TPs and FPs by estimating and evaluating the prediction uncertainties of CNN-based classifiers. Particularly, we want the predictive confidence measuring uncertainties to be high for TPs and low for FPs. To achieve this, we require a method that forces CNN-based classifiers to make FPs with low confidence (e.g., [0, 0.5[) and TPs with high confidence (e.g., [0.5, 1]). We therefore formulated the following three research questions:

(1) What method is required to force the predictive confidence of a CNN-based classifier to be high for TPs and low for FPs?

(2) Under what circumstances does the proposed method work?

(3) At what cost does the proposed method help to maintain a low confidence for FPs and a high confidence for TPs?

1.6. Challenges to overcome

To address the first research question, we need to overcome the following challenges.

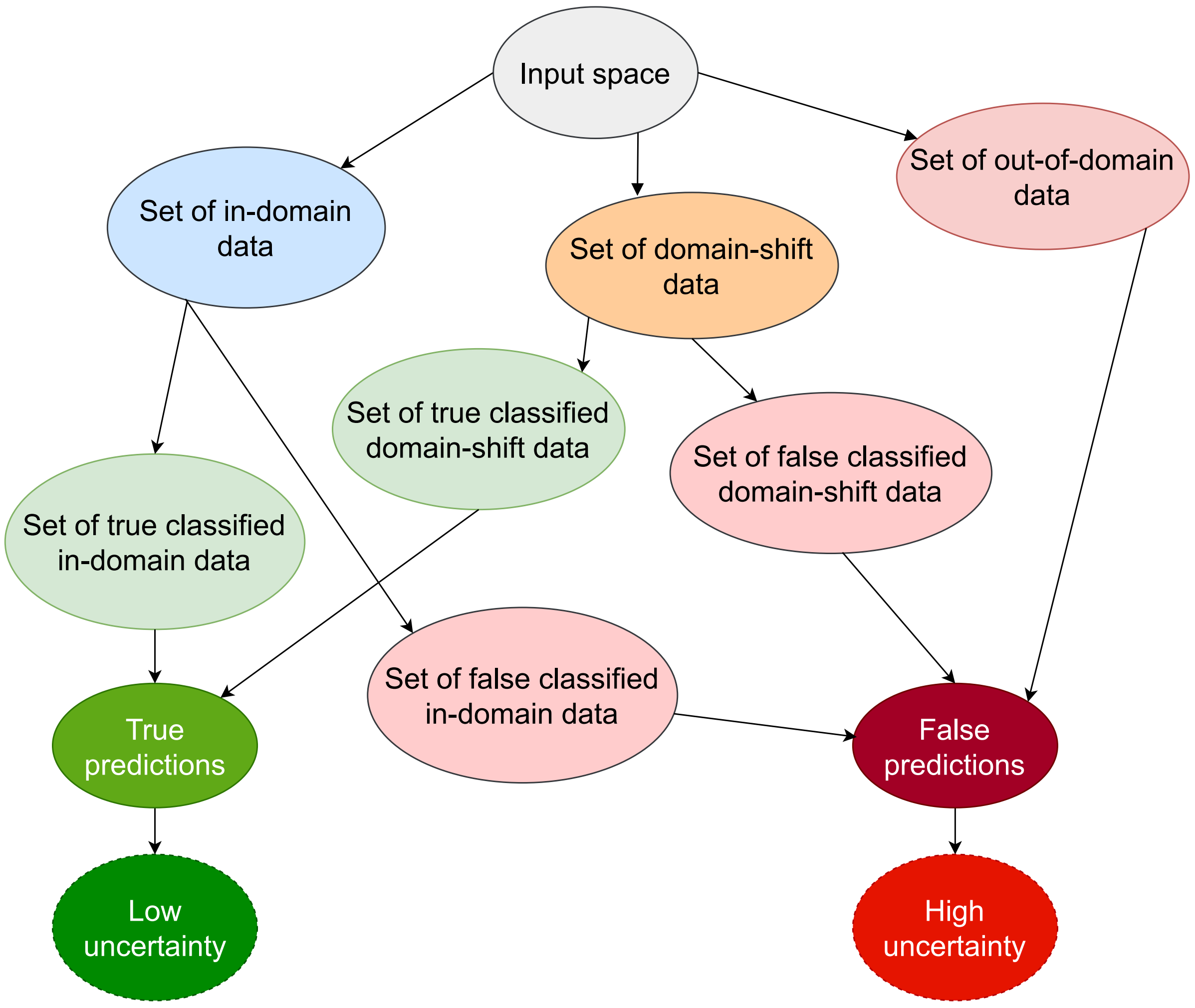
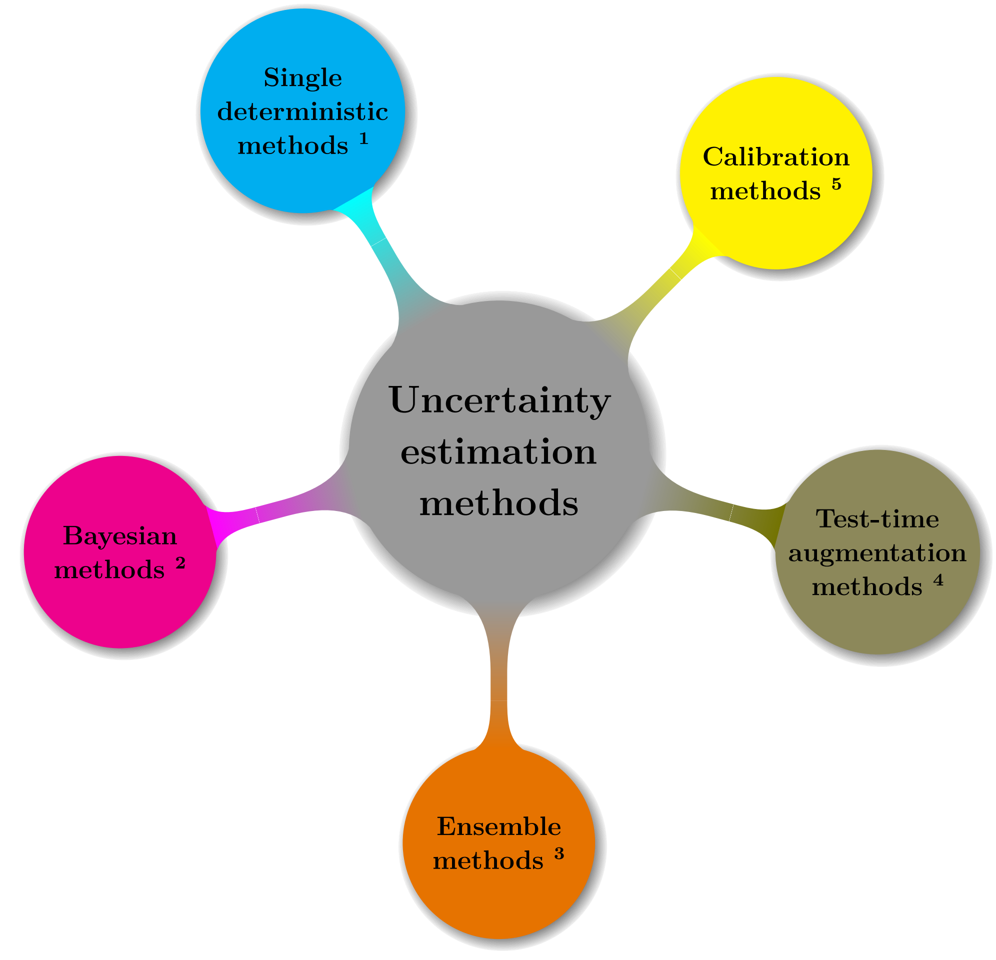

Input space
Set of in-domain data
Set of domain-shift data
Set of out-of-domain data
Set of true classified domain-shift data
Set of false classified domain-shift data
Set of true classified in-domain data
Set of false classified in-domain data
True predictions
False predictions
Low uncertainty
High uncertainty

Figure 4.: Overview of the input space of a classifier and the desired uncertainty profile

The challenge in understanding the functionality of CNNs The parameters of CNNs, which are in the order of thousands to millions, approximate a nonlinear function used to map the given input to the desired output. The nonlinear function was trained using the data from the prepared training dataset, the CNN architecture specified through a large number of hyperparameters (i.e., the number of layers and hidden units, and the types of layers such as convolution, pooling, and fully-connected layers), the training procedure specified through a learning algorithm (e.g., stochastic gradient descent (SGD)), a regularization method, and various hyperparameters (i.e., the number of training iterations, regularization strength, learning rate, and batch size). Since the nonlinear function is shaped by several factors, the learning process with high degree of randomness is no longer sufficiently understood. In addition, the approximation function is not easily interpretable by humans because it is affected by various parameters. Owing to the substantial complexity of CNNs (in terms of number of parameters), their inference processes are also no longer sufficiently understood. Consequently, the results generated were hard to understand or explain, which resulted into two complications. The first is the difficulty to impose some constraints on CNN parameters because we don't know the relevant parameters, which affect predictions and overall functioning of CNNs. Therefore, we treat CNNs as black-boxes because this allows us to focus on the prediction instead of the manner in which the prediction arrives. The second one is the difficulty to trace the source of uncertainties inherent in predictions. In other words, if the source of the uncertainty is a priori unknown to the modeler, then tracing it will be difficult or impossible at all owing to the substantial complexity of CNNs. Therefore, we (only) evaluated uncertainty of well-known sources. Particularly, we used five evaluation data (as shown in Appendix A.6) for different purposes. We used test data to evaluate the classification accuracy and calibration error and used subsets of correctly classified test data, OOD, swap, and noisy data to assess the ability of separating TPs and FPs caused by distribution-shift, structural perturbation, and Gaussian noise.

The challenge in choosing a principle for uncertainty estimation As shown in Figure 5, existing methods to estimate uncertainties adhere to five distinct principles. The first principle underlying *single deterministic methods* is using a single deterministic CNN and placing a distribution over the output for uncertainty quantification or deriving uncertainties from external information such as gradient or from uncertainty measures such as the predictive confidence and/or entropy. The main advantage of the methods following this principle is its computational efficiency in training and inference, because only one CNN has to be trained and evaluated. However, since a single CNN converges to a single local optimum in the solution space [97] and therefore represents a single solution (or opinion), the methods


Single deterministic methods [1]
Calibration methods [5]
Uncertainty estimation methods
Bayesian methods [2]
Test-time augmentation methods [4]
Ensemble methods [3]

[1] [43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55] [2] [56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76] [3] [15, 77, 78, 79, 80, 81, 82, 83] [4] [84, 85, 86, 87, 88, 89, 90] [5] [91, 92, 93, 94, 95, 96, 16, 28]

Figure 5.: Visualization of different methods to estimate uncertainties

following this principle can be very sensitive to the architecture and training procedure used to shape the solution. The second principle underlying *Bayesian methods* is combining the predictions of multiple stochastic CNNs to estimate uncertainties. The main advantage of the methods following this principle is that they evaluate multiple neighboring points within a certain region and therefore include the uncertainties around a single local optimum [97]. However, these methods are sensitive to the prior distribution and the approximate Bayesian inference for estimating the posterior distribution. Moreover, the true nature of the prior and posterior distribution is generally unknown, and specifying a meaningful distribution is challenging [98]. The third principle underlying *ensemble methods* is combining the predictions of multiple deterministic CNNs to estimate uncertainties. The main advantage of the methods following this principle is that they evaluate multiple CNNs representing multiple local optima [97]. However, these methods are computational and memory demanding because we have to store and evaluate multiple CNNs. The fourth principle underlying *test-time augmentation methods* is using a single deterministic CNN and augmenting the input data at test-time to generate several predictions to estimate uncertainties. The main advantage of the methods following this principle is that they use a single CNN to evaluate different simulated poses and/or views of the same object under different simulated environmental conditions such as illumination and lighting. However, for a given problem, the type of augmentation (i.e., simulation of pose, view, or environmental condition) and number of augmentations needed is a priori unknown and the computational cost increases with the number of augmentations. The fifth and last principle underlying *calibration methods* is modifying the optimization algorithm and/or training objective for building CNNs that are inherently calibrated or adjusting the predictions of single or multiple CNNs after the training process to reflect empirical outcomes (accuracy). Understanding the five principles and their various implementations is vital to decide which principle to follow and/or how to combine the existing principles. Correspondingly, we first summarized and discussed the existing methods in our survey paper [99]. One of the main suggestions of our survey paper is to develop methods to estimate uncertainties that combine the strengths of both Bayesian (single-mode exploration) and ensemble (multimode evaluation) principles. This suggestion was applied in studies by Kahn et al. [30]; Lutjens, Everett, and How [31]; and Wilson and Izmailov [74] in building an MMCD that utilized Bayesian inference in ensemble members. Specifically, the MMCD utilized MCD (the most widely used approximation for Bayesian inference) in ensemble members. Inspired by the MMCD, we developed MCA, a method that combines the strengths of ensemble and MCD. Similar to MMCD, MCA evaluates multiple solutions because it relies on an ensemble of CNNs and explores the uncertainty around the individual solutions thanks to features averaging, as discussed in Section 4.2.3.

The need for identical experimental setups According to Sinha et al. [100], one of the challenges in machine learning research is ensuring that empirical results from previous works are reproducible. The reproducibility of empirical results is a necessary step not only to verify the reliability of research findings but also to promote fair comparison of the existing methods. However, it is sometimes not feasible to reproduce all the experiments in a paper owing to several factors such as private datasets, computation availability for extensive tuning, the requirement of nonstandard compute infrastructure, or incorporation of sufficiently many baselines. Consequently, each paper code base (if available) differs in experimental settings, making it difficult to compare existing methods within a common benchmark. Therefore, we reimplemented all related methods, which aids in understanding the related methods and sheds light on the aspects of the implementation that could affect the results of the related methods.

1.7. Solution approaches

To address the first research question, we developed MCA and compared it to related methods, such as baseline (single CNN), MCD, ensemble of CNNs, and MMCD.

To address the second research question, we assessed the performance of MCA and related methods on four datasets (CIFAR10, Fashion-MNIST, MNIST, and GTSRB) to evaluate their ability to perform on tasks (datasets) with various difficulties. We expected the task difficulty to affect the performance of MCA and related methods because some datasets (e.g., GTSRB) have more noise inherent in their samples than others (e.g., MNIST). In addition, samples of some datasets (e.g., CIFAR10) are more difficult to learn than others (e.g., MNIST). We also assessed the performance of MCA and related methods on CNNs of three architectures (VGGNet, DenseNet, and ResNet) to evaluate their ability to perform on architectures with various design and configurations. We expected the architecture to affect the performance of MCA and related methods. This is because the architecture conditions the manner in which information are propagated from the input to subsequent layers and different architectures will result in different gradient calculations and therefore to different solutions. Further, we evaluated the impact of applying logit instead of probability averaging and the impact of reducing the regularization strength on the performance of MCA and related methods. This is due to the fact that logit averaging and the reduction of the strength of regularization can reduce the level of inductive biases inherent in CNNs and influence the confidence of MCA and related methods.

To address the third research question, we evaluated not only the ability of MCA and related methods to separate TPs and FPs but also the classification accuracy, calibration error, and inference time. We expected a good method for uncertainty estimation to preserve the classification accuracy on the test data, to exhibit a low calibration error on the test data, and to maintain a high degree of confidence for TPs and a low degree of confidence for FPs.

Table 1.: Summary of characteristics of MCA and related methods, such as baseline, MCD, ensemble, and MMCD. The symbols S and M (with $M << S$) refer to the number of stochastic forward passes in MCD and the number of members in MCA and ensemble, respectively

Properties	**Baseline**	**MCD**	**MMCD**	**Ensemble**	**MCA**
Nature of CNNs	Deterministic	Stochastic	Stochastic	Deterministic	Deterministic
Number of CNNs trained and evaluated	1	1	M	M	M
Number of feature extractors	1	1	M	M	M
Number of discriminators	1	1	M	M	M
Number of features evaluated by a single discriminator	1	S	S	1	M
Evaluated uncertainty associated with extracted features?	No	Yes	Yes	No	Yes
Number of predictions to generate	1	S	$M \cdot S$	M	M^2

1.8. Results

Table 1 presents some characteristics of MCA and related methods. Herein, we can derive that MCA is deterministic similar to baseline and ensemble, evaluates multiple CNNs similar to ensemble and MMCD, and explores the uncertainty associated with extracted features similar to MCD and MMCD, which are both stochastic. Experimental results showed improvement of MMCD and MCA over related methods, such as baseline, MCD, and ensemble. Specifically, similar to MMCD, MCA can preserve the accuracy of the underlying ensemble increasing the baseline accuracy, which can only be preserved by MCD. In addition, MMCD and MCA can improve the separability of TPs and FPs by increasing the calibration error and the inference time. However, applying logit instead of probability averaging in MCA and related methods or reducing the strength of regularization can reduce the calibration error consequently affecting the separability between TPs and FPs. Hence, there is a tradeoff between improving the calibration and improving the separability between TPs and FPs. Although the performance of all methods heavily relies on the dataset and/or architecture, MCD and MMCD are more sensitive to the dataset and/or architecture because of the sensitivity of the predefined distribution from which masks were drawn for feature sampling. Overall, the results showed that the developed MCA is an alternative to MMCD. Assuming that MMCD and MCA are well-specified for a given dataset and architecture, they can exhibit similar performance because they have the same purpose and underlying principle. However, the design process of MMCD can be more time-consuming and costly than that of MCA. This is because MMCD requires the specification of the predefined (prior) distribution from which masks were drawn for feature sampling, while MCA relies on features extracted from ensemble members for feature averaging. Moreover, the outcome of feature averaging operations is deterministic while that of feature sampling operations is stochastic. Therefore, feature sampling required more samples than feature averaging to evaluate the uncertainties associated with the extracted features. Because of all the advantages of MCA over MMCD, we can use MCA in lieu of MMCD for uncertainty estimation.

1.9. Contribution

The added value of this thesis is four-fold:

(1) We summarized and discussed existing methods for uncertainty estimation in our survey paper [99].

(2) We developed MCA for uncertainty estimation and compared it to related methods (i.e., baseline, MCD, ensemble of CNNs, and MMCD). The results of this work were published [101] and presented at the 2022 International Joint Conference on Robotics and Artificial Intelligence (JCRAI 2022), which took place in Chengdu, China, on October 14-16, 2022.

(3) We studied the impact of applying logit instead of probability averaging on the properties of MCA and related methods. This is based on the results of our research paper [102] published and presented at the 31st International Joint Conference on Artificial Intelligence and the 25th European Conference on Artificial Intelligence (IJCAI-ECAI-22) Workshop in Artificial Intelligence Safety (AISafety 2022), which took place in Vienna, Austria, on July 23-29, 2022.

(4) We studied the impact of reducing the strength of regularization on the properties of MCA and related methods. The results of this work were published [103] and presented at the 2022 5th International Conference on Algorithms, Computing and Artificial Intelligence (ACAI 2022), which took place in Sanya, China, on December 23, 2022.

1.10. Outline

This thesis is structured as follows.

Chapter 2 presents some fundamentals. We start this chapter with a definition of a classifier followed by the presentation of a CNN-based classifier and its core elements. We continue by analyzing uncertainties in a CNN-based classifier with an emphasis on sources and types of uncertainty. We end this chapter by presenting the approach to measure uncertainties and highlighting the poor quality of the uncertainties obtained with a single CNN (baseline).

Chapter 3 summarizes and discusses ensemble and Bayesian methods, which build the foundations of the developed MCA. This chapter is structured in two main sections. The first section presents the definition of a Bayesian neural network and explains the underlying principles governing a Bayesian neural network. Afterward, we present the inherent properties of a Bayesian neural network and discuss some popular variational inference techniques for approximating the posterior distribution of a Bayesian neural network. We end the first section by specifying the settings for building a Bayesian neural network. The second section presents the definition of an ensemble and then explains the underlying principles governing an ensemble and its inherent properties. We continue by analyzing the challenges faced when building an ensemble and discuss some popular existing methods to construct an ensemble. We end the second section by discussing the ensemble method in the context of uncertainty estimation and specifying some parameters for empirical evaluation of an ensemble.

Chapter 4 presents the developed MCA and theoretically compares it with related methods, such as baseline (single CNN), MCD, ensemble, and MMCD. We start this chapter by presenting the related methods and their shortcomings. We continue by presenting MCA and the feature averaging approach, which is at the core of MCA. Afterward, we highlight the importance of feature sampling (inherent in MCD and MMCD) and feature averaging (inherent in MCA) and compare them. We end this chapter by summarizing the results of the theoretical comparison of MCA and related methods and highlighting the implications of the results.

Chapter 5 empirically compares the developed MCA and related methods. We start this chapter by comparing the accuracy and calibration error of MCA and related methods. We continue by comparing the ability of MCA and related methods to separate TPs and FPs. We end this chapter by summarizing the results of the empirical comparison of MCA and related methods and highlighting the implications of the results.

Chapter 6 presents the impact of applying logit instead of probability averaging in MCA and related methods. We start this chapter by stating a rationale for applying logit instead of probability averaging. We continue by describing the process of applying logit instead of probability averaging in MCA and related methods. Afterward, we present the impact of applying logit instead of probability averaging on the accuracy and calibration error of MCA and related methods and on the ability of MCA and related methods to separate TPs and FPs. We end this chapter by summarizing the results of the impact of applying logit instead of probability averaging in MCA and related methods and highlighting the implications of the results.

Chapter 7 investigates the impact of reducing the regularization strength. We start this chapter by giving a rationale for reducing the regularization strength. We continue by describing the process of reducing the regularization strength. Afterward, we study the impact of reducing the strength of regularization on the classification accuracy and calibration error of MCA and related methods and on the ability of MCA and related methods to separate TPs and FPs. We end this chapter by summarizing the results of the impact of reducing the regularization strength and highlighting the implications of the results.

Chapter 8 summarizes a rationale and main objective of this thesis followed by the challenges we faced. Afterward, it restates the research questions and the methodologies used to address them. Finally, it discusses the results and their implications.

Chapter 9 concludes the thesis by summarizing the results, making recommendations, and highlighting limitations and future work directions.

2. Fundamentals

Herein, we assume that the reader is already familiar with the concepts of traditional deep learning such as artificial neural networks, training algorithms, supervision strategies, and loss functions. The unfamiliar reader is referred to studies by Jain, Mao, and Mohiuddin [104], Janocha and Czarnecki [105], and Shrestha and Mahmood [106] to learn these basic concepts.

2.1. Definition of a classifier

In the context of image classification, let the training data $D_{train} = \{x_i \in \mathbb{R}^{H\times W\times C}, y_i \in U^K\}_{i\in[1,N]}$ be a realization of independently and identically distributed random variables $(x, y) \in X \times Y$, where x_i denotes the i^{th} input and y_i is its corresponding one hot encoded class label from the set of standard unit vectors of $\mathbb{R}^K$, U^K. The symbols X and Y denote the input and label spaces, respectively. The symbol $H \times W \times C$ denotes the dimension of input images, where H, W, and C refer to the height, width, and number of channels, respectively. The symbols K and N denote the numbers of possible output classes and samples within the training data, respectively.

A classifier is a linear or nonlinear function f that maps input images $x_i \in \mathbb{R}^{H\times W\times C}$ to class labels $y_i \in U^K$, that is

$$f : x_i \in \mathbb{R}^{H\times W\times C} \to y_i \in U^K; \qquad f(x_i) = y_i \ . \tag{2.1}$$

Given a new data sample $x \in \mathbb{R}^{H\times W\times C}$, a classifier predicts the corresponding target $\hat{y} = f(x)$. Many linear (e.g., support vector machine) and nonlinear (e.g., neural network) classifiers proposed in the literature are summarized and discussed in the survey paper of Yuan et al. [107]. Since linear classifiers operate directly on data in the original input space, they cannot discover all hidden patterns in the given input. On the contrary, nonlinear classifiers operate on data mapped to a higher dimensional space and can therefore discover more hidden patterns in the given input. Consequently, nonlinear classifiers perform (in terms of accuracy) better than the linear ones. However, owing to the emergence of large datasets, increasing computational power, tremendous advances in deep learning, and the success of the submission of Krizhevsky et al. [108] to the ImageNet large-scale visual recognition challenge [109], CNNs, a special type of neural network architectures, have become the standard for modern classifiers. This is mainly attributed to the fact that CNNs do not rely on features engineered by human experts because they automatically learn how to extract and discriminate features at training.

2.2. A CNN-based classifier

Several architectures of CNNs such as AlexNet [108], VGGNet [110], ResNet [111], Wide-ResNet [112], ResNeXt [113], and DenseNet [114] have been proposed in the literature to improve the accuracy of image classification or to reduce computational costs. Rawat and Wang [115], Khan et al. [116], and Alzubaidi et al. [117] summarized and discussed all these architectures, which include multiple convolution, pooling, and fully connected layers. These layers form the building blocks of a CNN-based classifier.

2.2.1. Building blocks of a CNN-based classifier

A CNN-based classifier comprises convolution, pooling, and fully connected layers (see Figure 6). These layers are cascaded together in a hierarchical manner to form a deep structure so that the first layer extracts a set of primitive patterns from raw pixels of the input image, the second layer extracts patterns from the output feature maps of the first layer, the third layer extracts patterns from the output feature maps of the second layer, and so on. Consequently, more and more abstract representations are learned as the information flow from input to output.

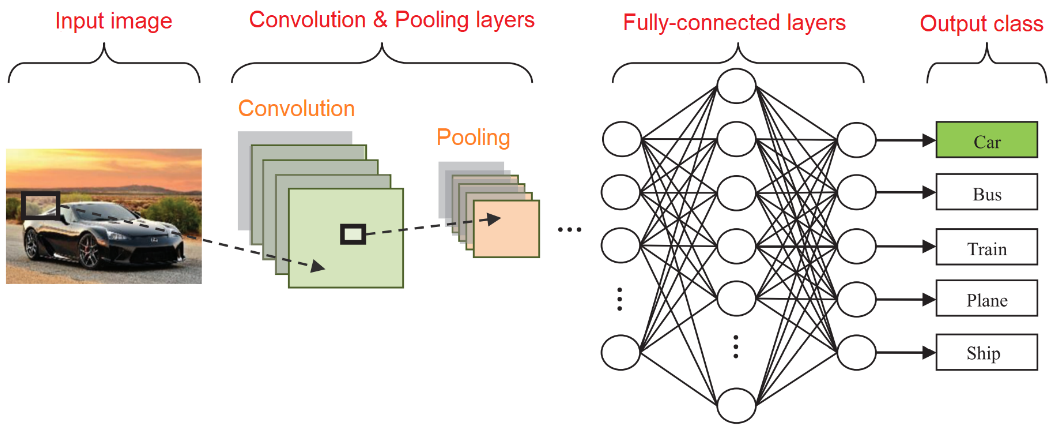


Figure 6.: Overview of a CNN architecture for image classification [115]

Convolution layers are feature extractors that learn the representations of their inputs. They extract features by convolving a convolution kernel[1] with the input (see

[1]A convolution kernel (also referred to as filter or mask) of a convolution layer represents the set

Figure 7). Neurons of convolution layers are arranged into feature maps and each neuron has a receptive field that depends on the size of the convolution kernel. Neurons within the same feature map have the same receptive field or use the same convolution kernel to perform the same operation on different parts of the image and to therefore extract the same feature. This procedure encourages weight sharing. Nonetheless, neurons of different feature maps within the same convolution layer use different sets of trainable weights and therefore extract different features at the same location. Particularly, given F convolution kernels, F different features can be learned, yielding to F feature maps. Mathematically, the operation performed by convolution neurons within a feature map is a discrete convolution. Let $l[i,j]$ denote the convolution kernel of size $I \times J$ with $0 \leq i < I$ and $0 \leq j < J$ and $d[i,j]$ denote a small location of the input image of size $I \times J$. The convolution of $l[i,j]$ with $d[i,j]$ yields the output feature map $o[i,j]$ defined as

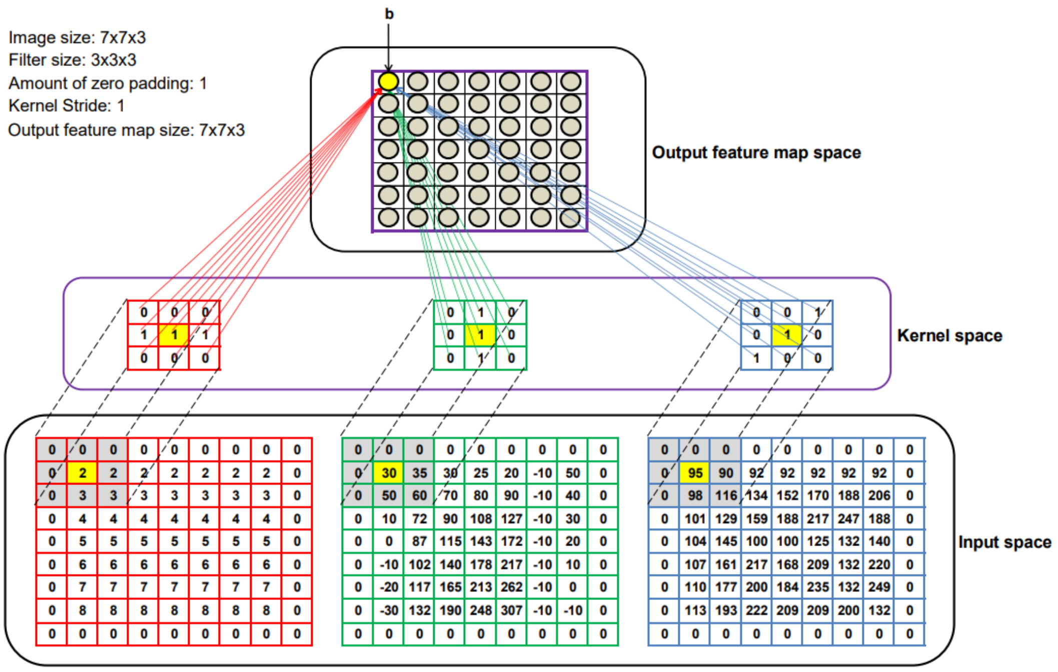
$$o[i,j] = l[i,j] * d[i,j] = \sum_{\widetilde{i}=1}^{I-1} \sum_{\widetilde{j}=1}^{J-1} d[i,j] \cdot l[i-\widetilde{i}, j-\widetilde{j}] \ ,$$

where $*$ is the (linear) convolution operator. *Nonlinear activation functions* are placed directly after convolution operations and perform element-wise operations on each output feature map. They facilitate the extraction of nonlinear features. In 2011, Glorot et al. [118] found that the rectified linear unit (RELU) activation function can perform better than conventional ones such as sigmoid and hyperbolic tangent. A year later, Krizhevsky et al [108] popularized the RELU activation function, which is defined as

$$RELU(a) = max(a,0)$$

where a is the output of a neuron within a feature map. The RELU activation function retains only the positive part of the input ($a > 0$) and reduces the negative part of the input ($a \leq 0$) to zero. Although this function can lead to faster convergence by accelerating training, it can negatively affect the training by blocking gradient backpropagation because the gradient is zero whenever a neuron is not active or saturates [118]. Therefore, many activation functions such as LRELU [119], PRELU [120], SRELU [121], and ELU [122] were proposed in the literature. However, despite its drawbacks, RELU remains the most widely used activation function, especially in the context of CNNs.

of adjustable weights used by all neurons within the same feature map. Convolution kernels are used to represent different processes such as smoothing, blurring, sharpening, and edge detection. Conventionally, symmetric convolution kernels are used, meaning that the origin of the convolution kernel is usually the center element, which is well defined when I and J are odd.



0	0	0
1	1	1
0	0	0

0	1	0
0	1	0
0	1	0

0	0	1
0	1	0
1	0	0

0	0	0	0	0	0	0	0	0
0	2	2	2	2	2	2	2	0
0	3	3	3	3	3	3	3	0
0	4	4	4	4	4	4	4	0
0	5	5	5	5	5	5	5	0
0	6	6	6	6	6	6	6	0
0	7	7	7	7	7	7	7	0
0	8	8	8	8	8	8	8	0
0	0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	0	0
0	30	35	30	25	20	-10	50	0
0	50	60	70	80	90	-10	40	0
0	10	72	90	108	127	-10	30	0
0	0	87	115	143	172	-10	20	0
0	-10	102	140	178	217	-10	10	0
0	-20	117	165	213	262	-10	0	0
0	-30	132	190	248	307	-10	-10	0
0	0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	0	0
0	95	90	92	92	92	92	92	0
0	98	116	134	152	170	188	206	0
0	101	129	159	188	217	247	188	0
0	104	145	100	100	125	132	140	0
0	107	161	217	168	209	132	220	0
0	110	177	200	184	235	132	249	0
0	113	193	222	209	209	200	132	0
0	0	0	0	0	0	0	0	0

Figure 7.: Illustration of a convolution operation with a convolution kernel size of 3×3 and an image size of 7×7. Convolution kernels always extend the full depth of the input image. Starting from the top-left corner of the input, the convolution kernel is moved from left to right with a stride of 1. Once the top-right corner is reached, the convolution kernel is moved with the same stride in downward direction, and again the kernel is moved from left to right corner while keeping the stride unchanged. This process is repeated until the kernel reaches the bottom-right corner. Zero padding is applied to fill the border of the image

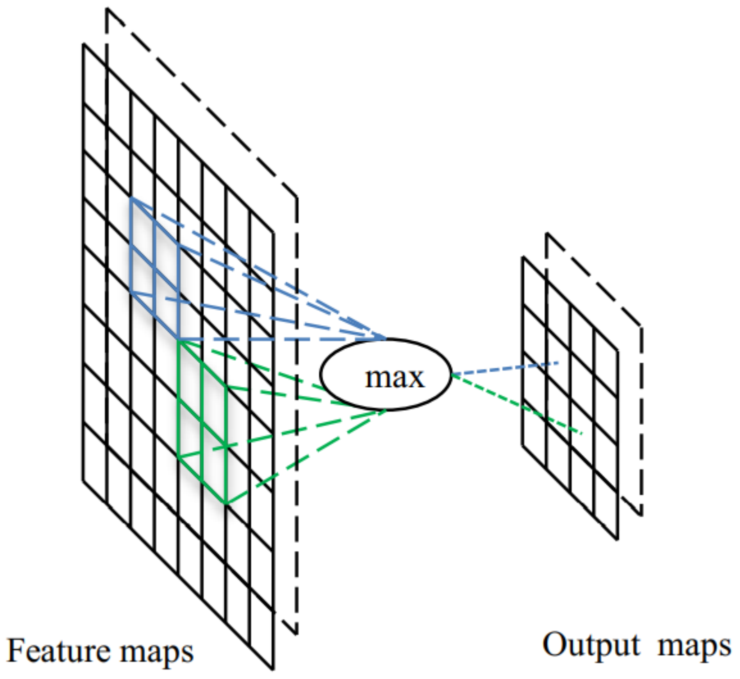


Figure 8.: Illustration of a max pooling operation with a pooling region size of 2×2 and a feature map size of 8×8. Starting from the top-left corner of the input, the pooling region is moved from left to right with a stride of 2. Once the top-right corner is reached, the pooling region is moved with the same stride in a downward direction, and again the region is moved from left to right corner while keeping the stride unchanged. This process is repeated until the region reaches the bottom-right corner. This figure is adapted from Guo et al. [123].

Pooling layers operate independently over each feature map and transform joint feature representations into more compact ones that preserve relevant information and discard irrelevant ones. They reduce the spatial resolution or dimension of feature maps and make extracted features invariant to small changes in position and appearance [124, 125]. Applying pooling is dividing the input into nonoverlapping two-dimensional regions and applying a pooling operation such as max pooling, which is the most widely used. Given a pooling region r within a set of activations $\{a_1, a_2, \ldots, a_{|r|}\}$, where $|r|$ denotes the cardinality [2] of r, the region output o_r obtained after applying the max-pooling operation is defined as

$$o_r = max(a_1, a_2, \ldots, a_{|r|}).$$

As an example, Figure 8 presents how a feature map with a size of 8×8 is reduced to a compact feature map with a size of 4×4 after applying max pooling with a pooling region with a size of 2×2 and stride of 2.

Fully connected layers are the last layers in the hierarchy of CNNs (see Figure 6). Usually, three fully connected layers are used to convert two-dimensional feature maps into one-dimensional vector. These layers further process features to make them more classifiable. Unlike pooling and convolution layers that perform local operations, fully connected layers perform global operations. In other words, the output of a fully connected neuron depends on all the elements of the input vector. While the first two fully connected layers can have different number of neurons with RELU activation function, the last fully connected layer includes K (representing the number of possible classes) output neurons with softmax activation function. The softmax activation function normalizes its inputs (referred to as logits and interpreted as found evidences for possible classes [126]) to produce discrete probabilities p_k (with $k = 1, \ldots, K$ and $\sum_{k=1}^{K} p_k = 1$) representing the probability that the input belongs to the class associated with the k^{th} output neuron. The discrete probabilities p_k are usually interpreted as the model confidence to its prediction. Given the logits $z = \begin{bmatrix} z_1 \\ . \\ . \\ . \\ z_K \end{bmatrix}$, the softmax function estimates $p = \begin{bmatrix} p_1 \\ . \\ . \\ . \\ p_K \end{bmatrix}$

[2]The cardinality of a finite set r denotes the number of its elements.

as

$$p = softmax(z) = \frac{1}{\sum_{k=1}^{K} \exp(\lambda z_k)} \begin{bmatrix} \exp(\lambda z_1) \\ . \\ . \\ . \\ \exp(\lambda z_K) \end{bmatrix}, \lambda > 0 \qquad (2.2)$$

where λ refers to the inverse temperature constant [127]. According to Gao and Pavel [127], when $\lambda = 1$, Equation 2.2 defines the standard softmax function. Given the discrete probabilities p_k, we can compute the predicted class label $\hat{y}$ as

$$\hat{y} = \arg\max_k (p_k). \qquad (2.3)$$

2.2.2. Supervised learning of a CNN-based classifier

Given a training data $D_{train} = \{x_i \in \mathbb{R}^{H \times W \times C}, y_i \in [0,1]^K]\}_{i \in [1,N]}$, we want to train a CNN that maps x_i to y_i. We start by initializing the parameters (biases and weights) with random values or using any other existing methods for parameter initialization such as transfer learning, which is summarized and discussed by Zhuang et al. [128] and Weiss, Khoshgoftaar, and Wang [129]. Afterward, we fed x_i into the CNN that estimates the discrete probability vector $\hat{p}_i$. The desired category for x_i should have the highest probability, but this is unlikely to happen before convergence at training [125]. Consequently, an objective (loss or cost) function that measures the error between $\hat{p}_i$ and y_i is estimated. Traditionally, the cross-entropy loss is used in conjunction with the softmax activation function [108, 110, 111, 114]. However, CNNs are trained using minibatches of size β. This consists of showing β training examples to the CNN and estimating the average cross-entropy loss as

$$L = \frac{1}{\beta} \sum_{i=1}^{\beta} L_i = \frac{1}{\beta} \sum_{i=1}^{\beta} y_i log(\hat{p}_i).$$

According to the backpropagation algorithm [130], to minimize the average cross-entropy loss (or reduce prediction errors), the CNN will modify its internal adjustable parameters $\theta = (w_1, ..., w_{|\theta|})$ by estimating the gradient vector $\nabla L = \frac{\partial L}{\partial \theta}$ that, for each internal adjustable parameter, indicates by what amount the error would increase or decrease. According to optimization algorithms such as SGD [131], the parameter θ is adjusted in the opposite direction to the gradient, that is,

$$\theta_{t+1} = \nu \theta_t - \gamma \nabla L,$$

where γ and ν are the learning rate and momentum, respectively. The process of minimizing the average cross-entropy loss is repeated for many batches of training data until the average cross-entropy loss stops decreasing. This indicates that the CNN has converged to the optimal local minima or has found the optimal configuration of parameters for performing inference after deployment.

2.3. Uncertainty in a CNN-based classifier

2.3.1. Uncertainty sources

As summarized and discussed in our survey paper [99], the following factors cause uncertainties in CNNs.

Variability in real-world situations Environmental changes, such as variations in temperature and illumination, affect the physical appearance of objects, in differences in size, shape, color, and background clutter. For instance, plants look very different after rain than after a drought. Cultural biases also affect the meaning given to physical objects. For instance, German traffic signs are different from Chinese ones, but such different signs can have the same meaning. *Distribution (or domain) shift* occurs when real-world situations are not similar to training ones. Distribution-shift can lead to FPs of CNNs owing to the perturbation of the given object size, shape, color, or background clutter, stemming from environmental changes, or the lack of knowledge about the given object, stemming from cultural biases. In summary, *variabilities in real-world situations can cause distribution-shift, which in turn can cause FPs.*

Errors and noise inherent to measurement systems The sensing model (or data acquisition system) for acquiring inputs for CNNs, e.g., the camera used to acquire images, is prone to several errors such as limited information in the measurements. The lack of sufficient information in the measurements can be attributed to the low resolution of the camera or the large distance between the physical object and camera, which results in an object with a small size in the given input image. Moreover, sensor noise inherent in the measurements and stemming from changes in temperature, illumination, motion, and mechanical stress, etc. can further limit the amount of information in the measurements. The lack of sufficient information in the measurements results in *distribution- (or domain-) shift.* Distribution-shift can lead to FPs of CNNs owing to the lack of knowledge about the given object. Overall, *errors and noise inherent in the measurement systems can cause distribution-shift, which in turn can cause FPs.*

Errors in architecture specification and training procedure As modelers, we have multiple alternative learning algorithms summarized and discussed by Pouyanfar et al.

[132] and Shrestha and Mahmood [106] and architectures summarized and discussed by Rawat and Wang [115], Khan et al. [116], and Alzubaidi et al. [117] to choose from. In addition, there are various hyperparameters for architecture specification (e.g., number of features, number of layers, kernel and pooling size, and input size) or for training specification (e.g., batch size, learning rate, momentum, and regularization parameters) to tune. This large spectrum in design choice makes the task of choosing the best architecture and finding the best settings of parameters challenging and error-prone. Usually, popular learning algorithms, architectures and hyperparameters are used for a given problem (dataset) with no guarantee that the design choices are the optimal ones. Consequently, errors in the architecture specification and training procedure caused by our ignorance in optimal design choices can limit the ability of CNNs in extracting domain-specific knowledge during training. This can cause CNNs to not properly learn objects of some specific classes during training and to therefore classify them with a low confidence or to even misclassify them. In summary, *errors in the architecture specification and training procedure can limit the ability of CNNs to extract domain-specific knowledge, resulting in low confident predictions or FPs.*

Errors caused by unknown data A CNN is usually trained to classify a limited number of objects, for example, classification of three objects: bus, car, and truck. However, in the real-world, the number of possible objects is unlimited. This means that images including unknown objects will be processed by the CNN at inference. For instance, a CNN trained to classify images of three objects (bus, car, and truck) can try to classify, for example, an image of a pedestrian at inference. Herein, the lack of knowledge about the pedestrian object will hinder the CNN to extract the meaningful features to classify a pedestrian. An alternative explanation is the absence of an output neuron, which maps the input to the pedestrian class. Therefore, the image of a pedestrian will be mapped to one of the three possible classes (bus, car, or truck), which results in FPs. In the literature, unknown data are also referred to as OOD data. In summary, *the lack of domain-specific knowledge about unknown (OOD) data can cause FPs.*

2.3.2. Uncertainty type

As explained above, factors such as the variability in real-world situations, errors, and noise inherent in the measurement systems can cause a distribution-shift, which in turn can cause FPs. Herein, the cause of the false prediction is inherent in the input data. Therefore, we refer to the uncertainty stemming from errors inherent in the input data as *data-dependent uncertainty* (also referred to as *statistical or aleatoric uncertainty* [133]). Data uncertainty is attributable to the lack of sufficient information to represent the real-world object in measurement data. Hence, the model cannot classify the data accordingly because the missing information is a priori unknown to the model and modeler. Since data uncertainty occurs in the presence of distribution-shift examples (inputs drawn from a shifted version of the training data distribution), it can be referred to as *domain-shift uncertainty* based on the source of the input data distribution.

In addition, other factors such as errors in the architecture specification and training procedure can limit the ability of CNNs to extract domain-specific knowledge, which results in low confident predictions or FPs. Herein, the cause of the low confident predictions or FPs is inherent in the CNN. We therefore refer to the uncertainty stemming from errors inherent in the CNN as *model uncertainty* (also referred to as *systemic or epistemic uncertainty* [133]). Model uncertainty can further be subdivided into *structural uncertainty* (wherein we have significant doubts that the model is even structurally correct) and *parametric uncertainty* (wherein we believe that the structure of the model is correct but are uncertain about the correct values of model parameters). Structural uncertainty and parametric uncertainty can be reduced by improving the architecture specification and improving the training procedure, respectively. Since model uncertainty occurs in the presence of in-domain examples (input drawn from a data distribution assumed to be equal to the training data distribution), it can be referred to as *in-domain uncertainty* based on the source of the input data distribution.

Finally, the lack of domain-specific knowledge about unknown data can cause FPs. Herein, the cause of the FPs is inherent in the distribution (or domain) of the input data that are incompatible with that of the training data. We therefore refer to the uncertainty stemming from the distributional incompatibility between the training and input data as *distributional uncertainty*. Since distributional uncertainty occurs in the presence of out-of-distribution or OOD examples (inputs drawn from the space of unknown data), it can be referred to as *OOD uncertainty* based on the source of the input data distribution.

2.3.3. Uncertainty estimation

Based on the source of uncertainty, we can distinguish between model, data, and distributional uncertainties, which occur in the presence of in-domain, domain-shift, and OOD examples, respectively, as shown in Figure 10. Regardless, we want to estimate the uncertainty associated with a prediction, that is, the *predictive uncertainty or total uncertainty*, that summarizes the model, data, and distributional uncertainties. Therefore, we can estimate the predictive uncertainty u (or predictive confidence) based on the discrete probabilities p_k as

$$u = \max_k (p_k) \ . \tag{2.4}$$

The predictive confidence measures the probability mass assigned to the predicted class label and can be used as a measure of uncertainty. However, it is widely discussed that a CNN is often overconfident and its predictive confidence is often poorly calibrated, leading to inaccurate uncertainty estimates [134, 27, 45, 47]. For instance, Figure 9 presents how a CNN falsely classifies rotated MNIST digits with a high confidence (low uncertainty). Therefore, to improve the predictive confidence in a CNN-based classifier, we develop MCA.

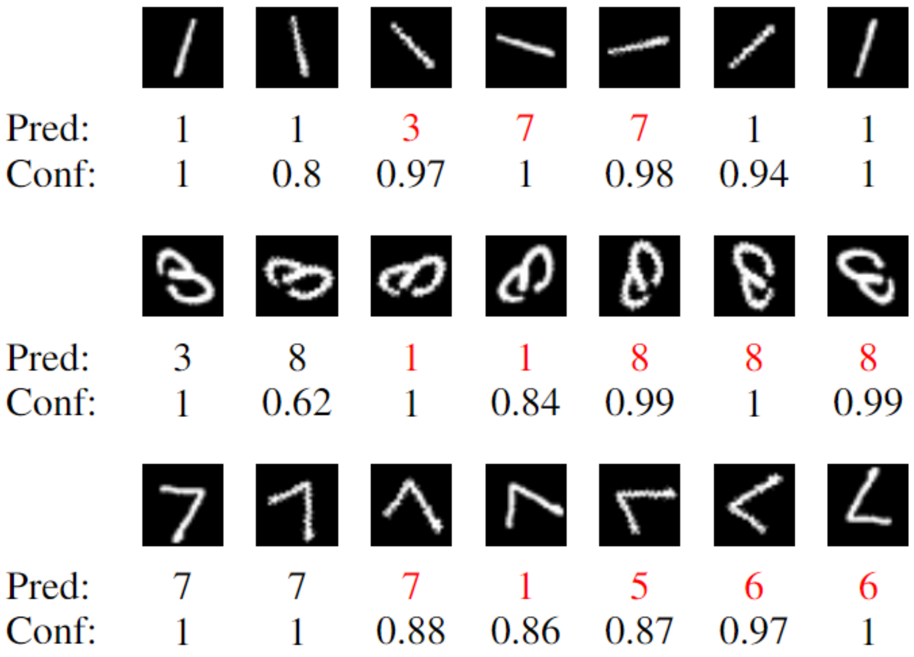

Pred: 1 1 3 7 7 1 1
Conf: 1 0.8 0.97 1 0.98 0.94 1
Pred: 3 8 1 1 8 8 8
Conf: 1 0.62 1 0.84 0.99 1 0.99
Pred: 7 7 7 1 5 6 6
Conf: 1 1 0.88 0.86 0.87 0.97 1

Figure 9.: Predictions obtained via a LeNet network trained on handwritten digits (from 0 to 9) of MNIST and evaluated on different rotations of test samples. For some rotations, the network exhibits a high confidence on the false class owing to confusion (e.g., 3 is confused with 8) or unknown representations. These examples show how a CNN-based classifier can generate overconfident FPs under data distribution shifts. This figure is adapted from Gawlikowski et al. [99]

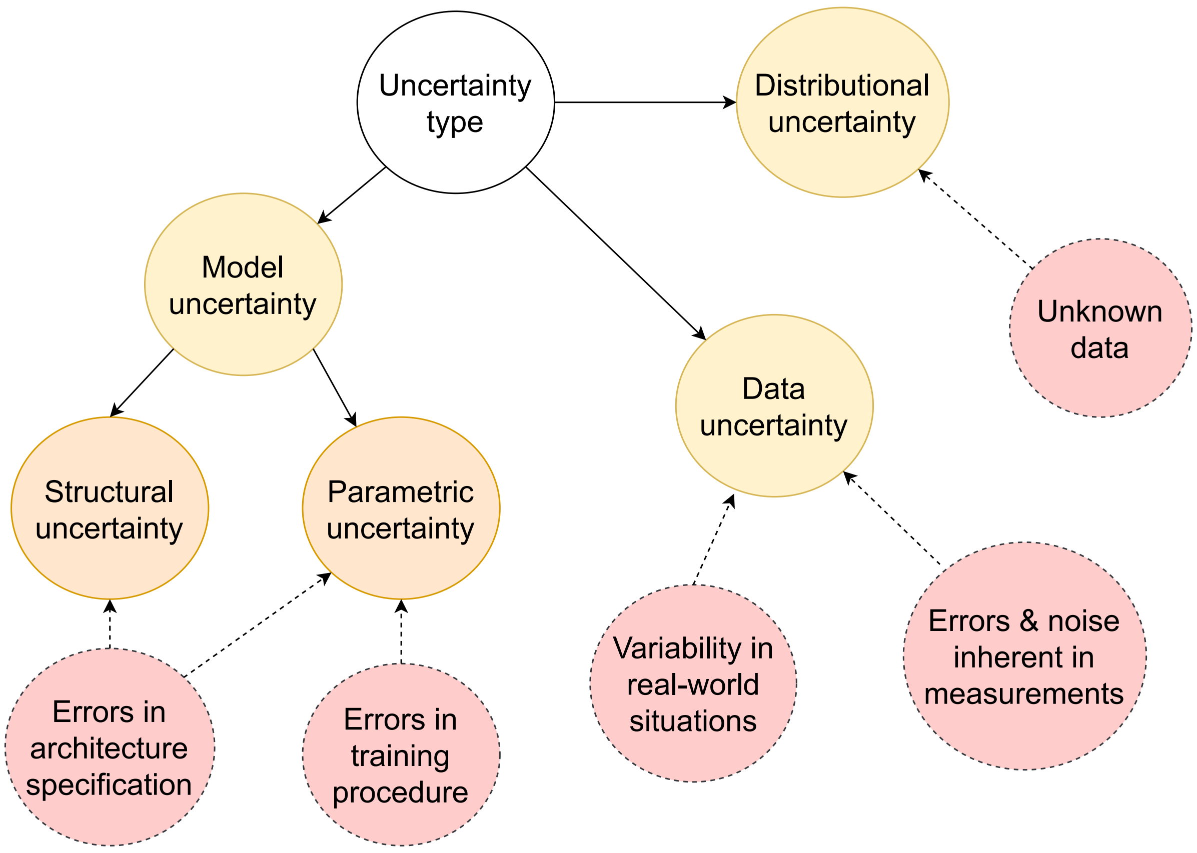

Uncertainty type
Distributional uncertainty
Model uncertainty
Data uncertainty
Unknown data
Structural uncertainty
Parametric uncertainty
Errors in architecture specification
Errors in training procedure
Variability in real-world situations
Errors & noise inherent in measurements

Figure 10.: Overview of uncertainty types and sources

3. Related works

Works related to ensembles and Bayesian neural networks build the foundations of the developed MCA. Therefore, we summarize and discuss only these works in this thesis. Other works related to single deterministic, test-time data augmentation, and calibration methods for uncertainty estimation were summarized and discussed in our survey paper [99].

3.1. Bayesian neural networks

3.1.1. Definition of a Bayesian neural network

Contrary to a point estimate (deterministic) neural network, a Bayesian neural network places prior probability distributions over its parameters (weights). In other words, neuron weights of a Bayesian neural network are random variables, which means that neuron activations and model outputs are also random variables. Given a neural network f with a prior distribution $p(\theta)$ placed over the neural network weights θ, the uncertainty or degree of belief with respect to weights is achieved by inferring (or updating) the prior distribution after encountering data to yield a posterior distribution $p(\theta|D_{train})$ encoding the model uncertainty. According to Bayes theorem [135], $p(\theta|D_{train})$ is defined as

$$p(\theta|D_{train}) = \frac{p(D_{train}|\theta)p(\theta)}{p(D_{train})} \propto p(D_{train}|\theta)p(\theta), \tag{3.1}$$

where the normalization constant $p(D_{train})$ is called the model evidence and is defined as

$$p(D_{train}) = \int p(D_{train}|\theta)p(\theta)d\theta, \tag{3.2}$$

where $p(D_{train}|\theta)$ is the likelihood that the data in D_{train} are realizations of the distribution approximated by the neural network parameterized with θ. It encodes the data uncertainty. Given an estimate of the posterior $p(\theta|D_{train})$, the prediction of an output $\hat{y}$ for a new input data x is obtained by Bayesian model averaging. Bayesian model averaging consists of marginalizing the likelihood $p(D_{train}|\theta)$ with the posterior distribution, that is

$$p(\hat{y}|x, D_{train}) = \int p(\hat{y}|x, \theta)p(\theta|D_{train})d\theta. \tag{3.3}$$

Consequently, we accurately weigh the evidence for different hypotheses. For instance, if the most likely hypothesis predicts *“class A”* but the posterior places more total mass on hypothesis predicting *“class B”* we prefer predicting *“class B”*.

3.1.2. Principles of a Bayesian neural network

Bayesian neural network is preferably used owing to the belief that marginalizing the likelihood $p(D_{train}|\theta)$ with the posterior $p(\theta|D_{train})$ will help make better decisions. This fact is supported by the following justifications.

Bayesian neural networks are ensembles of (stochastic) neural networks [60, 136] Marginalizing $p(D_{train}|\theta)$ with the posterior $p(\theta|D_{train})$ simulates multiple stochastic neural networks parametrized by θ with their associated probability distribution $p(\theta)$. Herein, Bayesian model averaging derives the predictions of all simulated stochastic neural networks to estimate the prediction of an output $\hat{y}$ for a new input data x. In addition, it can reduce overfitting [56] because simulated stochastic neural networks often exhibit a low level of confidence than the underlying point estimate neural network. Moreover, Bayesian model averaging can improve accuracy and calibration of neural networks [74].

Bayesian neural networks take uncertainty into account A Bayesian neural network evaluates a single local optimum in the solution space and the uncertainty around this local optimum [97]. This means that the prediction of an output $\hat{y}$ for a new input data x is also influenced by the neighboring points within a certain region around the solution. This helps to quantify the uncertainty associated with a prediction. Consequently, Bayesian model averaging offers a mathematically grounded tool to deal with uncertainties.

3.1.3. Properties of a Bayesian neural network

Casting a point estimate neural network to a Bayesian neural network involves defining the prior distribution to impose on parameters and specifying the approximate of the posterior distribution. Therefore, the prior and approximate of posterior distributions constitute the main properties of a Bayesian neural network.

Prior distribution Integrating prior knowledge into neural networks is often considered as imposing soft constraints on parameters or activations using regularization approaches such as dropout [9] and variants [10, 11], batch normalization [12, 13], and data augmentation techniques [6, 7, 8]. Consequently, using a regularization approach in a point estimate neural network can be seen as setting a prior distribution [137] from a Bayesian point of view. Since different problems specified through different datasets require different categories of prior knowledge, choosing the appropriate prior distribution is critical. Particularly, a technique used in one field may not directly lead to improvement in another field, although it will be useful if it does. Because of our lack of knowledge (as modelers) regarding the true distribution to place over parameters and its variations from one task to another, specifying a task-specific prior distribution for neural networks is a challenge that is less understood [98].

Approximate of posterior distribution Bayesian model averaging, as presented in Equation 3.3, involves marginalizing the likelihood $p(D_{train}|\theta)$ with the posterior distribution. However, directly inferring the posterior $p(\theta|D_{train})$ is challenging if not impossible in deep neural networks. This is because the size of the data and the number of parameters are too large for the use-cases of deep neural networks and the integral in Equation 3.3 is not computationally tractable as the size of the data and number of parameters increase [56]. Therefore, the posterior distribution is typically approximated via *sampling approaches* [138], *Laplace approximation* [139, 140], and *variational inference* [141, 142]. Sampling approaches are nonparametric and build a representation of weights from which realizations are sampled. This representation is not restricted by a type of distribution, that is, it can be multi-modal or non-Gaussian. Laplace approximation estimates the log of the posterior distribution to derive a normal distribution over weights. Therefore, it is restricted to a type of distribution, that is, the normal distribution. Variational inference methods approximate the posterior distribution by optimizing over a predefined, well-known, and tractable distribution. Several approximations for the posterior distribution exist in literature, however, reviewing all of them is beyond the scope of this thesis. The interested reader is referred to the studies of Gawlikowski et al. [99], Mena et al. [143], and Jospin et al. [144] for a review of different approximations for the posterior distribution. Herein, we focus on the most popular approach, that is, the variational inference.

3.1.4. Variational inference techniques

Variational inference is a well-known technique used in statistics to approximate the intractable posterior distribution $p(\theta|D_{train})$ by optimizing over a predefined, well-known, and tractable distribution $q(\theta)$. The optimization objective is to minimize the error between the surrogate (approximate) distribution $q(\theta)$ and $p(\theta|D_{train})$, that is

$$q(\theta) = \arg\min_q KL(q||p) \tag{3.4}$$

where the error between $q(\theta)$ and $p(\theta|D_{train})$ is measured using the Kullback-Leibler (KL) divergence defined as

$$\mathrm{KL}(q||p) = \mathbb{E}_q \left[\log \frac{q(\theta)}{p(\theta|D_{train})} \right]. \tag{3.5}$$

Since the posterior $p(\theta|D_{train})$ is unknown, the Kullback-Leibler divergence defined in Equation 3.5 cannot be directly minimized. Through some algebraic manipulations, Yang [145] reformulated the Kullback-Leibler divergence as

$$\mathrm{KL}(q||p) = -ELBO + \log p(\hat{y}|x) \tag{3.6}$$

where $ELBO$ (evidence lower bound) is defined by

$$ELBO = \mathbb{E}_q \left[\log \frac{p(\hat{y}|x, \theta)}{q(\theta)} \right] . \tag{3.7}$$

Therefore, minimizing the negative ELBO is equivalent to minimizing the Kullback-Leibler divergence. In addition, it is possible to approximate the posterior $p(\theta|D_{train})$ by minimizing the negative ELBO. Hinton and van Camp [141] and Barber et al. [142] presented the early examples of research minimizing the Kullback-Leibler divergence (or negative ELBO) between the true posterior and a surrogate distribution. Recently, several variational inference techniques for deep neural networks such as Bayes by hypernet [146], Bayes by backprop [58], multiplicative normalizing flows [63], or the application of stochastic elements have been proposed in the literature.

Bayes by hypernet Similar to a hypernetwork [146] that outputs parameters of a target network, a Bayesian hypernetwork hn takes random noise $\epsilon \sim \mathcal{N}(0, I)$ as input and outputs independent samples approximating the surrogate distribution $q(\theta)$ placed over the parameters θ of a target network, that is

$$q(\theta) = q(hn(\epsilon)) \quad \epsilon \sim \mathcal{N}(0, I). \tag{3.8}$$

The Bayesian hypernetwork and the target network form a single model that is trained by backpropagation [24]. According to Krueger et al. [24], a Bayesian hypernetwork is a variational inference method, which can represent a complex multimodal approximate posterior with correlations between parameters.

Bayes by backprop [58] directly model each weight as a random variable with a mean μ and standard deviation σ, instead of approximating the posterior distribution using a hypernetwork. Bayes by backprop is a backpropagation-compatible algorithm for learning the parameters μ and σ describing the probability distribution placed over a weight of a neural network. According to Krueger et al. [24], Bayes by backprop is a special sample of a Bayesian hypernetwork, where the hypernetwork only performs an element-wise scale ($\epsilon \sim \mathcal{N}(0, I)$) and shift ($\mu$) of the input noise ϵ resulting in a factorial Gaussian distribution.

Normalizing flows To scale variational inference to large neural networks and datasets, normalizing flows aim to transform a simple probability distribution (source) to a more complex one (target). Herein, the main challenge is finding the deterministic map that transforms the source to the target distribution with conservation of probability mass. This is achieved, for example, by applying a sequence of invertible transformations until a desired level of complexity is attained [57]. The length of the sequence of transformations affects the complexity of the posterior distribution. However, different transformations result in different characteristics of the posterior distribution.

Stochastic elements Existing stochastic elements to regularize deep neural networks such as dropout [9] and variants [10, 11], and batch normalization [12, 13] can be used for variational inference. MCD is the most widely known [56, 60, 62], where dropout layers, such as Bernoulli distributed random variables, are used during training for regularization and during inference for approximating the posterior distribution. Different extensions evaluating the use of dropout layers in pooling and/or convolutional layers instead of fully-connected layers [70], or different dropout masks (such as Gaussian, Bernoulli, or a cascade of Gaussian and Bernoulli [70]), or different dropout strategies (such as DropConnect that drop connections instead of activations [71, 61] or structured dropout that drop layers, blocks, or channels [69]) were investigated. Supported with a lot of evidence, a combination of different extensions can lead to higher accuracy or better representation of uncertainty estimates [61, 70]. Approximation of the posterior distribution with batch normalization has also been suggested in the literature [68].

In summary, several variational inference techniques can be utilized to approximate the posterior distribution $p(\theta|D_{train})$. These techniques either train a hypernetwork to predict the posterior using Bayes by hypernet, model all weights as random Gaussian variables using Bayes by backprop, transform a simple well-known probability distribution to a complex one via normalizing flows, or take advantage of existing stochastic elements in deep neural networks. Using different techniques results in different approximations of the posterior distribution. For instance, Krueger et al. [24] experimentally showed that Bayes by hypernet is a better regularizer and defends better against adversarial examples than MCD and Bayes by backprop. However, MCD performs better than Bayes by hypernet on detecting anomalies. Notwithstanding these results, the experiments were conducted on shallower (small) neural networks. In addition, Blundell et al. [58] empirically showed that Bayes by backprop yields comparable performance (in terms of accuracy) than MCD on MNIST classification on a fully connected neural network. However, there appears to be no reported studies regarding complex problems using deep neural networks. Louizos and Welling [63] empirically showed that multiplicative normalizing flows generate better uncertainty estimates than MCD, however, their experiments were conducted on shallower neural networks. Consequently, it remains confusing which of the three techniques, namely, Bayes by hypernet, Bayes by backprop, and normalizing flows, can be utilized on complex problems using deep neural networks. Particularly, training a hypernetwork or modelling all weights as random variables can be more time-consuming and requires a large memory demand than applying stochastic elements. In addition, using stochastic elements for variational inference is easy to implement and allows cheap samples from $q(\theta)$. Furthermore, several works [147, 148, 149] have demonstrated the practical values of MCD. However, applying Bayes by backprop results in a full Bayesian neural network because all parameters are given prior distributions, whereas applying MCD results in a sub-Bayesian neural network because only parameters of the last layers are given a prior distribution. Therefore, Zeng et al. [150] investigated the position and number of Bayesian layers required to approximate a full Bayesian neural network. They found that only few Bayesian layers placed closed to the output of a deep neural network are sufficient. Similarly, Brosse et al. [151] evaluated the quality of the uncertainty estimates obtained by making only the last layer Bayesian. They found that the last layer Bayesian neural networks perform similarly well than the full Bayesian neural networks. Kristiadi et al. [152] complemented the empirical evidence of Brosse et al. [151] with a theoretical justification that supports why just making the last layer Bayesian is cost-efficient and sufficient in quantifying uncertainty. According to Zeng, Lesnikowski, and Alvarez [150] and Brosse et al. [151], having multiple Bayesian layers in a Bayesian neural network may compromise the accuracy and does not improve uncertainty estimates. On the contrary, the more Bayesian the layers are, for example, by using high dropout probability, the more uncertainties we can capture at the cost of sacrificing the accuracy [150, 70]. Taken together, all these works demonstrate that a single MCD layer placed closed to the output of a deep neural network, for example, at the input of the first fully connected layer, is sufficient for qualitatively quantifying uncertainty. Moreover, these works provide empirical evidence that demonstrate how MCD is sensitive to the dropout strategy, probability, and mask.

3.1.5. Summary and implications

The main idea of the Bayesian principle is to define parameters of a point estimate neural network as random variables by specifying a prior distribution to impose on parameters. This is based on a rationale that marginalizing the likelihood with the posterior distribution will help make better decisions. Moreover, the main challenge in building a Bayesian neural network is correctly specifying the prior and posterior distributions. The specification of the prior distribution is often achieved using regularization approaches. Among the existing regularization approaches, we will use dropout, batch normalization, and standard label-preserving data augmentation techniques such as rotation, translation, flipping, shearing, and additive Gaussian noise for integrating prior knowledge into neural networks. The posterior distribution is often approximated via variational inference techniques. Among existing variational inference techniques, we will use the popular MCD (see Section 4.1.2), which was utilized in some studies [56, 60, 78, 83, 79] for uncertainty estimation. Furthermore, we will use MCD because its practical value has been proven in several works [147, 148, 149].

3.2. Ensembles

3.2.1. Definition of an ensemble

While a Bayesian neural network is an *implicit or stochastic ensemble* [9], an ensemble (also called a committee [153], multiple classifier systems [154], or mixture of experts [155]) denotes an *explicit or deterministic ensemble* that consists of a set of deterministic neural networks referred to as *ensemble members*. The ensemble members are independently trained and stored, which results in a linear increase in the required memory and computation power with each additional member. This is true for both training and testing. Given an ensemble $f : X \to Y$ with members $f_m : X \to Y$ for $m \in 1, 2, ..., M$, the ensemble prediction is obtained, for example, by averaging over the predictions of the members such as CNNs, that is

$$f(x) = \frac{1}{M} \sum_{m=1}^{M} f_m(x) \ .$$

3.2.2. Principles of an ensemble

The main rationale for utilizing an ensemble is the fact that a group of decision makers make better decisions than a single decision maker. Hansen and Salamon [156], Simonyan and Zisserman [110], and He et al. [111] provided empirical evidence that support this argument by proving that an ensemble improves accuracy. In addition, related justifications have been given to support this argument as follows.

An ensemble approximates the posterior distribution of a Bayesian neural network According to Blundell et al. [58], performing Bayesian model averaging to estimate the posterior distribution is equivalent to using an ensemble of an infinite number of neural networks. Therefore, an ensemble is a finite sample approximation to the posterior distribution [157, 158, 159]. Wilson et al. [74] empirically demonstrate that an ensemble can provide a better approximation to the posterior distribution than standard Bayesian approaches. Specifically, while standard Bayesian approaches reformulate the posterior distribution using Bayes theorem [135], an ensemble approximates the posterior distribution by learning several different parameter settings and averaging over the resulting neural networks.

An ensemble captures the distribution of possible solutions According to Kawaguchi [160], given a neural network to train, the number of possible solutions (local minima) increases exponentially with the number of parameters. Therefore, a trained neural network is one of the possible solutions that exhibit the same accuracy on the training data. By combining multiple neural networks trained independently, an ensemble captures the distribution of all possible solutions and reduces the risk of choosing the wrong solution.

An ensemble enriches the space of representable functions [161] For various applications, correctly modeling the true unknown function using a neural network may not be possible owing to the limited number of parameters. Therefore, by combining multiple neural networks, it may be possible to expand the space of representable functions and to therefore correctly approximate the true unknown function. Domingos [162] supported this argument by empirically demonstrating that an ensemble constructed based on bagging (explained in Section 3.2.6) reduces the classification error because it changes the model space that better fits the input domain.

An ensemble reduces estimation and optimization errors Building a neural network is prone to several errors stemming from the design process (e.g., errors in architecture specification) and/or training process (e.g, errors in parameter initialization). These design and training errors induce a bias and variance in the neural networks. Many empirical studies [161, 163] confirmed that an ensemble reduces the bias and variance inherent in individual ensemble members.

3.2.3. Properties of an ensemble

An ensemble can be characterized based on two main properties, namely, redundancy and diversity, as shown in Figure 11.

Ensemble redundancy

Ensemble redundancy (or redundancy in prediction space of multiple neural networks) is the ability of ensemble members to make similar predictions of the inputs. To explain the advantages of redundancy, imagine that we have an ensemble of three members. If the three members are not redundant in the domain of all possible inputs, then two of the three members will always make wrong predictions. Therefore, the ensemble will always make wrong predictions and will therefore be inaccurate. To avoid this inaccuracy, constructing redundant ensemble members is critical. Tumer and Ghosh [164] supported this argument by showing how correlation among individual ensemble members can affect the accuracy of an ensemble.

Ensemble diversity

Ensemble members must have different characteristics; otherwise, there would be no performance improvement if they have similar ones. Diversity is the differences between

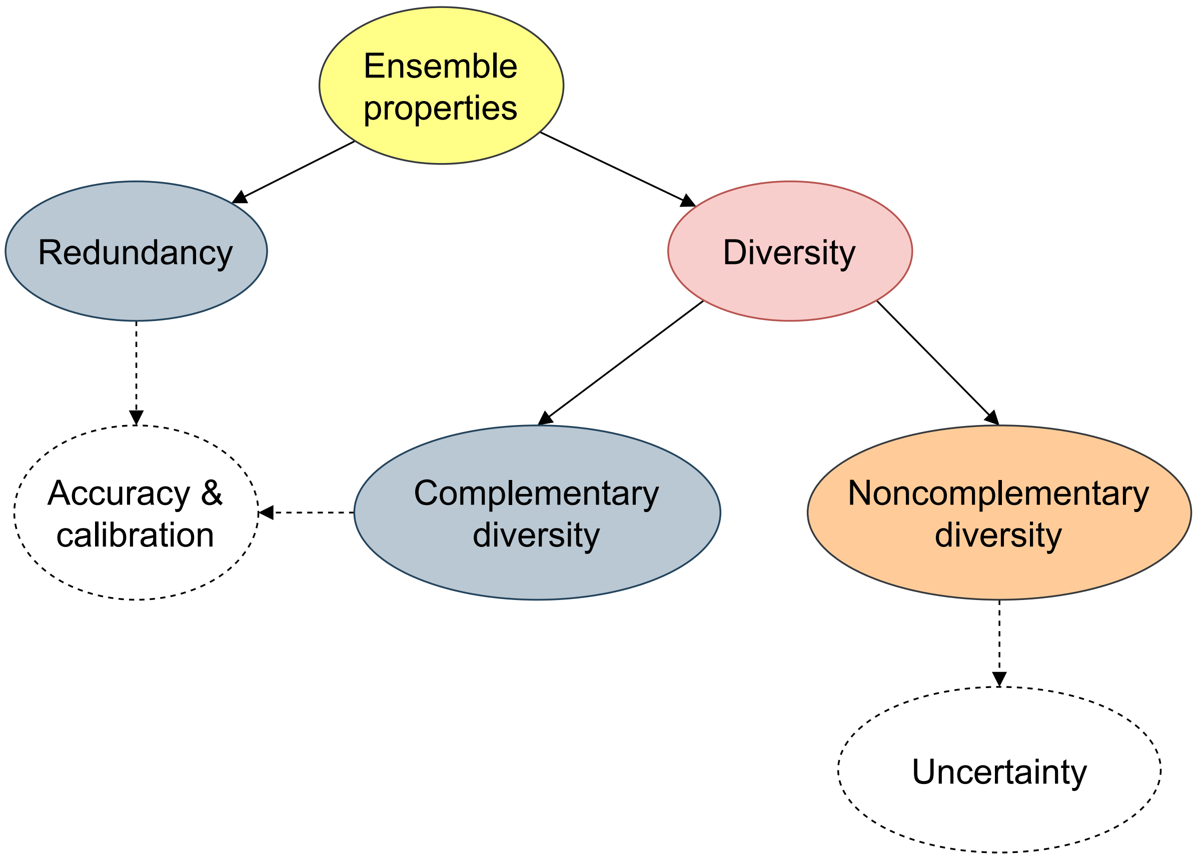


Figure 11.: Overview of the properties of an ensemble such as redundancy and (complementary and noncomplementary) diversity. Accuracy is driven by redundancy and complementary diversity, while uncertainty is driven by noncomplementary diversity, which negatively affects accuracy and calibration

ensemble members and can be monitored by analyzing their prediction errors. Diversity exists because ensemble members have different modalities [1]. Moreover, it has two types: *complementary* and *noncomplementary* diversity.

Complementary diversity Diversity in ensemble members is complementary, if there is enough redundancy to compensate for the prediction errors. In other words, if some of the ensemble members make prediction errors for a given input, the others should make accurate predictions to correct them. The complementary diversity reduces generalization error and therefore boosts accuracy. Specifically, the accuracy of an ensemble is obtained based on two regions of the input domain: the redundant region, where all members agree, and the complementary diversity region, where members compensate for the errors of others. Krogh et al. [165] and Hansen and Salamon [156] provided theoretical and empirical evidence stating that diversity in error distributions across ensemble members can improve ensemble performance. In addition, according to Zhang et al. [69], the more accurate (redundant) and the more (complementary) diverse the ensemble members, the more accurate the ensemble. Furthermore, according to Zhang et al. [69] and further supported by Rahaman and Thiery [81], confidence calibration is correlated to (complementary) diversity and the more (complementary) diverse the ensemble members, the better the ensemble calibration.

Noncomplementary diversity Diversity in ensemble members is noncomplementary, if there is not enough redundancy to compensate for the prediction errors. In other words, if some of the ensemble members make prediction errors for a given input, the others cannot make accurate predictions to correct them. Hence, the noncomplementary diversity can increase the model uncertainty at the cost of reducing the accuracy and/or increasing the calibration error.

3.2.4. Ensembles and uncertainty estimation

Using an ensemble for uncertainty estimation has a long history. Parker [166] reviewed some works that use ensembles to investigate uncertainty in the climate context. However, it is only until the work of Lakshminarayanan et al. [15] that the application of ensembles for uncertainty estimation in deep learning has gained momentum. Some related works [78, 83, 79] compared ensembles to other methods of uncertainty estimation such as MCD and concluded that ensembles performed better than MCD. Moreover, Gustafsson et al. [83] experimentally showed that ensembles are more reliable and suitable to real-life applications than MCD. These findings aligned with the results reported by Beluch et al. [78] who found that ensemble methods generate more accurate and better calibrated

[1]Modality here is the manner in which ensemble members extract and explore feature representations.

predictions than MCD on active learning tasks. While Ashukha et al. [167] evaluated the performance of ensembles in capturing in-domain uncertainty, Lakshminarayanan, Pritzel, and Blundell [15] and Ovadia et al. [79] evaluated the performance of ensembles in capturing OOD uncertainty. Moreover, Rahaman and Thiery [81], Ashukha et al. [167], and Wu and Gales [168] calibrated ensembles via temperature scaling to improve the quality of uncertainty estimates.

3.2.5. Challenges in building ensembles

Building diverse ensembles is challenging. One obstacle is the fact that the ensemble members are trained for the same task using the same training data and are therefore often highly redundant. Another obstacle is the fact that diversity and redundancy are conflicting concepts because we cannot maximize diversity without minimizing redundancy and vice versa. Moreover, introducing different types of diversity in ensembles requires different strategies and may be conflicting. For instance, introducing noncomplementary diversity for driving model uncertainty may harm the calibration. Furthermore, we still do not have a clear understanding of diversity. For instance, given an existing method to build an ensemble, it remains unclear what type (and to what extent) of diversity the method introduced.

3.2.6. Techniques for building ensembles

Different techniques to build an ensemble exist in the literature, which were originally proposed to improve accuracy [156]. This means that all existing works that utilized an ensemble to improve uncertainty or calibration adopt a technique that was initially proposed to improve accuracy. We reviewed and discussed the most popular approaches for introducing diversity in an ensemble, such as random initialization, data shuffling, bagging, boosting, data augmentation, and network architecture. Other approaches, such as random subspace [169], learning under a unified ensemble-aware loss [170], snapshot ensembles [171], and learning rate scheduling [172] are rarely used in the literature, especially in the context of uncertainty estimation; hence, they are not reviewed and discussed in this thesis.

Random initialization is the process of assigning random values to parameters (weights and biases) of ensemble members. It is well known that two identical architectures optimized with different initializations will converge to different solutions [97], leading to decorrelated prediction errors. This is because different initializations have different gradient calculations, using which parameters are found to minimize the given error function. Hansen et al. [156], Krizhevsky et al. [108], and Simonyan et al. [110] utilized the randomness introduced in the process of initializing parameters to build ensembles for improving accuracy.

Data shuffling is the process of randomly selecting training data points for building minibatches (small to large number of training data points). Particularly, to limit the sensitivity of gradient calculations to a single training data point, training of neural networks is conducted on minibatches. Different minibatches result in different gradient calculations and subsequently, to different solutions. The randomness introduced in the process of building minibatches can introduce diversity in ensemble members. In the literature, data shuffling is always used in addition to random initialization; this is exemplified in the work of Laschiminarayanan et al. [15], where they used random initialization and data shuffling to build an ensemble for uncertainty estimation.

Bagging (bootstrap aggregation) [173] uses subsets of the training data uniformly sampled with replacement from the original training data to train ensemble members. The random process of sampling training data points with replacement results in subsets of the training data including similar training data points while missing others. Therefore, bagging not only diversifies the distribution but also reduces the size of the training data of single ensemble members and therefore encourages specialization. Krogh et al. [165] utilized bagging to build an ensemble to boost the accuracy. However, according to Lee et al. [170], bagging can result in poorly calibrated ensembles caused by (noncomplementary) diversity owing to the modification of the training data distribution.

Boosting uses subsets of the original training data, similar to bagging, however, the ensemble members are trained in sequence and each one is trained to focus on the mistakes of others [174]. Herein, training data points falsely classified by already trained ensemble members are included in the training data of the next ensemble member to be trained and are given more importance. Therefore, the training data for the next ensemble member to train is obtained in a deterministic way, contrary to bagging, wherein they are obtained randomly and independently from the performance of previous trained members. Moghimi et al. [175] applied boosting to CNNs to create an ensemble and improve its accuracy.

Data augmentation modifies the training data distribution while increasing the size of the training data in contrast to bagging and boosting wherein they reduce it. Augmenting the size of the training data can be achieved using existing data augmentation techniques reviewed by Shorten et al. [6]. Different data augmentation techniques result in different gradient calculations and solutions. Consequently, the randomness introduced in a single or different data augmentation approaches can be used to introduce diversity in ensemble members. For instance, Nanni et al. [176] used different data augmentation approaches to improve the accuracy of an ensemble for bioimage classification. Another example is the work of Guo and Gould [177], wherein they improved the performance of an ensemble for object detection by training ensemble members on different subsets of the training data augmented using existing benchmarking datasets. Specifically, they augmented the PASCAL VOC dataset [178] with the Microsoft COCO dataset [179]. However, according to Wen et al. [82], data augmentation approaches such as MixUp [180] can improve the accuracy of an ensemble at the cost of impairing the calibration. Maroñas et al. [181] and Rahaman and Thiery [81] provided empirical evidence supporting this negative impact of MixUp training on ensemble calibration. Wen et al. [82] argued that label smoothing [91, 94] in MixUp training is responsible for impairing the calibration. On the contrary, Maroñas et al. [181] and Rahaman and Thiery [81] argued that the distributional shift (or data uncertainty) resulting from convex mixing of pairs of images is responsible for the negative impact of MixUp training on the calibration of an ensemble.

Network architecture involves introducing diversity through the structure (architecture) rather than through the parameters using methods such as random initialization, bagging, boosting, or data augmentation. Specifically, the architecture of a neural network conditions the manner in which information are propagated from the input to subsequent layers. Different architectures result in different gradient calculations and therefore to different solutions. According to Zhang et al. [69], architectural or structural diversity can help promote model diversity. This is exemplified in their work, wherein they dropped structure elements (channels or blocks) instead of processing units (neurons) to promote model diversity and improve confidence calibration. In addition, Herron et al. [182] used neural architecture search to build an ensemble of structural diverse neural networks and achieved an increased accuracy owing to the structural diversity. Moreover, Guo and Gould [177] improved the accuracy of an ensemble to detect objects by building ensemble members using different architectures. Particularly, they used the GoogLeNet [183] architecture for some ensemble members and the VGG-16 [110] for others.

In summary, while some techniques for building an ensemble such as random initialization, data shuffling, and network architecture train ensemble members on the same original training data, other methods such as bagging, boosting, and data augmentation train them on different modified training data. Therefore, redundancy is ensured in an ensemble by utilizing the same original training data or including similar training data points in modified training data. Livieris et al. [184] compared bagging and boosting strategies for building ensembles. They concluded that bagging performs better on a small number of ensemble members while boosting performs better on a large number of ensemble members. Therefore, since a small number of ensemble members are preferred over a large one to keep the requirement in memory and computational demand in minimum, bagging strategies are preferred over boosting ones. However, Lee et al. [170] experimentally showed that the diversity introduced in ensemble members via random parameter initialization is more useful than that introduced via bagging. They concluded that random initialization may be preferred over bagging for building an ensemble because of the large parameter space and the requirement for large training data. Moreover, according to Lee et al. [170], bagging can result in poorly calibrated ensembles. Although data augmentation enlarges the size of the training data, which helps meet the requirement for large training data, it can result in poorly calibrated ensembles, especially when using modern data augmentation techniques such as MixUp [180]. This is exemplified in the studies of Maronas, Ramos, and Paredes [181] and Rahaman and Thiery [81]. Consequently, standard label-preserving data augmentation techniques such as rotation, translation, and flipping, are preferred over modern ones for building calibrated ensembles. Nonetheless, we note that the calibration of an ensemble can be improved using post-hoc methods such as temperature scaling [16] as exemplified in the studies of Rahaman and Thiery [81], Ashukha et al. [167], and Wu and Gales [168].

3.2.7. Summary and implications

The main idea behind the ensemble principle is combining the individual predictions of multiple diverse neural networks (ensemble members) to estimate uncertainty. The ensemble members need to be as accurate and diverse as possible for the ensemble principle to be effective. The main challenge when building ensembles is to tradeoff redundancy and diversity among the ensemble members. Therefore, several techniques (reviewed and discussed in Section 3.2.6) were proposed in the literature. Among these techniques, we used random initialization, data shuffling, and standard (label-preserving) data augmentation techniques such as rotation, translation, flipping, shearing, and additive Gaussian noise to build ensembles because these techniques do not (strongly) affect the calibration of an ensemble. However, the needed memory and computational requirements increased linearly with the number of ensemble members for training and inference, thereby limiting the deployment of an ensemble in many practical applications where the available computation power or memory is limited. Notwithstanding this, several researchers [79, 167] have shown that using a small number of ensemble members is sufficient to achieve a good-performing ensemble. For this reason, we will use a small (5-20) number of ensemble members to keep the requirement in memory and computation at minimum.

4. Uncertainty estimation methods

Herein, we are not concerned with improving the classification accuracy, reducing the number of parameters, or improving the training or inference time of CNN-based classifiers. Rather, our goal is to improve the quality of the predictive uncertainty. Particularly, we want the predictive confidence measuring uncertainty to be high for TPs and low for FPs. Therefore, we developed MCA, which is inspired by the MMCD, which combines the strength of both ensemble and MCD.

4.1. Related methods

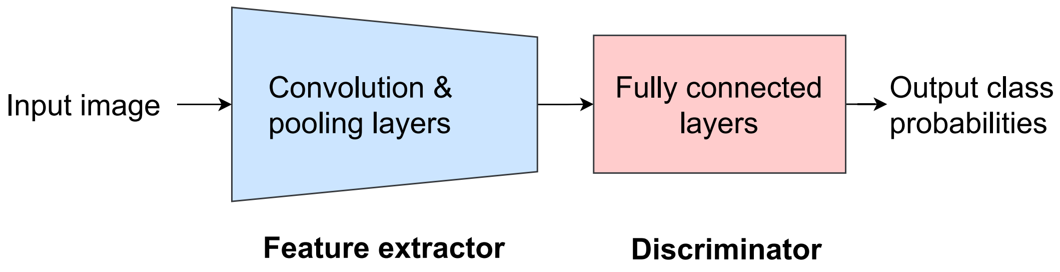


Figure 12.: Illustration of the two main modules of a CNN-based classifier. The feature extractor is achieved using convolution and pooling layers, while the discriminator is achieved using fully connected layers. The parameters of the two modules are jointly learned during the training process

4.1.1. Baseline

Baseline is a single CNN. As explained in Section 2.2, a CNN is a neural network with multiple layers such as convolution, pooling, and fully connected layers. CNN-based classifiers for image classification are designed to extract relevant features directly from raw pixel of the given input image and to discriminate them. Subsequently, such classifiers include two main modules (see Figure 12): a *features extractor* and *discriminator*. Therefore, a CNN-based classifier is a composite of two nonlinear functions

$f_{FeatureExtractor}()$ and $f_{Discriminator}()$ parameterized by trainable model parameters (i.e., the neural network weights). Therefore

$$f : x \in \mathbb{R}^{H\times W\times C} \to \hat{y} \in U^K; \qquad f_{Discriminator}(f_{FeatureExtractor}(x)) = p(\hat{y}|x) \ . \tag{4.1}$$

One of the main drawbacks of baseline is that it relies on a unique solution shaped by a single CNN and the discriminator of this single CNN does not evaluate the uncertainty associated with the extracted features.

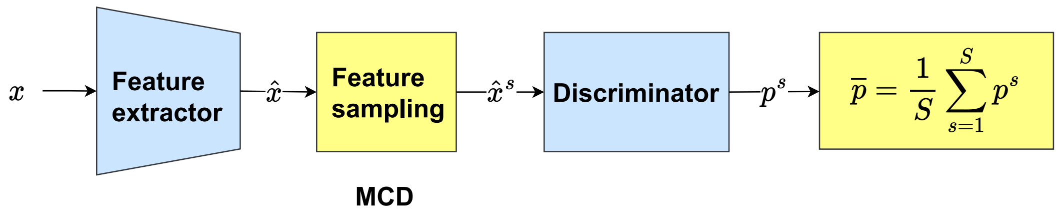


Figure 13.: Overview of prediction estimation via MCD applying feature sampling in a single CNN

4.1.2. Monte Carlo dropout

MCD is one of the most widely used variational inference methods for approximating a Bayesian neural network. As shown in Figure 13, MCD mainly samples $\hat{x} = f_{FeatureExtractor}(x)$ with masks derived from known distributions such as Gaussian, Bernoulli, or cascade of Gaussian and Bernoulli distribution [70]. In other words, MCD is achieved using a nonlinear function $f_{FeatureSampling}()$ parameterized by random variables obtained from known distributions, that is

$$f_{MCD} : \hat{x} \in \mathbb{R}^F \to \hat{x}^s \in \mathbb{R}^F; \qquad \hat{x}^s = f_{FeatureSampling}(\hat{x}) \ , \tag{4.2}$$

where F is the dimension of the feature vector $\hat{x}$. In our study [70], we compared the quality of uncertainty estimates obtained via dropout sampling with masks derived from Gaussian, Bernoulli, or cascade of Gaussian and Bernoulli distributions. Among these, we used the cascade of Gaussian and Bernoulli distributions, which reduces the number of active neurons owing to the Bernoulli distribution and helps strengthen/weaken the magnitude of active neurons owing to the Gaussian distribution. Herein, sampling is achieved with random variables obtained from the cascade of Gaussian and Bernoulli distributions and MCD samples $\hat{x}$, as shown in Figure 17a and exemplified in Figure 14. MCD estimates $\overline{p}(\hat{y}|x)$ by the mean of S feature sampling operations, that is

$$\overline{p}(\hat{y}|x) \quad \approx \quad \frac{1}{S}\sum_{s=1}^{S} p^s(\hat{y}|x) \quad \approx \quad \frac{1}{S}\sum_{s=1}^{S} f_{Discriminator}(\hat{x}^s). \tag{4.3}$$

$$\hat{x} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix} \qquad \alpha^1 = \begin{bmatrix} 0.1 \\ 0.2 \\ 0.5 \\ 0.4 \\ 0.5 \end{bmatrix} \qquad \beta^1 = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \qquad \hat{x}^1 = \begin{bmatrix} 0.1 \\ 0 \\ 1.5 \\ 1.6 \\ 0 \end{bmatrix}$$

$$\alpha^2 = \begin{bmatrix} 0.5 \\ 0.5 \\ 0.1 \\ 0.3 \\ 0.5 \end{bmatrix} \qquad \beta^2 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} \qquad \hat{x}^2 = \begin{bmatrix} 0 \\ 0 \\ 0.3 \\ 1.2 \\ 2.5 \end{bmatrix}$$

Figure 14.: An example showing two feature sampling operations, where $\hat{x}$ is the original feature vector, α^1 and α^2 are the sampling masks including random variables obtained from a Gaussian distribution at the first and second feature sampling operation, β^1 and β^2 are the sampling masks including random variables obtained from a Bernoulli distribution at the first and second feature sampling operation, and $\hat{x}^1 = \hat{x} * \alpha^1 * \beta^1$ and $\hat{x}^2 = \hat{x} * \alpha^2 * \beta^2$ are the perturbed feature vectors obtained from the first and second feature sampling operation.

We refer to MCD as an average of S stochastic CNNs. Although the discriminator of MCD evaluates the uncertainty associated with the extracted features, MCD relies on a unique solution shaped by a single CNN. This means that the S stochastic CNNs are bounded around the same unique solution. Moreover, the modeler can incorrectly specify the predefined distribution from which masks will be obtained for feature sampling (or feature perturbation). Since the architecture and dataset affect the predefined distribution [185], specifying the optimal parameters (e.g., dropout probability) for a given dataset and architecture can be time-consuming and costly and can therefore increase the complexity of building MCD.

4.1.3. Deep ensemble

Deep ensemble is an ensemble (see Section 3.2) of deep neural networks. It was utilized by Lakshminarayanan, Pritzel, and Blundell [15]; Ashukha et al. [167]; Beluch et al. [78]; Gustafsson, Danelljan, and Schon [78]; and Ovadia et al. [79] to estimate uncertainties.

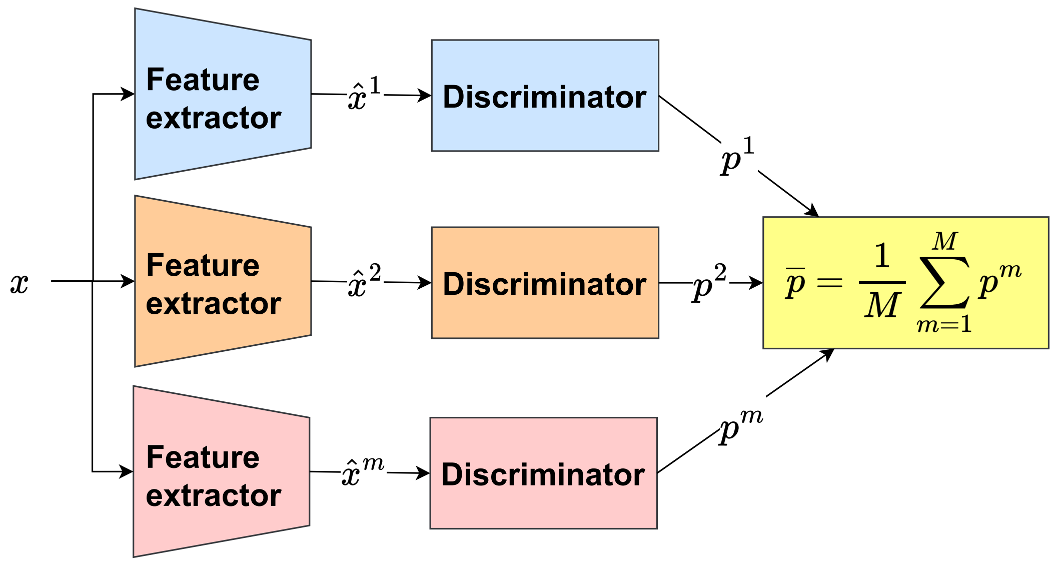


Figure 15.: Overview of prediction estimation via ensemble

As shown in Figure 15, given a set of CNNs $f_m = f_{Discriminator_m}(f_{FeatureExtractor_m}(\cdot))$ for $m \in 1, 2, ..., M$, the ensemble prediction $\overline{p}(\hat{y}|x)$ is estimated by averaging over the predictions of the CNNs, that is

$$\overline{p}(\hat{y}|x) = \frac{1}{M}\sum_{m=1}^{M} p^m(\hat{y}|x) = \frac{1}{M}\sum_{m=1}^{M} f_{Discriminator_m}(f_{FeatureExtractor_m}(x)) \; . \quad (4.4)$$

We refer to a deep ensemble (or an ensemble for short) as an average of M deterministic CNNs. Although an ensemble overcomes the drawback of baseline and MCD by relying on multiple solutions shaped by multiple CNNs, it does not evaluate the uncertainty associated with the extracted features. Specifically, the discriminators of the members of an ensemble do not evaluate the uncertainty associated with the outputs of the feature extractors.

4.1.4. Mixture of Monte Carlo dropout

MMCD was utilized by Kahn et al. [30]; Lutjens, Everett, and How [31]; and Wilson and Izmailov [74] to estimate uncertainties. While MCD relies on a single solution shaped by a single CNN but additionally evaluates the uncertainty associated with the extracted features, an ensemble evaluates multiple solutions shaped by multiple CNNs but does not evaluate the uncertainty associated with the extracted features. Therefore, as shown in Figure 16, MMCD utilizes MCD on the members of an ensemble to capitalize the power and overcome the drawback of an ensemble by evaluating the uncertainty associated with the extracted features. Given a set of CNNs $f_m = f_{Discriminator_m}(f_{FeatureExtractor_m}(\cdot))$ for

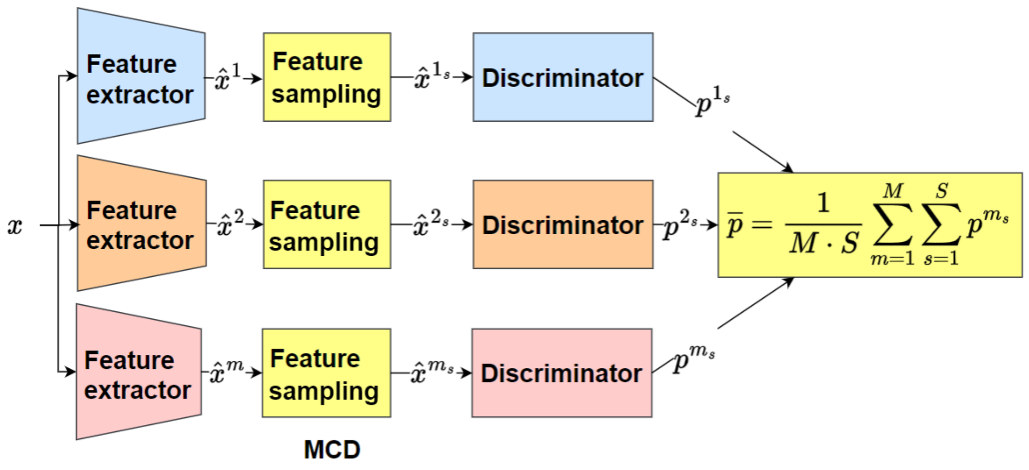
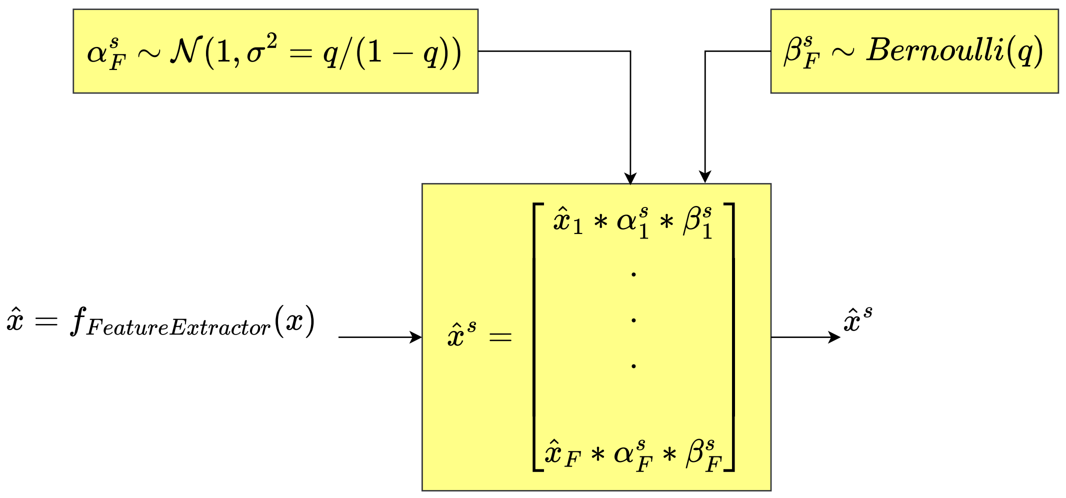


Figure 16.: Overview of prediction estimation via MMCD by applying feature sampling in multiple CNNs

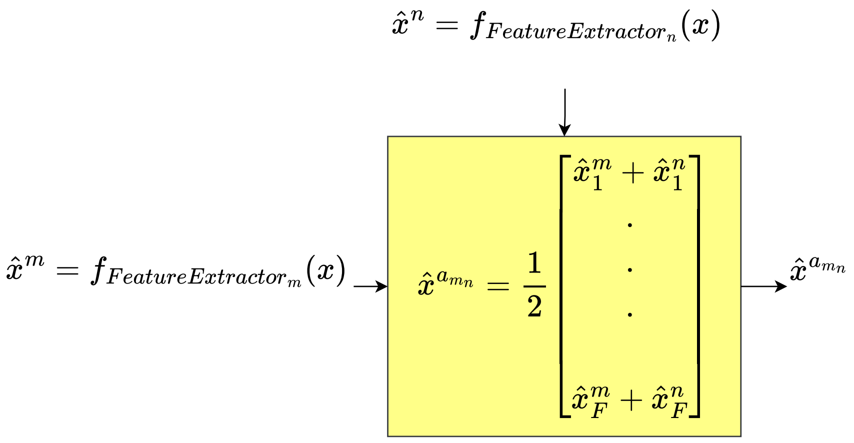
$m \in 1, 2, ..., M$, the MMCD prediction $\overline{p}(\hat{y}|x)$ is estimated as

$$\overline{p}(\hat{y}|x) \quad \approx \quad \frac{1}{M \cdot S} \sum_{m=1}^{M} \sum_{s=1}^{S} p^{m_s}(\hat{y}|x) \quad \approx \quad \frac{1}{M \cdot S} \sum_{m=1}^{M} \sum_{s=1}^{S} f_{Discriminator_m}(\hat{x}^{m_s}), \qquad (4.5)$$

where $\hat{x}^{m_s} = f_{FeatureSampling}(\hat{x}^m)$ is a feature sampled from $\hat{x}^m = f_{FeatureExtractor_m}(x)$, as shown in Figure 17a and exemplified in Figure 14. We refer to MMCD as an average of $M \cdot S$ stochastic CNNs. Although MMCD overcomes the drawback of an ensemble by applying MCD in ensemble members to force their discriminators to evaluate the uncertainty associated with the extracted features, MMCD can experience some of the problems of MCD. Particularly, the modeler of MMCD can incorrectly specify the predefined distribution from which masks will be obtained for feature sampling. Moreover, the design process of MMCD can be more complex than that of an ensemble because the predefined distribution needs to be fine-tuned for a given architecture and dataset.


$\alpha^s_F \sim \mathcal{N}(1, \sigma^2 = q/(1-q))$
$\beta^s_F \sim Bernoulli(q)$
$\hat{x} = f_{FeatureExtractor}(x)$
$\hat{x}^s = \begin{bmatrix} \hat{x}_1 * \alpha^s_1 * \beta^s_1 \\ \cdot \\ \cdot \\ \cdot \\ \hat{x}_F * \alpha^s_F * \beta^s_F \end{bmatrix}$
$\hat{x}^s$

(a) feature sampling


$\hat{x}^n = f_{FeatureExtractor_n}(x)$
$\hat{x}^m = f_{FeatureExtractor_m}(x)$
$\hat{x}^{a_{mn}} = \frac{1}{2} \begin{bmatrix} \hat{x}^m_1 + \hat{x}^n_1 \\ \cdot \\ \cdot \\ \cdot \\ \hat{x}^m_F + \hat{x}^n_F \end{bmatrix}$
$\hat{x}^{a_{mn}}$

(b) feature averaging

Figure 17.: Comparison of feature sampling (inherent in MMCD) and feature averaging (inherent in MCA)

4.2. Monte Carlo averaging

4.2.1. Motivation

MMCD evaluates the uncertainty associated with the extracted features by obtaining masks from a predefined distribution. The nature and/or parameters specifying this distribution are a priori unknown (to the modeler) and vary depending on the dataset and architecture. Hence, the process of building an MMCD can be time-consuming and costly because the predefined distribution needs to be fine-tuned for a given dataset and architecture. Therefore, we developed MCA, which doesn't require a predefined distribution (from which masks will be obtained for feature sampling). Similar to MMCD, MCA relies on multiple solutions shaped by multiple CNNs and evaluates the uncertainty associated with the extracted features. However, instead of applying MCD on ensemble members, similar to MMCD, for feature perturbation, MCA averages features extracted by the different ensemble members. Specifically, MCA evaluates the uncertainty associated with the features extracted by one ensemble member by, sequentially and in a pairwise manner, averaging or perturbing them with the features extracted by other ensemble members. This is based on a rationale that features extracted by ensemble members are different and can therefore be used for feature perturbation.

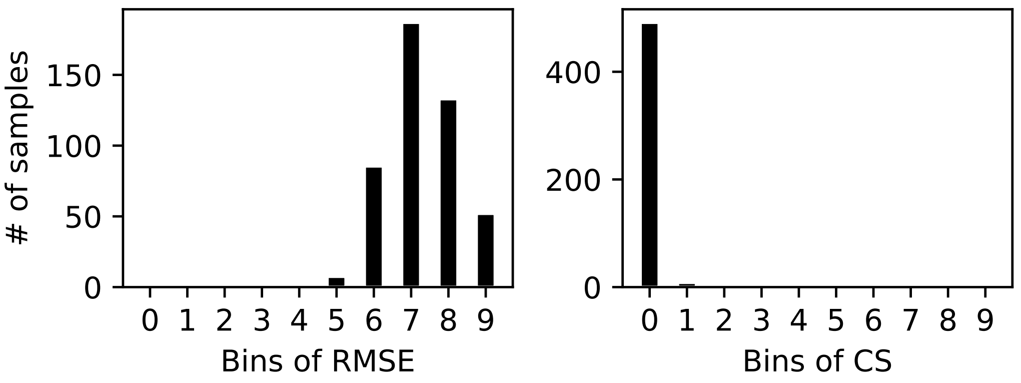


Figure 18.: Histograms of root mean square error (RMSE) and cosine similarity (CS) measuring the similarity between features extracted by different ensemble members. Lower bins include lower values, while higher bins include higher values of RMSE and CS. The results were obtained via CIFAR10 using an ensemble of CNNs trained with strong regularization (SR) and using the *subset of correctly classified test data*, as shown in Appendix A.6

$$\hat{x}^1 = \begin{bmatrix}1\\2\\3\\4\\5\end{bmatrix} \qquad \hat{x}^{a_{1_2}} = \begin{bmatrix}1\\1\\2\\3\\4\end{bmatrix} \qquad \hat{x}^{a_{1_3}} = \begin{bmatrix}0.5\\1\\2\\2\\6\end{bmatrix}$$

$$\hat{x}^2 = \begin{bmatrix}1\\0\\1\\2\\3\end{bmatrix} \qquad \hat{x}^{a_{2_1}} = \begin{bmatrix}1\\1\\2\\3\\4\end{bmatrix} \qquad \hat{x}^{a_{2_3}} = \begin{bmatrix}0.5\\0\\1\\1\\5\end{bmatrix}$$

$$\hat{x}_3 = \begin{bmatrix}0\\0\\1\\0\\7\end{bmatrix} \qquad \hat{x}^{a_{3_1}} = \begin{bmatrix}0.5\\1\\2\\2\\6\end{bmatrix} \qquad \hat{x}^{a_{3_2}} = \begin{bmatrix}0.5\\0\\1\\1\\5\end{bmatrix}$$

Figure 19.: An example showing feature averaging operations performed by the first, second, and third ensemble members. $\hat{x}^1$, $\hat{x}^2$, and $\hat{x}^3$ are the feature vectors extracted by the first, second, and third ensemble members, respectively. $\hat{x}^{a_{1_2}} = \frac{1}{2}(\hat{x}^1 + \hat{x}^2)$, $\hat{x}^{a_{1_3}} = \frac{1}{2}(\hat{x}^1 + \hat{x}^3)$, $\hat{x}^{a_{2_1}} = \frac{1}{2}(\hat{x}^2 + \hat{x}^1)$, $\hat{x}^{a_{2_3}} = \frac{1}{2}(\hat{x}^2 + \hat{x}^3)$, $\hat{x}^{a_{3_1}} = \frac{1}{2}(\hat{x}^3 + \hat{x}^1)$, and $\hat{x}^{a_{3_2}} = \frac{1}{2}(\hat{x}^3 + \hat{x}^2)$ are pairwise averaged features, which are used to evaluate the uncertainty associated with extracted features. Particularly, the discriminator of the first ensemble member processes $\hat{x}^1$, $\hat{x}^{a_{1_2}}$, and $\hat{x}^{a_{1_3}}$. The discriminator of the second ensemble member processes $\hat{x}^2$, $\hat{x}^{a_{2_1}}$, and $\hat{x}^{a_{2_3}}$. Lastly, the discriminator of the third ensemble member processes $\hat{x}^3$, $\hat{x}^{a_{3_1}}$, and $\hat{x}^{a_{3_2}}$.

4.2.2. Analyzing features extracted by ensemble members

Given members $m \in 1, 2, ..., M$ and $n \in 1, 2, ..., M$ (with $n \neq m$), we measure the similarity between feature vector $\hat{x}^m = f_{FeatureExtractor_m}(x)$ and $\hat{x}^n = f_{FeatureExtractor_n}(x)$ via root mean square error (RMSE) and cosine similarity (CS). RMSE measures the difference in magnitude between $\hat{x}^m$ and $\hat{x}^n$ and determines whether the two vectors are similar in terms of magnitude. CS measures the cosine of the angle between the two vectors and determines whether the two vectors are pointing in (roughly) the same direction. The RMSE is defined as

$$RMSE = \sqrt{\frac{1}{F}\sum_{i=1}^{F}(\hat{x}_i^m - \hat{x}_i^n)^2}, \tag{4.6}$$

and the CS is defined as

$$CS = \frac{\hat{x}^m \cdot \hat{x}^n}{||\hat{x}^m|| \cdot ||\hat{x}^n||}, \tag{4.7}$$

where $||\hat{x}^m||$ is the Euclidean norm of vector $\hat{x}^m$. The lower (closer to 0) the RMSE, the smaller the error between $\hat{x}^m$ and $\hat{x}^n$ and the greater the match between the two vectors in terms of magnitude. Moreover, the higher (closer to 1) the CS, the smaller the angle between $\hat{x}^m$ and $\hat{x}^n$ and the greater the match between the two vectors in terms of orientation. Figure 18 presents the histograms of RMSE and CS measuring the similarity between features extracted by two members of an ensemble. The results show that only higher bins of RMSE and lower bins of CS are populated, that is, the RMSE and CS between $\hat{x}^m$ and $\hat{x}^n$ are higher (closer to 1) and lower (closer to 0), respectively. This implies that *features extracted by ensemble members are different in terms of magnitude and orientation*, that is, *they extract different features from the same given input*. To further support this argument, we evaluated the classification accuracy when discriminators of ensemble member m (only) evaluate features extracted by ensemble member n which is denoted by $n \neq m$. Hence, we estimate $\overline{p}(\hat{y}|x)$ as

$$\overline{p}(\hat{y}|x) \quad = \quad \frac{1}{M^2}\sum_{m=1}^{M}\sum_{n=1}^{M} p^{m_n}(\hat{y}|x) \quad = \quad \frac{1}{M^2}\sum_{m=1}^{M}\sum_{n=1}^{M} f_{Discriminator_m}(\hat{x}^n) \ . \tag{4.8}$$

Afterward, we observed that the accuracy drastically declined, meaning that discriminators of ensemble member m cannot correctly evaluate features extracted by ensemble member n. For instance, the accuracy of an ensemble of DenseNet trained on CIFAR10 using strong regularization (SR) drops to 17.73% from 89.50% when we estimated $\overline{p}(\hat{y}|x)$ as showed in Equation 4.8. This implies that *features extracted by the ensemble members are different and cannot be evaluated by discriminators of other ensemble members*. Hence, the feature vector $\hat{x}^n$ of ensemble member n can be used to perturb the feature vector $\hat{x}^m$ of ensemble member m.

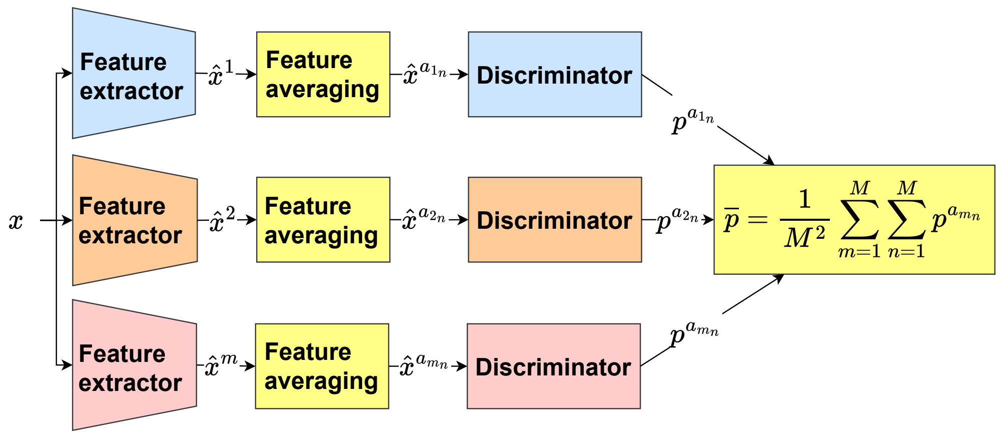


Figure 20.: Overview of prediction estimation via MCA by applying feature averaging in multiple CNNs

4.2.3. Applying feature averaging in ensemble members

To perturb the feature vector $\hat{x}^m$ of ensemble member m using the feature vector $\hat{x}^n$ of ensemble member n, we must make sure that the discriminator of ensemble member m will classify the perturbed features $\hat{x}^{a_{mn}}$ to the same class associated with $\hat{x}^m$ to preserve the classification accuracy. Therefore, MCA performs a pairwise averaging of $\hat{x}^m$ and $\hat{x}^n$ to obtain $\hat{x}^{a_{mn}}$. Particularly, MCA sequentially perturbs $\hat{x}_m$ by averaging it with $\hat{x}_n$, as shown in Figure 17b and exemplified in Figure 19. Given a set of CNNs $f_m = f_{Discriminator_m}(f_{FeatureExtractor_m}(\cdot))$, MCA estimates $\overline{p}(\hat{y}|x)$ as

$$\overline{p}(\hat{y}|x) \quad = \quad \frac{1}{M^2}\sum_{m=1}^{M}\sum_{n=1}^{M} p^{a_{mn}}(\hat{y}|x) \quad = \quad \frac{1}{M^2}\sum_{m=1}^{M}\sum_{n=1}^{M} f_{Discriminator_m}(\hat{x}^{a_{mn}}). \tag{4.9}$$

where $\hat{x}^{a_{mn}} = \frac{1}{2}\hat{x}^m + \frac{1}{2}\hat{x}^n$. By averaging features of ensemble members in a pairwise manner, MCA forces discriminators to evaluate the uncertainty associated with the extracted features. We refer to MCA as an average of M^2 deterministic CNNs. We continued without considering that if MCA does not perform the pairwise averaging and instead averages the features of more than two ensemble members, then the level of perturbation in the averaged features will drastically increase. This will have a negative impact on the classification accuracy. Therefore, it is recommended to integrate MCA with the pairwise averaging proposed in this thesis. Moreover, future works can study weighing approaches for the pairwise averaging of features of ensemble members.

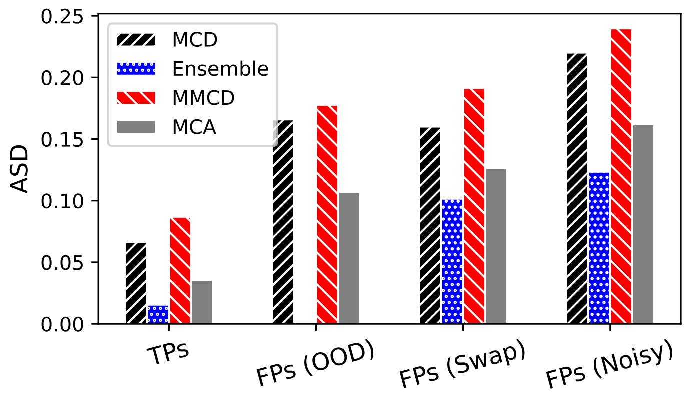
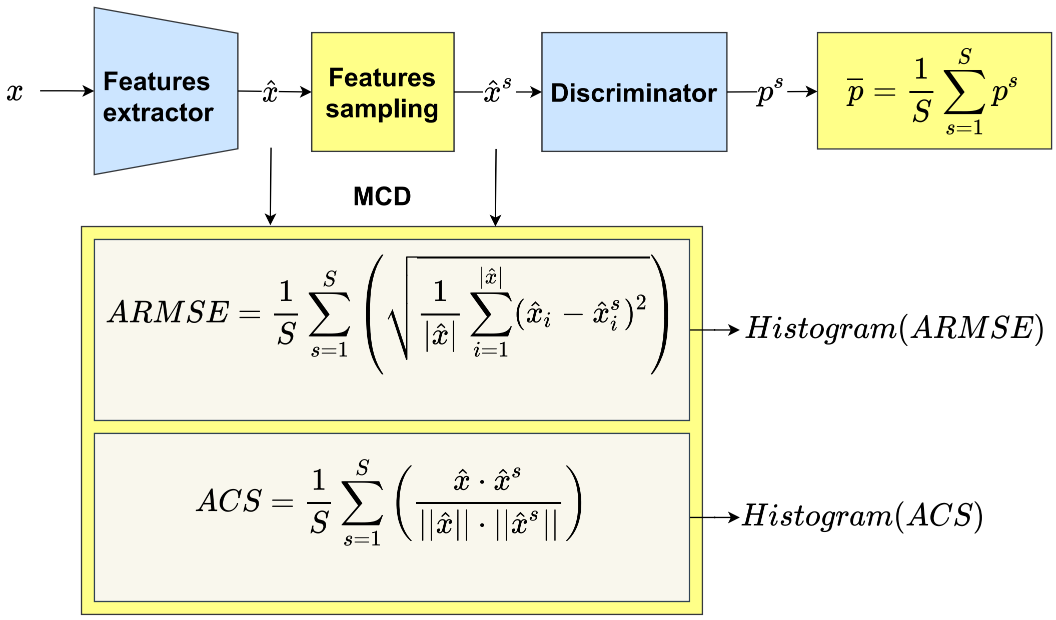


Figure 21.: Comparison of the average standard deviation (ASD) obtained on CIFAR10 using CNNs with large capacities trained using weak regularization (WR) and evaluated on TPs and FPs. TPs were obtained on *subsets of test data correctly classified*, while FPs were obtained on *swap*, *noisy*, or *OOD* data, as described in Appendix A.6

4.3. Importance of feature averaging or feature sampling

It is clear that feature averaging (inherent in MCA) and feature sampling (inherent in MMCD) are used to evaluate the uncertainty associated with the extracted features of ensemble members. Therefore, both approaches improve the predictive uncertainty of the underlying ensemble. However, the manner of improving the predictive uncertainty remains ambiguous. To clarify this, we argue that by evaluating the uncertainty associated with the features extracted by the ensemble members, MCA and MMCD can capture the diversity between the ensemble members better than the underlying ensemble. In other words, feature sampling and averaging help to better capture the variability between ensemble members and to therefore improve the predictive uncertainty of the underlying ensemble. To provide empirical evidence supporting this argument, Figure 21 presents the average standard deviation (ASD) between predictions of the underlying ensemble, MMCD, and MCA. The results show that for FPs owing to OOD data, the ASD of the underlying ensemble is lower than that of MMCD and MCA. This means that the underlying ensemble fails to capture the variability between ensemble members, while MMCD and MCA succeed owing to feature sampling and feature averaging, respectively.



x
Features extractor
$\hat{x}$
Features sampling
$\hat{x}^s$
Discriminator
p^s
$\overline{p} = \frac{1}{S}\sum_{s=1}^{S} p^s$
MCD
$ARMSE = \frac{1}{S}\sum_{s=1}^{S}\left(\sqrt{\frac{1}{|\hat{x}|}\sum_{i=1}^{|\hat{x}|}(\hat{x}_i - \hat{x}_i^s)^2}\right)$
$Histogram(ARMSE)$
$ACS = \frac{1}{S}\sum_{s=1}^{S}\left(\frac{\hat{x}\cdot\hat{x}^s}{||\hat{x}||\cdot||\hat{x}^s||}\right)$
$Histogram(ACS)$

Figure 22.: Illustration of the process of extracting statistical data, such as the average cosine similarity (ACS) and average root mean square error (ARMSE), from the sampled features. The statistical data are visualized using histograms, as shown in Figure 23

4.4. Comparison of feature sampling and feature averaging

Feature averaging (inherent in MCA) and feature sampling (inherent in MMCD) are types of feature perturbation. We then evaluated how each perturbation modifies the original features. For this purpose, we measure the similarity (using the RMSE and the CS) between the original feature vector $\hat{x}$ and the sampled ($\hat{x}^s$) or averaged ($\hat{x}^a$) feature vector. Specifically, we estimated the average cosine similarity (ACS) and average root mean square error (ARMSE) over perturbed samples. Figure 22 presents the process of building histograms of ACS and ARMSE from the sampled features. The same procedure is applied for building histograms from the averaged features. Figure 23 presents the histograms of ACS and ARMSE obtained via feature sampling and averaging. The results show that lower bins of ARMSE and higher bins of ACS are more populated via feature averaging than feature sampling. This means that feature averaging results in lower ARMSE and higher ACS than feature sampling. Moreover, this implies that *feature averaging produces features that are more similar to the original ones than feature sampling*, that is, *feature averaging preserves similarity better than feature sampling.* This indicates that *feature averaging can preserve the classification accuracy better than*

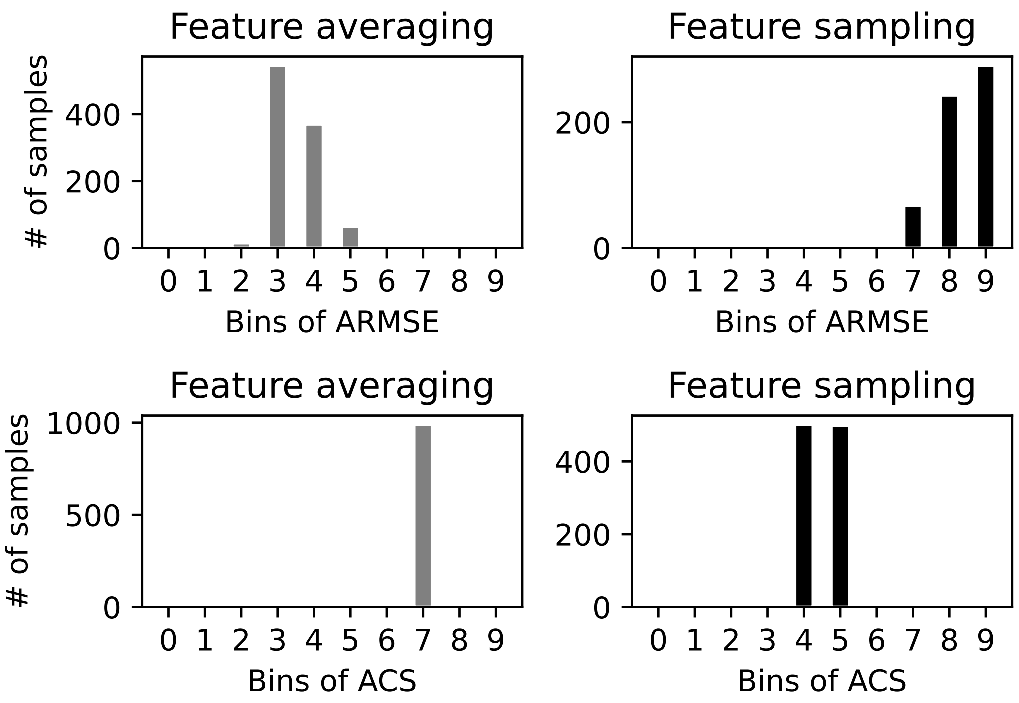


Figure 23.: Comparison of histograms of ACS and ARMSE obtained from the sampled and averaged features. Lower bins include lower values, while higher bins include higher values of ACS and ARMSE. The results were obtained via CIFAR10 using an ensemble of CNNs trained with SR and using the *subset of correctly classified test data*, as described in Appendix A.6

feature sampling. We continued without considering that feature sampling results in higher ARMSE and lower ACS owing to dropout that eliminates some relevant features. By fine-tuning the dropout probability, we can achieve performance similar to MCA; however, this fine-tuning process can be time-consuming and costly.

4.5. Summary and implications

We aim to improve the separability of TPs and FPs by estimating and evaluating the uncertainty associated with predictions of CNN-based classifiers. Particularly, we want the predictive confidence measuring uncertainty to be high for TPs and low for FPs. Therefore, we developed MCA inspired by the MMCD, which combines the strength of ensemble and MCD. Similar to baseline, MCD relies on a unique solution shaped by a single CNN and evaluates the uncertainty associated with the extracted features. On the contrary, an ensemble evaluates multiple solutions shaped by multiple CNNs but does not evaluate the uncertainty associated with the extracted features. By applying MCD in ensemble members, MMCD evaluates multiple solutions shaped by multiple CNNs similar to an ensemble and also evaluates the uncertainty associated with the extracted features similar to MCD. However, MMCD can have the same drawback as MCD, that is, the modeler of MMCD can incorrectly specify the predefined distribution from which masks will be obtained for feature sampling. Moreover, the design process of MMCD can be more complex than that of an ensemble because the predefined distribution needs to be fine-tuned to a given architecture and dataset. Therefore, MCA replaces feature sampling operations (inherent in MMCD) with feature averaging operations. Particularly, MCA evaluates the uncertainty associated with the features extracted by one ensemble member by sequentially averaging it in a pairwise manner with the features extracted by other ensemble members. This is based on a rationale that features extracted by ensemble members are different. Feature sampling and averaging help to better capture the variabilities of ensemble members and to therefore improve the predictive uncertainty of the underlying ensemble. However, feature averaging can preserve the classification accuracy better than feature sampling, because feature averaging can preserve the similarity between perturbed features and the original ones better than feature sampling. In addition, feature averaging relies on features extracted by ensemble members, while feature sampling requires a predefined distribution, which relies on the dataset and architecture. Moreover, the outcome of feature averaging operations is deterministic while that of feature sampling operations is stochastic. Therefore, feature sampling can required more samples than feature averaging to evaluate the uncertainty associated with extracted features, owing to this stochastic nature. Consequently, this can increase the inference time. To provide empirical evidence supporting these theoretical findings, we empirically compared MCA and related methods.

5. Empirical comparison of MCA and related methods

We empirically compared MCA and related methods such as baseline (single CNN), MCD, ensemble, and MMCD. We expect a good method to estimate uncertainties to preserve the accuracy while giving a good estimate of uncertainty (confidence). Therefore, we empirically compared all methods with respect to the accuracy and quality of confidence. We evaluated the quality of confidence by assessing the degree of confidence calibration and the ability to separate TPs and FPs. Specifically, we assessed the degree of confidence calibration by evaluating the calibration errors. In addition, we assessed the ability to separate TPs and FPs by evaluating the average confidence. Appendix A.5 presents all evaluation metrics (accuracy, expected calibration error (ECE), and average confidence). In addition, Appendix A.2 presents the experimental architectures. Moreover, Appendix A.1 presents the experimental datasets. Particularly, experiments were conducted on CIFAR10, MNIST, and Fashion-MNIST and GTSRB evaluated on DenseNets, VGGNets, and ResNets, respectively. Appendix A.4 presents all inference details.

5.1. Analyzing accuracy and calibration error

Tables 2 and 3 present the classification accuracy (CA), average confidence (AC), and ECE of MCA and related methods for CNNs with small and large capacities, respectively.

The results show that MCD can preserve the accuracy of the underlying baseline, especially for CNNs with large capacities. However, it can reduce the accuracy of the underlying baseline for CNNs with small capacities. For instance, as shown in Table 3, on CIFAR10, MCD decreased the accuracy of baseline to 85.76% from 86.02% for CNNs with large capacities. However, as shown in Table 2, on the same CIFAR10, MCD decreased the accuracy of baseline to 77.96% from 83.07% for CNNs with small capacities. This means that the decrease in the accuracy of baseline caused by MCD is minimal for CNNs with large capacities but large for CNNs with small capacities. Notwithstanding this, as shown in Table 2, whether MCD will (significantly) decrease the accuracy of baseline depends on the capacity of the underlying CNN and the dataset. For instance, as shown in Table 2, on MNIST, MCD decreased the accuracy of baseline to 98.26% from 98.39%.

Table 2.: classification accuracy (CA), average confidence (AC) (in bracket), and ECE obtained via CNNs with small capacities (see Table 13) trained using SR (see Table 9). The results were obtained using *test data*, as described in Appendix A.6

	CA (AC) [%]↑	**ECE** [%]↓
CIFAR10 (DenseNets)		
Baseline	83.07 (89.80)	7.00
MCD	77.96 (59.44)	18.53
Ensemble	89.46 (84.64)	**5.15**
MMCD	86.67 (57.58)	29.10
MCA	89.03 (68.04)	21.01
Fashion-MNIST (ResNets)		
Baseline	90.90 (91.82)	**1.68**
MCD	88.98 (74.80)	14.28
Ensemble	92.74 (87.99)	5.11
MMCD	91.47 (71.38)	20.10
MCA	92.32 (72.86)	19.55
MNIST (VGGNets)		
Baseline	98.39 (98.07)	**0.83**
MCD	98.26 (88.76)	9.54
Ensemble	98.95 (97.67)	1.39
MMCD	98.83 (89.41)	9.45
MCA	98.61 (88.43)	10.22
GTSRB (ResNets)		
Baseline	97.12 (98.79)	1.79
MCD	97.11 (91.91)	5.37
Ensemble	98.57 (97.59)	**1.13**
MMCD	98.50 (91.58)	6.93
MCA	97.95 (94.67)	3.34

Table 3.: Classification accuracy (CA), average confidence (AC) (in bracket), and ECE obtained via CNNs with large capacities (see Table 14) trained using SR (see Table 9). The results were obtained using *test data*, as described in Appendix A.6

	CA (AC) [%]↑	**ECE** [%]↓
CIFAR10 (DenseNets)		
Baseline	86.02 (86.36)	**2.06**
MCD	85.76 (70.58)	15.22
Ensemble	90.15 (83.48)	6.75
MMCD	89.67 (69.85)	19.82
MCA	89.75 (69.27)	**20.49**
Fashion-MNIST (ResNets)		
Baseline	88.58 (85.22)	**3.70**
MCD	89.07 (71.60)	17.48
Ensemble	92.99 (86.87)	6.34
MMCD	93.09 (75.64)	17.46
MCA	92.96 (69.88)	**23.09**
MNIST (VGGNets)		
Baseline	98.18 (98.14)	**0.74**
MCD	98.15 (94.53)	3.88
Ensemble	99.11 (98.25)	1.12
MMCD	99.13 (94.78)	4.48
MCA	99.13 (85.34)	**13.81**
GTSRB (ResNets)		
Baseline	93.41 (97.17)	3.87
MCD	93.38 (90.30)	3.54
Ensemble	94.68 (92.55)	**2.59**
MMCD	94.71 (85.74)	9.05
MCA	94.62 (77.56)	**17.18**

In addition, the results show that an ensemble can preserve or improve the accuracy of the underlying baseline for CNNs with small and large capacities. For instance, as shown in Table 2, on MNIST, ensemble increased the accuracy of baseline to 98.95% from 98.39%. However, on CIFAR10, ensemble increased the accuracy of baseline to 89.46% from 83.07%. Table 3 presents a similar pattern.

Moreover, the results show that MMCD can preserve the accuracy of the underlying ensemble, especially for CNNs with large capacities. However, it can reduce the accuracy of the underlying ensemble for CNNs with small capacities. For instance, as shown in Table 3, on CIFAR10, MMCD decreased the accuracy of the underlying ensemble to 89.67% from 90.15% for CNNs with large capacities. However, as shown in Table 2, on the same CIFAR10, MMCD decreased the accuracy of the underlying ensemble to 86.67% from 89.46% for CNNs with small capacities. This means that the decrease in the accuracy of the underlying ensemble caused by MMCD is minimal for CNNs with large capacities but large for CNNs with small capacities. Notwithstanding this, as shown in Table 2, whether MMCD will (significantly) decrease the accuracy of the underlying ensemble depends on the capacity of the underlying CNNs and the dataset. For instance, as shown in Table 2, on MNIST, MMCD decreased the accuracy of the underlying ensemble to 98.83% from 98.95%.

Further, the results show that MCA preserves the accuracy of the underlying ensemble for all datasets and for CNNs with small and large capacities. For instance, as shown in Table 2, on CIFAR10, MCA decreased the accuracy of the underlying ensemble to 89.03% from 89.46% for CNNs with small capacities. In addition, as shown in Table 3, on the same CIFAR10, MCA decreased the accuracy of the underlying ensemble to 89.75% from 90.15% for CNNs with large capacities.

Finally, the results show that whether the calibration error of baseline is better than that of an ensemble depends on the capacity of the underlying CNNs and the dataset. For instance, as shown in Table 3, on all datasets except GTSRB, the ECE of baseline is lower than that of ensemble. However, as shown in Table 2, on all datasets except CIFAR10, the ECE of baseline is lower than that of ensemble. In addition, the results show that an ensemble is better calibrated than MCD, MMCD, and MCA owing to the low ECE of ensemble. Moreover, the results show that MCD is (often) better calibrated than MMCD and MCA owing to the low ECE of MCD. However, whether MMCD is better calibrated than MCA depends on the capacity of the underlying CNNs and the dataset. For instance, as shown in Table 3, on CIFAR10, the ECE of MMCD is lower than that of MCA for CNNs with large capacities. On the contrary, as shown in Table 2, on the same CIFAR10, the ECE of MCA is lower than that of MMCD for CNNs with small capacities. However, as shown in Tables 2 and 3, on MNIST, the ECE of MMCD is lower than that of MCA for CNNs with small and large capacities.

5.2. Analyzing the ability to separate TPs and FPs

To evaluate the ability of MCA and related methods to separate TPs and FPs, we compared their average confidence on evaluation data causing TPs and FPs. Specifically, we obtained TPs on *subsets of correctly classified test data*, while FPs caused by struc-

Table 4.: The average confidence (AC) obtained via CNNs with small capacities (see Table 13) trained using SR (see Table 9) and evaluated on datasets, thereby generating TPs and FPs

	TPs ↑	**FPs (OOD)** ↓	**FPs (Swap)** ↓	**FPs (Noisy)** ↓
CIFAR10 (DenseNets) : AC [%]				
Baseline	**97.60**	35.64	74.89	100.00
MCD	67.92	22.48	42.50	41.55
Ensemble	97.51	39.82	61.20	51.25
MMCD	68.74	**21.78**	**37.98**	**26.29**
MCA	82.11	33.76	42.39	37.34
Fashion-MNIST (ResNets) : AC [%]				
Baseline	**97.52**	74.23	78.04	99.99
MCD	81.53	**41.49**	50.26	39.76
Ensemble	96.46	70.00	58.73	26.57
MMCD	79.30	45.19	**41.57**	**21.90**
MCA	80.84	44.56	43.05	35.04
MNIST (VGGNets) : AC [%]				
Baseline	**99.28**	81.95	74.22	100.00
MCD	90.56	**59.03**	49.37	**34.76**
Ensemble	99.23	80.97	60.62	77.05
MMCD	90.84	60.10	**43.68**	50.77
MCA	89.49	62.38	46.18	78.09
GTSRB (ResNets) : AC [%]				
Baseline	**99.90**	86.46	72.90	45.84
MCD	93.20	**40.41**	41.33	27.27
Ensemble	99.83	88.71	45.28	26.53
MMCD	93.21	48.31	**30.64**	**17.56**
MCA	97.17	76.86	34.94	21.89

Table 5.: The average confidence (AC) obtained via CNNs with large capacities (see Table 14) trained using SR (see Table 9) and evaluated on datasets, thereby generating TPs and FPs

	TPs ↑	**FPs (OOD)** ↓	**FPs (Swap)** ↓	**FPs (Noisy)** ↓
CIFAR10 (DenseNets) : AC [%]				
Baseline	95.93	88.29	57.88	64.43
MCD	82.69	35.72	38.56	39.18
Ensemble	**96.40**	38.41	50.28	51.98
MMCD	83.47	24.34	**35.63**	**27.97**
MCA	**83.20**	**19.40**	36.17	30.10
Fashion-MNIST (ResNets) : AC [%]				
Baseline	91.89	70.40	53.34	99.73
MCD	78.32	49.57	39.80	69.67
Ensemble	**95.06**	49.83	55.71	58.04
MMCD	83.26	37.40	44.21	38.81
MCA	**77.12**	**32.83**	**37.83**	**35.83**
MNIST (VGGNets) : AC [%]				
Baseline	**99.38**	60.86	61.58	99.30
MCD	96.00	44.93	49.39	93.28
Ensemble	99.34	55.95	52.59	81.46
MMCD	95.88	48.23	43.02	69.89
MCA	**86.40**	**38.88**	**34.91**	**58.49**
GTSRB (ResNets) : AC [%]				
Baseline	**99.71**	56.87	53.64	94.44
MCD	94.07	26.09	31.23	43.29
Ensemble	99.13	34.10	39.20	32.67
MMCD	92.22	17.93	27.58	21.87
MCA	**84.32**	**16.13**	**21.27**	**12.01**

tural perturbation of objects were obtained on *swap* data. In addition, FPs caused by perturbation of objects with Gaussian noise were obtained on *noisy* data, and FPs caused by objects of unknown domain were obtained on *OOD* data. Appendix A.6 presents all evaluation data. Tables 4 and 5 present the results obtained on these evaluation data for CNNs with small capacities and large capacities, respectively. Note that TPs and FPs are separable when the confidence for TPs is high and the confidence for FPs is low. Therefore, we expect the average confidence to be high on evaluation data causing TPs and low on evaluation data causing FPs.

The results show that the average confidence of ensemble for TPs is similar to or even higher than that of baseline, which is true for all datasets and for CNNs with small and large capacities. However, the average confidence of MCD, MMCD, and MCA for TPs is (often) lower than that of the underlying baseline. For instance, as shown in Table 5, on CIFAR10, ensemble increased the average confidence for TPs of baseline to 96.40% from 95.93%, while MCD reduces it to 82.69%. Similarly, MMCD and MCA reduced the average confidence for TPs of the underlying ensemble to around 83% from 96.40%. This means that *while an ensemble can preserve or increase the degree of confidence for TPs, MCD, MMCD, and MCA can reduce it.* In addition, the results show that *the decreased degree of confidence (caused by MCD and MMCD) for TPs depends on the datasets and the capacity of the underlying CNNs.* For instance, as shown in Table 5, on CIFAR10 evaluated on CNNs with large capacities, MMCD reduced the average confidence for TPs of the underlying ensemble to 83.47% from 96.40%. On the contrary, as shown in Table 4, on the same CIFAR10 evaluated on CNNs with small capacities, MMCD reduced the average confidence for TPs of the underlying ensemble to 68.74% from 97.51%. Herein, the reduced average confidence for TPs is larger for CNNs with small capacities than for CNNs with large capacities.

Furthermore, the results show that the average confidence of baseline for all FPs is (often) larger than that of all other methods, which is true for all experiments. This means that *MCD, ensemble, MMCD, and MCA reduced the degree of confidence for FPs.* In addition, the results imply that *whether an ensemble reduces the confidence for FPs better than MCD depends on the dataset, capacity of the underlying CNNs, and/or type of FPs.* For instance, as shown in Table 5, on all datasets except CIFAR10, the average confidence of ensemble for FPs owing to noisy data is lower than that of MCD. However, on all datasets including CIFAR10, the average confidence of ensemble for FPs owing to swap and OOD data is higher than that of MCD. Table 4 presents a similar pattern between ensemble and MCD.

In addition, the results show that the average confidence of MMCD and MCA on all FPs are (often) lower than that of the underlying ensemble, which is true for all experiments. This means that *MMCD and MCA reduced the confidence of the underlying ensemble for all FPs.* Finally, the results show that MCA can maintain a low confidence for all FPs similar to or sometimes even better than MMCD.

Table 6.: Mean and standard deviation of inference time (in seconds) obtained over 100 test samples via CNNs with small capacities trained using SR

	Time [s] ↓
CIFAR10 (DenseNets)	
Baseline	0.03 ± 0.01
MCD	0.50 ± 0.02
Ensemble	0.57 ± 0.03
MMCD	10.58 ± 0.30
MCA	2.46 ± 0.32
Fashion-MNIST (ResNets)	
Baseline	0.03 ± 0.01
MCD	0.54 ± 0.03
Ensemble	0.72 ± 0.01
MMCD	10.58 ± 0.28
MCA	2.33 ± 0.11
MNIST (VGGNets)	
Baseline	0.02 ± 0.01
MCD	0.80 ± 0.05
Ensemble	0.49 ± 0.02
MMCD	13.21 ± 0.49
MCA	2.94 ± 0.38
GTSRB (ResNets)	
Baseline	0.05 ± 0.01
MCD	0.98 ± 0.09
Ensemble	1.12 ± 0.08
MMCD	15.74 ± 0.70
MCA	3.59 ± 0.22

Table 7.: Mean and standard deviation of inference time (in seconds) obtained over 100 test samples via CNNs with large capacities trained using SR

	Time [s] ↓
CIFAR10 (DenseNets)	
Baseline	0.06 ± 0.01
MCD	1.33 ± 0.09
Ensemble	1.22 ± 0.05
MMCD	22.30 ± 0.42
MCA	5.00 ± 0.25
Fashion-MNIST (ResNets)	
Baseline	0.06 ± 0.01
MCD	1.14 ± 0.07
Ensemble	1.21 ± 0.05
MMCD	21.72 ± 0.56
MCA	4.95 ± 0.14
MNIST (VGGNets)	
Baseline	0.03 ± 0.01
MCD	1.72 ± 0.22
Ensemble	0.76 ± 0.05
MMCD	25.45 ± 0.96
MCA	4.92 ± 0.15
Inference time$^*[s]$	
Baseline	0.08 ± 0.01
MCD	1.53 ± 0.11
Ensemble	1.69 ± 0.10
MMCD	$\mathbf{30.31 \pm 1.04}$
MCA	6.58 ± 0.32

5.3. Analyzing the inference time

To evaluate the computational cost for uncertainty estimation, we compared the inference time (in seconds) of MCA and related methods. Tables 6 and 7 summarize the results for CNNs with small and large capacities, respectively. The results show that the inference time is larger for CNNs with large capacities than CNNs with small capacities. This means that *the increase in the capacity of CNNs increased the inference time.* Moreover, the results show that MMCD has the largest inference time followed by MCA. Particularly, *the inference time of MMCD is four times larger than that of MCA, which is* *also significantly larger than that of ensemble and MCD.* However, *whether the inference time of ensemble is larger than that of MCD depends on the architecture and/or capacity of CNNs.* For instance, as shown in Table 6, on CIFAR10, the inference time of MCD is larger than that of the ensemble. On the contrary, as also shown in Table 6, on the same CIFAR10, the inference time of ensemble is larger than that of MCD. Finally, the results show that baseline has the smallest inference time among the methods.

5.4. Summary and implications

We empirically compared MCA and related methods (such as baseline (single CNN), MCD, ensemble, and MMCD) based on the results from experiments conducted on four datasets using three different architectures. The comparison was done considering their accuracy, calibration errors, ability to separate TPs and FPs based on the evaluation of the degree of confidence, and inference time.

Empirical results show that while MCD can preserve or decrease the accuracy of the underlying baseline depending on the capacity of the underlying CNN and the dataset, an ensemble can preserve or increase it. This finding is supported by previous studies [110, 111], which demonstrated that an ensemble can increase accuracy. In addition, Section 3.2.2 discusses the possible reasons explaining why an ensemble can improve accuracy. However, while MMCD can preserve or decrease the accuracy of the underlying ensemble depending on the capacity of the underlying CNNs and the dataset, MCA can only preserve it. We argue that the decreased accuracy of baseline and ensemble caused by MCD and MMCD, respectively, results from the incorrect specification of the predefined distribution from which masks for feature sampling (inherent in MCD and MMCD) are obtained. Specifically, the parameters shaping the predefined distribution need to be fine-tuned according to the capacity of the underlying CNNs and the dataset. In other words, the dropout probability of 0.5 is not optimal for all CNNs of various capacities and all datasets. Therefore, given a specific CNN and dataset, one needs to find the optimal dropout probability. The process of finding the optimal dropout probability can make the design of MCD and MMCD time-consuming and costly. We note that MCA preserves the accuracy of the underlying ensemble for all datasets and CNNs of various capacities because feature averaging (inherent in MCA) preserves the similarity between augmented and original features (see Section 4.4).

Further, the results show that whether ensemble is better calibrated than baseline depends on the capacity of the underlying CNNs and the dataset. Notwithstanding this, an ensemble is better calibrated than MCD, which is in turn (often) better calibrated than MMCD and MCA. This is because MCD, MMCD, and MCA reduce the degree of confidence for TPs and the larger the decrease in the degree of confidence for TPs, the larger the calibration error. Moreover, an ensemble can reduce the degree of confidence for FPs, thereby increasing or preserving the degree of confidence for TPs. On the contrary, MCD, MMCD, and MCA can reduce the degree of confidence for FPs, thereby reducing the degree of confidence for TPs. For MCD and MMCD, the decreased degree of confidence for TPs depends on the dataset and capacity of the underlying CNNs. In addition, whether MMCD is better calibrated than MCA depends on the capacity of the underlying CNNs and the dataset.

Furthermore, whether an ensemble reduces the degree of confidence on FPs better than MCD depends on the dataset, capacity of the underlying CNNs, and/or type of FPs. This finding implies that we cannot claim that an ensemble captures uncertainty (e.g., degree of confidence) better than MCD and vice versa. This contradicts with some previous studies [78, 83, 79], which claim that an ensemble captures uncertainty better than MCD. Notwithstanding this, MMCD and MCA reduced the confidence of the underlying ensemble on all FPs. This means that MMCD and MCA capture uncertainty better than an ensemble. This is because MMCD and MCA evaluate not only multiple features extracted by ensemble members similar to an ensemble but also the uncertainty associated with the individual feature. By evaluating the uncertainty associated with the individual feature thanks to feature sampling (inherent in MMCD) and feature averaging (inherent in MCA), MMCD and MCA capture the diversity between the ensemble members better than an ensemble and therefore improve the uncertainty. In addition, the results show that MCA can maintain a low confidence for all FPs similar to or sometimes even better than MMCD.

Finally, the results show that baseline has the smallest inference time compared to other methods. This is not surprising because baseline evaluates a single CNN and performs a single forward pass, while other methods evaluate multiple CNNs or perform multiple forward passes. This means that MCD, ensemble, MMCD, and MCA improve uncertainty at the cost of increasing the inference time. Notwithstanding this, whether the inference time of ensemble is larger than that of MCD depends on the architecture and/or capacity of CNNs. Moreover, although the inference time of MCA is significantly larger than that of ensemble and MCD, it is four times smaller than that of MMCD.

Overall, the empirical results imply the dominance of MMCD and MCA over previous methods, such as MCD and ensemble. In addition, assuming that MMCD and MCA are well-specified for a given dataset and architecture, then both methods can have similar performance because they have the same purpose and underlying principle. However, the design process of MMCD is more complex than that of MCA. This is because MMCD requires the specification of a predefined distribution, from which masks will be obtained for feature sampling, while MCA relies on features extracted by ensemble members for feature averaging. Moreover, the inference time of MMCD is four times larger than that of MCA owing to the large amount of feature sampling operations.

However, the main drawback of MMCD and MCA is that they reduce not only the degree of confidence for FPs but also for TPs. The decreased degree of confidence for

TPs can increase the calibration error. If the confidence drop for TPs is too large, it can harm the separability between FPs and TPs, for example, when the degree of confidence for TPs falls in the intervals 0% and 50%. Therefore, ameliorating the confidence drop for TPs can further improve the performance of MMCD and MCA. In Chapter 6 and Chapter 7, we attempted to solve this issue.

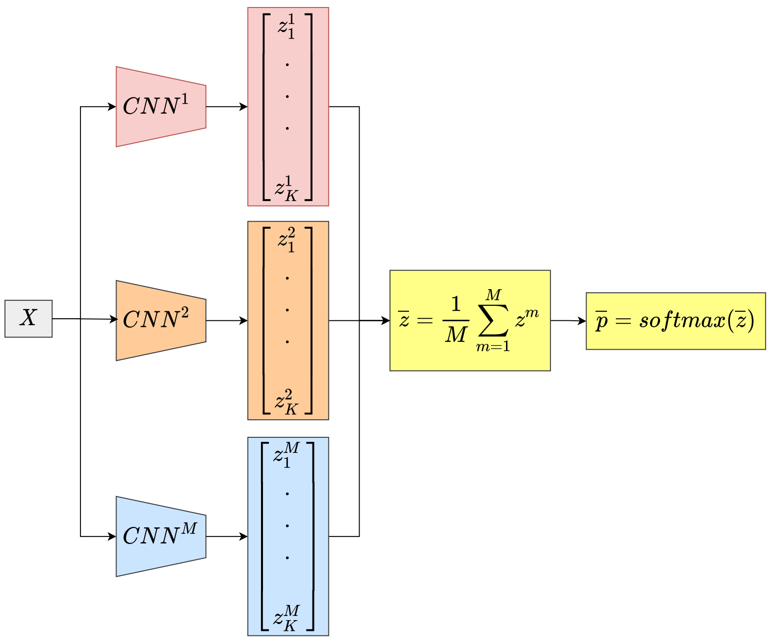
6. Addressing underconfidence by averaging logit instead of probability

6.1. Motivation

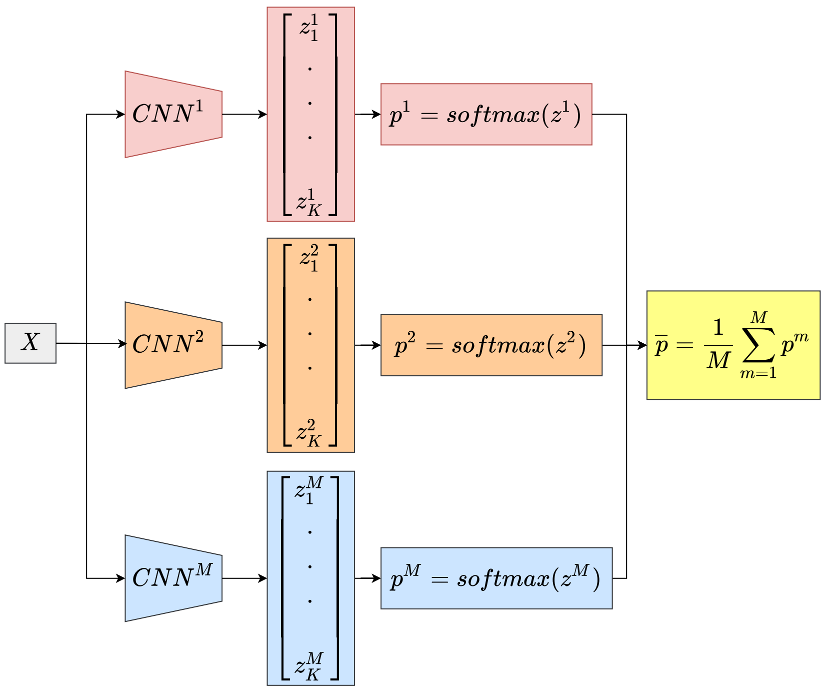
The main drawback of MCA and MMCD is that they reduce not only the confidence for FPs but also for TPs. Although the confidence drop for FPs is relevant, the confidence drop for TPs is alarming. In addition, we argue that the confidence drop for TPs is caused by inductive biases inherent in ensemble members or introduced by feature sampling (inherent in MMCD) or feature averaging (inherent in MCA). Therefore, we utilized logit instead of probability averaging in MCA and related methods (as shown in our research paper [102]) to reduce the level of inductive biases negatively influencing the confidence.

6.2. Using logit instead of probability averaging

In the CNN-based classifiers, the predictions of multiple ensemble members are combined by averaging softmax outputs (probabilities). However, as derived from our research paper [102], we can average softmax inputs (logits) instead of probabilities to reduce the influence of inductive biases and increase the confidence of the ensemble prediction, as shown in Figure 24 taken from [102]. Logit averaging is due to the rationale that logits, which are established evidences for possible classes [126], are continuous values normalized by the softmax function to produce discrete probabilities. The softmax normalization of continuous values (logits) to discrete values (probabilities) causes robustness to changes in magnitudes of logits and a possible loss of information. The possible loss of information owing to softmax normalization and/or the inductive biases (inherent in logits) can negatively affect the confidence of individual ensemble members, which limits the confidence of the ensemble. However, by applying logit averaging, we can reduce the inductive biases and avoid the loss of information at the cost of being sensitive to changes in the magnitudes of logits. Therefore, we can increase the confidence of the ensemble, as shown in Figure 25. Intuitively, logit averaging provides the best evidence (characterized by a low level of uncertainty caused by the reduction of inductive biases) for



(a) Logit averaging



(b) Probability averaging

Figure 24.: An illustration of the difference between logit and probability averaging. Herein, M CNNs are averaged to estimate the ensemble prediction

$$z^1 = \begin{bmatrix}50\\30\\10\end{bmatrix} \qquad z^2 = \begin{bmatrix}5\\3\\2\end{bmatrix} \qquad z^3 = \begin{bmatrix}3\\2\\1\end{bmatrix} \qquad z^4 = \begin{bmatrix}2\\1\\0\end{bmatrix} \qquad \overline{z} = \begin{bmatrix}15\\9\\3.25\end{bmatrix}$$

$$\Big\downarrow \text{Softmax}$$

$$\underbrace{p^1 = \begin{bmatrix}1\\0\\0\end{bmatrix} \qquad p^2 = \begin{bmatrix}0.85\\0.11\\0.04\end{bmatrix} \quad p^3 = \begin{bmatrix}0.66\\0.25\\0.09\end{bmatrix} \quad p^4 = \begin{bmatrix}0.67\\0.24\\0.09\end{bmatrix}}_{\overline{p} = \begin{bmatrix}0.79\\0.15\\0.06\end{bmatrix}} \qquad p_{\overline{z}} = \begin{bmatrix}1\\0\\0\end{bmatrix}$$

Figure 25.: An example showing how logit averaging increased the confidence of an ensemble of four deterministic CNNs: Herein, $\overline{z} = \frac{1}{4}\sum_{m=1}^{4} z^m$, $p_{\overline{z}} = softmax(\overline{z})$, and $\overline{p} = \frac{1}{4}\sum_{m=1}^{4} p^m$ with $p^1 = softmax(z^1)$, $p^2 = softmax(z^2)$, $p^3 = softmax(z^3)$, and $p^4 = softmax(z^4)$. We can see that logit averaging $p_{\overline{z}}$ results in more confident predictions than probability averaging $\overline{p}$. This is because logit averaging is more sensitive to the magnitude of logits. Here, z^m with large values contribute the most to $\overline{z}$. Particularly, $\overline{z}$ is mostly influenced by the values of z^1, that is, the contributions of z^2, z^3, and z^4 to $\overline{z}$ are low. On the contrary, $\overline{p}$ is influenced by the values of all probability vectors p^m and is less sensitive to the magnitude of individual logits z^m

making decisions. However, probability averaging provides the best confidence regarding decisions made using (weak) evidence (characterized by a high level of uncertainty caused by inductive biases). This implies that a decision made based on probability averaging considers more uncertainty than the one based on logit averaging. To reduce the level of uncertainty in ensemble predictions, we can apply logit instead of probability averaging. As shown in Figure 24, given an ensemble of M deterministic CNNs with logits z^m, the average logit $\overline{z}$ can be estimated by

$$\overline{z} = \frac{1}{M}\sum_{m=1}^{M} z^m = \frac{1}{M}\sum_{m=1}^{M} f_m(x). \tag{6.1}$$

and the predicted probability vector of the ensemble can be reformulated as

$$\overline{p}(\hat{y}|x) = softmax(\overline{z}). \tag{6.2}$$

Given MCD representing an ensemble of S stochastic CNNs with logits z^s, we can estimate the average logit $\overline{z}$ as

$$\overline{z} \approx \frac{1}{S}\sum_{s=1}^{S} z^s \approx \frac{1}{S}\sum_{s=1}^{S} f_s(x), \tag{6.3}$$

and reformulate the predicted probability vector of MCD, as denoted in Equation 6.2. Similarly, given MMCD representing an ensemble of $M \cdot S$ stochastic CNNs with logits z^{m_s}, we can estimate the average logit $\overline{z}$ as

$$\overline{z} \approx \frac{1}{M \cdot S}\sum_{m=1}^{M}\sum_{s=1}^{S} z^{m_s} \approx \frac{1}{M \cdot S}\sum_{m=1}^{M}\sum_{s=1}^{S} f_{m_s}(x), \tag{6.4}$$

and reformulate the predicted probability vector of MMCD, as denoted in Equation 6.2. Finally, given MCA representing an ensemble of $M \cdot M$ deterministic CNNs with logits z^{m_n}, we can estimate the average logit $\overline{z}$ as

$$\overline{z} \approx \frac{1}{M \cdot M}\sum_{m=1}^{M}\sum_{n=1}^{M} z^{m_n} \approx \frac{1}{M \cdot M}\sum_{m=1}^{M}\sum_{n=1}^{M} f_{m_n}(x), \tag{6.5}$$

and reformulate the predicted probability vector of MCA, as shown in Equation 6.2. In the following sections, we evaluated the impact of applying logit instead of probability averaging in MCA and related methods regarding their accuracy, calibration error, and ability to separate TPs and FPs.

6.3. Impact of logit averaging on accuracy and calibration error

Table 8 summarizes the classification accuracy, average confidence, and ECE of MCA and related methods for logit and probability averaging. The results show that the classification accuracy is nearly the same for both averaging, which indicates that *logit averaging preserves the accuracy.* In addition, the results show that the average confidence is always larger for logit than probability averaging, which indicates that *logit averaging increases the degree of confidence.* Figure 25 presents the manner in which the degree of confidence increases. Further, the results show that the ECE is always smaller for logit than probability averaging, which means that *logit averaging can reduce the calibration error of MCA and related methods.* This is because the increased degree of confidence caused by logit averaging reduces the gap between the average confidence and classification accuracy. For instance, on CIFAR10, applying logit instead of probability averaging in MMCD reduces the gap between the average confidence and classification accuracy to 10.34(=|89.64-79.30|)% from 19.82(=|89.67-69.85|)%.

Table 8.: Classification accuracy (CA), average confidence (AC) (in bracket), and ECE for probability averaging (PA) and logit averaging (LA) obtained via CNNs with large capacities (summarized in Table 14) trained using SR (summarized in Table 9). The results were obtained using *test data*, as described in Appendix A.6

	CA (AC) ↑		**ECE ↓**	
	PA	LA	PA	LA
CIFAR10 (DenseNets)				
MCD	85.76 (70.58)	85.70 (**77.12**)	15.22	**8.61**
Ensemble	90.15 (83.48)	90.15 (**87.94**)	6.75	**2.68**
MMCD	89.67 (69.85)	89.64 (**79.30**)	19.82	**10.37**
MCA	89.75 (69.27)	89.64 (**74.22**)	20.49	**15.43**
Fashion-MNIST (ResNets)				
MCD	89.07 (71.60)	88.98 (**77.49**)	17.48	**11.51**
Ensemble	92.99 (86.87)	92.96 (**90.07**)	6.34	**3.31**
MMCD	93.09 (75.64)	93.08 (**83.69**)	17.46	**9.47**
MCA	92.96 (69.88)	92.89 (**77.52**)	23.09	**15.46**
MNIST (VGGNets)				
MCD	98.15 (94.53)	98.20 (**96.48**)	3.88	**1.94**
Ensemble	99.11 (98.25)	99.11 (**98.94**)	1.12	**0.56**
MMCD	99.13 (94.78)	99.12 (**97.55**)	4.48	**1.76**
MCA	99.13 (85.34)	99.15 (**91.71**)	13.81	**7.47**
GTSRB (ResNets)				
MCD	93.38 (90.30)	93.40 (**94.87**)	3.54	**1.90**
Ensemble	94.68 (92.55)	94.77 (**96.54**)	2.59	**2.09**
MMCD	94.71 (85.74)	94.68 (**94.37**)	9.05	**1.52**
MCA	94.62 (77.56)	94.53 (**86.32**)	17.18	**8.23**

6.4. Impact of logit averaging on the ability to separate TPs and FPs

Figure 27 presents the average confidence of MCA and related methods on evaluation data causing TPs and FPs. The results show that the average confidence for both TPs and FPs increased when we applied logit instead of probability averaging. This indicates that *applying logit instead of probability averaging in MCA and related methods can increase the degree of confidence for both TPs and FPs.* Moreover, the increased degree of confidence caused by logit averaging is sometimes of a huge margin, especially on FPs owing to noisy data. For instance, on CIFAR10, the average confidence of the ensemble on noisy data increases from around 50% to 85% when we applied logit instead

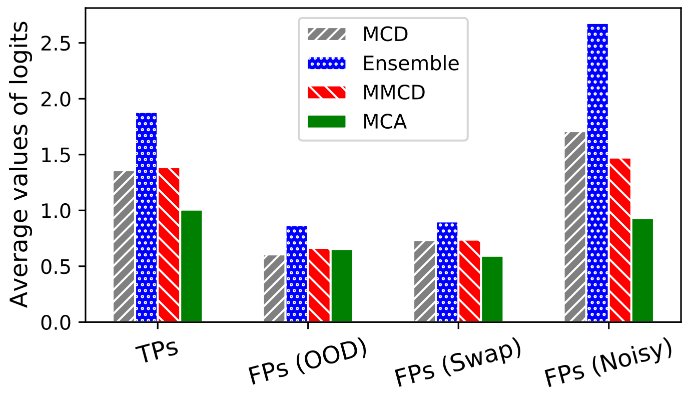


Figure 26.: Average values of logits obtained for TPs and FPs: TPs were obtained on *subsets of test data correctly classified*. FPs were obtained on *swap*, *noisy*, or *OOD* data, as described in Appendix A.6. This example shows that FPs caused by noisy data can increase the magnitude of logit values (see the ensemble). This experiment was conducted on CIFAR10 via CNNs with large capacities (summarized in Table 14) trained using SR (summarized in Table 9)

of probability averaging. This is because noisy data can increase the magnitude of logits, as shown in Figure 26, and logit averaging is more sensitive to changes in magnitude of logits than probability averaging (see Figure 25). This means that *logit averaging can negatively affect the separability between TPs and FPs, since it can maintain a high degree of confidence for both TPs and FPs.*

6.5. Summary and implications

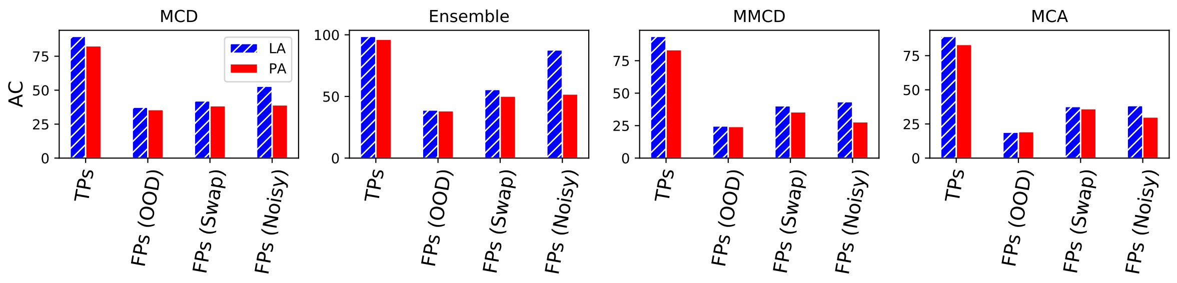
To ameliorate the confidence drop for TPs, we apply logit instead of probability averaging in MCA and related methods to reduce the level of inductive biases (inherent in ensemble members), which are responsible for the confidence drop for TPs. To evaluate the impact of applying logit averaging on properties (classification accuracy, calibration error, and ability to separate TPs and FPs) of MCA and related methods, we compared the results between logit and probability averaging.

The results show that logit averaging preserves the accuracy but increases the degree of confidence for both TPs and FPs. This is based on the rationale that logit averaging can preserve the position of the max element of individual logit vectors but is more sensitive to the magnitude of logit values than probability averaging. In other words, logit values with large magnitude contribute the most to the average logit. Herein, the magnitude of logit values induces a nonuniform weighting for logit averaging, which is not the case for probability averaging.

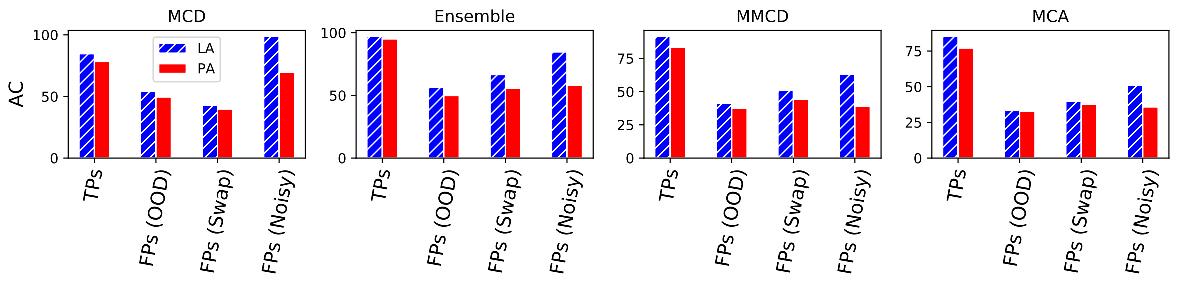
In addition, the results show that the increased degree of confidence caused by logit averaging can reduce the calibration error owing to the fact that it can reduce the gap between the average confidence and the classification accuracy.

However, this increased degree of confidence can negatively affect the separability of TPs and FPs, especially FPs owing to noisy data. This is because noisy data can increase the magnitude of logits and logit averaging is more sensitive to changes in the magnitude of logits than probability averaging.

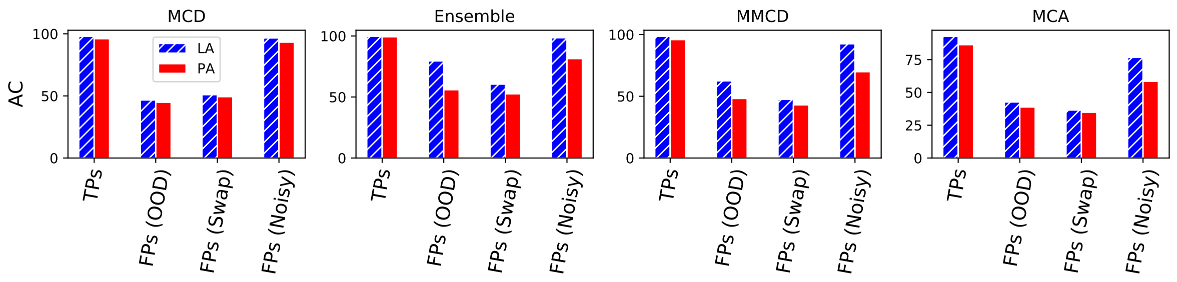
Since applying logit instead of probability averaging can reduce the calibration error at the cost of affecting the ability to separate TPs and FPs, then reducing the calibration error on test data and improving the ability to separate TPs and FPs are contradicting goals. Improving one may be detrimental to the other. Furthermore, given two models A and B, if A is better calibrated than B (with respect to the ECE), then A does not necessarily separate TPs and FPs better than B. This means that existing methods for confidence calibration may not help to improve the separability between TPs and FPs. To verify this hypothesis, *it is recommended that future works should evaluate the ability of existing methods for confidence calibration to maintain a low degree of confidence for FPs. We also recommend researchers evaluate not only the calibration error of a proposed method for confidence calibration but also the ability of the proposed method to maintain a low degree of confidence for FPs. Finally, for mission- and safety-critical applications where the separability of TPs and FPs is critical, we suggest to apply probability averaging (as it is traditionally done) to avoid the negative impact of logit averaging on the separability of TPs and FPs.*



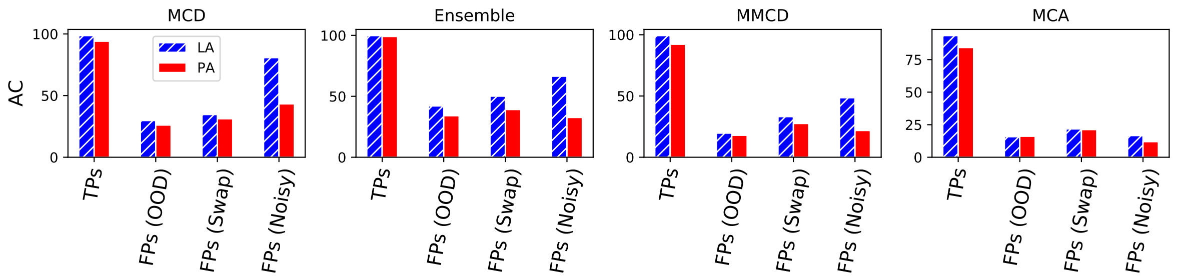
(a) CIFAR10



(b) Fashion-MNIST



(c) MNIST



(d) GTSRB

Figure 27.: The average confidence (AC) for probability averaging (PA) and logit averaging (LA) obtained via CNNs with large capacities (summarized in Table 14) trained using SR (summarized in Table 9) and evaluated on datasets generating TPs and FPs. TPs were obtained on *subsets of test data correctly classified*. FPs were obtained on *swap*, *noisy* and *OOD* data, as described in Appendix A.6

7. Addressing underconfidence by reducing the regularization strength

7.1. Motivation

In Chapter 6, to ameliorate the confidence drop for TPs, we applied logit instead of probability averaging in MCA and related methods to reduce the level of inductive biases, which are responsible for the confidence drop for TPs. We empirically found that applying logit instead of probability averaging in MCA and related methods can reduce the calibration error at the cost of affecting the ability to separate TPs and FPs. We therefore suggest to apply probability averaging (as it is traditionally done) to avoid the negative impact of logit averaging on the separability of TPs and FPs. However, the confidence drop for TPs still remains a concern. Therefore, we propose an alternative approach consisting of reducing the regularization strength applied at training, as discussed in our paper [103].

7.2. Regularization strength

Regularization is an approach for modifying the training data, optimization algorithm, or training objective to limit the growth of the parameters and, thus, speed up training, prevent overfitting, and/or improve generalization performance. From a Bayesian point of view, regularization can be seen as encoding specific kinds of prior knowledge. The strength of regularization can be controlled by reducing the number of regularization approaches applied and/or reducing the values of regularization hyperparameters. To evaluate the impact of reducing the strength of regularization on MCA and related methods, we compared the results obtained with weak regularization (WR) and strong regularization (SR), as shown in Table 9. Particularly, we compared the accuracy, calibration error, and ability of MCA and related methods to separate TPs and FPs for WR and SR. We observed that WR results in overconfident CNNs, while SR results in underconfident CNNs, respectively.

Table 9.: Summary of values assigned to regularization hyperparameters

Hyperparameters	**WR**	**SR**
Is batch normalization applied after max pooling layers?	Yes	Yes
Probability of dropout applied at inputs to max pooling layers	-	0.05
Probability of dropout applied at inputs to fully connected layers	0.01	0.5
Rotation range [degree]	-	[-45, +45]
Width and height shift range [pixel]	-	[-5, +5]
Scale intensity range	-	[0.9, 1.2]
Shear intensity range	-	0.1
Additive Gaussian noise standard deviation range	-	0.5

7.3. Effect of reducing regularization strength on accuracy and calibration error

Table 10 presents the classification accuracy, average confidence, and ECE of MCA and related methods for WR and SR.

The results show that, on CIFAR10, the classification accuracy of MCA and related methods (slightly) increased when using SR instead of WR. On the contrary, on Fashion-MNIST, the classification accuracy of MCA and related methods (slightly) decreased when using SR instead of WR. This means that, *the decreased strength of regularization can increase or decrease the classification accuracy depending on the dataset and/or architecture.*

In addition, the results show that *the decreased strength of regularization increases the average confidence.* For instance, on CIFAR10, the average confidence of ensemble increases to 89% from around 83% when using WR instead of SR.

Moreover, the results show that the ECE of MCA and related methods decreases when using WR instead of SR. For instance, on CIFAR10, the ECE of ensemble decreases to 2.47% from 6.75% when using WR instead of SR. This means that *the decreased strength of regularization decreases the calibration error of MCA and related methods.* This is because the decreased strength of regularization increases the average confidence and therefore reduces the gap between the accuracy and average confidence. For instance, on CIFAR10, the average confidence of ensemble increases from 83.48% to 89.26% when using WR instead of SR. Consequently, the gap between the classification accuracy and average confidence decreased to $1.24(= |88.02 - 89.26|)\%$ from $6.67 = |90.15 - 83.48|)\%$.

Table 10.: Classification accuracy (CA), average confidence (AC) (in bracket), and ECE obtained via CNNs with large capacities (summarized in Table 14) trained using WR and SR (summarized in Table 9). The results were obtained using *test data*, as described in Appendix A.6

	CA (AC) ↑		**ECE ↓**	
	WR	SR	WR	SR
CIFAR10 (DenseNets)				
MCD	83.56 (76.80)	85.76 (70.58)	**6.93**	15.22
Ensemble	88.02 (89.26)	90.15 (83.48)	**2.47**	6.75
MMCD	87.62 (74.62)	89.67 (69.85)	**13.07**	19.82
MCA	87.81 (82.96)	89.75 (69.27)	**4.98**	20.49
Fashion-MNIST (ResNets)				
MCD	92.23 (92.60)	89.07 (71.60)	**1.51**	17.48
Ensemble	94.52 (94.72)	92.99 (86.87)	**1.39**	6.34
MMCD	94.34 (91.05)	93.09 (75.64)	**3.54**	17.46
MCA	94.48 (92.06)	92.96 (69.88)	**2.81**	23.09
MNIST (VGGNets)				
MCD	99.30 (99.49)	98.15 (94.53)	**0.51**	3.88
Ensemble	99.62 (99.50)	99.11 (98.25)	**0.34**	1.12
MMCD	99.63 (99.36)	99.13 (94.78)	**0.47**	4.48
MCA	99.61 (98.55)	99.13 (85.34)	**1.20**	13.81
GTSRB (ResNets)				
MCD	94.33 (94.17)	93.38 (90.30)	**1.41**	3.54
Ensemble	97.24 (96.03)	94.68 (92.55)	**1.54**	2.59
MMCD	96.88 (93.13)	94.71 (85.74)	**3.88**	9.05
MCA	97.16 (93.22)	94.62 (77.56)	**4.02**	17.18

7.4. Effect of reducing regularization strength on ability to separate TPs and FPs

Figure 28 presents the average confidence of MCA and related methods for WR and SR.

The results show that *the decreased strength of regularization can increase the average confidence for TPs.* For instance, on Fashion-MNIST, the average confidence of MCD for TPs increases to 97% from around 78% when using WR instead of SR. In addition, on the same Fashion-MNIST, while the average confidence of MCD for TPs increases to 97% from around 78%, the average confidence of ensemble increases to 99% from around 95% when using WR instead of SR. Moreover, on MNIST, the average confidence of MCD for TPs increases to 99% from around 96%, while the average confidence of ensemble remains constant at 99% when using WR instead of SR. This means that *the level of increase in the average confidence for TPs caused by the reduction in the strength of regularization depends on the dataset, architecture, and/or model type.*

Further, on CIFAR10, the average confidence of ensemble for FPs due to OOD data increases to 99% from around 38% when using WR instead of SR. This indicates that the decreased strength of regularization can also increase the average confidence for FPs. Moreover, on CIFAR10 and Fashion-MNIST, the average confidence of ensemble for FPs due to OOD data is higher, but on MNIST, it is lower for WR than SR. In addition, on CIFAR10 and MNIST, the average confidence of MMCD for FPs due to swap data is higher, but on Fashion-MNIST, it is lower for WR than SR. This indicates that *whether the decrease in the strength of regularization will increase the average confidence for FPs depends on the dataset, architecture, and/or model type.* Finally, on MNIST, the average confidence of MCD for FPs due to noisy data is lower, but for FPs due to OOD data, it is higher for WR than SR. In addition, on Fashion-MNIST, the average confidence of ensemble and MMCD for FPs due to swap and noisy data is lower, but for FPs due to OOD data, it is higher for WR than SR. This indicates that *whether the decrease in the strength of regularization will increase the average confidence for FPs depends on the FP type.*

7.5. Summary and implications

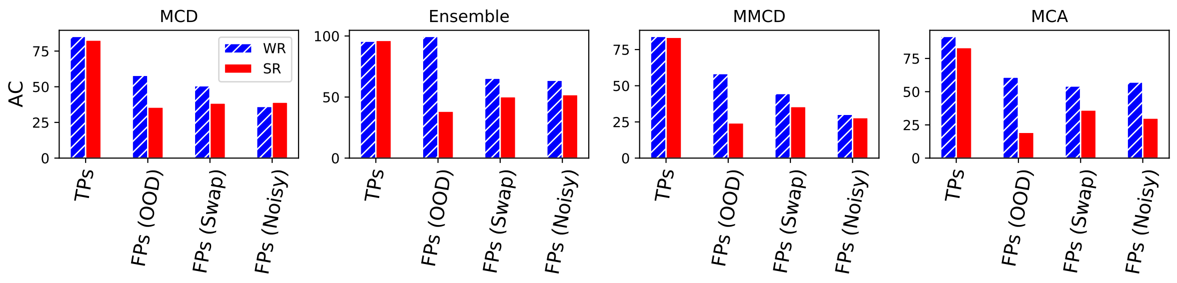
To ameliorate the confidence drop for TPs caused by inductive biases inherent in ensemble members, we proposed to reduce the strength of regularization applied at training (as discussed in our paper [103]). To evaluate the impact of the reduced strength of regularization on properties (classification accuracy, calibration error, and ability to separate TPs and FPs) of MCA and related methods, we compared results for WR and SR.

The results show that the decrease in the strength of regularization can increase or decrease the classification accuracy depending on the dataset and/or architecture. This is not surprising because the optimal strength of regularization depends on the dataset and/or architecture. This is because some datasets (e.g., GTSRB) have more noise inherent in their samples than others (e.g., MNIST) and samples of some datasets (e.g., CIFAR10) are more difficult to learn than of the others (e.g., MNIST). In addition, some architectures (e.g., DenseNet and ResNet) introduce a certain level of implicit regularization via the network depth or width. Therefore, the increased strength of regularization may not always improve the classification accuracy as is claimed by Ioffe and Szegedy [12] and Hinz, Barros, and Wermter [186].

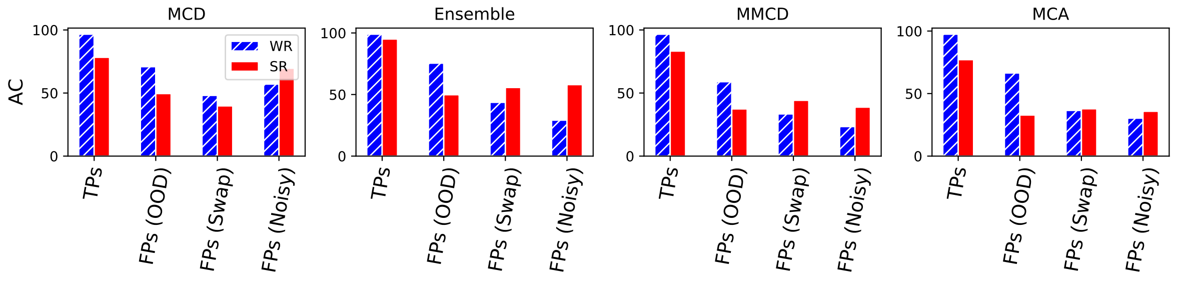
The results also show that the decreased strength of regularization decreases the calibration error of MCA and related methods. This is because the decrease in the strength of regularization increases the degree of confidence and therefore the average confidence. This increase in the average confidence reduces the gap between the classification accuracy and average confidence and therefore improves the calibration error.

Furthermore, the decreased strength of regularization can increase the degree of confidence for both TPs and FPs, and the level of increase in the degree of confidence depends on the dataset, architecture, model type, and/or FP type. This is because the decreased strength of regularization can increase the values of logits, and the increased values of logits can increase the degree of confidence for both TPs and FPs depending on the dataset, architecture, model type, and/or FP type. This implies that the decreased strength of regularization can negatively affect the separability between TPs and FPs (based on the evaluation of the degree of confidence) depending on the dataset, architecture, model type, and/or FP type, because a WR can maintain a high degree of confidence for both TPs and FPs.

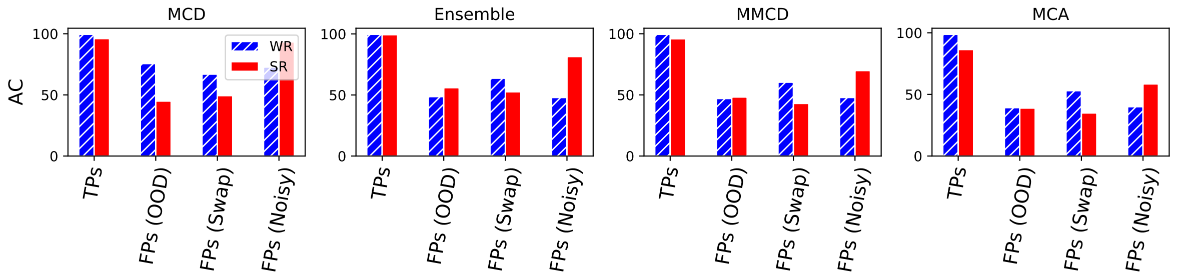
Overall, the reduced strength of regularization can ameliorate the confidence drop for TPs and therefore improves the calibration error at the cost of affecting the separability between TPs and TPs. This finding suggests that reducing the calibration error on test data and improving the ability to separate TPs and FPs are two contradicting goals. Improving one may be detrimental to the other. Further, given two models A and B, if A is better calibrated than B (with respect to the ECE), then A does not necessarily separate TPs and FPs better than B. This means that existing methods for confidence calibration may not help to improve the separability between TPs and FPs. To verify this hypothesis, it is recommended that future works should evaluate the ability of existing methods for confidence calibration to maintain a low degree of confidence for FPs. We also recommend researchers to evaluate not only the calibration error of a proposed method for confidence calibration but also the ability of the proposed method to maintain a low degree of confidence for FPs. Finally, for mission- and safety-critical applications where the separability of TPs and FPs is critical, we suggest to apply MCA and related methods with CNNs trained using SR (as it is traditionally done) to avoid the negative impact of WR on the separability of TPs and FPs.



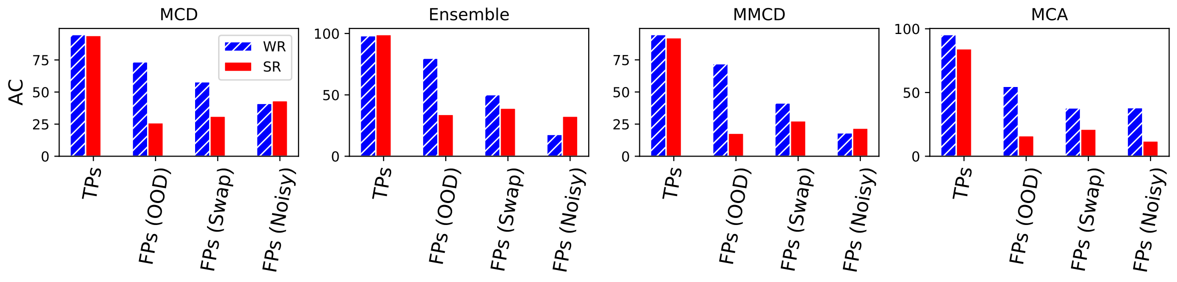
(a) CIFAR10



(b) Fashion-MNIST



(c) MNIST



(d) GTSRB

Figure 28.: The average confidence (AC) obtained via CNNs with large capacities (summarized in Table 14) trained using WR and SR (summarized in Table 9) and evaluated on datasets generating TPs and FPs. TPs were obtained on *subsets of test data correctly classified*. FPs were obtained on *swap*, *noisy*, and *OOD* data, as described in Appendix A.6

8. Discussion

8.1. Research objective and questions

A CNN-based classifier is prone to overfitting and lacks robustness to undesired inputs (e.g., OOD, swap, or noisy examples). Therefore, it can generate overconfident false predictions (FPs), which can be dangerous and costly, especially when it is part of the decision-making unit of a safety- and/or mission-critical application. For instance, overconfident FPs can

(a) Cause collisions in robotic applications

(b) Provide false treatments in medical applications, or

(c) Increase cost in financial applications.

To avoid these consequences and encourage the widespread use of CNN-based classifiers in safety- and/or mission-critical applications, we aim to prevent FPs by improving the separability of FPs and true predictions (TPs). Therefore, we seek to improve the quality of the confidence (measuring uncertainty) in terms of maintaining a high confidence for TPs and low confidence for FPs. This is based on the hypothesis that if the confidence is high (e.g.,]0.5, 1]) for TPs and low (e.g., [0, 0.5]) for FPs, then both TPs and FPs can be well-separated using a threshold. However, we require a method that forces CNN-based classifiers to generate FPs with low confidence and TPs with high confidence. Therefore, we formulated the three following research questions:

(1) What method forces the predictive confidence to be high for TPs and low for FPs?

(2) Under what circumstances does the method work?

(3) At what cost the method helps to maintain a low confidence for FPs and a high confidence for TPs?

8.2. Research methodology

To address the first research question (**What method forces the predictive confidence to be high for TPs and low for FPs?**), we overcame the following challenges presented in Section 1.6:

(a) The challenge in understanding the underlying functionality of CNNs. This challenge introduces two difficulties. The first difficulty is our inability to impose some constraints on parameters of CNNs because we do not know the relevant parameters, the manner in which the relevant parameters affect predictions, and the constraints to impose on them. To overcome this first difficulty, we treated CNNs as black-boxes. This allows us to focus on the prediction instead on the manner in which the prediction arrives. The second difficulty is our inability to trace the source of uncertainty inherent in a prediction. To overcome this second difficulty, we (only) evaluate uncertainty of well-known sources. Specifically, we used five evaluation data for different purposes. We used *test data* to evaluate the classification accuracy and calibration error. In addition, we used *subsets of correctly classified test data*, *OOD*, *swap*, and *noisy* data to evaluate the ability to separate TPs and FPs caused by distribution shift, structural perturbation, and noise, respectively. These data were presented in Appendix A.6.

(b) The challenge in choosing a principle for uncertainty estimation. Specifically, existing methods for estimating uncertainty adhere to five distinct principles. Understanding the five distinct principles, as well as, their various implementations was mandatory in deciding which principle to follow and/or how to combine the existing principles. To overcome this challenge, we summarized and discussed existing methods for estimating uncertainty in our survey paper [99]. One of the main suggestions of our survey paper is to build methods for uncertainty estimation that combine the strengths of both Bayesian (single-mode exploration) and ensemble (multimode evaluation) principles. We followed this suggestion and developed Monte Carlo averaging (MCA), which is similar to mixture of Monte Carlo dropout (MMCD), which combines the strengths of both ensemble and Monte Carlo dropout (MCD).

(c) The need for identical experimental setups to compare the developed MCA and related methods (baseline (single CNN), MCD, ensemble, and MMCD). To overcome this challenge, we reimplemented baseline, MCD, ensemble, and MMCD. This was helpful in understanding the methods and it sheds light on the aspects of the implementation that could affect the results.

In summary, to address the first research question, we developed MCA and empirically compared it to related methods. To tackle the second research question (**Under what circumstances the developed method works?**), we evaluated the performance of the developed and related methods on four different datasets (CIFAR10, Fashion-MNIST, MNIST, and GTSRB) to assess their ability to perform on datasets with different difficulties. We expect the performance of all methods to depend on the task difficulty. This is because some datasets (e.g., GTSRB) have more noise inherent in their samples than others (e.g., MNIST). Besides, samples of some datasets (e.g., CIFAR10) are more difficult to learn than samples of others (e.g., MNIST). We also evaluated the performance of the developed and related methods on CNNs of three different architectures (VGGNet, DenseNet, and ResNet) to assess their ability to perform on different architectures. We expect the performance of all methods to depend on the architecture. This is because the architecture conditions the manner in which information is propagated from the input to subsequent layers, and different architectures will result in different gradient calculations and therefore to different solutions. Further, we investigated the impact of applying logit instead of probability averaging in the developed and related methods, as well as, the effect of reducing the regularization strength on the performance of the developed and related methods. This was motivated by the hypothesis that both logit averaging and the reduction of the strength of regularization can reduce the level of inductive biases inherent in CNNs and therefore help to address underconfidence in the developed and related methods. To approach the third research question (**At what cost the developed method helps to maintain a low confidence for FPs and a high confidence for TPs?**), in addition to analyze the ability of the developed and related methods to separate TPs and FPs, we also analyzed the classification accuracy, the calibration error, and the inference time. We expect a good method for uncertainty estimation to preserve the classification accuracy on the test data, to exhibit a low calibration error on test data, and to maintain a high degree of confidence for TPs and low degree of confidence for FPs.

8.3. Research results

Theoretically, MCD relies on a unique solution obtained using a single CNN similar to baseline; however, it considers the uncertainty associated with the extracted features. In contrast to MCD, ensemble evaluates multiple solutions obtained using multiple CNNs but does not consider the uncertainty associated with the extracted features. Similar to ensemble, MMCD evaluates multiple solutions obtained using multiple CNNs but also considers the uncertainty associated with the extracted features similar to MCD. However, the modeler of MCD or MMCD can incorrectly specify the predefined distribution from which masks will be obtained for feature sampling. Moreover, the design process of MCD and MMCD can be more time-consuming and costly than that of ensemble because the predefined distribution needs to be fine-tuned for a given architecture and dataset. To address the limitations of MMCD, the developed MCA replaces feature sampling operations (inherent in MMCD) with feature averaging operations. Particularly, the developed MCA evaluates the uncertainty associated with the features extracted from one ensemble member by sequentially averaging them in a pair-wise manner with the features extracted from other ensemble members. This is based on the rationale that the features extracted from different ensemble members are different and therefore represent a source of noisy information used for feature perturbation. Feature sampling (inherent in MMCD) and feature averaging (inherent in MCA) help to better capture the variability between ensemble members and to therefore improve the predictive uncertainty of the underlying ensemble.

Empirically, baseline has the smallest inference time compared to other methods. This is not surprising because baseline evaluates a single CNN and performs a single forward pass, while other methods evaluate multiple CNNs or perform multiple forward passes. Therefore, MCD, ensemble, MMCD, and MCA improve the baseline uncertainty at the cost of increasing the inference time. Notwithstanding this, whether the inference time of ensemble is larger than that of MCD depends on the architecture and/or capacity of CNNs. Although the inference time of MCA is significantly larger than those of baseline, MCD, and ensemble, it is four times smaller than that of MMCD.

In addition, the empirical results show that while MCD can preserve or decrease the accuracy of the underlying baseline depending on the capacity of the underlying CNN and/or the dataset, ensemble can preserve or increase the accuracy. This finding is supported by previous studies [110, 111], which demonstrated that an ensemble can increase the accuracy. Section 3.2.2 discusses the possible reasons explaining why an ensemble can yield enhanced accuracy. However, while MMCD can preserve or decrease the accuracy of the underlying ensemble depending on the capacity of the underlying CNNs and/or the dataset, MCA can only preserve it. We argue that the decreased baseline and ensemble accuracy caused by MCD and MMCD, respectively, resulted from the incorrect specification of the predefined distribution from which masks for feature sampling (inherent in MCD and MMCD) are obtained. We also argue that MCA preserves the accuracy of the underlying ensemble for all datasets and CNNs of various capacities because feature averaging (inherent in MCA) preserves the similarity between perturbed and original features (Section 4.4).

In addition, the empirical results show that whether an ensemble is better calibrated than baseline depends on the capacity of the underlying CNNs and/or the dataset. Notwithstanding this, an ensemble is better calibrated than MCD, which is in turn (often) better calibrated than MMCD and MCA. This is because MCD, MMCD, and MCA reduce the degree of confidence for TPs and the larger the decrease, the larger the calibration error. Besides, an ensemble can reduce the degree of confidence for FPs but increase or preserve the degree of confidence for TPs. On the contrary, MCD, MMCD, and MCA can reduce the degree of confidence for FPs at the cost of reducing the degree of confidence for TPs. For MCD and MMCD, the level of decrease in the degree of confidence for TPs depends on the dataset and the capacity of the underlying CNNs. However, whether MMCD is better calibrated than MCA depends on the capacity of the underlying CNNs and/or the dataset.

Moreover, the baseline (often) maintain a high degree of confidence for both TPs and

FPs. This means that baseline cannot separate TPs and FPs better than other methods. However, whether ensemble reduces the degree of confidence for FPs better than MCD depends on the dataset, capacity of the underlying CNNs, and/or type of FPs. This result implies that we cannot claim that ensemble can separate TPs and FPs better than MCD and vice versa. In other words, we cannot claim that ensemble captures uncertainty better than MCD. This contradicts with the previous studies [78, 83, 79], which claim that ensemble captures uncertainty better than MCD. Notwithstanding this, MMCD and MCA can maintain a low degree of confidence for FPs better than baseline, MCD, and ensemble. This means that both MMCD and MCA can separate TPs and FPs better than other methods. This is because MMCD and MCA evaluate not only multiple features extracted by different members similar to ensemble but also the uncertainty associated with the extracted features owing to feature sampling (inherent in MMCD) and feature averaging (inherent in MCA). In addition, MCA can maintain a low confidence for FPs similar to or sometimes even better than MMCD. This means that MCA can separate TPs and FPs similar to or sometimes even better than MMCD. This is not surprising because MCA and MMCD have the same purpose and rationale.

The main drawback of MMCD and MCA is that they do not only reduce the degree of confidence for FPs but also for TPs. To ameliorate the confidence drop for TPs, we applied logit instead of probability averaging in MCA and related methods (as discussed in our research paper [102]) to reduce the level of inductive biases (inherent in ensemble members), which are responsible for the confidence drop for TPs. The results show that logit averaging preserved the accuracy but increased the degree of confidence for both TPs and FPs. This is because logit averaging preserves the position of the max element of individual logit vectors but is more sensitive to the magnitude of logit values than probability averaging. Therefore, the magnitude of logit values induces a nonuniform weighting for logit averaging, which is not the case for probability averaging. In addition, the increased degree of confidence caused by logit averaging can reduce the calibration error. This is because the increased degree of confidence can reduce the gap between the average confidence and the classification accuracy. However, it can negatively affect the separability of TPs and FPs, especially FPs owing to noisy data. This is because noisy data can increase the magnitude of logits and logit averaging is more sensitive to changes in the magnitude of logits than probability averaging.

As an alternative approach to ameliorate the confidence drop for TPs, we reduced the strength of regularization applied at training (as discussed in our paper [103]). The results of the empirical comparison of MCA and related methods for weak regularization (WR) and strong regularization (SR) show that the decreased strength of regularization can increase or decrease the classification accuracy depending on the dataset and/or architecture. This is not surprising because the optimal regularization strength depends on the dataset and/or architecture. In addition, some datasets (e.g., GTSRB) have more noise inherent in their samples than others (e.g., MNIST) and samples of some datasets (e.g., CIFAR10) are more difficult to learn than samples of others (e.g., MNIST). Moreover, some architectures (e.g., DenseNet and ResNet) introduce a certain level of implicit regularization via the network depth or width. Therefore, the increased strength of regularization may not always improve the classification accuracy as is claimed by Ioffe and Szegedy [12] and Hinz, Barros, and Wermter [186]. In addition, the decreased strength of regularization decreases the calibration error of MCA and related methods. This is because the decreased strength of regularization increases the degree of confidence. The increase in the degree of confidence reduces the gap between the classification accuracy and average confidence and therefore improves the calibration error. Moreover, the decreased strength of regularization can increase the degree of confidence for both TPs and FPs, and the level of increase in the degree of confidence depends on the dataset, architecture, model type, and/or FP type. This is because the decreased strength of regularization can increase the values of logits, and the increased values of logits can increase the degree of confidence for both TPs and FPs depending on the dataset, architecture, model type, and/or FP type. This implies that the decreased strength of regularization can negatively affect the separability between TPs and FPs depending on the dataset, architecture, model type, and/or FP type because a WR can maintain a high degree of confidence for both TPs and FPs.

9. Summary

We developed Monte Carlo averaging (MCA) to improve the quality of the predictive confidence measuring uncertainty for improving the separability of true predictions (TPs) and false predictions (FPs). The developed MCA enforces CNN-based classifiers to generate FPs with low confidence (e.g., [0, 0.5]) and TPs with high confidence (e.g.,]0.5, 1]). We theoretically and empirically compared the developed MCA with related methods, such as baseline (single CNN), Monte Carlo dropout (MCD), ensemble, and mixture of Monte Carlo dropout (MMCD). We also studied the circumstances under which the developed and related methods perform and analyzed the cost for maintaining a low confidence for FPs and high confidence for TPs. The results show that:

(1) While MCD can preserve or decrease baseline accuracy depending on the capacity of the underlying CNN and/or dataset, ensemble can preserve or increase baseline accuracy. While MMCD can preserve or decrease ensemble accuracy depending on the capacity of the underlying CNNs and/or dataset, MCA can only preserve ensemble accuracy.

(2) Whether ensemble is better calibrated than baseline depends on the capacity of the underlying CNNs and/or the dataset. However, ensemble is better calibrated than MCD, which is in turn (often) better calibrated than MMCD and MCA. Moreover, whether MCA is better calibrated than MMCD depends on the capacity of the underlying CNNs and/or the dataset.

(3) Baseline cannot separate TPs and FPs better than other methods. Whether ensemble can separate TPs and FPs better than MCD depends on the dataset, capacity of the underlying CNNs, and/or the type of FPs. However, MMCD and MCA can separate TPs and FPs better than other methods. Besides, MCA can separate TPs and FPs similar to or sometimes even better than MMCD.

(4) The inference time of MCA is significantly larger than that of baseline, MCD, and ensemble but it is four times smaller than that of MMCD.

Overall, the results imply the dominance of MMCD and MCA over previous methods, such as baseline, MCD, and ensemble. Specifically, MMCD and MCA can improve the separability of TPs and FPs at the cost of increasing the calibration error and the inference time. The increased calibration error is based on the rationale that MMCD and MCA do not only reduce the degree of confidence for FPs but also for TPs. To ameliorate the confidence drop for TPs and therefore improve the calibration error, we applied logit instead of probability averaging in MCA and related methods. The results show that logit averaging:

(5) preserves the classification accuracy of MCA and related methods, but increases the degree of confidence for both TPs and FPs

(6) reduces the calibration error of MCA and related methods

(7) negatively affect the separability of TPs and FPs.

As an alternative approach to ameliorate the confidence drop for TPs and therefore reduce the calibration error, we reduced the strength of regularization applied at training. The empirical results of the comparison of MCA and related methods for weak regularization (WR) and strong regularization (SR) show that the decrease in the strength of regularization:

(8) increases or decreases the classification accuracy of MCA and related methods depending on the dataset and/or architecture

(9) reduces the calibration error of MCA and related methods

(10) affects the separability between TPs and FPs depending on the dataset, architecture, model, and/or FP type.

Since applying logit instead of probability averaging in MCA and related methods or reducing the strength of regularization can ameliorate the confidence drop for TPs and therefore reduce the calibration error at the cost of harming the separability between TPs and FPs. We argue that reducing the calibration error on test data and improving the ability to separate TPs and FPs are two contradicting goals. Improving one may be detrimental to the other. In addition, given two models A and B, if A is better calibrated than B, then A does not necessarily separate TPs and FPs better than B. This means that existing methods for confidence calibration may not help to improve the separability between TPs and FPs. To verify this hypothesis, it is recommended that future works investigate the ability of existing methods for confidence calibration to maintain a low degree of confidence for FPs. We also recommend researchers evaluate not only the calibration error of a proposed method for confidence calibration, but also the ability of the proposed method to maintain a low degree of confidence for FPs. Notwithstanding this, for mission- and/or safety-critical applications where the separability of TPs and FPs is critical, we suggest to apply probability averaging (as it is traditionally done) to avoid the negative impact of logit averaging on the separability of TPs and FPs. We also suggest applying MCA and related methods with CNNs trained using SR (as it is traditionally done) to avoid the negative impact of WR on the separability of TPs and FPs.

9.1. Conclusion

Based on the results presented so far, we conclude that the developed MCA is an alternative to MMCD. Assuming that both MMCD and MCA are well-specified for a given dataset and architecture, they can have similar performance because they have the same purpose and rationale. Specifically, both approaches evaluate multiple features extracted from multiple ensemble members and evaluate the uncertainty associated with the extracted features thanks to feature averaging (inherent in MCA) and feature sampling (inherent in MMCD). However, the design process of MMCD can be more time-consuming and costly than that of MCA. This is because MMCD requires the specification of a prior distribution (which is sensitive to the dataset and architecture) from which masks will be obtained for feature sampling, while MCA relies on features extracted from ensemble members for feature averaging. Moreover, the outcomes of feature averaging and sampling operations are deterministic and stochastic, respectively. Therefore, feature sampling required more samples than feature averaging to evaluate the uncertainty associated with the extracted features. Particularly, empirical results show that the inference time of MMCD is four times larger than that of MCA. In summary, the developed MCA

- is four times faster than,
- has the same purpose and rationale as, and
- performs similar to or sometimes even better than the state-of-the-art MMCD.

Owing to all the advantages of MCA over MMCD, we preferred MCA instead of MMCD for applications (such as collision prediction [30], door recognition for visual-based robot navigation [32], and pedestrian detection [33]) where the separability of TPs and FPs is critical and where the computational budget is limited. MCA can enable CNN-based classifiers to explicitly decide to ignore uncertain predictions or pass them to human experts [60]. In addition, MCA can benefit other fields (such as active learning [187, 188, 150], online learning [78], and reinforcement learning [60, 31, 58]) where uncertainty is required. Moreover, MCA can benefit the field of explainable artificial intelligence, which explores the role of uncertainty to explain predictions and increase the social acceptance of CNN-based decision-making systems [189, 190]. Finally, MCA opens new perspectives to fuse (or average) features of ensemble members, aiming at representing more sophisticated forms of the prior distribution. Besides, feature averaging is a specific type of feature augmentation, which can be used for regularizing an ensemble of neural networks trained under a unified loss.

9.2. Limitations

One of the limitations of MCA is that it can improve the separability of TPs and FPs at the cost of increasing the calibration error on the test data. A fundamental open question now is: *How much can we compromise calibration to achieve better separation between TPs and FPs?* To address this question, one should first understand that the ECE is a measure to evaluate in-domain uncertainty. Specifically, a small value of ECE indicates that MCA or related methods can separate TPs and FPs due to incorrect classifications of in-domain examples. This is because a small value of ECE indicates that MCA or related methods can assign a high confidence for TPs and low confidence for FPs. However, a large value of ECE does not necessarily mean that MCA or related methods cannot separate TPs and FPs. For instance, the ECE is huge for MCA and MMCD because these two methods reduce the confidence for both TPs and FPs, not because they cannot maintain a high confidence for TPs and a low confidence for FPs. Overall, the ECE and therefore the calibration error can be huge because MCA or related methods (1) can reduce the confidence for TPs (as in this thesis) or (2) cannot separate TPs and FPs. A huge ECE is acceptable for the first reason. However, for the second reason, a huge ECE is not acceptable in applications where the separability of TPs and FPs is critical.

Another limitation of MCA is that it relies on multiple members similar to an ensemble and MMCD. A large number of members may require a large amount of storage memory. For instance, each MCA investigated in this thesis is based on an ensemble of 20 CNNs. The 20 CNNs require approximately 59 megabytes of storage memory, which can inhibit the use of MCA in applications with limited storage memory. Therefore, it is recommended that future research explore pruning methods reviewed by Tsoumakas, Partalas, and Vlahavas [191] to reduce the number of ensemble members to three or five and to therefore reduce the storage memory requirement.

Finally, in this thesis, all empirical results were obtained using benchmarking datasets, which are possibly different from real-world application datasets. Although the developed MCA can perform similarly well and sometimes even better than the state-of-the-art MMCD on benchmarking datasets, the applicability of the developed MCA on real-world application datasets remains uncertain. Therefore, it is recommended that future works evaluate the developed MCA on real-world applications, such as collision prediction [30], door recognition for visual-based robot navigation [32], and pedestrian detection [33].

A. Experimental setup

Our experimental setup includes experimental datasets, experimental architectures, training details, evaluation metrics, and evaluation data.

Table 11.: Summary of experimental datasets

Datasets	**Image type**	**Image size [Pixels]**	**Training data size**	**Test data size**	**No. of classes**	**No. of images per class**
MNIST	Grayscale	32×32	60000	10000	10	Balanced
Fashion-MNIST	Grayscale	32×32	60000	10000	10	Balanced
GTSRB	Grayscale	32×32	39209	12630	43	Unbalanced
CIFAR10	Color	$32 \times 32 \times 3$	50000	10000	10	Balanced

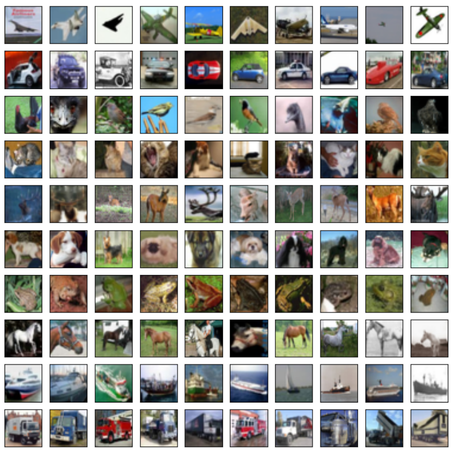
A.1. Experimental datasets

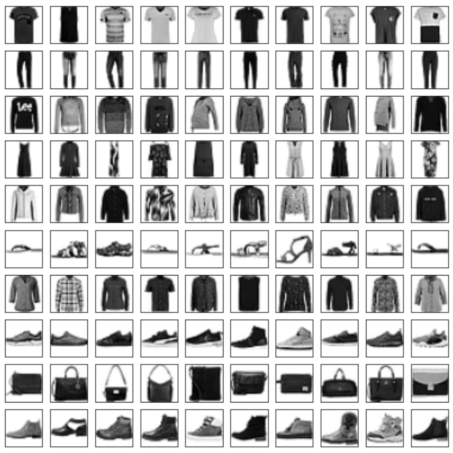
To evaluate the ability of MCA and related methods to perform on tasks (datasets) with various difficulties, we conducted experiments on four benchmarking datasets: MNIST [125], Fashion-MNIST [192], CIFAR10 [193], and GTSRB [194]. Figure 29 presents some image examples of these datasets.

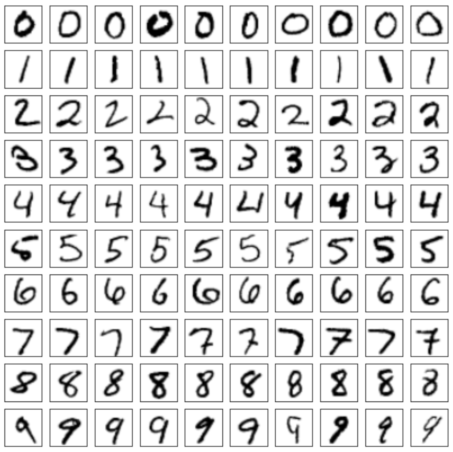
MNIST contains grayscaled images of ten categories of digits ranging from 0 to 9. The digits are centered inside images of 28x28 pixels. The number of images per digit is balanced. MNIST includes 60,000 and 10,000 images for training and testing, respectively.

Fashion-MNIST contains grayscaled images of ten categories of Zalando's articles. The articles are inside images of 28x28 pixels. The number of images per articles is balanced. Similar to MNIST, Fashion-MNIST includes 60,000 and 10,000 images for training and testing, respectively.

CIFAR10 includes colored images of ten categories of common objects. The objects appear in various poses and are not always centered inside the images of 32x32


(a) CIFAR10


(b) Fashion-MNIST


(c) MNIST

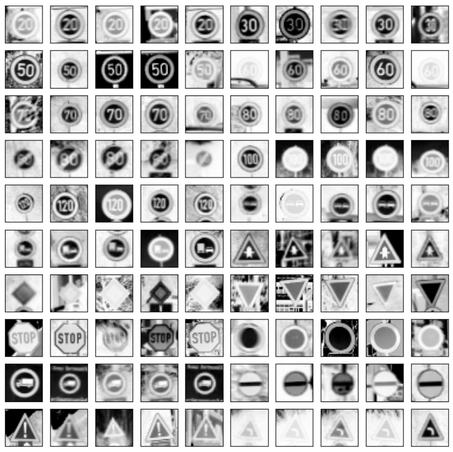

(d) GTSRB

Figure 29.: Examples of images of the experimental datasets

pixels. The number of images per category is uniformly distributed. CIFAR10 includes 50,000 and 10,000 images for training and testing, respectively.

GTSRB includes colored images of 43 categories of common German traffic signs. The traffic signs have different poses and are not always centered inside the images. The images have sizes varying from 15x15 to 222x192 pixels. The number of images per traffic sign is not uniformly distributed. GTSRB includes 39,209 and 12,630 images for training and testing, respectively.

Table 12.: Capacity or number of parameters (in millions) of experimental architectures

	Small capacity	**Large capacity**	**Increase factor**
CIFAR10 (DenseNets)	5.02	21.36	4.25
Fashion-MNIST (ResNets)	7.15	31.27	4.37
MNIST (VGGNets)	6.15	26.86	4.37
GTSRB (ResNets)	31.40	7.22	4.35

Table 13.: Summary of CNN architectures with small capacities: $\left[conv3 \times 3 - 64\right]$ denotes a convolution operation with 64 convolution filters of size 3×3. $\left[\cdot\right] \times 3$ denotes 3 consecutive operations of $\left[\cdot\right]$. $max2 \times 2$ denotes a max pooling operation over a 2×2 pixel window, with stride 2. $FC - 2048$ denotes 2048 fully connected neurons. The GlobalAveragePooling2D [195] operation is used to reduce the spatial dimension of inputs to fully connected layers

Layers	**VGGNet**	**ResNet**	**DenseNet**
Input			
Convolution	$\left[conv3 \times 3 - 64\right] \times 3$	$\begin{bmatrix} conv1 \times 1 - 64 \\ conv3 \times 3 - 64 \\ conv1 \times 1 - 64 \end{bmatrix} \times 3$	$\begin{bmatrix} conv1 \times 1 \\ conv3 \times 3 \end{bmatrix} \times 6$
Pooling	$max2 \times 2$	$max2 \times 2$	$\begin{bmatrix} conv1 \times 1 \\ max2 \times 2 \end{bmatrix}$
Convolution	$\left[conv3 \times 3 - 256\right] \times 3$	$\begin{bmatrix} conv1 \times 1 - 256 \\ conv3 \times 3 - 256 \\ conv1 \times 1 - 256 \end{bmatrix} \times 3$	$\begin{bmatrix} conv1 \times 1 \\ conv3 \times 3 \end{bmatrix} \times 6$
Pooling	$max2 \times 2$	$max2 \times 2$	$\begin{bmatrix} conv1 \times 1 \\ max2 \times 2 \end{bmatrix}$
GlobalAveragePooling2D			
$FC - 2048$			
$FC - 2048$			
$FC - K$			

A.2. Experimental architectures

To evaluate the ability of MCA and related methods to perform on different architectures, we conducted experiments on three popular architectures: VGGNet [110], ResNet [111], and DenseNet [114]. Tables 13 and 14 present these architectures for networks with small and large capacities, respectively. Table 12 presents the capacities of these networks for the experimental datasets.

VGGNet is an architecture with convolution, pooling, and fully connected layers. Each convolution layer performs three consecutive convolution operations with convolution filter size of 3×3. In addition, each convolution layer includes many convolution filters: 64, 128, 256, or 512. Convolution neurons are equipped with RELU activation functions. Convolution layers are followed by max pooling layers. Max pooling is performed over a 2×2 pixel window, with stride 2. The last max pooling layer is followed by three fully connected layers. The first two fully connected layers include 2048 or 4096 neurons equipped with RELU activation functions, and the third fully connected layer includes K neurons equipped with softmax activation functions.

ResNet is an architecture similar to the VGGNet. However, its convolution layers replace some 3×3 convolution filters with 1×1 ones and include residual (or identity shortcut) connections used to alleviate the vanishing-gradient problem. Convolution layers combine features of identity connections with their outputs through summation before they are passed on to the subsequent (max pooling) layers.

DenseNet is an architecture similar to the ResNet, but with densely connected convolution layers. Specifically, each convolution layer obtains additional inputs from all preceding convolution layers. DenseNet alleviates the vanishing-gradient problem with dense connections instead of residual ones similar to ResNet. While ResNet combines features through summation, DenseNet combines features by concatenating them, which increases the number of channels of inputs to max pooling layers. To reduce the number of channels of inputs to max pooling layers, DenseNet performs 1×1 convolution operations at inputs to max pooling layers. Herein, we set the grow rate conditioning the number of feature maps per convolution layers to 24.

A.3. Training details

All experiments were coded in TensorFlow. All CNNs were trained using the categorical cross-entropy, SGD, and hyperparameters, as shown in Table 15. All images were standardized and normalized by dividing pixel values by 255. Moreover, all CNNs were regularized using dropout layers (using a cascade of Gaussian and Bernoulli distributions) placed at inputs to max pooling layers and batch normalization [12] layers placed after max pooling layers. In addition, dropout layers were placed at inputs to fully connected layers. Further, we regularized using standard data augmentation techniques such as rotation, (vertical and horizontal) translations, scaling, shearing, and additive Gaussian noise (AGN).

A.4. Inference details

At inference, we applied feature sampling inherent in MCD/MMCD and feature averaging inherent in MCA at inputs to the first fully connected layer. MCD samples features using masks obtained from a cascade of Bernoulli and Gaussian distribution [70] and using a dropout probability of 0.5. MCD performs 100 stochastic forward passes ($T = 100$) and therefore samples 100 features. Ensemble, MMCD and MCA include 20 deterministic CNNs ($M = 20$).

A.5. Evaluation metrics

We expect a good method for estimating uncertainty to preserve the classification accuracy, while giving a good estimate of uncertainty. Therefore, we compared MCA and related methods with respect to the classification accuracy and quality of uncertainty. If not stated otherwise, N denotes the evaluation data size.

A.5.1. Evaluating the classification accuracy

The classification accuracy measures how well a trained CNN generalizes to evaluation (e.g., test) data. We expect the classification accuracy to be high on evaluation data obtained from the training data distribution. In contrast, we expect the classification accuracy to be low on evaluation data obtained far from the training data distribution (e.g., OOD examples). Given an evaluation data, the classification accuracy (CA) is estimated as

$$CA = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}(\hat{y}_i = y_i) \tag{A.1}$$

where $\mathbb{1}(\hat{y}_i = y_i)$ is 1 if the condition holds or 0 otherwise.

A.5.2. Evaluating the quality of uncertainty

We evaluated the quality of uncertainty by assessing the degree of confidence calibration and by assessing the ability to separate TPs and FPs based on evaluation of the degree of confidence.

Assessing the degree of confidence calibration

We assessed the degree of confidence calibration by evaluating the calibration error, which is the difference between the predicted probabilities (evaluated based on the average confidence) and true probabilities (evaluated based on the CA). We measured the calibration error using the ECE utilized by Naeini et al. [196]; Guo et al. [16]; Laves et al. [197]; Zhang, Dalca, and Sabuncu [69]; Thulasidasan et al. [198]; Liang et al. [199]; Wen et al. [82]; and Wu Gales [168]. Low and high values of ECE indicate low and high calibration errors, respectively. The ECE is denoted as in Equation A.4. It sorts and groups predictions of an evaluation data into B equally spaced bins and weighted the difference between the average confidence and CA of bins. The bin b_τ denotes the set of indices of evaluation samples whose confidences fall into the interval $I_\tau =]\frac{\tau-1}{B}, \frac{\tau}{B}]$. The expected accuracy $acc(b_\tau)$ of bin b_τ is denoted as in Equation A.3. The expected confidence $conf(b_\tau)$ within bin b_τ is denoted as in Equation A.2. Confidences are well-calibrated when $acc(b_\tau) = conf(b_\tau)\ \forall\ \tau \in [1, B]$.

$$conf(b_\tau) = \frac{1}{|b_\tau|} \sum_{i \in b_\tau} u_i \tag{A.2}$$

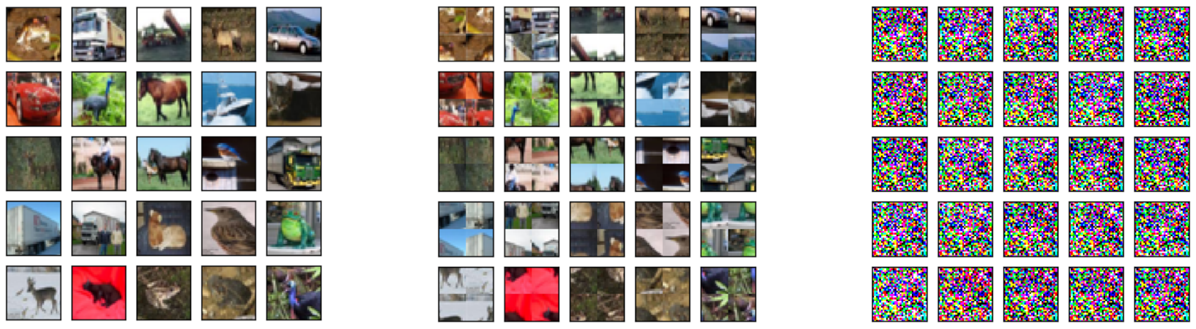
$$acc(b_\tau) = \frac{1}{|b_\tau|} \sum_{i \in b_\tau} \mathbb{1}(\hat{y}_i = y_i) \tag{A.3}$$

$$ECE = \sum_{\tau=1}^{B} \frac{|b_\tau|}{N} |conf(b_\tau) - acc(b_\tau)| \tag{A.4}$$

Assessing the ability to separate TPs and FPs

We assessed the ability to separate TPs and FPs by evaluating the average confidence. While the degree of confidence can be used as a measure of uncertainty, the average confidence can be used to evaluate the ability of CNN-based classifiers to separate TPs and FPs. Specifically, we expect the average confidence to be high on evaluation data generating TPs and low on evaluation data generating FPs. Therefore, the average confidence (AC) is estimated as

$$AC = \frac{1}{N} \sum_{i=1}^{N} u_i\ . \tag{A.5}$$



(a) Test data (b) Swap data (c) Noisy data

Figure 30.: Examples of evaluation data for experiments conducted on CIFAR10

A.6. Evaluation data

We used five evaluation data for different purposes: *test*, *subsets of correctly classified test*, *OOD*, *swap*, and *noisy* data. While test data are used for evaluating the CA and ECE, subsets of correctly classified test, OOD, swap, and noisy data are used for evaluating the ability to separate TPs and FPs.

Test data are experimental data, such as the MNIST, CIFAR10, Fashion-MNIST, and GTSRB that include both correctly classified and misclassified test data. Test data are used to evaluate the CA and ECE. We expect the CA to be high and the ECE to be low on test data.

Subsets of correctly classified test data include 1000 correctly classified test data of experimental data. Since CNNs will generate (only) TPs on these data, we used these data to evaluate the average confidence for TPs.

Swap data are simulated using subsets of correctly classified test data structurally perturbed by dividing images into four regions and diagonally permuting the regions. As shown in Figure 30b, the upper left is permuted with the bottom right and the upper right is permuted with the bottom left region. Swap data include structurally perturbed objects within the given images. Since the structure of objects within swap images is destroyed, we expect CNNs to generate (only) FPs on swap data. Therefore, we used these data to evaluate the average confidence for FPs due to structural perturbation.

Noisy data are simulated using subsets of correctly classified test data perturbed by applying AGN of standard deviation of 500. As shown in Figure 30c, noisy data include noise within the given images. Since CNNs will generate (only) FPs on noisy data, we used these data to evaluate the average confidence for FPs due to noise.

OOD data are simulated using 1000 test data of CIFAR100. Since CNNs will generate (only) FPs on these data, we used these data to evaluate the average confidence for FPs due to domain shift.

In general, we expect the average confidence to be high (e.g., $[50\%, 100\%]$) for TPs and low (e.g., $[0\%, 50\%[$) for FPs.

Table 14.: Summary of CNN architectures with large capacities: $\begin{bmatrix} conv3 \times 3 - 64 \end{bmatrix}$ denotes a convolution operation with 64 convolution filters of size 3×3. $\begin{bmatrix} \cdot \end{bmatrix} \times 3$ denotes 3 consecutive operations of $\begin{bmatrix} \cdot \end{bmatrix}$. $max2 \times 2$ denotes a max pooling operation over a 2×2 pixel window, with stride 2. $FC - 4096$ denotes 4096 fully connected neurons

Layers	**VGGNet**	**ResNet**	**DenseNet**
Input			
Convolution	$\begin{bmatrix} conv3 \times 3 - 64 \end{bmatrix} \times 3$	$\begin{bmatrix} conv1 \times 1 - 64 \\ conv3 \times 3 - 64 \\ conv1 \times 1 - 64 \end{bmatrix} \times 3$	$\begin{bmatrix} conv1 \times 1 \\ conv3 \times 3 \end{bmatrix} \times 6$
Pooling	$max2 \times 2$	$max2 \times 2$	$\begin{bmatrix} conv1 \times 1 \\ max2 \times 2 \end{bmatrix}$
Convolution	$\begin{bmatrix} conv3 \times 3 - 64 \end{bmatrix} \times 3$	$\begin{bmatrix} conv1 \times 1 - 64 \\ conv3 \times 3 - 64 \\ conv1 \times 1 - 64 \end{bmatrix} \times 3$	$\begin{bmatrix} conv1 \times 1 \\ conv3 \times 3 \end{bmatrix} \times 6$
Pooling	$max2 \times 2$	$max2 \times 2$	$\begin{bmatrix} conv1 \times 1 \\ max2 \times 2 \end{bmatrix}$
Convolution	$\begin{bmatrix} conv3 \times 3 - 128 \end{bmatrix} \times 3$	$\begin{bmatrix} conv1 \times 1 - 128 \\ conv3 \times 3 - 128 \\ conv1 \times 1 - 128 \end{bmatrix} \times 3$	$\begin{bmatrix} conv1 \times 1 \\ conv3 \times 3 \end{bmatrix} \times 6$
Pooling	$max2 \times 2$	$max2 \times 2$	$\begin{bmatrix} conv1 \times 1 \\ max2 \times 2 \end{bmatrix}$
Convolution	$\begin{bmatrix} conv3 \times 3 - 256 \end{bmatrix} \times 3$	$\begin{bmatrix} conv1 \times 1 - 256 \\ conv3 \times 3 - 256 \\ conv1 \times 1 - 256 \end{bmatrix} \times 3$	$\begin{bmatrix} conv1 \times 1 \\ conv3 \times 3 \end{bmatrix} \times 6$
Pooling	$max2 \times 2$	$max2 \times 2$	$\begin{bmatrix} conv1 \times 1 \\ max2 \times 2 \end{bmatrix}$
Convolution	$\begin{bmatrix} conv3 \times 3 - 512 \end{bmatrix} \times 3$	$\begin{bmatrix} conv1 \times 1 - 512 \\ conv3 \times 3 - 512 \\ conv1 \times 1 - 512 \end{bmatrix} \times 3$	$\begin{bmatrix} conv1 \times 1 \\ conv3 \times 3 \end{bmatrix} \times 6$
Pooling	$max2 \times 2$	$max2 \times 2$	$\begin{bmatrix} conv1 \times 1 \\ max2 \times 2 \end{bmatrix}$
$FC - 4096$			
$FC - 4096$			
$FC - K$			

Table 15.: Summary of values assigned to SGD hyperparameters

Hyperparameters	**Values**
Learning rate (γ)	0.02
Batch size (β)	128
Momentum (ν)	0.9
Epoch	100

List of Figures

List of Tables

Acronyms

AC average confidence. 58, 59, 61, 71, 74, 77, 80, 96, 104, 105

ACS average cosine similarity. 55, 56, 103

AGN additive Gaussian noise. 95, 97

ARMSE average root mean square error. 55, 56, 103

ASD average standard deviation. 54, 103

CA classification accuracy. 58, 59, 71, 77, 95–97, 105

CNN convolutional neural network. iv–vii, xiii, 1–4, 6, 7, 9–15, 17–19, 22–27, 36, 41, 44–48, 50, 53, 54, 56–65, 67–72, 74, 75, 77, 79–85, 87, 89, 90, 93–99, 101–106

CS cosine similarity. 50, 52, 55, 103

ECE expected calibration error. 58–60, 70, 71, 73, 76, 77, 79, 90, 96, 97, 105

FP false prediction. iv–vii, xiv, xv, 2–4, 6, 7, 9, 12, 13, 15, 16, 24–27, 44, 54, 56, 58, 60–62, 64–67, 70–75, 77–90, 96–98, 101–105

LA logit averaging. 71, 74, 104, 105

MCA Monte Carlo averaging. iv–vii, xiv, 11–16, 27, 29, 44, 49, 50, 53–67, 70–72, 75–79, 82–91, 94, 95, 103, 105

MCD Monte Carlo dropout. iv–vii, x, 11–15, 33–35, 39, 40, 44–48, 50, 56–65, 70, 71, 77, 78, 82–85, 87, 95, 102, 105

MMCD mixture of Monte Carlo dropout. iv–vii, x, 11–15, 44, 47–50, 54–67, 70, 71, 77, 78, 82–85, 87, 89, 90, 95, 102, 103, 105

OOD out-of-domain. 3, 7, 9, 25, 26, 40, 54, 61, 62, 72, 74, 78, 80–82, 95, 97, 98, 103, 104

PA probability averaging. 71, 74, 104, 105

RELU rectified linear unit. 19, 22, 94

RMSE root mean square error. 50, 52, 55, 103

SGD stochastic gradient descent. 9, 23, 94, 100, 106

SR strong regularization. 50, 52, 56, 59, 61, 63, 71, 72, 74–80, 85, 88, 89, 103–105

TP true prediction. iv–vii, xiv, xv, 7, 9, 12, 13, 15, 16, 44, 54, 56, 58, 60–62, 64–67, 70–75, 77–90, 96–98, 103–105

WR weak regularization. 54, 75–80, 85, 86, 88, 89, 103–105

References

[1] Mohammad Mahdi Bejani and Mehdi Ghatee. “A systematic review on overfitting control in shallow and deep neural networks”. In: *Artificial Intelligence Review* 54.8 (2021), pp. 6391–6438.

[2] Lutz Prechelt. “Early stopping-but when?” In: *Neural Networks: Tricks of the Trade*. Springer, 1998, pp. 55–69.

[3] Yuan Yao, Lorenzo Rosasco, and Andrea Caponnetto. “On early stopping in gradient descent learning”. In: *Constructive Approximation* 26.2 (2007), pp. 289–315.

[4] Anders Krogh and John Hertz. “A simple weight decay can improve generalization”. In: *Advances in Neural Information Processing Systems* 4 (1991).

[5] Christos Louizos, Max Welling, and Diederik P Kingma. “Learning sparse neural networks through L_0 regularization”. In: *arXiv preprint arXiv:1712.01312* (2017).

[6] Connor Shorten and Taghi M Khoshgoftaar. “A survey on image data augmentation for deep learning”. In: *Journal of Big Data* 6.1 (2019), pp. 1–48.

[7] Cherry Khosla and Baljit Singh Saini. “Enhancing performance of deep learning models with different data augmentation techniques: A survey”. In: *2020 International Conference on Intelligent Engineering and Management (ICIEM)*. IEEE. 2020, pp. 79–85.

[8] Humza Naveed. “Survey: Image mixing and deleting for data augmentation”. In: *arXiv preprint arXiv:2106.07085* (2021).

[9] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. “Dropout: A simple way to prevent neural networks from overfitting”. In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[10] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. “Regularization of neural networks using dropconnect”. In: *International Conference on Machine Learning*. PMLR. 2013, pp. 1058–1066.

[11] Mostafa Rahmani and George K Atia. "Data dropout in arbitrary basis for deep network regularization". In: *2018 52nd Asilomar Conference on Signals, Systems, and Computers*. IEEE. 2018, pp. 66–70.

[12] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International Conference on Machine Learning*. PMLR. 2015, pp. 448–456.

[13] Nils Bjorck, Carla P Gomes, Bart Selman, and Kilian Q Weinberger. "Understanding batch normalization". In: *Advances in Neural Information Processing Systems* 31 (2018).

[14] Ismoilov Nusrat and Sung-Bong Jang. "A comparison of regularization techniques in deep neural networks". In: *Symmetry* 10.11 (2018), p. 648.

[15] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. "Simple and scalable predictive uncertainty estimation using deep ensembles". In: *arXiv preprint arXiv:1612.01474* (2016).

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

[17] Vladimir Vapnik. "Principles of risk minimization for learning theory". In: *Advances in Neural Information Processing Systems*. 1992, pp. 831–838.

[18] Dan Hendrycks and Thomas Dietterich. "Benchmarking neural network robustness to common corruptions and perturbations". In: *arXiv preprint arXiv:1903.12261* (2019).

[19] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and harnessing adversarial examples". In: *arXiv preprint arXiv:1412.6572* (2014).

[20] Anh Nguyen, Jason Yosinski, and Jeff Clune. "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 427–436.

[21] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. "Adversarial machine learning at scale". In: *arXiv preprint arXiv:1611.01236* (2016).

[22] Cihang Xie, Mingxing Tan, Boqing Gong, Alan Yuille, and Quoc V Le. "Smooth adversarial training". In: *arXiv preprint arXiv:2006.14536* (2020).

[23] Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya Nori, and Antonio Criminisi. "Measuring neural net robustness with constraints". In: *Advances in Neural Information Processing Systems* 29 (2016).

[24] David Krueger, Chin-Wei Huang, Riashat Islam, Ryan Turner, Alexandre Lacoste, and Aaron Courville. “Bayesian hypernetworks”. In: *arXiv preprint arXiv:1710.04759* (2017).

[25] Lewis Smith and Yarin Gal. “Understanding measures of uncertainty for adversarial example detection”. In: *arXiv preprint arXiv:1803.08533* (2018).

[26] Omer Faruk Tuna, Ferhat Ozgur Catak, and M Taner Eskil. “Closeness and uncertainty aware adversarial examples detection in adversarial machine learning”. In: *arXiv preprint arXiv:2012.06390* (2020).

[27] Dan Hendrycks and Kevin Gimpel. “A baseline for detecting misclassified and out-of-distribution examples in neural networks”. In: *arXiv preprint arXiv:1610.02136* (2016).

[28] Shiyu Liang, Yixuan Li, and R. Srikant. “Enhancing the reliability of out-of-distribution image detection in neural networks”. In: *6th International Conference on Learning Representations*. 2018.

[29] Harry A Pierson and Michael S Gashler. “Deep learning in robotics: A review of recent research”. In: *Advanced Robotics* 31.16 (2017), pp. 821–835.

[30] Gregory Kahn, Adam Villaflor, Vitchyr Pong, Pieter Abbeel, and Sergey Levine. “Uncertainty-aware reinforcement learning for collision avoidance”. In: *arXiv preprint arXiv:1702.01182* (2017).

[31] Björn Lütjens, Michael Everett, and Jonathan P How. “Safe reinforcement learning with model uncertainty estimates”. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 8662–8668.

[32] Wei Chen, Ting Qu, Yimin Zhou, Kaijian Weng, Gang Wang, and Guoqiang Fu. “Door recognition and deep learning algorithm for visual based robot navigation”. In: *2014 IEEE International Conference on Robotics and Biomimetics (robio 2014)*. IEEE. 2014, pp. 1793–1798.

[33] Wanli Ouyang and Xiaogang Wang. “Joint deep learning for pedestrian detection”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2013, pp. 2056–2063.

[34] Ashesh Jain, Hema S Koppula, Shane Soh, Bharad Raghavan, Avi Singh, and Ashutosh Saxena. “Brain4cars: Car that knows before you do via sensory-fusion deep learning architecture”. In: *arXiv preprint arXiv:1601.00740* (2016).

[35] Varun Gulshan, Lily Peng, Marc Coram, Martin C Stumpe, Derek Wu, Arunachalam Narayanaswamy, Subhashini Venugopalan, Kasumi Widner, Tom Madams, Jorge Cuadros, et al. “Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs”. In: *Jama* 316.22 (2016), pp. 2402–2410.

[36] Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebastian Thrun. “Dermatologist-level classification of skin cancer with deep neural networks”. In: *Nature* 542.7639 (2017), pp. 115–118.

[37] Babak Ehteshami Bejnordi, Mitko Veta, Paul Johannes Van Diest, Bram Van Ginneken, Nico Karssemeijer, Geert Litjens, Jeroen AWM Van Der Laak, Meyke Hermsen, Quirine F Manson, Maschenka Balkenhol, et al. “Diagnostic assessment of deep learning algorithms for detection of lymph node metastases in women with breast cancer”. In: *Jama* 318.22 (2017), pp. 2199–2210.

[38] Jian Huang, Junyi Chai, and Stella Cho. “Deep learning in finance and banking: A literature review and classification”. In: *Frontiers of Business Research in China* 14.1 (2020), pp. 1–24.

[39] Ahmet Murat Ozbayoglu, Mehmet Ugur Gudelek, and Omer Berat Sezer. “Deep learning for financial applications: A survey”. In: *Applied Soft Computing* 93 (2020), p. 106384.

[40] Björn Rafn Gunnarsson, Seppe Vanden Broucke, Bart Baesens, Marıa Óskarsdóttir, and Wilfried Lemahieu. “Deep learning for credit scoring: Do or don’t?” In: *European Journal of Operational Research* 295.1 (2021), pp. 292–305.

[41] Svitlana Galeshchuk and Sumitra Mukherjee. “Deep networks for predicting direction of change in foreign exchange rates”. In: *Intelligent Systems in Accounting, Finance and Management* 24.4 (2017), pp. 100–110.

[42] Thomas Fischer and Christopher Krauss. “Deep learning with long short-term memory networks for financial market predictions”. In: *European Journal of Operational Research* 270.2 (2018), pp. 654–669.

[43] Philipp Oberdiek, Matthias Rottmann, and Hanno Gottschalk. “Classification uncertainty of deep neural networks based on gradient information”. In: *IAPR Workshop on Artificial Neural Networks in Pattern Recognition*. Springer. 2018, pp. 113–125.

[44] Andrey Malinin and Mark Gales. “Reverse KL-Divergence Training of Prior Networks: Improved Uncertainty and Adversarial Robustness”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. dÁlché-Buc, E. Fox, and R. Garnett. 2019, pp. 14547–14558.

[45] Murat Sensoy, Lance Kaplan, and Melih Kandemir. “Evidential deep learning to quantify classification uncertainty”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 3179–3189.

[46] Andrey Malinin and Mark Gales. “Predictive uncertainty estimation via prior networks”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 7047–7058.

[47] Marcin Możejko, Mateusz Susik, and Rafał Karczewski. “Inhibited softmax for uncertainty estimation in neural networks”. In: *arXiv preprint arXiv:1810.01861* (2018).

[48] Jay Nandy, Wynne Hsu, and Mong Li Lee. “Towards Maximizing the Representation Gap between In-Domain &amp; Out-of-Distribution Examples”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin. 2020, pp. 9239–9250.

[49] Maithra Raghu, Katy Blumer, Rory Sayres, Ziad Obermeyer, Bobby Kleinberg, Sendhil Mullainathan, and Jon Kleinberg. “Direct uncertainty prediction for medical second opinions”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 5281–5290.

[50] Tiago Ramalho and Miguel Miranda. “Density Estimation in Representation Space to Predict Model Uncertainty”. In: *Engineering Dependable and Secure Machine Learning Systems: Third International Workshop, EDSMLS 2020, New York City, NY, USA, February 7, 2020, Revised Selected Papers*. Vol. 1272. Springer Nature. 2020, p. 84.

[51] Jinsol Lee and Ghassan AlRegib. “Gradients as a measure of uncertainty in neural networks”. In: *2020 IEEE International Conference on Image Processing*. IEEE. 2020, pp. 2416–2420.

[52] Theodoros Tsiligkaridis. “Information Robust Dirichlet Networks for Predictive Uncertainty Estimation”. In: *arXiv preprint arXiv:1910.04819* (2019).

[53] Qingyang Wu, He Li, Weijie Su, Lexin Li, and Zhou Yu. “Quantifying Intrinsic Uncertainty in Classification via Deep Dirichlet Mixture Networks”. In: *arXiv preprint arXiv:1906.04450* (2019).

[54] Joost Van Amersfoort, Lewis Smith, Yee Whye Teh, and Yarin Gal. “Uncertainty Estimation Using a Single Deep Deterministic Neural Network”. In: *Proceedings of the 37th International Conference on Machine Learning*. PMLR, 2020, pp. 9690–9700.

[55] Yen-Chang Hsu, Yilin Shen, Hongxia Jin, and Zsolt Kira. “Generalized odin: Detecting out-of-distribution image without learning from out-of-distribution data”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 10951–10960.

[56] Yarin Gal and Zoubin Ghahramani. “Bayesian convolutional neural networks with Bernoulli approximate variational inference”. In: *arXiv preprint arXiv:1506.02158* (2015).

[57] Danilo Rezende and Shakir Mohamed. “Variational inference with normalizing flows”. In: *International Conference on Machine Learning*. 2015, pp. 1530–1538.

[58] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. “Weight uncertainty in neural networks”. In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning-Volume 37*. 2015, pp. 1613–1622.

[59] Durk P Kingma, Tim Salimans, and Max Welling. “Variational dropout and the local reparameterization trick”. In: *Advances in Neural Information Processing Systems*. 2015, pp. 2575–2583.

[60] Yarin Gal and Zoubin Ghahramani. “Dropout as a bayesian approximation: Representing model uncertainty in deep learning”. In: *International Conference on Machine Learning*. 2016, pp. 1050–1059.

[61] Patrick McClure and Nikolaus Kriegeskorte. “Robustly representing uncertainty through sampling in deep neural networks”. In: *arXiv preprint arXiv:1611.01639* (2016).

[62] Yarin Gal, Jiri Hron, and Alex Kendall. “Concrete dropout”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 3581–3590.

[63] Christos Louizos and Max Welling. “Multiplicative normalizing flows for variational Bayesian neural networks”. In: *International Conference on Machine Learning*. 2017, pp. 2218–2227.

[64] Shengyang Sun, Changyou Chen, and Lawrence Carin. “Learning structured weight uncertainty in bayesian neural networks”. In: *Artificial Intelligence and Statistics*. 2017, pp. 1283–1292.

[65] Alex Kendall and Yarin Gal. “What uncertainties do we need in bayesian deep learning for computer vision?” In: *Advances in Neural Information Processing Systems*. 2017, pp. 5574–5584.

[66] Guodong Zhang, Shengyang Sun, David Duvenaud, and Roger Grosse. “Noisy natural gradient as variational inference”. In: *International Conference on Machine Learning*. 2018, pp. 5852–5861.

[67] A Wu, S Nowozin, E Meeds, RE Turner, JM Hernández-Lobato, and AL Gaunt. “Deterministic variational inference for robust Bayesian neural networks”. In: *7th International Conference on Learning Representations, ICLR 2019*. 2019.

[68] Andrei Atanov, Arsenii Ashukha, Dmitry Molchanov, Kirill Neklyudov, and Dmitry Vetrov. “Uncertainty estimation via stochastic batch normalization”. In: *International Symposium on Neural Networks*. Springer. 2019, pp. 261–269.

[69] Zhilu Zhang, Adrian V Dalca, and Mert R Sabuncu. “Confidence calibration for convolutional neural networks using structured dropout”. In: *arXiv preprint arXiv:1906.09551* (2019).

[70] Cedrique Rovile Njieutcheu Tassi. "Bayesian convolutional neural network: Robustly quantify uncertainty for misclassifications detection". In: *Pattern Recognition and Artificial Intelligence: Third Mediterranean Conference, MedPRAI 2019, Istanbul, Turkey, December 22–23, 2019, Proceedings 3*. Springer. 2020, pp. 118–132.

[71] Aryan Mobiny, Hien V Nguyen, Supratik Moulik, Naveen Garg, and Carol C Wu. "DropConnect is effective in modeling uncertainty of Bayesian deep networks". In: *arXiv preprint arXiv:1906.04569* (2019).

[72] Wesley J Maddox, Pavel Izmailov, Timur Garipov, Dmitry P Vetrov, and Andrew Gordon Wilson. "A simple baseline for bayesian uncertainty in deep learning". In: *Advances in Neural Information Processing Systems*. 2019, pp. 13153–13164.

[73] Kazuki Osawa, Siddharth Swaroop, Mohammad Emtiyaz E Khan, Anirudh Jain, Runa Eschenhagen, Richard E Turner, and Rio Yokota. "Practical deep learning with bayesian principles". In: *Advances in neural information processing systems*. 2019, pp. 4287–4299.

[74] Andrew G Wilson and Pavel Izmailov. "Bayesian deep learning and a probabilistic perspective of generalization". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin. Vol. 33. 2020, pp. 4697–4708.

[75] Sebastian Farquhar, Lewis Smith, and Yarin Gal. "Try depth instead of weight correlations: Mean-field is a less restrictive assumption for deeper networks". In: *arXiv preprint arXiv:2002.03704* (2020).

[76] Jongseok Lee, Matthias Humt, Jianxiang Feng, and Rudolph Triebel. "Estimating model uncertainty of neural networks in sparse information form". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 5702–5713.

[77] Apoorv Vyas, Nataraj Jammalamadaka, Xia Zhu, Dipankar Das, Bharat Kaul, and Theodore L Willke. "Out-of-distribution detection using an ensemble of self supervised leave-out classifiers". In: *Proceedings of the European Conference on Computer Vision*. 2018, pp. 550–564.

[78] William H Beluch, Tim Genewein, Andreas Nürnberger, and Jan M Köhler. "The power of ensembles for active learning in image classification". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 9368–9377.

[79] Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, David Sculley, Sebastian Nowozin, Joshua Dillon, Balaji Lakshminarayanan, and Jasper Snoek. “Can you trust your model’s uncertainty? Evaluating predictive uncertainty under dataset shift”. In: *Advances in Neural Information Processing Systems*. 2019, pp. 13991–14002.

[80] Matias Valdenegro-Toro. “Deep sub-ensembles for fast uncertainty estimation in image classification”. In: *Bayesian Deep Learning Workshop at Neural Information Processing Systems 2019*. 2019.

[81] Rahul Rahaman and Alexandre H Thiery. “Uncertainty Quantification and Deep Ensembles”. In: *stat* 1050 (2020), p. 20.

[82] Yeming Wen, Ghassen Jerfel, Rafael Muller, Michael W Dusenberry, Jasper Snoek, Balaji Lakshminarayanan, and Dustin Tran. “Combining ensembles and data augmentation can harm your calibration”. In: *International Conference on Learning Representations*. 2021.

[83] Fredrik K Gustafsson, Martin Danelljan, and Thomas B Schon. “Evaluating scalable bayesian deep learning methods for robust computer vision”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 2020, pp. 318–319.

[84] Guotai Wang, Wenqi Li, Sébastien Ourselin, and Tom Vercauteren. “Automatic brain tumor segmentation using convolutional neural networks with test-time augmentation”. In: *International MICCAI Brainlesion Workshop*. Springer. 2018, pp. 61–72.

[85] Murat Seckin Ayhan and Philipp Berens. “Test-time data augmentation for estimation of heteroscedastic aleatoric uncertainty in deep neural networks”. In: *Medical Imaging with Deep Learning Conference*. 2018.

[86] Guotai Wang, Wenqi Li, Michael Aertsen, Jan Deprest, Sébastien Ourselin, and Tom Vercauteren. “Aleatoric uncertainty estimation with test-time augmentation for medical image segmentation with convolutional neural networks”. In: *Neurocomputing* 338 (2019), pp. 34–45.

[87] Nikita Moshkov, Botond Mathe, Attila Kertesz-Farkas, Reka Hollandi, and Peter Horvath. “Test-time augmentation for deep learning-based cell segmentation on microscopy images”. In: *Scientific reports* 10.1 (2020), pp. 1–7.

[88] Divya Shanmugam, Davis Blalock, Guha Balakrishnan, and John Guttag. “When and why test-time augmentation works”. In: *arXiv preprint arXiv:2011.11156* (2020).

[89] Dmitry Molchanov, Alexander Lyzhov, Yuliya Molchanova, Arsenii Ashukha, and Dmitry Vetrov. “Greedy policy search: A simple baseline for learnable test-time augmentation”. In: *arXiv preprint arXiv:2002.09103* 2.7 (2020).

[90] Ildoo Kim, Younghoon Kim, and Sungwoong Kim. “Learning loss for test-time augmentation”. In: *Advances in Neural Information Processing Systems*. 2020, pp. 4163–4174.

[91] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. “Rethinking the inception architecture for computer vision”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 2818–2826.

[92] Gabriel Pereyra, George Tucker, Jan Chorowski, Łukasz Kaiser, and Geoffrey Hinton. “Regularizing neural networks by penalizing confident output distributions”. In: *arXiv preprint arXiv:1701.06548* (2017).

[93] Kimin Lee, Honglak Lee, Kibok Lee, and Jinwoo Shin. “Training confidence-calibrated classifiers for detecting out-of-distribution samples”. In: *International Conference on Learning Representations*. 2018.

[94] Rafael Müller, Simon Kornblith, and Geoffrey E Hinton. “When does label smoothing help?” In: *Advances in Neural Information Processing Systems*. 2019, pp. 4694–4703.

[95] Bindya Venkatesh and Jayaraman J Thiagarajan. “Heteroscedastic calibration of uncertainty estimators in deep learning”. In: *arXiv preprint arXiv:1910.14179* (2019).

[96] Jonathan Wenger, Hedvig Kjellström, and Rudolph Triebel. “Non-parametric calibration for classification”. In: *International Conference on Artificial Intelligence and Statistics*. 2020, pp. 178–190.

[97] Stanislav Fort, Huiyi Hu, and Balaji Lakshminarayanan. “Deep ensembles: A loss landscape perspective”. In: *arXiv preprint arXiv:1912.02757* (2019).

[98] Shengyang Sun, Guodong Zhang, Jiaxin Shi, and Roger Grosse. “Functional variational bayesian neural networks”. In: *arXiv preprint arXiv:1903.05779* (2019).

[99] Jakob Gawlikowski, Cedrique Rovile Njieutcheu Tassi, Mohsin Ali, Jongseok Lee, Matthias Humt, Jianxiang Feng, Anna Kruspe, Rudolph Triebel, Peter Jung, Ribana Roscher, et al. “A survey of uncertainty in deep neural networks”. In: *Artificial Intelligence Review* 56.Suppl 1 (2023), pp. 1513–1589.

[100] Koustuv Sinha, Joelle Pineau, Jessica Forde, Rosemary Nan Ke, and Hugo Larochelle. “NeurIPS 2019 reproducibility challenge”. In: *ReScience C* 6.2 (2020), p. 11.

[101] Cedrique Rovile Njieutcheu Tassi, Anko Börner, and Rudolph Triebel. “Monte Carlo averaging for uncertainty estimation in neural networks”. In: *Journal of Physics: Conference Series*. Vol. 2506. 1. IOP Publishing. 2023, p. 012004.

[102] Cedrique Rovile Njieutcheu Tassi, Jakob Gawlikowski, Auliya Unnisa Fitri, and Rudolph Triebel. “The impact of averaging logits over probabilities on ensembles of neural networks.” In: *AISafety@ IJCAI*. 2022.

[103] Cedrique Rovile Njieutcheu Tassi, Anko Börner, and Rudolph Triebel. “Regularization Strength Impact on Neural Network Ensembles”. In: *Proceedings of the 2022 5th International Conference on Algorithms, Computing and Artificial Intelligence*. 2022, pp. 1–9.

[104] Anil K Jain, Jianchang Mao, and K Moidin Mohiuddin. “Artificial neural networks: A tutorial”. In: *Computer* 29.3 (1996), pp. 31–44.

[105] Katarzyna Janocha and Wojciech Marian Czarnecki. “On loss functions for deep neural networks in classification”. In: *arXiv preprint arXiv:1702.05659* (2017).

[106] Ajay Shrestha and Ausif Mahmood. “Review of deep learning algorithms and architectures”. In: *IEEE Access* 7 (2019), pp. 53040–53065.

[107] Guo-Xun Yuan, Chia-Hua Ho, and Chih-Jen Lin. “Recent advances of large-scale linear classification”. In: *Proceedings of the IEEE* 100.9 (2012), pp. 2584–2603.

[108] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in Neural Information Processing Systems* 25 (2012).

[109] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. “Imagenet large scale visual recognition challenge”. In: *International Journal of Computer Vision* 115.3 (2015), pp. 211–252.

[110] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).

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

[112] Sergey Zagoruyko and Nikos Komodakis. “Wide residual networks”. In: *arXiv preprint arXiv:1605.07146* (2016).

[113] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. “Aggregated residual transformations for deep neural networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 1492–1500.

[114] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. "Densely connected convolutional networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 4700–4708.

[115] Waseem Rawat and Zenghui Wang. "Deep convolutional neural networks for image classification: A comprehensive review". In: *Neural Computation* 29.9 (2017), pp. 2352–2449.

[116] Asifullah Khan, Anabia Sohail, Umme Zahoora, and Aqsa Saeed Qureshi. "A survey of the recent architectures of deep convolutional neural networks". In: *Artificial Intelligence Review* 53.8 (2020), pp. 5455–5516.

[117] Laith Alzubaidi, Jinglan Zhang, Amjad J Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, J Santamarıa, Mohammed A Fadhel, Muthana Al-Amidie, and Laith Farhan. "Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions". In: *Journal of Big Data* 8.1 (2021), pp. 1–74.

[118] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep sparse rectifier neural networks". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings. 2011, pp. 315–323.

[119] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. "Rectifier nonlinearities improve neural network acoustic models". In: *Proceedings of the International Conference on Machine Learning*. Vol. 30. 1. Citeseer. 2013, p. 3.

[120] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[121] Xiaojie Jin, Chunyan Xu, Jiashi Feng, Yunchao Wei, Junjun Xiong, and Shuicheng Yan. "Deep learning with s-shaped rectified linear activation units". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 30. 1. 2016.

[122] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. "Fast and accurate deep network learning by exponential linear units (elus)". In: *arXiv preprint arXiv:1511.07289* (2015).

[123] Yanming Guo, Yu Liu, Ard Oerlemans, Songyang Lao, Song Wu, and Michael S Lew. "Deep learning for visual understanding: A review". In: *Neurocomputing* 187 (2016), pp. 27–48.

[124] Y-Lan Boureau, Jean Ponce, and Yann LeCun. "A theoretical analysis of feature pooling in visual recognition". In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. 2010, pp. 111–118.

[125] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[126] Murat Sensoy, Lance Kaplan, and Melih Kandemir. “Evidential deep learning to quantify classification uncertainty”. In: *Advances in Neural Information Processing Systems* 31 (2018).

[127] Bolin Gao and Lacra Pavel. “On the properties of the softmax function with application in game theory and reinforcement learning”. In: *arXiv preprint arXiv:1704.00805* (2017).

[128] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. “A comprehensive survey on transfer learning”. In: *Proceedings of the IEEE* 109.1 (2020), pp. 43–76.

[129] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. “A survey of transfer learning”. In: *Journal of Big Data* 3.1 (2016), pp. 1–40.

[130] Randall J Erb. “Introduction to backpropagation neural network computation”. In: *Pharmaceutical Research* 10.2 (1993), pp. 165–170.

[131] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. “On the importance of initialization and momentum in deep learning”. In: *International Conference on Machine Learning*. PMLR. 2013, pp. 1139–1147.

[132] Samira Pouyanfar, Saad Sadiq, Yilin Yan, Haiman Tian, Yudong Tao, Maria Presa Reyes, Mei-Ling Shyu, Shu-Ching Chen, and Sundaraja S Iyengar. “A survey on deep learning: Algorithms, techniques, and applications”. In: *ACM Computing Surveys (CSUR)* 51.5 (2018), pp. 1–36.

[133] Eyke Hüllermeier and Willem Waegeman. “Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods”. In: *Machine Learning* 110.3 (2021), pp. 457–506.

[134] Vishal Thanvantri Vasudevan, Abhinav Sethy, and Alireza Roshan Ghias. “Towards better confidence estimation for neural models”. In: *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2019, pp. 7335–7339.

[135] Christopher Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, 2006.

[136] Zhi-Hua Zhou. *Ensemble methods: Foundations and algorithms*. CRC press, 2012.

[137] Laurent Valentin Jospin, Hamid Laga, Farid Boussaid, Wray Buntine, and Mohammed Bennamoun. “Hands-on Bayesian neural networks—A tutorial for deep learning users”. In: *IEEE Computational Intelligence Magazine* 17.2 (2022), pp. 29–48.

[138] Radford M Neal. *Bayesian training of backpropagation networks by the hybrid Monte Carlo method*. Tech. rep. Citeseer, 1992.

[139] John S Denker and Yann LeCun. “Transforming neural-net output levels to probability distributions”. In: *Advances in Neural Information Processing Systems*. 1991, pp. 853–859.

[140] David JC MacKay. “A practical Bayesian framework for backpropagation networks”. In: *Neural Computation* 4.3 (1992), pp. 448–472.

[141] Geoffrey E Hinton and Drew Van Camp. “Keeping the neural networks simple by minimizing the description length of the weights”. In: *Proceedings of the Sixth Annual Conference on Computational Learning Theory*. 1993, pp. 5–13.

[142] David Barber and Christopher M Bishop. “Ensemble learning in Bayesian neural networks”. In: *Nato ASI Series F Computer and Systems Sciences* 168 (1998), pp. 215–238.

[143] José Mena, Oriol Pujol, and Jordi Vitrià. “A survey on uncertainty estimation in deep learning classification systems from a Bayesian perspective”. In: *ACM Computing Surveys (CSUR)* 54.9 (2021), pp. 1–35.

[144] Laurent Valentin Jospin, Wray Buntine, Farid Boussaid, Hamid Laga, and Mohammed Bennamoun. “Hands-on Bayesian Neural Networks–a Tutorial for Deep Learning Users”. In: *arXiv preprint arXiv:2007.06823* (2020).

[145] Xitong Yang. “Understanding the variational lower bound”. In: *variational lower bound, ELBO, hard attention* 13 (2017), pp. 1–4.

[146] David Ha, Andrew Dai, and Quoc V Le. “Hypernetworks”. In: *arXiv preprint arXiv:1609.09106* (2016).

[147] Zach Eaton-Rosen, Felix Bragman, Sotirios Bisdas, Sébastien Ourselin, and M Jorge Cardoso. “Towards safe deep learning: Accurately quantifying biomarker uncertainty in neural network predictions”. In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2018, pp. 691–699.

[148] Antonio Loquercio, Mattia Segu, and Davide Scaramuzza. “A general framework for uncertainty estimation in deep learning”. In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 3153–3160.

[149] Marc Rußwurm, Syed Mohsin Ali, Xiao Xiang Zhu, Yarin Gal, and Marco Körner. "Model and data uncertainty for satellite time series forecasting with deep recurrent models". In: *2020 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*. 2020.

[150] Jiaming Zeng, Adam Lesnikowski, and Jose M Alvarez. "The relevance of Bayesian layer positioning to model uncertainty in deep Bayesian active learning". In: *arXiv preprint arXiv:1811.12535* (2018).

[151] Nicolas Brosse, Carlos Riquelme, Alice Martin, Sylvain Gelly, and Éric Moulines. "On last-layer algorithms for classification: Decoupling representation from uncertainty estimation". In: *arXiv preprint arXiv:2001.08049* (2020).

[152] Agustinus Kristiadi, Matthias Hein, and Philipp Hennig. "Being bayesian, even just a bit, fixes overconfidence in relu networks". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 5436–5446.

[153] Dan Cireşan, Ueli Meier, Jonathan Masci, and Jürgen Schmidhuber. "A committee of neural networks for traffic sign classification". In: *The 2011 International Joint Conference on Neural Networks*. IEEE. 2011, pp. 1918–1921.

[154] Michał Woźniak, Manuel Grana, and Emilio Corchado. "A survey of multiple classifier systems as hybrid systems". In: *Information Fusion* 16 (2014), pp. 3–17.

[155] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. "Adaptive mixtures of local experts". In: *Neural Computation* 3.1 (1991), pp. 79–87.

[156] Lars Kai Hansen and Peter Salamon. "Neural network ensembles". In: *IEEE transactions on pattern analysis and machine intelligence* 12.10 (1990), pp. 993–1001.

[157] Pedro Domingos. "Bayesian averaging of classifiers and the overfitting problem". In: *ICML*. Vol. 747. Citeseer. 2000, pp. 223–230.

[158] Thomas P Minka. "Bayesian Model Averaging Is Not Model Combination. 2002". In: *Comment available electronically at http://www. stat. cmu. edu/minka/papers/bma. html* ().

[159] Kristine Monteith, James L Carroll, Kevin Seppi, and Tony Martinez. "Turning Bayesian model averaging into Bayesian model combination". In: *The 2011 International Joint Conference on Neural Networks*. IEEE. 2011, pp. 2657–2663.

[160] Kenji Kawaguchi. "Deep learning without poor local minima". In: *arXiv preprint arXiv:1605.07110* (2016).

[161] Thomas G Dietterich. "Ensemble methods in machine learning". In: *International Workshop on Multiple Classifier Systems*. Springer. 2000, pp. 1–15.

[162] Pedro M Domingos. “Why Does Bagging Work? A Bayesian Account and its Implications.” In: *KDD*. Citeseer. 1997, pp. 155–158.

[163] Robi Polikar. “Ensemble learning”. In: *Ensemble Machine Learning*. Springer, 2012, pp. 1–34.

[164] Kagan Tumer and Joydeep Ghosh. “Analysis of decision boundaries in linearly combined neural classifiers”. In: *Pattern Recognition* 29.2 (1996), pp. 341–348.

[165] Anders Krogh, Jesper Vedelsby, et al. “Neural network ensembles, cross validation, and active learning”. In: *Advances in Neural Information Processing Systems* 7 (1995), pp. 231–238.

[166] Wendy S Parker. “Ensemble modeling, uncertainty and robust predictions”. In: *Wiley Interdisciplinary Reviews: Climate Change* 4.3 (2013), pp. 213–223.

[167] Arsenii Ashukha, Alexander Lyzhov, Dmitry Molchanov, and Dmitry Vetrov. “Pitfalls of in-domain uncertainty estimation and ensembling in deep learning”. In: *arXiv preprint arXiv:2002.06470* (2020).

[168] Xixin Wu and Mark Gales. “Should ensemble members be calibrated?” In: *arXiv preprint arXiv:2101.05397* (2021).

[169] Tin Kam Ho. “The random subspace method for constructing decision forests”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20.8 (1998), pp. 832–844.

[170] Stefan Lee, Senthil Purushwalkam, Michael Cogswell, David Crandall, and Dhruv Batra. “Why M heads are better than one: Training a diverse ensemble of deep networks”. In: *arXiv preprint arXiv:1511.06314* (2015).

[171] Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E Hopcroft, and Kilian Q Weinberger. “Snapshot ensembles: Train 1, get m for free”. In: *arXiv preprint arXiv:1704.00109* (2017).

[172] Jun Yang and Fei Wang. “Auto-ensemble: An adaptive learning rate scheduling based deep learning model ensembling”. In: *IEEE Access* 8 (2020), pp. 217499–217509.

[173] Leo Breiman. “Bagging predictors”. In: *Machine Learning* 24.2 (1996), pp. 123–140.

[174] Yoav Freund, Robert E Schapire, et al. “Experiments with a new boosting algorithm”. In: *International Conference on Machine Learning*. Vol. 96. Citeseer. 1996, pp. 148–156.

[175] Mohammad Moghimi, Serge J Belongie, Mohammad J Saberian, Jian Yang, Nuno Vasconcelos, and Li-Jia Li. “Boosted convolutional neural networks.” In: *BMVC*. Vol. 5. 2016, p. 6.

[176] Loris Nanni, Sheryl Brahnam, and Gianluca Maguolo. “Data augmentation for building an ensemble of convolutional neural networks”. In: *Innovation in Medicine and Healthcare Systems, and Multimedia*. Singapore: Springer Singapore, 2019, pp. 61–69.

[177] Jian Guo and Stephen Gould. “Deep CNN ensemble with data augmentation for object detection”. In: *arXiv preprint arXiv:1506.07224* (2015).

[178] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. “The pascal visual object classes (voc) challenge”. In: *International Journal of Computer Vision* 88.2 (2010), pp. 303–338.

[179] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. “Microsoft coco: Common objects in context”. In: *European Conference on Computer Vision*. Springer. 2014, pp. 740–755.

[180] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. “mixup: Beyond empirical risk minimization”. In: *International Conference on Learning Representations*. 2018.

[181] Juan Maroñas, Daniel Ramos, and Roberto Paredes. “Improving calibration in mixup-trained deep neural networks through confidence-based loss functions”. In: *arXiv preprint arXiv:2003.09946* (2020).

[182] E. J. Herron, S. R. Young, and T. E. Potok. “Ensembles of networks produced from neural architecture search”. In: *International Conference on High Performance Computing*. Springer. 2020, pp. 223–234.

[183] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. “Going deeper with convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[184] Ioannis E Livieris, Lazaros Iliadis, and Panagiotis Pintelas. “On ensemble techniques of weight-constrained neural networks”. In: *Evolving Systems* (2020), pp. 1–13.

[185] Alex Hernández-Garcıa and Peter König. “Data augmentation instead of explicit regularization”. In: *arXiv preprint arXiv:1806.03852* (2018).

[186] Tobias Hinz, Pablo Barros, and Stefan Wermter. “The effects of regularization on learning facial expressions with convolutional neural networks”. In: *International Conference on Artificial Neural Networks*. Springer. 2016, pp. 80–87.

[187] Yarin Gal, Riashat Islam, and Zoubin Ghahramani. “Deep bayesian active learning with image data”. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 1183–1192.

[188] Remus Pop and Patric Fulop. “Deep ensemble bayesian active learning: Addressing the mode collapse issue in monte carlo dropout via ensembles”. In: *arXiv preprint arXiv:1811.03897* (2018).

[189] Yunfeng Zhang, Q Vera Liao, and Rachel KE Bellamy. “Effect of confidence and explanation on accuracy and trust calibration in AI-assisted decision making”. In: *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency.* 2020, pp. 295–305.

[190] Umang Bhatt, Javier Antorán, Yunfeng Zhang, Q Vera Liao, Prasanna Sattigeri, Riccardo Fogliato, Gabrielle Melançon, Ranganath Krishnan, Jason Stanley, Omesh Tickoo, et al. “Uncertainty as a form of transparency: Measuring, communicating, and using uncertainty”. In: *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society.* 2021, pp. 401–413.

[191] Grigorios Tsoumakas, Ioannis Partalas, and Ioannis Vlahavas. “A taxonomy and short review of ensemble selection”. In: *Workshop on Supervised and Unsupervised Ensemble Methods and Their Applications.* 2008, pp. 1–6.

[192] Han Xiao, Kashif Rasul, and Roland Vollgraf. “Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms”. In: *arXiv preprint arXiv:1708.07747* (2017).

[193] Alex Krizhevsky, Geoffrey Hinton, et al. “Learning multiple layers of features from tiny images”. In: (2009).

[194] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and Christian Igel. “Detection of traffic signs in real-world images: The German traffic sign detection benchmark”. In: *The 2013 International Joint Conference on Neural Networks (IJCNN).* IEEE. 2013, pp. 1–8.

[195] Min Lin, Qiang Chen, and Shuicheng Yan. “Network in network”. In: *arXiv preprint arXiv:1312.4400* (2013).

[196] Mahdi Pakdaman Naeini, Gregory F Cooper, and Milos Hauskrecht. “Obtaining well calibrated probabilities using bayesian binning”. In: *Proceedings of the... AAAI Conference on Artificial Intelligence. AAAI Conference on Artificial Intelligence.* Vol. 2015. NIH Public Access. 2015, p. 2901.

[197] Max-Heinrich Laves, Sontje Ihler, Karl-Philipp Kortmann, and Tobias Ortmaier. “Well-calibrated model uncertainty with temperature scaling for dropout variational inference”. In: *arXiv preprint arXiv:1909.13550* (2019).

[198] Sunil Thulasidasan, Gopinath Chennupati, Jeff A Bilmes, Tanmoy Bhattacharya, and Sarah Michalak. “On mixup training: Improved calibration and predictive uncertainty for deep neural networks”. In: *Advances in Neural Information Processing Systems.* 2019, pp. 13888–13899.

[199] Gongbo Liang, Yu Zhang, Xiaoqin Wang, and Nathan Jacobs. “Improved trainable calibration method for neural networks on medical imaging classification”. In: *arXiv preprint arXiv:2009.04057* (2020).