On Confidence, Agreement and Calibration of Deep Ensembles

by

Alim Manjiyani

Thesis submitted to the University of Ottawa
in partial fulfillment of the requirements for the degree of
Master of Science
in
Computer Science

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Examining Committee

The following served on the Examining Committee for this thesis.

Carleton Member: Oliver Van Kaick
Associate Professor, School of Computer Science
Carleton Univeristy

Internal Member: Paula Branco
Assistant Professor, School of Electrical Engineering & Computer Science
University of Ottawa

Supervisor(s): Yongyi Mao
Professor, School of Electrical Engineering & Computer Science
University of Ottawa

Declaration of Authorship

I hereby certify that this thesis is entirely my own original work except where otherwise indicated. I am aware of the University of Ottawa regulations concerning plagiarism, including those regarding consequent disciplinary actions. Any use of the works of any other author, in any form, is properly acknowledged at their point of use.

Abstract

Deep ensembles, composed of multiple independently trained Deep neural networks (DNNs), have demonstrated strong predictive power and reliability in various machine learning applications such as classification [6], uncertainty estimation [21], anomaly detection [11], medical diagnostics [1] etc. This paper delves into the intricate interplay between confidence, agreement, and calibration within deep ensembles, shedding light on their relationships and combined properties.

We begin this thesis by providing a brief yet insightful background of Calibration, its various notions, evaluation methods, and the existing techniques to achieve calibration. We extend on Generalization Disagreement Equality (GDE) [16] - a natural phenomenon of deep ensembles, which states that the agreement and accuracy of a deep ensemble are equal in expectation over population. We introduce a more generalized version of this phenomenon and call it Generalized GDE (GGDE). Similar to [16], we provide empirical evidence of GGDE and provide the sufficient calibration conditions required for GGDE. We further study the interaction between agreement and the true-confidence (accuracy) and establish bounds on their absolute difference. As a result, we shed light upon the possibility of instance-wise GGDE and provide a suitable calibration condition for the same.

In the second half of the thesis, we formulate the notion of "Confidence Calibration" and emphasize its importance theoretically. We study its properties and unveil its interesting properties. Remarkably, our work suggests that Confidence Calibration can serve as a theoretical foundation for understanding the effectiveness of Label Smoothing [34], thus establishing a connection between calibration and regularization.

We end the thesis by combining the properties of GGDE, Label Smoothing, and Confidence Calibration to provide a novel training algorithm termed Agreement Guided Label Smoothing (AGLS). We provide a theoretical justification for its functioning and demonstrate its effectiveness empirically. By advancing our understanding of Confidence Calibration, this work contributes to the broader goal of enhancing the trustworthiness and applicability of deep learning models in various domains.

Acknowledgements

In the game of life, where the quest for knowledge resembles an intricate and challenging adventure, there are those who have been my invaluable teammates, whose support and presence have been nothing short of remarkable.

To my mentor and supervisor Professor Yongyi Mao, you've been the seasoned captain of our squad and the MVP, guiding us through the complex levels of research with wisdom and expertise. Your strategies and insights have been the ultimate power-ups on this thrilling quest.

To my fellow players in the academic arena: Runzhi, Ziqiao, Zixuan and Zhiyi, your collaboration and teamwork have made each level of this journey both exciting and conquerable. Together, we've faced the tough bosses, cracked the codes, and celebrated our victories in this virtual world of science.

And to my family: Dad, Mom and my Sister; the ultimate cheerleaders, the source of power to my gaming console, who make my real-life adventure worth every challenge, your support and belief in my abilities have been the extra lives that keep me going, no matter how tough the obstacles.

This thesis is like the ultimate boss battle, and I couldn't have defeated it without each one of you. It is the result of our collective efforts, our shared strategies, and our determination to level up in the game of knowledge. Each one of you has played a vital role in this adventure, and for that, I'm deeply appreciative. As this chapter of the game concludes, I look forward to what the next level has in store for us. Thank you for being my trusted allies in this epic quest of the game of life.

Dedication

This work is dedicated to my mother Rukshana Manjiyani, my father Karim Manjiyani, and my sister Sneha Manjiyani. Your belief in my dreams, your sacrifices, and your ceaseless encouragement have been the unspoken dedication page of my thesis. This achievement is as much yours as it is mine, and I dedicate it to each of you with all my heart.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Deep neural networks (DNNs) have demonstrated remarkable success across various machine learning tasks, from computer vision [18] and speech recognition [15] to natural language processing [25] and bioinformatics [45], [2]. However, their reliability and robustness in real-world applications have been a subject of concern. One of the key challenges in deploying DNNs is the need for well-calibrated confidence estimates.

Calibration refers to the alignment between the predicted confidence scores produced by a DNN and the actual correctness probabilities of those predictions [28]; [8]. A well-calibrated model is not only aware of its own uncertainty but also provides reliable estimates of the likelihood that a given prediction is correct. Such calibration is crucial for various downstream tasks, including decision-making systems that rely on model confidence to guide their actions.

We start this thesis by investigating the concept of Calibration in detail. Specifically, we compare all the existing notions of calibration, the most widely used techniques of model-calibration and the methods to evaluate the level of calibration. We find that there are several notions of calibration, but they can be segregated based on their quantities of interest. To name a few, top-class calibration ([13]; [32]) is concerned with the predicted class with the highest confidence value for a given instance, whereas class-wise calibration is top-class calibration concerned with the confidence of all the classes and have been studied under different names [40]; [38]; [16]. Similarly, when the notion of calibration covers confidence of all classes on average, it is referred to as class-aggregated calibration [16] or static calibration [32]. The difference in these notions arise from the confidence function or the confidence estimation method being used.

Although multiple studies contribute to the fact that ensembles of several independently

and stochastically trained models tend to be naturally calibrated [13], [21], [9], [38], [3], [26], the work [16] pointed out that even a pair of models (ensemble of size two), appears to be well calibrated, where the agreement of the models aligns with a confidence function —a phenomenon termed Generalization Disagreement Equality (GDE). The authors [16] provide calibration conditions which are sufficient but not necessary for GDE to hold. GDE provides a way to estimate the accuracy of the model with a fresh unlabelled dataset. However, GDE requires that the hypothesis space of the classifier that constitutes the ensemble is restricted to only hard classifiers (classifiers that make deterministic predictions).

In this thesis, we extend on GDE and discover that we need not restrict the hypothesis space to only hard classifiers. We name this phenomenon **Generalized Generalization Disagreement Equality (GGDE)** and show that it holds through various experiments. Similar to [16], we discuss the sufficient calibration condition that may lead to GGDE and in doing so, we also provide a simpler interpretation to the class-aggregated calibration given by [16]. We continue to study this interplay between the agreement and accuracy of an ensemble and establish some bounds on their absolute difference. We then investigate the possibility of instance-wise GGDE and provide a calibration condition that implies instance-wise GGDE. As GGDE and GDE allow us to estimate the accuracy without needing the true labels, we investigate the performance of agreement of ensembles as a confidence score function in a selective classification setting, where the choice of such function is of great importance.

In the second part of this thesis, we formulate the notion of **Confidence Calibration**, which appears as a case of top-class calibration but reveals some interesting and desirable properties. Specifically, a classifier that is perfectly calibrated in this sense must agree with the ground-truth classifier. From pre-deep-learning era to the present, calibrating a neural network has been a major interest, and can be broadly divided into pre-processing methods and post-processing methods. Label smoothing [34] is one such widely adopted pre-processing method that is used for regularizing DNNs during training. It aims to mitigate overconfidence and encourage models to be more cautious in their predictions. Remarkably, our work suggests that Confidence Calibration can serve as a theoretical framework for understanding the effectiveness of Label Smoothing. This connection sheds new light on the interplay between calibration and regularization techniques.

Leveraging the GDE phenomenon and the benefits of label smoothing and the properties of a Confidence-Calibrated model, we propose a training algorithm named **Agreement Guided Label Smoothing (AGLS)** and provide a theoretical justification that explains its functioning. As this algorithm utilizes an unlabeled validation set, it can also be viewed as a special case of the Transductive Learning algorithms [44]. Our empirical results demonstrate the effectiveness of this approach in improving the reliability and

generalization performance of DNNs, without needing to search tediously for the smoothing hyper-parameters as in label smoothing.

We summarize our contributions in this thesis as follows:

- We extend on the work of Jiang *et al* [16] and discover the GGDE phenomenon and provide empirical evidence to support it.
- We formulate the notion of Confidence Calibration and emphasize its potential relationship with Label Smoothing.
- We present a practical method to exploit Confidence Calibration and its properties with the help of label smoothing and GGDE, and provide empirical evidence of its benefits.

The outline of this thesis is as follows: in Chapter 2, we present the preliminaries about machine learning and the related mathematical background. In Chapter 3, we introduce calibration and discuss its various notions, techniques of calibration, and methods of evaluating calibration. In Chapter 4, we introduce GDE and its calibration conditions and extend on it by formulating GGDE. Furthermore, we discuss the relationship between agreement and accuracy and discuss its impact on selective classification. In Chapter 5, we formulate the notion of Confidence Calibration, discuss its properties and explore its connection to Label Smoothing. In Chapter 6, we unite Confidence Calibration, GGDE and Label Smoothing to provide a novel training algorithm named AGLS and show its effectiveness empirically.

Chapter 2

Preliminaries

2.1 Machine Learning

"Machine Learning" (ML): a term coined in 1959 in reference to *self-teaching computers*, has become an umbrella term for solving problems that are too expensive or challenging for human programmers by allowing the machine to learn on their own as if they were discovering the solutions independently. ML finds its roots in the fields of probability theory, statistics, optimization *etc.* It is a powerful technology that has transformed various industries and applications by allowing systems to automatically improve their performance through experience. ML algorithms can be divided into 4 different categories: Supervised, Unsupervised, Semi-supervised, and Reinforcement learning. In this work, we focus mainly on Supervised and Semi-Supervised learning.

2.1.1 Supervised Learning

In this approach, the algorithm is trained on labeled data, where each input is associated with a correct output. The model learns to map inputs to outputs. Consider a labeled dataset $S := \{(x_i, y_i)\}_{i=1}^{i=n}$ of size n, where each x_i represent an input from the input space $\mathcal{X}$ and y_i represent its correct label that lives in output space $\mathcal{Y}$. These samples are drawn independently from an underlying distribution $\mathbb{P}_{X,Y}$, where X and Y are the random variables corresponding to input instances x_i's and their corresponding output y_i's. The objective of supervised learning is to learn a function $h : X \to Y$ which gives the best performance according to some evaluation metric. In other words, supervised learning aims to learn the posterior distribution $\mathbb{P}_{Y|X}$.

2.1.2 Semi-Supervised Learning

Semi-supervised learning deals with problems that involve learning from a mixture of labeled and unlabeled data. This approach is particularly useful in situations where obtaining a large amount of labeled data is expensive or time-consuming, but unlabeled data is abundant. It leverages the benefits of both labeled and unlabeled data to improve model performance.

2.2 Parameterized and Probabilistic Models

Parameterized model, also known as parametric model, is a type of model used in machine learning and statistics. In this context, "parametric" refers to the model structure, which is defined by a set of parameters. These parameters are numerical values that are learned from training data to make predictions, describe patterns, or represent relationships in the data. Specifically, a parametric model is a family of probability distributions that has a finite number of parameters, i.e. if Θ represents the parameter space, then a model is said to be parametric if $\Theta \in \mathbb{R}^d$ for some finite positive integer d.

A Probabilistic model, on the other hand, is a model that provides a probabilistic distribution over the possible outcomes. Unlike deterministic models that make deterministic predictions, probabilistic models provide a measure of uncertainty. They estimate the likelihood or probability of different outcomes. A probabilistic model can either be parametric or non-parametric. It's crucial to highlight that specific methods exist for transforming a deterministic model into a probabilistic one, and conversely, for converting a probabilistic model into a deterministic one.

While a parametric model is defined by a fixed set of parameters that determine the model's structure and relationships, a probabilistic model focuses on providing a distribution of probabilities for outcomes, allowing for more nuanced and uncertain predictions. They are related concepts, but they serve different purposes, with probabilistic models offering a richer representation of uncertainty. In this work, we target parameterized probabilistic and parameterized deterministic models, whose setup will be discussed as we gradually introduce other aspects required to explain the machine learning stratagem.

2.3 Classification Problems

Classification problems are the most common type of problem tackled by machine learning algorithms. They are a type of supervised machine learning task where the goal is to categorize or assign predefined labels or classes to input instances based on their characteristics or features. In classification, you aim to build a model that can make predictions about which category a new, unseen data point belongs to. These categories are typically discrete and represent different classes or groups. Any model which corresponds to a classification task is called a classifier. Consider a classification problem that involves classifying an input instance $x \in \mathcal{X}$ into K different categories, such a problem is called a K-class classification problem or a Multi-class Classification. A classifier for such a problem is any function $h : \mathcal{X} \to [\mathcal{K}]$, where $[K]$ denotes the set of K-labels $\{1, 2, \ldots, K\}$. Such a function is called a "hypothesis" and the space of all such functions is called the "hypothesis space" $\mathcal{H}$. A function that maps all the inputs to their actual/true classes is called a ground-truth classifier and is denoted by f.

It is important to note that a parametric classifier/hypothesis with parameter space $\Theta \in \mathbb{R}^d$, restricts the hypothesis space by assuming that the underlying mapping can be adequately described by a specific set of parameters. In other words, it imposes a predefined structure on the hypothesis space. Hence, each parameterized function/classifier $h \in \mathcal{H}$ can be distinguished based on the underlying set of parameters $\theta \in \Theta$, and we denote a classifier h governed by parameters θ as h_θ.

2.3.1 Binary Classification

As the name suggests, in binary classification, there are only two possible classes or categories and the goal of the model is to predict whether the given data point belongs to one class or the other. Given a dataset $S := \{x_i, y_i\}$, let X and Y represent the random variables corresponding to the input x and output y respectively. In the binary classification case, the space of outputs $\mathcal{Y}$ takes on only two values each corresponding to a class, i.e. $\mathcal{Y} = \{0, 1\}$. In this case, a deterministic classifier would output the predicted class/label $\hat{Y} = \{0, 1\}$ for the given instance x, whereas a probabilistic model would output a distribution over all possible classes i.e. $\mathbb{P}_{\hat{Y}|X}$, where $\mathbb{P}_{\hat{Y}|X}[\hat{Y} = 0|X] + \mathbb{P}_{\hat{Y}|X}[\hat{Y} = 1|X] = 1$.

In this scenario, a probabilistic classifier can be used as a deterministic classifier by interpreting its output as follows:

$$\hat{Y} = \begin{cases} 1, & \text{if} \quad \mathbb{P}_{\hat{Y}|X}[\hat{Y} = 1|X] > 0.5 \\ 0, & \text{otherwise} \end{cases}$$

2.3.2 Multi-Class Classification

In Multi-class classification, the output can have more than 2 classes. Let's assume we have a K-class classification, i.e. $\mathcal{Y} = \{1, 2, \ldots, K\}$. Generally, there are two strategies to solve this problem: One-vs-one (OvO) and One-vs-Rest(OvR). Both of these strategies solve the multi-class problem by splitting it into smaller binary-class problems

OvO strategy involves learning $\frac{K(K-1)}{2}$ binary classifiers, where each pair of classes is assigned a binary classifier. In this case, each binary classifier predicts a class for the given input and the class that is predicted the most is considered the predicted class.

On the other hand, the OvR strategy learns only K classifiers each corresponding to a class, with the instances of that class as positive and all other instances as negatives. This strategy requires each classifier to produce a score for its decision rather than a discrete class label. Let $h(x)_k$ denote the score obtained from the k^{th} classifier, where $k \in \{1, \ldots, K\}$ and $x \in \mathcal{X}$. The class label $\hat{y}$ with the highest score is chosen as the prediction for an unseen instance x, i.e.

$$\hat{y} = \text{argmax}_{k \in \{1,\ldots,K\}} h_k(x)$$

2.3.3 Common Datasets for Classification

The **CIFAR-10** and **CIFAR-100** [17] dataset are the most popular dataset used for image classification. They consist of 32 × 32 color images which represent 10 and 100 classes respectively. The dataset comprises of 60,000 images which is then split into a training set containing 50,000 images and a testing set containing 10,000 images. Both of these datasets contain the same images, but CIFAR-10 dataset categorizes them into 10 categories whereas CIFAR-100 categorizes them into 100 categories.

Another popular dataset is the **SVHN** [30] dataset. It is a widely used real-world image dataset generally used for object recognition tasks. Similar to the CIFAR-10/100 dataset, SVHN consists of 32 × 32 color images, each associated with a label ranging from 0 to 9 (10 classes). The dataset comprises around 100,000 images of small cropped digits of house numbers obtained from Google Street View images. The training set of SVHN contains 73,257 images, while the test set contains 26,032 images. The dataset also contains 531,131 additional, somewhat less challenging samples as supplementary training data.

2.4 Training

We say that a parametric model is "learning" if it gradually predicts the correct output more and more accurately. This process of "learning" is called **training**, and the data used for that purpose is called training data. Generally, an algorithm is used to provide feedback to the model on its performance and update its parameters so as to give a better result, this algorithm is called the training algorithm.

2.4.1 Hyperparameters

Hyperparameters is a set of parameters for the training algorithm used to train the model. They govern the learning procedure of a model. Unlike the parameters of the model which are trained throughout the training procedure, the hyperparameters are not trainable and are optimized after observing how their change affects the performance of a trained model.

2.4.2 Loss Functions

Loss Functions are the building blocks of a training procedure. Roughly speaking, loss functions reflect the distance between the predicted output and the actual output for a given input instance. In a K-class classification setting, we denote the loss function for any parametric classifier $h_\theta : \mathcal{X} \to [K]$ with parameters $\theta \in \Theta$, as $\ell(h_\theta(x), y)$, for a given $x \in \mathcal{X}$ and $y \in [K]$. Let $S := \{x_i, y_i\}_{i=1}^n$ denote a training set of size n for a K-class classification problem, then the empirical loss on S is defined as:

$$\hat{R}_S(h_\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell(h_\theta(x_i), y_i) \tag{2.1}$$

Empirical loss is also known as **empirical risk**. Intuitively, a bad classifier would correspond to a high loss, and on the contrary, a good classifier would correspond to a low loss. Thus, the objective of training a classifier is to find the optimal set of parameters θ^* that has the lowest empirical risk on S, i.e.

$$\theta^* = \arg\min_{\theta \in \Theta} \hat{R}_S(h_\theta)$$

This training scheme is also known as **empirical risk minimization (ERM)**.

In a classification problem, the classifier's uncertainty plays an important role in predicting the class label and is informative of the reliability of the predictions of the classifier. Therefore, we wish to train a parameterized probabilistic classifier $h_\theta : \mathcal{X} \to \Delta([K])$, where $\Delta([K])$ is the space of all probability distributions on $[K]$. The loss of the classifier in this case would be a measure of the difference between the true distribution $\mathbb{P}_{Y|X}$ and the predicted distribution $\mathbb{P}_{\hat{Y}|X}$, where $Y, \hat{Y} \in [K]$. One such statistical measure of how one probability distribution is different from another is KL-Divergence [20]. Minimizing the KL-Divergence between $\mathbb{P}_{Y|X}$ and $\mathbb{P}_{\hat{Y}|X}$ is equivalent to minimizing the **cross-entropy (CE) loss**, which is defined as:

$$\ell_{\text{CE}}(h_\theta, x, y) = -\sum_{i=1}^{K} \mathbb{P}_{Y|X}[Y = i|X = x] \log \left(\mathbb{P}_{\hat{Y}|X}[\hat{Y} = i|X = x] \right) \tag{2.2}$$

2.4.3 Training Setup

In classification problems, the setup of training targets is crucial. Generally, the training set consists of discrete values as labels/targets for a given instance, in such cases we must transform the target to a probability distribution over all the labels to be able to minimize the cross-entropy loss over the training set.

Consider a K-class classification problem with a dataset $S := \{x_i, y_i\}_{i=1}^m$ of size m, where each y_i is a discrete value representing a label for the instance x_i that lives in $[K]$. To convert such discrete labels to a probability distribution over labels, in practice, we encode them as **one-hot** vectors. Consider an input feature $x \in \mathcal{X}$, whose label $y = c$ where $c \; in[K]$, we denote its corresponding one hot vector y $:= [y_1, y_2, \ldots, y_K]$, which consists of only 0s and a single 1, created according to:

$$y_i = \begin{cases} 1, & \text{if} \quad i = c \\ 0, & \text{otherwise} \end{cases}$$

for all $i \in [K]$. These one-hot vectors are now treated as probability distributions over labels, such that the probability of the correct label is 1.

2.4.4 Gradient Descent

Gradient descent is a fundamental optimization algorithm used for training neural networks and many other machine learning models. The primary goal of gradient descent is to

minimize a cost or loss function by iteratively adjusting the model's parameters (weights and biases) in the direction that reduces the cost. This direction is obtained through the gradient/slope of the function at any given instance and we want to travel step by step in the direction where the slope is descending, hence the name gradient descent. Let θ represent the parameters of a given classifier h, here is a high-level overview of the gradient descent approach:

1. Initialize θ randomly.
2. Make prediction using θ.
3. Compute the loss.
4. Compute the gradient of the loss with respect to θ and update θ:

$$\theta^{\text{new}} = \theta^{\text{old}} - \eta \times \frac{1}{|S|} \sum_{(x,y)\in S} \nabla_\theta \quad \ell(\theta, x, y)$$

5. Repeat step 2 to 5

where η is called the learning rate and $|\cdot|$ is the cardinality of the set S. The learning rate η helps us control the length of the step. It is important to note that this approach of minimizing the loss function may not always give an optimal solution, for example when the step size is too large, we may never converge to the global minimum of the loss function and keep oscillating around it, or when the step is too small we may get stuck at local minima. However, with careful design of the algorithm, we can find an optimal solution.

2.4.5 Stochastic Gradient Descent

In **Stochastic Gradient Descent (SGD)**, the gradient is computed randomly (stochastically) from a single sample from S, instead of using the entire training set S. The idea behind this approach is to introduce randomness so as to be able to escape local minima. Here is an overview of SGD:

1. Initialize θ randomly.
2. Make prediction using θ.

3. Choose a sample $(x, y) \in S$ randomly.
4. Compute the loss on (x, y).
5. Compute the gradient of the loss with respect to θ and update theta

$$\theta^{\text{new}} = \theta^{\text{old}} - \eta \times \frac{1}{|S|} \nabla_\theta \quad \ell(\theta, x, y)$$

6. Repeat step 2 to 5

2.4.6 Mini-Batch Gradient Descent

Mini-Batch Gradient Descent or mini-batch SGD, is the most widely used form of gradient descent algorithm. It's a compromise between SGD and full-batch gradient descent. In each iteration, a small random batch of training examples is used to compute the gradient and update the parameters. This approach provides a balance between the noise introduced by SGD and the computational efficiency of full-batch gradient descent. Moreover, using the entire dataset (gradient descent) for each update can be computationally expensive, especially for large datasets combined with the fact that neural networks can have a large number of parameters loading the entire dataset into memory for each update can be impractical or impossible. On the other hand, using a single example (stochastic gradient descent) can lead to slow convergence due to noisy updates. The overview of the algorithm is described below:

1. Initialize θ randomly.
2. Make prediction using θ.
3. Draw a mini-batch $B \subset S$ randomly.
4. Compute the loss on B.
5. Compute the gradient of the loss with respect to θ and update theta

$$\theta^{\text{new}} = \theta^{\text{old}} - \eta \times \frac{1}{|S|} \sum_{(x,y) \in B} \nabla_\theta \quad \ell(\theta, x, y)$$

6. Repeat step 2 to 5

2.4.7 Practical Implementation

Generally, when we implement a mini-batch SGD we use a slightly different version of the algorithm than the one described above. This version grants us the ability to take more control over the training process and manipulate it to a certain extent.

In this version, each batch is drawn randomly, but without replacement, and drawing all the batches marks the end of an **epoch**. Moreover, the size of the mini-batch and the number of epochs along with the learning rate are treated as the standard hyperparameters of the training algorithm. Algorithm 2.1 describes the procedure discussed above, where $\lceil \cdot \rceil$ is the ceil function:

Algorithm 2.1 Mini-Batch SGD

Input:
Training set S
Learning rate η
Batch size b
Epochs e

Output:
θ

$n_b \leftarrow \lceil \frac{|S|}{b} \rceil$ ▷ Number of batches
Initialize θ randomly
$c \leftarrow 0$ ▷ Current Epoch
while $c \leq e$ **do**
  $c \leftarrow c + 1$
  $S' \leftarrow$ Randomly Shuffle S
  $S' = \{S'_1, S'_2, \ldots, S'_{n_b}\}$ ▷ Divided into n_b batches
  $i \leftarrow 0$
  while $i \leq n_b$ **do**
    $i \leftarrow i + 1$ ▷ Compute loss for batch S'_i
    Update θ

$$\theta \leftarrow \theta - \eta \times \frac{1}{|S|} \sum_{(x,y) \in S'_i} \nabla_\theta \quad \ell(\theta, x, y)$$

  end while
end while
return θ

2.5 Performance Evaluation

2.5.1 Testing

In the context of machine learning, "testing" refers to the evaluation of a trained machine learning model's performance on a dataset that it has not seen during training. This process is an essential part of the machine learning workflow and helps assess how well the model can generalize to new, unseen data.

The primary goal of testing is to assess the model's performance. It helps determine how well the model can make predictions on new, previously unseen data. Common evaluation metrics include accuracy, precision, recall, F1-score, mean squared error, and many others, depending on the type of machine learning task (classification, regression, etc.).

To be able to perform testing, we generally split the dataset into two parts namely, training set and testing set. While the model learns from the data in the training set, the samples in the testing set remains unseen by the model.

2.5.2 Underfitting

Underfitting is a common issue in machine learning and occurs when a model is too simple to capture the underlying patterns in the data. In other words, the model is not able to fit the training data well, and as a result, its performance on both the training data and unseen data (validation or test data) is suboptimal. Underfit models are said to have high bias and low variance.

A common solution to underfitting is to increase model complexity, increase the size of the training set or to tune the hyperparameters of the algorithm under use.

2.5.3 Overfitting

Overfitting is another common problem in machine learning and is the opposite of underfitting, where a model learns the training data too well, to the extent that it captures noise and random fluctuations in the data, rather than the underlying patterns. This results in a model that performs extremely well on the training data but poorly on unseen or validation data. Overfit models have low bias but high variance. Neural networks often contain a large number of parameters and in such cases overfitting is unavoidable if trained long enough.

2.5.4 Generalization

Generalization refers to the ability of a machine learning model to perform well on data it has never seen before, specifically on unseen or out-of-sample data. It is the ultimate goal of machine learning algorithms, as it reflects the model's capacity to capture underlying patterns, structures, or relationships in the data rather than merely memorizing the training data. Also, the difference between the training accuracy and the testing accuracy is referred to as the **generalization gap**.

2.6 Regularization

2.6.1 Introduction

Regularization is a technique used in machine learning to prevent overfitting and improve the generalization of a model. Regularization methods generally introduce a penalty term into the model's loss function, discouraging overly complex or large parameter values, which can lead to overfitting.

2.6.2 Early Stopping

Early stopping is a regularization technique that aims to find the optimal balance between model complexity and generalization. In this method, we first separate some samples from the training data and call it validation data/set. We then monitor the performance of the model after each updation step in the algorithm. The idea is to stop training the model when its performance on the validation set starts to degrade, indicating that it has reached the point of overfitting.

It is an effective and practical way to find a model that generalizes well to unseen data without requiring extensive hyperparameter tuning. However, it's essential to choose the early stopping criteria carefully to strike the right balance between training time and model performance.

2.6.3 Weight Decay

Weight decay is a regularization technique that discourages the model's parameters, also known as weights, from taking on large values. It does this by adding a penalty

term to the loss function during training, which is proportional to the sum of the squared values of the model's weights. The penalty term is controlled by a hyperparameter, often denoted as λ. Weight decay is also known as L2 regularization or ridge regularization as the penalty term is nothing but the L_2 norm or the Euclidean distance of the underlying vector. Mathematically, the loss function incorporating the weight decay is shown below:

$$\ell^{wd}(\theta, x, y) = \ell(\theta, x, y) + \lambda \cdot \|\theta\|_2$$

where $\|v\|_2 = \sqrt{\sum_{i=1}^{n} v_i^2}$ is the L_2 norm of a vector v of size n.

2.6.4 Data Augmentation

As discussed earlier that increasing the size of the training dataset can reduce overfitting, **Data Augmentation** is one such technique that artificially increase the size of a training dataset by applying various transformations to the existing data. Figure 2.1 shows various methods incorporated by Data Augmentation.

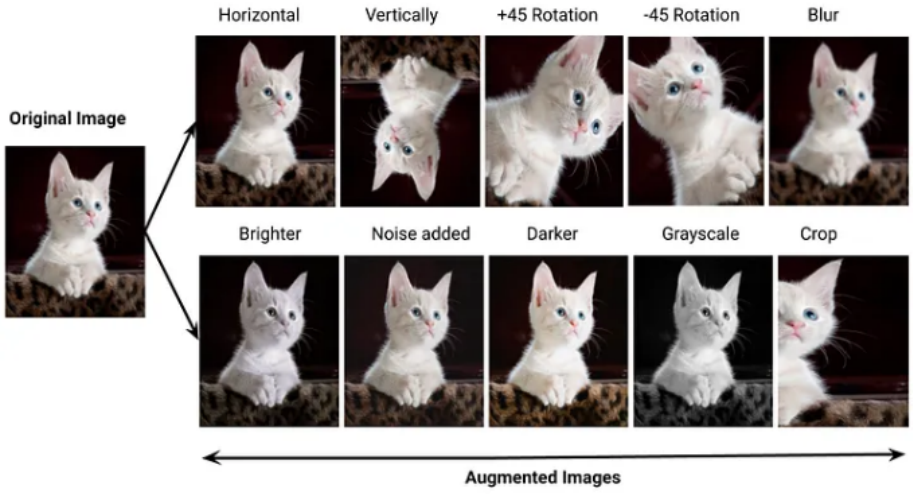


Figure 2.1: Data Augmentation methods. Image adapted from [35]

2.6.5 Label Smoothing

Label smoothing [34] is a regularization technique that addresses the problem of over-confidence in machine learning models, especially deep neural networks. When training a model for classification, it is common practice to use one-hot encoded labels, where the true class is assigned a probability of 1, and all other classes have a probability of 0.

Label smoothing introduces a small amount of uncertainty into the training process by replacing the one-hot encoded target labels with a smoothed distribution. Instead of assigning a probability of 1 to the true class, a value slightly less than 1 (e.g., 0.9) is assigned, and the remaining probability mass is distributed among other classes, including those that are not the true class. Typically, the distribution is uniform or nearly uniform.

Let f denote the ground-truth classifier of a K-class classification problem. We know that the ground truth-classifier only outputs one-hot labels and we train our neural network to minimize the cross-entropy between the model's prediction $h(x)$ and the ground-truth $f(x)$. The label smoothing technique allows us to control the amount of smoothing of ground-truth through a hyperparameter ϵ. The updated ground-truth using this ϵ is as follows:

$$f^{\mathrm{LS}}(x)_i := (1-\epsilon)f(x)_i + \frac{\epsilon}{K}$$

for all $x \in \mathcal{X}$ and $i \in [1, 2, \ldots, K]$. One might wonder that doing so would increase the computation cost while computing the cross-entropy loss as the ground-truth is no longer one-hot, but remarkably the cross-entropy loss takes a rather simple form as shown below:

$$\ell_{\mathrm{CE}}(h_\theta, x, y^{\mathrm{LS}}) = \ell_{\mathrm{CE}}(h_\theta, x, y) + \ell_{\mathrm{CE}}(h_\theta, x, u(K))$$

where y^{LS} is the label smoothed ground-truth and $u(K)$ represents the uniform distribution over K labels.

2.7 Artificial Neural Network

2.7.1 Artificial Neurons

An artificial neuron, also known as a perceptron, is a computational model inspired by the fundamental building block of the biological nervous system: the biological neuron [19].

The concept of artificial neurons in machine learning and neural networks draws inspiration from how biological neurons function in the human brain.

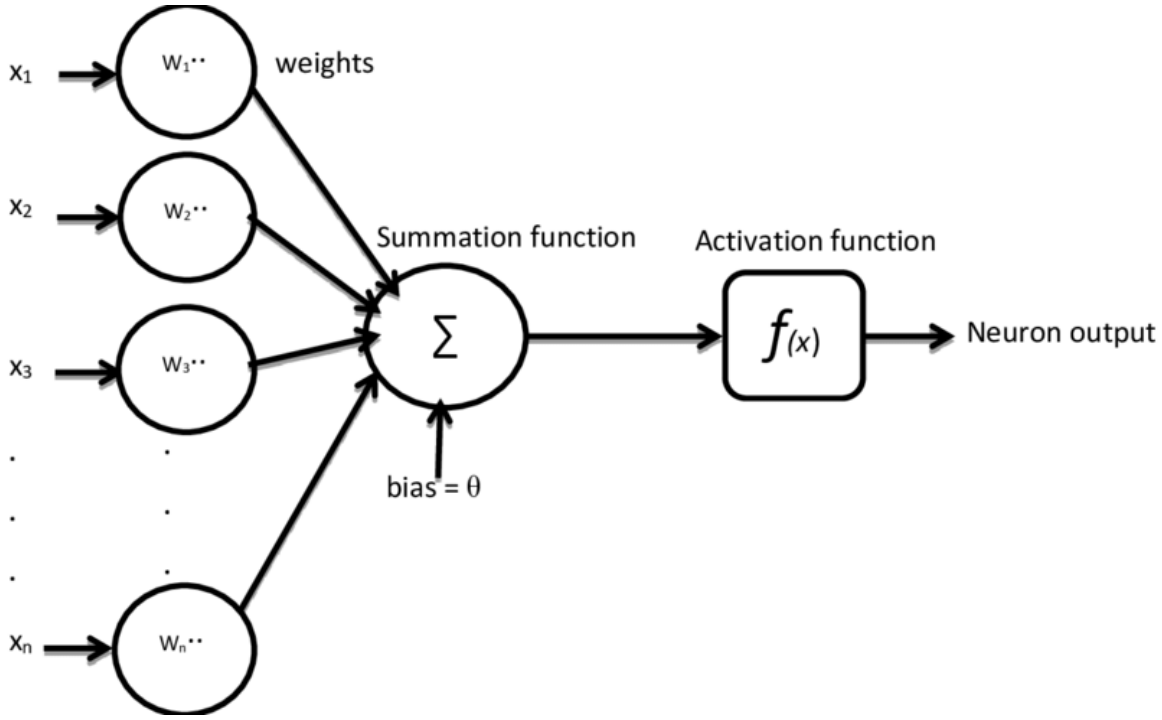


Figure 2.2: Artificial Neuron. Image adapted from [39]

Similar to how a biological neuron receives inputs through its dendrites from other neurons or sensory organs, an artificial neuron receives **input** data, often represented as numerical values, from other neurons in the network or external sources. These inputs are aggregated and processed mathematically.

In biological neurons, the strength of the connections between neurons (synaptic weights) influences the transmission of signals. These synaptic strengths determine how much weight each input has in the summation of signals. Comparably, artificial neurons use weights assigned to each input to scale and combine them. The **weighted sum** represents the neuron's attempt to weigh the importance of different inputs, just as synaptic strengths do in the brain.

Moreover, in the brain, once the integrated inputs surpass a certain threshold, a biological neuron generates an action potential, an electrical signal to communicate with other neurons.In a similar fashion, an artificial neuron applies an **activation function** to the

weighted sum of inputs. This activation function introduces non-linearity and controls the neuron's output. If the output surpasses a predefined threshold (usually determined by the activation function), the neuron becomes "active" and passes information to subsequent neurons.

The artificial neuron's motivation from biology lies in its attempt to capture the basic principles of input integration, weighted summation, and activation, which are fundamental to information processing in biological neurons. However, it's important to note that artificial neurons are highly simplified abstractions created for computational purposes and do not capture the full complexity and richness of biological neurons. Nonetheless, this abstraction has been instrumental in developing neural networks, enabling them to learn complex patterns and perform various tasks in machine learning.

Figure 2.2 shows the structure of an artificial neuron, that contains the inputs x_i's, the weights w_i's, a summation function $\sum$, and an activation function f. An input first x_i travels through the weights along its path w_i and end up at the summation function, whose output is the same of all the input-weight products, i.e. $\sum_i x_i \cdot w_i$ plus some bias b. This value is then passed through an activation function. The activation function $f : \mathbb{R} \to \mathbb{R}$ then computes the final output of the neuron. Mathematically, the output $o(x)$ of the artificial neuron with weights $w = \{w_1, w_2, \ldots, 2_n\}$ and an activation function ϕ, for an input $x = \{x_1, x_2, \ldots, x_n\}$ is defined as:

$$o(x) = \phi\left(\sum_{i=1}^{n} x_i \cdot w_i\right)$$

These neurons when stacked one after the other constitute the **Feed Forward Neural Networks**, where the chain of neurons allows the flow of information uni-directionally. In this thesis, we work only on such feed-forward neural networks.

2.7.2 Layered Structure

A layered structure is a defining characteristic of artificial neural networks, where neurons are organized into layers. The structured architecture, coupled with the network's ability to learn from data, enables neural networks to perform a wide range of tasks, from classification and regression to complex pattern recognition. The layers of the layered structure can be divided into three categories namely: input layer, hidden layer and output layer.

The first layer, known as the **input layer**, is responsible for receiving the initial data or features. Each neuron in this layer represents a specific feature or attribute of the input

data. The input layer is directly connected to the external data source, such as an image or a text document.

Between the input layer and the output layer, there may be one or more **hidden layers**. These hidden layers contain neurons that perform computations on the input data. Hidden layers are responsible for capturing complex patterns and relationships within the data. The number of hidden layers and the number of neurons in each hidden layer can vary, depending on the complexity of the problem.

The final layer, known as the **output layer**, produces the network's predictions or outputs. The structure of the output layer depends on the type of task the network is designed for. For example, in a classification task, the output layer may consist of neurons corresponding to different classes, while in a regression task, there might be a single neuron producing a continuous numerical output.

2.7.3 Activation Functions

Activation functions are critical components in artificial neural networks, particularly in the hidden layers of deep neural networks. They serve two primary purposes: introducing non-linearity and controlling neuron activation. It is important to recall the fact that training algorithms depend upon the gradient of the loss function, which would imply that any activation function used must be differentiable for the network to train. Some commonly used functions and their derivatives are list below:

Sigmoid Function or **Logistic Sigmoid**:

$$\sigma(x) := \frac{1}{1 + e^{-x}}$$

$$\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x))$$

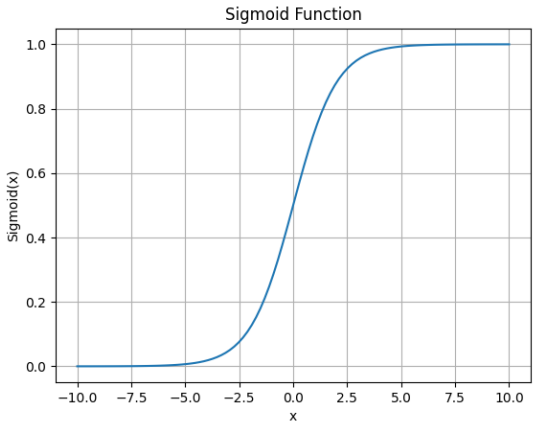


Figure 2.3: Sigmoid Function

Rectified Linear Unit (ReLU):

$$\mathrm{ReLU}(x) := \max(x, 0)$$

$$\mathrm{ReLU}'(x) = \begin{cases} 1, & \text{if} \quad x \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

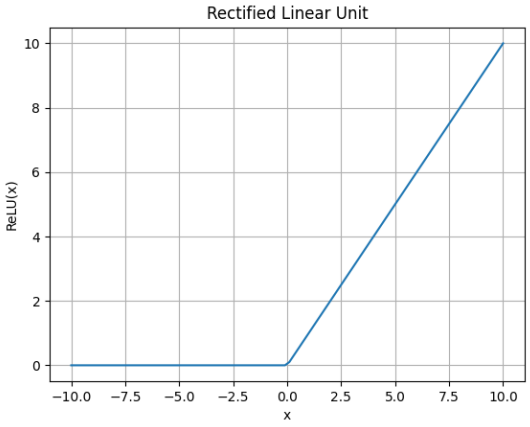


Figure 2.4: Rectified Linear Unit

Leaky Rectified Linear Unit (LReLU):

$$\text{LReLU}(x) := \max(0.01 \cdot x, x)$$

$$\text{ReLU}'(x) = \begin{cases} 1, & \text{if} \quad x \geq 0 \\ 0.01, & \text{otherwise} \end{cases}$$

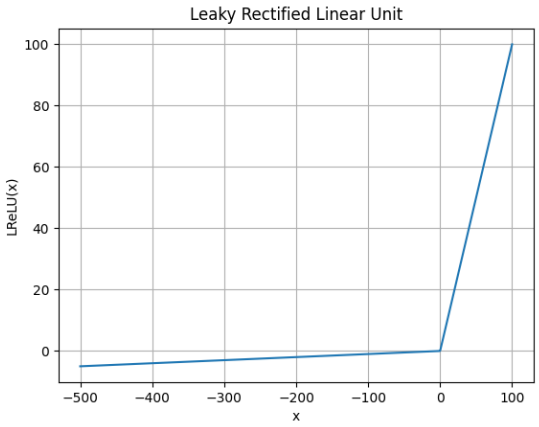


Figure 2.5: Leaky ReLU

Hyperbolic Tangent (tanh):

$$\tanh(x) := \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$\tanh'(x) = 1 - \tanh^2(x)$$

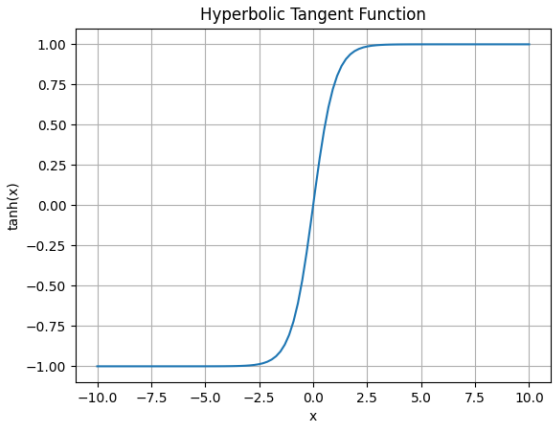


Figure 2.6: Hyperbolic Tangent Function

Softmax Function The softmax function is often used in multi-class classification problems. The softmax function takes as input a vector of real numbers, often referred to as logits or scores (the output of neural network), and produces an output that represents the probability of each class. The softmax function is a crucial component of the output layer in neural networks when the goal is to perform multi-class classification.

The softmax function's output for an input logit $z = \{z_1, z_2, \ldots, z_n\}$ is as follows:

$$\sigma_{\text{SM}}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \tag{2.3}$$

2.7.4 Multilayer Perceptron

A **Multilayer Perceptron (MLP)** is a versatile fully connected neural network architecture with input, hidden, and output layers that can model complex patterns and relationships within data, making it a valuable tool for a wide range of machine learning and deep learning tasks.

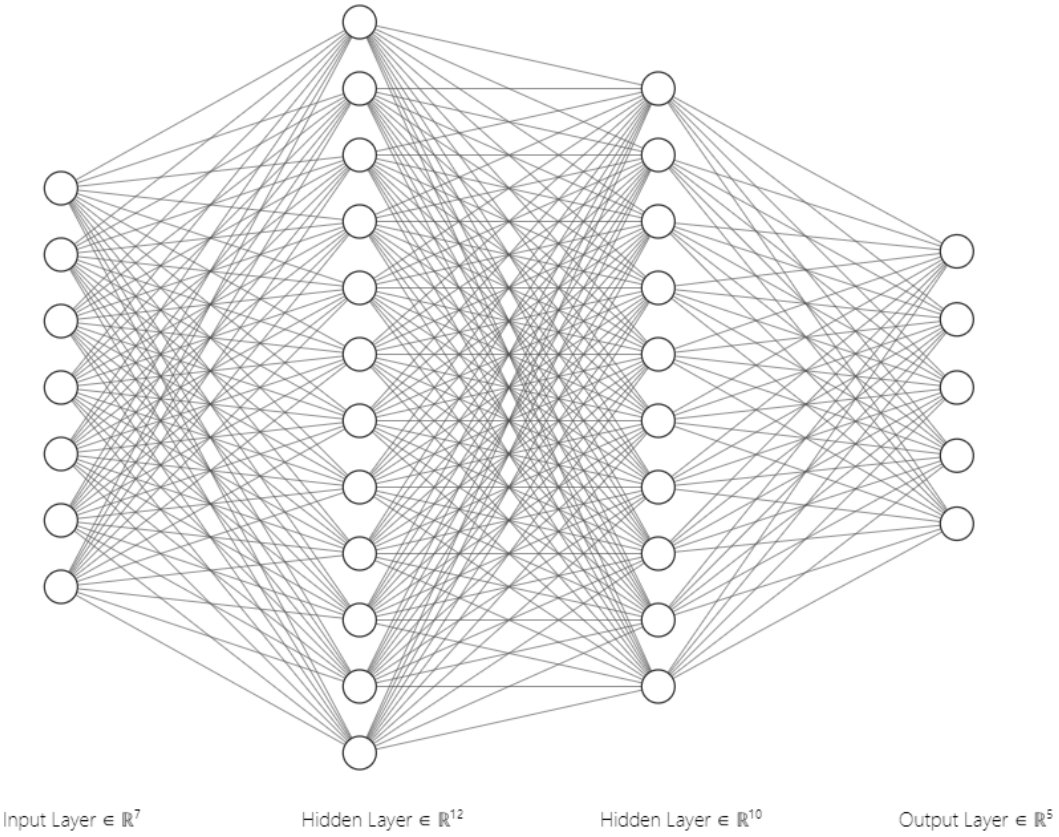


Figure 2.7: 3-Layer MLP [22]

Figure 2.7 shows an example of a MLP with an input layer, an output layer and two hidden layers. When an MLP has multiple hidden layers, it is considered a **Deep Neural Network(DNN)**. Deep learning involves training deep neural networks to automatically learn features and representations from data. Deep MLPs are a fundamental component of deep learning and are used in various applications, such as image and speech recognition, natural language processing, and recommendation systems.

2.8 Convolutional Neural Networks

Convolutional Neural Networks (CNNs), often referred to as ConvNets, are a specialized type of artificial neural network designed for processing and analyzing visual data, particularly images and video. CNNs have revolutionized the field of computer vision and are widely used for tasks like image classification, object detection, facial recognition, and more. Figure 2.8 illustrates the structure of a CNN. In this thesis we use a CNN called Visual Geometry Group (VGG)-16 [23] and comprises multiple convolutional layers and fully connected layers. VGG models have a deep and regular structure with small 3x3 convolutional filters and max-pooling layers. We now discuss the layers that make up a CNN

2.8.1 Convolutional Layers

The core building blocks of CNNs are convolutional layers. Unlike the layers of MLP, these layers apply a series of convolutional filters (also known as kernels) to the input image. Each filter is a small matrix often with dimension 3×3 or 5×5, that slides over the input, performing element-wise multiplications and summations. Each filter is treated as a set of weights, which are learned through the training process. The result of the element-wise multiplications at each position is summed to produce a single value. This process is repeated for each position, generating a two-dimensional array of values. This array is known as a feature map. This operation helps detect local patterns and features in the image, such as edges, textures, and shapes.

Generally, similar to MLP, a non-linear activation function, such as ReLU (Rectified Linear Unit), is applied element-wise to the feature maps after convolution with the aim of introducing non-linearity and allowing the network to capture complex relationships within the data.

2.8.2 Pooling Layers

Pooling layers follow convolutional layers and reduce the spatial dimensions of the feature maps produced by the convolutions. Common pooling techniques include max-pooling and average-pooling. Pooling helps reduce computational complexity, increase translation invariance, and reduce the risk of overfitting. Just as their name suggests, max-pooling over a region outputs the maximum value in that region whereas average-pooling returns the average value of that region.

2.8.3 Fully Connected Layers

Now that the job of feature extraction is taken care of, after several convolutional and pooling layers, CNNs often have one or more fully connected layers, also known as dense layers. These layers perform the final classification or regression tasks based on the features learned in the previous layers. It is important to note that the convolutional layers and the pooling layers work in unison to be able to learn through the algorithm.

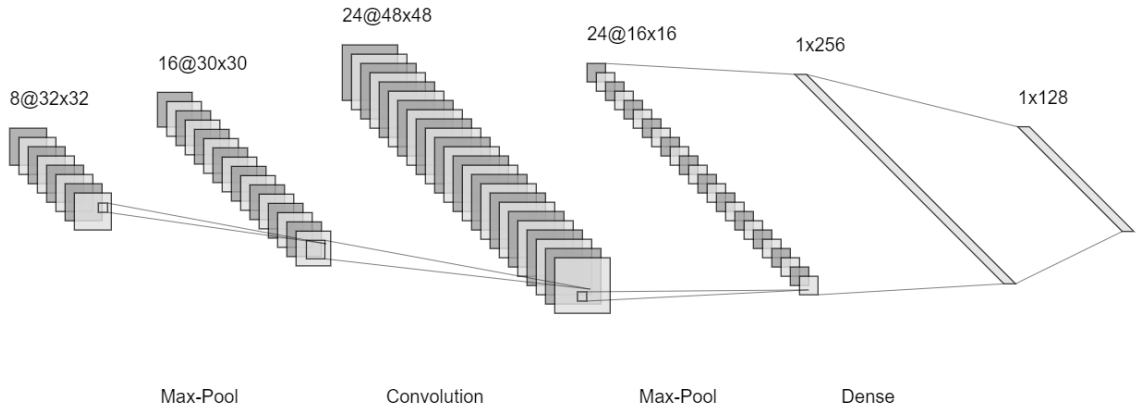


Figure 2.8: A Convolution Neural Network [22]

2.8.4 Residual Networks

Residual Networks [14], often referred to as ResNets, are a type of deep neural network architecture that has had a significant impact on the field of deep learning and computer vision. ResNets were introduced to address the challenges of training very deep neural networks, one such important challenge was the "vanishing gradient" problem. As the name

suggests, it is the problem of deep neural networks where the gradients of the loss function become extremely small as they are backpropagated through many layers. This can hinder the training process and lead to suboptimal or even poor performance in deep networks. They solve this by using a clever design that includes shortcut or skip connections, allowing the network to effectively learn residual functions.

ResNets introduce a fundamental building block called a "residual block." Each residual block contains two main paths for data flow:

Identity Path: This is a shortcut connection where the input data is passed directly to the output of the block without any transformation. It's called the identity path because it's akin to passing the input data unchanged.

Residual Path: This path contains a sequence of convolutional layers and activation functions that aim to learn a residual function. The residual function captures the difference between the desired output and the input.

The main idea of ResNets is to learn the residual(difference) between the desire output and the input data for every layer. And hence, the the neural network focuses on the sum of identity path output and the residual path output.

In this thesis we mainly use ResNet18 [14] and Wide-ResNet [42] (WRN) 22-2 in our experiments, where the numbers 18 and 22 represent the depth (layers) of the neural networks.

Chapter 3

Calibration

3.1 Setup

Consider a K-way classification task. Let $\mathcal{X}$ be the input space and μ denote the true distribution of input instance X over $\mathcal{X}$. Let $S := \{x_1, x_2, \ldots, x_m\}$ denote the set of input instances used for training, which are drawn i.i.d from μ. With a slight abuse of notation, we will also use S to denote the empirical distribution on $\mathcal{X}$ given by S. Any ambiguity should be resolvable from the context. Let$[K] := \{1, 2, \ldots, K\}$ be the set of all class labels. Let $\Delta([K])$ be the space of all probability distributions on $[K]$. A classifier h is a function mapping $\mathcal{X}$ to $\Delta([K])$, which maps any $x \in \mathcal{X}$ to a soft label, or a K-vector $h(x) = [h(x)_1, h(x)_2, \ldots, h(x)_K]$. We will use h_{H} to denote the "hard" version of the classifier h by setting $h_{\mathrm{H}}(x)_y = 1$, where $y = \arg\max_{k \in [K]} h(x)_k$, and $h_{\mathrm{H}}(x)_{y'} = 0$ for all $y' \neq y$. It is clear that $h_{\mathrm{H}}(x) \in \Delta([K])$. For any classifier h, soft or hard, when we use it to assign a label y for instance x, we follow the rule of sampling label y from the distribution $h(x)$. Note that if h is a soft classifier, the standard rule for predicting the label is identical to sampling the label y from $h_{\mathrm{H}}(x)$.

Moving forward, we denote the ground-truth classifier by f and consider it as a hard classifier. For any given classifier h, we define its soft accuracy and hard accuracy as follows:

$$\mathrm{SoftAcc}_{\mu}(h) := \mathbb{E}_{x \sim \mu}\left[\mathbb{P}_{Y \sim f(x), \hat{Y} \sim h(x)}[Y = \hat{Y} | X = x]\right]$$

$$\mathrm{HardAcc}_{\mu}(h) := \mathbb{E}_{x \sim \mu}\left[\mathbb{P}_{Y \sim f(x), \hat{Y} \sim h_{\mathrm{H}}(x)}[Y = \hat{Y} | X = x]\right]$$

In practice, one usually evaluates a classifier by its hard accuracy. Notably, only focusing on

hard accuracy and completely ignoring the soft accuracy discards the fact that the output $h(x)$ of the classifier h may not be a one-hot vector, for example, when h is obtained by training a neural network model.

3.2 Confidence of a Classifier

Before we introduce the Calibration of a classifier, we must understand the confidence of a classifier. Classifier's confidence or prediction's confidence, is a measure of how certain or reliable the classifier is in its predictions. It quantifies the degree of trust or certainty the classifier has in the correctness of its output for a given input. High confidence indicates that the classifier is very sure about its prediction, while low confidence suggests uncertainty or a lack of confidence in the prediction. In the context of classification tasks, classifier's confidence is typically expressed as a probability or a score.

Given a classifier h, one may define a "confidence function" G_h, mapping $\mathcal{X}$ to $[0, 1]$, attempting to indicate the confidence level of prediction based on $h(x)$. The "true-confidence" is another word for the accuracy of the classifier, and the goal of any confidence function is to mimic the true-confidence of the classifier.

3.3 Introduction to Calibration

Calibration refers to the alignment between the predicted confidence scores produced by a DNN and the actual correctness probabilities of those predictions [28], [8]. A well-calibrated model is not only aware of its own uncertainty but also provides reliable estimates of the likelihood that a given prediction is correct. Such calibration is crucial for various downstream tasks, including decision-making systems that rely on model confidence to guide their actions.

3.4 Notions of Calibration

There are several notions of calibration, but they can be segregated based on their quantities of interest. Specifically, **top-class calibration** [13], [32] is concerned with the predicted class with the highest confidence value for a given instance, whereas **class-wise calibration** is top-class calibration concerned with the confidence of all the classes and

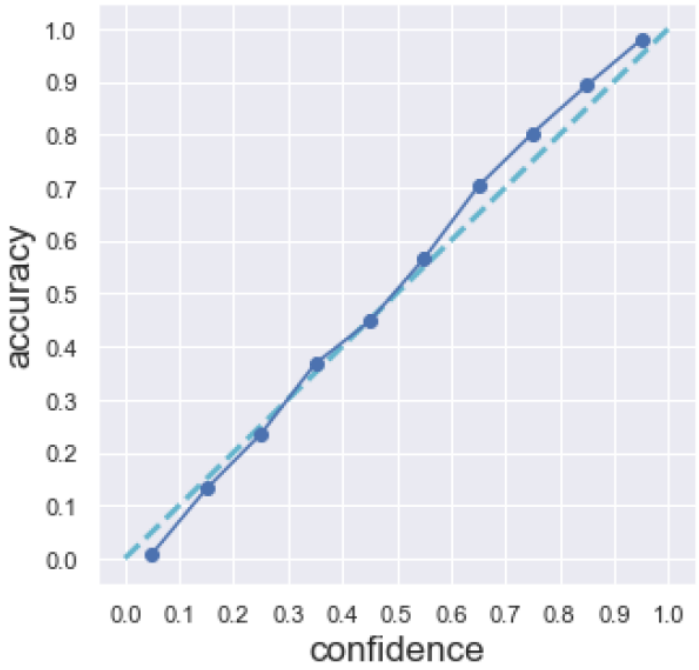


Figure 3.1: A Well-Calibrated model

have been studied under different names [40], [38], [16]. A model h is said to be top-class calibrated if:

$$\mathbb{P}_{Y\sim f(X),\hat{Y}\sim h_{\mathrm{H}}(X)}\left[Y=\hat{Y}|G_h(X)=q\right]=q \tag{3.1}$$

and a model h is said to be class-wise calibrated if, for all $k\in[K]$:

$$\mathbb{P}_{Y\sim f(X)}\left[Y=k|G_h(X)_k=q\right]=q \tag{3.2}$$

for all $q\in[0,1]$ for which the probability on L.H.S is well defined and the confidence function $G_h(x)=[G_h(x)_1,\ldots,G_h(x)_K]$ is a mapping from $\mathcal{X}$ to $\Delta([K])$.

Similarly, when the notion of calibration covers the confidence of all classes on average, it is referred to as **class-aggregated calibration** ([16]) or **static calibration** [32]. A model h is said to be class-aggregated calibrated if:

$$\frac{\sum_{k=1}^{K} \mathbb{P}_{Y \sim f(X)}[Y = k, G_h(X)_k = q]}{\sum_{k=1}^{K} \mathbb{P}[G_h(X)_k = q]} = q \tag{3.3}$$

for all $q \in [0, 1]$ for which the probability on L.H.S is well defined.

Apart from the calibration notions mentioned above, a calibration notion that is stricter than the ones discussed is that of **strong calibration** [37]; [36], which implies that one can fully trust the output probabilities of any class for any given instance. A model h is said to be strongly calibrated if, for all $k \in [K]$:

$$\mathbb{P}_{Y \sim f(x)} [Y = k | h(X)] = h(X)_k$$

The difference in these notions arises from the confidence function or the confidence estimation method being used.

3.5 Methods of Calibration

Moving on to the methods of calibration, these methods can be divided into two categories: pre-processing methods that involve tweaking the model before and during its training to achieve a better-calibrated model, and post-processing methods that modify the model after it has finished training. Pre-processing methods such as label smoothing [34], data-augmentation [43], calibration layers [43], and focal loss minimization [27] face a slight disadvantage when compared to post-processing methods in terms of computational costs, whereas post-processing methods have been the center of attention for a long time, from pre-DNN era methods such as Platt scaling [33], histogram binning [40], and isotonic regression [41] to newer methods such as temperature scaling [13], disagreement [16], and p-NormSoftmax [7].

Generally, post-processing methods require a hold-out validation set, which in practice can be the same set used for hyperparameter tuning, it is also assumed that the training, validation, and testing set are drawn from the same distribution. Moreover, the goal of a post-processing method is to generate calibrated probabilities. For example, consider Temperature scaling (TS) that is a simple extension of Platt scaling. TS uses a single scalar parameter $T > 0$ for all classes $k \in K$, and for a given logit vector z_i, the confidence assigned to that logit $\hat{q}_i$ is:

$$\hat{q}_i = \max_{k \in [K]} \sigma_{\text{SM}}(z_i/T)_k$$

where σ_{SM} is the softmax function as defined in equation 2.3.

3.6 Evaluation

There are several measures that are used to evaluate the extent to which a model is calibrated. For all the notions mentioned above, a natural measure is to compute the absolute difference between L.H.S and R.H.S expected over all possible values of $q \in [0, 1]$ for which the probability in L.H.S is well-defined. For instance, the calibration error of a model h corresponding to a top-class calibration would be:

$$\int_q \left| q - \mathbb{P}_{Y \sim f(X), \hat{Y} \sim h_{\mathrm{H}}(X)} \left[Y = \hat{Y} | G_h(X) = q \right] \right| \, dq$$

Or,

$$\mathbb{E}_{q \sim P_G} \left[\left| \mathbb{P}(Y = \hat{Y} | G_h(X) = q) - q \right| \right]$$

where, P_G is the *p.d.f* of G_h.

Since we don't have access to the true distribution μ, we can only compute the above quantity over a finite set. **Expected Calibration Error (ECE)** [29] is a widely used technique that approximately measures the top-class calibration error in expectation by discreetly splitting the probability intervals into a fixed number of bins, and assigning each bin the probability value that encompasses it. Specifically,

$$\mathrm{ECE} := \sum_{b=1}^{B} \frac{n_b}{N} \left| \mathrm{HardAcc}_b(h) - \mathrm{Conf_h(b)} \right| \tag{3.4}$$

where n_b is the number of predictions in bin b, N is the total number of data points, and $\mathrm{HardAcc}_b(h)$ and $\mathrm{Conf_h(b)}$ are the expected hard accuracy and the assigned confidence of bin b.

In high-stakes scenarios demanding dependable confidence assessments, it becomes essential to reduce the maximum potential disparity between confidence levels and actual accuracy. In such cases, the **Maximum Calibration Error (MCE)** [29] is used. MCE estimates an upper bound on the deviation $|\mathrm{HardAcc}_b(h) - \mathrm{Conf_h(b)}|$ across all $b \in [B]$, i.e.

$$\mathrm{MCE} := \max_{b \in [B]} \left| \mathrm{HardAcc}_b(h) - \mathrm{Conf_h(b)} \right| \tag{3.5}$$

Nixon *et al* [32] argues that measures such as ECE and MCE are reductive in a multi-class setting as they are computed only over the predicted class's confidence. Moreover, since most of the DNNs are overconfident [10], [31], [13], the use of equally spaced bins

results in a large leftward skew in the output probabilities, with the left region being sparsely populated and the right region densely populated in a plot of bins vs accuracy. Therefore they introduce **Static Calibration Error (SCE)** as an extension of ECE. SCE can also be viewed as the class-aggregated calibration error. It is defined as:

$$\text{SCE} = \frac{1}{K} \sum_{k=1}^{K} \sum_{b=1}^{B} \frac{n_{bk}}{N} \left| \text{acc}(b,k) - \text{conf}(b,k) \right| \tag{3.6}$$

where $\text{acc}(b,k)$ and $\text{conf}(b,k)$ are the accuracy and confidence of bin b for class label k, respectively; n_{bk} is the number of predictions in bin b for class k, and N is total number of predictions.

Chapter 4

GDE and Generalized GDE (GGDE)

4.1 Generalization Disagreement Equality (GDE)

We now take a slight digression to elaborate on the findings of Jiang *et al* [16] and its implications on the calibration of a DNN.

Before we can define GDE, we must first define an "ensemble". Let $\mathcal{H}_{\mathcal{A}}$ be a distribution over the space of all classifiers that characterize the output of a stochastic training algorithm $\mathcal{A}$. That is, the output of $\mathcal{A}$ is regarded as being drawn from $\mathcal{H}_{\mathcal{A}}$. For the purpose of stating GDE, we restrict $\mathcal{H}_{\mathcal{A}}$ to be a distribution over hard classifiers.

We define classifier $\tilde{h}$ by aggregating the random classifier ensemble specified by $\mathcal{H}_{\mathcal{A}}$ as follows:

$$\tilde{h}(x)_k = \mathbb{E}_{h\sim\mathcal{H}_{\mathcal{A}}}\left[\mathbb{P}_{Y\sim h(x)}(Y=k)\right]$$

We define Agreement for any classifier h and h' on a given instance $x \in \mathcal{X}$:

$$\text{Agree}(h(x), h'(x)) := \mathbb{P}_{Y\sim h(x), Y'\sim h'(x)}[Y = Y']. \tag{4.1}$$

From the above equation, it is easy to realize that $\text{Agree}(h(x), f(x))$ is the accuracy of the model h for instance x. Moreover, it is easy to realize that agreement when expected over all pairs of models $h, h' \in \mathcal{H}_{\mathcal{A}}$, is the agreement of the average model (ensemble model) $\tilde{h}$ with itself, i.e.

$$\mathbb{E}_{h,h'\sim\mathcal{H}_{\mathcal{A}}}\left[\text{Agree}(h(x), h'(x))\right] = \text{Agree}(\tilde{h}(x), \tilde{h}(x))$$

Proof:

$$\begin{aligned}
\mathbb{E}_{h,h'\sim\mathcal{H}_{\mathcal{A}}}\left[\text{Agree}(h(x),h'(x))\right] &= \mathbb{E}_{h,h'\sim\mathcal{H}_{\mathcal{A}}}\left[\mathbb{P}_{Y\sim h(x).\hat{Y}\sim h'(x)}[Y=\hat{Y}]\right] \\
&= \mathbb{E}_{h,h'\sim\mathcal{H}_{\mathcal{A}}}\left[\sum_{k=1}^{K}\mathbb{P}_{Y\sim h(x)}[Y=k].\mathbb{P}_{\hat{Y}\sim h'(x)}[\hat{Y}=k]\right] \\
&= \sum_{k=1}^{K}\mathbb{E}_{h\sim\mathcal{H}_{\mathcal{A}}}\left[\mathbb{P}_{Y\sim h(x)}[Y=k]\right]\mathbb{E}_{h'\sim\mathcal{H}_{\mathcal{A}}}\left[\mathbb{P}_{\hat{Y}\sim h'(x)}[\hat{Y}=k]\right] \\
&= \sum_{k=1}^{K}\mathbb{P}_{Y\sim\tilde{h}(x)}[Y=k].\mathbb{P}_{\hat{Y}\sim\tilde{h}(x)}[\hat{Y}=k] \\
&= \mathbb{P}_{Y,\hat{Y}\sim\tilde{h}(x)}[Y=\hat{Y}] \\
&= \text{Agree}(\tilde{h}(x),\tilde{h}(x))
\end{aligned}$$

□

We say that the stochastic algorithm $\mathcal{A}$ satisfies the **Generalization Disagreement Equality (GDE)** on μ if

$$\mathbb{E}_{h,h'\sim\mathcal{H}_{\mathcal{A}}}\left[\mathbb{E}_{x\sim\mu}\left[\text{Agree}(h_{\text{H}}(x),h'_{\text{H}}(x))\right]\right] = \text{HardAcc}_{\mu}(\tilde{h}) \tag{4.2}$$

It is important to note that [16] defines GDE for "hard classifiers". In other words, an algorithm $\mathcal{A}$ satisfies GDE when the average hard agreement amongst the ensembles is equal to the average hard accuracy of the ensembles; when averaged over all data points.

Jiang *et al* [16] point out that even a pair of deep neural networks/models (ensemble of size two), obtained through standard training (minimizing cross-entropy loss), satisfy GDE (See Figure 4.1).

To show GDE holds empirically, they observed the average disagreement and average error values for several models on several datasets. Each variation inside this model-dataset matchup has a unique hyperparameter configuration, and for each configuration, they train two copies of models which can further be categorized based on the source of stochasticity. They use four sources of stochasticity namely, DiffInit, DiffOrder, DiffData and AllDiff. In summary, DiffInit stands for models having random initialization, DiffOrder stands for models seeing the training data in a different order, DiffData stands for models seeing different (disjoint) training data and when all of the above conditions are applied together it is denoted as AllDiff.

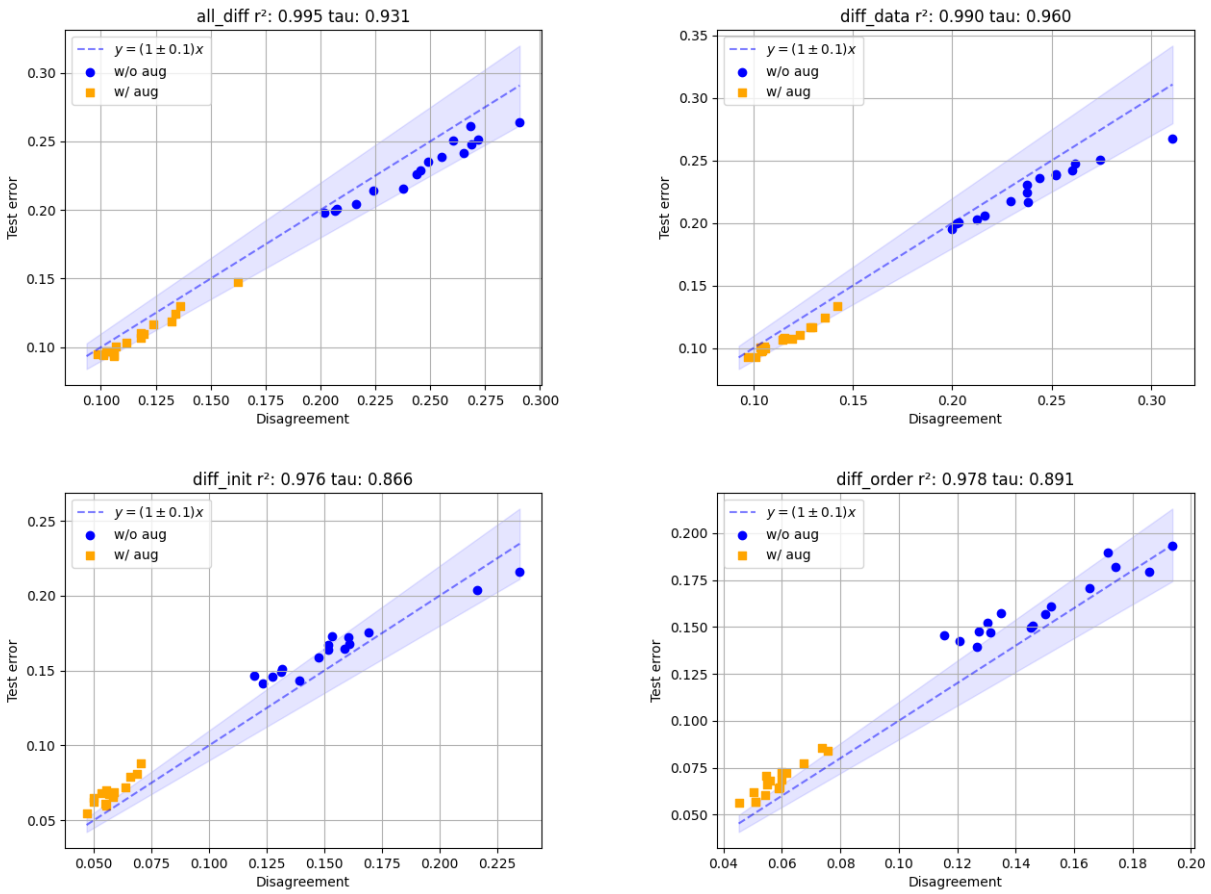


Figure 4.1: GDE on Resnet18-CIFAR10

4.2 Calibration for GDE

A natural question would be to ask: what conditions give rise to the GDE phenomenon? Jiang *et al* answer this question by stating that the ensemble is well-calibrated and provides formulations of some calibration conditions that may lead to GDE. They formulate two notions of calibration, namely class-wise and class-aggregated calibration.

The ensemble model $\tilde{h}$ satisfies **class-wise calibration** (CWC) on μ if for any confidence value $q \in [0, 1]$ and for any class $k \in [K]$,

$$\mathbb{P}[Y = k|\tilde{h}(X)_k = q] = q$$

The ensemble model $\tilde{h}$ satisfies **class-aggregated calibration (CAC)** on μ if for each $q \in [0, 1]$,

$$\frac{\sum_{k=1}^{K} \mathbb{P}_{Y \sim f(X)}[Y = k, \tilde{h}(X)_k = q]}{\sum_{k=1}^{K} \mathbb{P}[\tilde{h}(X)_k = q]} = q$$

They state that these calibration conditions are sufficient but not necessary for GDE to hold.

4.3 GGDE

In this thesis, we extend on GDE phenomenon. We found that GDE is satisfied irrespective of the type of classifiers (soft or hard) that constitute the ensemble (See Section 4.5). We define this phenomenon as Generalized Generalization Disagreement Equality (GGDE) and remove the restriction on $\mathcal{H}_{\mathcal{A}}$ to be a distribution only over hard classifiers.

Definition 4.1. *We say that the stochastic algorithm $\mathcal{A}$ satisfies the **Generalized GDE, or GGDE** on μ if*

$$\begin{aligned} \mathbb{E}_{h,h' \sim \mathcal{H}_{\mathcal{A}}} \left[\mathbb{E}_{x \sim \mu} \left[\text{Agree}(h(x), h'(x)) \right] \right] &= \text{SoftAcc}_{\mu}(\tilde{h}) \\ &= \mathbb{E}_{h \sim \mathcal{H}_{\mathcal{A}}} \left[\mathbb{E}_{x \sim \mu} \left[\text{Agree}(h(x), f(x)) \right] \right] \end{aligned} \tag{4.3}$$

Similar to [16], we believe that this result is due to the well-calibrated nature of ensembles. Now, we provide a condition, in the same virtue of [16], under which GGDE holds.

Definition 4.2. *We say that the ensemble $\mathcal{H}_{\mathcal{A}}$ satisfies Generalized Class Aggregated Calibration (GCAC) on μ if for each $q \in [0, 1]$,*

In a nutshell, the GCAC condition simply requires the random variables "Agreement" and "Accuracy" have the same distribution. This is in fact also what the "Class Aggregated Calibration" of [16] dictates, put in the simplest form.

Proof:

$$\frac{\sum_k \mathbb{P}_{x\sim\mu, Y\sim f(x)}[Y=k, \mathbb{P}_{\hat{Y}\sim\tilde{h}(x)}[\hat{Y}=k]=q]}{\sum_k \mathbb{P}_{x\sim\mu}[\mathbb{P}_{\hat{Y}\sim\tilde{h}(x)}[\hat{Y}=k]=q]} = q$$

$$\sum_k \mathbb{P}_{x\sim\mu, Y\sim f(x)}[Y=k, \mathbb{P}_{\hat{Y}\sim\tilde{h}(x)}[\hat{Y}=k]=q] = q.\sum_k \mathbb{P}_{x\sim\mu}[\mathbb{P}_{\hat{Y}\sim\tilde{h}(x)}[\hat{Y}=k]=q]$$

$$\mathbb{P}_{x\sim\mu}[\mathbb{P}_{\hat{Y}\sim\tilde{h}(x), Y\sim f(x)}[\hat{Y}=Y]=q] = \sum_k q.\mathbb{P}_{x\sim\mu}[\mathbb{P}_{\hat{Y}\sim\tilde{h}(x)}[\hat{Y}=k]=q]$$

$$\mathbb{P}[\text{Agree}(\tilde{h}(X), f(X))=q] = \sum_k \mathbb{P}_{x\sim\mu, \hat{Y}'\sim\tilde{h}(x)}[\hat{Y}'=k|\mathbb{P}_{\hat{Y}\sim\tilde{h}(x)}[\hat{Y}=k]=q].\mathbb{P}_{x\sim\mu}[\mathbb{P}_{\hat{Y}\sim\tilde{h}(x)}[\hat{Y}=k]=q]$$

$$\mathbb{P}[\text{Agree}(\tilde{h}(X), f(X))=q] = \sum_k \mathbb{P}_{x\sim\mu, \hat{Y}'\sim\tilde{h}(x)}[\hat{Y}'=k, \mathbb{P}_{\hat{Y}\sim\tilde{h}(x)}[\hat{Y}=k]=q]$$

$$\mathbb{P}[\text{Agree}(\tilde{h}(X), f(X))=q] = \mathbb{P}_{x\sim\mu}[\mathbb{P}_{\hat{Y}\sim\tilde{h}(x), \hat{Y}'\sim\tilde{h}(x)}[\hat{Y}=\hat{Y}']=q]$$

$$\mathbb{P}[\text{Agree}(\tilde{h}(X), f(X))=q] = \mathbb{P}[\text{Agree}(\tilde{h}(X), \tilde{h}(X))=q]$$

Which is the definition of GCAC. □

Theorem 4.1. *If $\mathcal{H}_{\mathcal{A}}$ satisfies GCAC, then GGDE holds.*

Proof: By definition, GCAC means that "Agreement" and "Accuracy" as Random Variables of X have the same *p.d.f*, which implies that they have the same mean and are equal in expectation over X, i.e.

$$\mathbb{E}_{x\sim\mu}\left[\text{Agree}(\tilde{h}(x), \tilde{h}(x))\right] = \text{SoftAcc}_{\mu}(\tilde{h}) = \mathbb{E}_{x\sim\mu}\left[\text{Agree}(\tilde{h}(x), f(x))\right] \tag{4.4}$$

□

4.4 Experimental Setup

We performed experiments similar to those in [16] and show that GGDE indeed holds closely in neural network models trained with SGD. Results can be found in Section 4.5. We report our results on GGDE using ResNet18 (ImgNet version) ([14]), WideResNet WRN-22(depth)-2(widen-factor) with 0.0 dropout ([42]) and CIFAR-VGG16-BN ([23]). Three datasets are used in our experiments. CIFAR-10 , CIFAR-100 ([17]) and SVHN ([30]).

Each variation inside the network-dataset matchup has a unique hyperparameter configuration, and for each configuration, we train two copies of models which can further be categorized based on the stochasticity introduced as in [16]. In summary, DiffInit stands for models having random initialization, DiffOrder stands for models seeing the training data in a different order, DiffData stands for models seeing different (disjoint) training data and when all of the above conditions are applied together it is denoted as AllDiff.

For CIFAR-10 and CIFAR-100 we use the following hyperparameter configurations:

1. initial learning rate: { 0.1, 0.05 }
2. weight decay { 0.0, 0.0001 }
3. batch size { 100, 200 }
4. data augmentation { No, Yes }

For the SVHN dataset we use a different set of hyperparameter configurations as the models are trained only on 5% of the training dataset available.

1. initial learning rate: { 0.01, 0.005 }
2. weight decay { 0.0, 0.0001 }
3. batch size { 64, 128 }
4. data augmentation { No, Yes }

Data augmentation used involves RandomCrop of size 32 and padding 4, followed by RandomHorizontalFlip. All models are trained using SGD with a momentum of 0.9. The learning rate decays 10× every 50 epochs. Training stops when training accuracy reaches 100% or the number of epochs exceeds 200. The plots contain information about correlation using r^2 correlation coefficient and Kendall's Ranking coefficient (tau).

4.5 Results

Figures 4.2 - 4.10 show that GGDE holds closely for 9 different network-dataset configurations each having 4 different sources of stochasticity and each source of stochasticity having 16 different ensemble configurations. The shaded region is used to highlight the closeness of soft disagreement between these network pairs (ensembles of size 2) with the respective soft test error, and hence these shaded regions are not the same throughout these results.

A natural question that one may raise is how much deviation from $y = x$ in these graphs is permissible to be able to state that GGDE holds. To answer this question, we devise a hypothesis testing setup which allows us to investigate the acceptable amount of deviation for GGDE with statistical significance. This hypothesis testing setup is inspired from the statistical t-test and is as follows:

- Null Hypothesis H_0: Error falls outside $[(1-\alpha)\text{Disagreement}, (1+\alpha)\text{Disagreement}]$
- Let $\mathcal{H}_{\mathcal{A}}$ be a distribution over the space of all classifiers that characterize the output of a stochastic training algorithm $\mathcal{A}$.
- Let Err and Dis denote the random variables (R.V.s) for Error and Disagreement respectively, having the distribution $\mathcal{H}_{\mathcal{A}}$.
- Define R.V. $C = Err/Dis$.
- We compute $c_i = err_i/dis_i$, for all $i \in [1, 2, \ldots, n]$, for the sample of size n.
- Let $\bar{c} = \frac{\sum_{i=1}^{n} c_i}{n}$ be the sample mean and $\sigma = \sqrt{\frac{\sum_i (c_i - \bar{c})^2}{n-1}}$ be the sample standard deviation.
- Define R.V. $T = \frac{C-1}{\sigma/\sqrt{n}}$.
- Then $H_0 \Leftrightarrow C \geq 1 + \alpha$ or $C \leq 1 - \alpha$.
- OR $H_0 \Leftrightarrow T \geq \frac{\alpha}{\sigma/\sqrt{n}}$ or $T \leq \frac{-\alpha}{\sigma/\sqrt{n}}$.
- Assuming that T follows Student's t-distribution
- $\mathbb{P}[H_0] = \mathbb{P}[T \geq \frac{\alpha}{\sigma/\sqrt{n}}] + \mathbb{P}[T \leq \frac{-\alpha}{\sigma/\sqrt{n}}]$
- $\mathbb{P}[H_0] = 2 * \mathbb{P}[T \geq \frac{\alpha}{\sigma/\sqrt{n}}]$

Table 4.1: p-values for two-tailed t-test with 10% deviation ($\alpha = 0.1$)

Networks	CIFAR-10				SVHN(5%)				CIFAR-100			
	all_diff	diff_data	diff_order	diff_init	all_diff	diff_data	diff_order	diff_init	all_diff	diff_data	diff_order	diff_init
Resnet18	4.10e-11	1.81e-10	6.83e-06	2.74e-06	3.05e-13	1.18e-13	0.0007	0.0035	3.27e-15	3.34e-14	1.16e-05	3.37e-05
WRN 22-2	4.15e-11	3.57e-11	3.70e-10	5.63e-12	5.01e-12	1.51e-11	8.15e-05	0.0003	6.02e-15	3.67e-16	6.00e-13	1.99e-12
CIFAR-VGG16-BN	1.88e-11	1.24e-11	6.14e-10	1.05e-10	5.86e-09	8.66e-09	1.17e-05	0.0001	1.09e-14	1.57e-14	2.00e-13	5.16e-13

Table 4.2: p-values for two-tailed t-test with 5% deviation ($\alpha = 0.05$)

Networks	CIFAR-10				SVHN(5%)				CIFAR-100			
	all_diff	diff_data	diff_order	diff_init	all_diff	diff_data	diff_order	diff_init	all_diff	diff_data	diff_order	diff_init
Resnet18	4.93e-07	1.78e-06	0.0042	0.0024	5.83e-09	2.40e-09	0.0529	0.1045	7.95e-11	7.29e-10	0.0058	0.0107
WRN 22-2	4.98e-07	4.37e-07	3.26e-06	8.44e-08	7.60e-08	2.04e-07	0.0173	0.0334	1.42e-10	9.62e-12	1.09e-08	3.29e-08
CIFAR-VGG16-BN	2.48e-07	1.71e-07	4.98e-06	1.12e-06	3.09e-05	4.19e-05	0.0059	0.0197	2.52e-10	3.57e-10	3.93e-09	9.49e-09

Table 4.3: p-values for two-tailed t-test with 2.5% deviation ($\alpha = 0.025$)

Networks	CIFAR-10				SVHN(5%)				CIFAR-100			
	all_diff	diff_data	diff_order	diff_init	all_diff	diff_data	diff_order	diff_init	all_diff	diff_data	diff_order	diff_init
Resnet18	0007	0.0018	0.1135	0.0891	3.08e-05	1.52e-05	0.3100	0.4012	8.77e-07	5.74e-06	0.1296	0.1663
WRN 22-2	0.0008	0.0007	0.0027	0.0002	0.0002	0.0004	0.2012	0.2600	1.45e-06	1.36e-07	5.02e-05	0.0001
CIFAR-VGG16-BN	0.0004	0.0003	0.0035	0.0013	0.0102	0.0121	0.1298	0.2115	2.36e-06	3.16e-06	2.25e-05	4.51e-05

The test above is a two-tailed t-test and It will allow us to reject the null hypothesis for a given value of deviation (α) with 95% significance if the p-value of this test is below 0.05.

Results in Tables 4.1,4.2 and 4.3 allow us to accept/reject the null hypothesis that Error falls outside the interval $[(1-\alpha)\text{Disagreement}, (1+\alpha)\text{Disagreement}]$ for $\alpha \in [0.1, 0.05, 0.025]$ respectively.

Rejecting the null hypothesis for a given value of α means that we "reject" that Error falls outside the interval $[(1-\alpha)\text{Disagreement}, (1+\alpha)\text{Disagreement}]$ for that α and the corresponding p-value indicates the probability of null hypothesis holding.

- For $\alpha = 0.1$ we reject the null hypothesis 36/36 times.
- For $\alpha = 0.05$ we reject the null hypothesis 34/36 times.
- For $\alpha = 0.025$ we reject the null hypothesis 26/36 times.

Therefore with a statistical significance of at least 95% we can say that Error lies within 10% deviation of disagreement for all the test-cases; Error lies within 5% deviation for 34/36 test-cases and within 2.5% deviation for 26/36 test-cases.

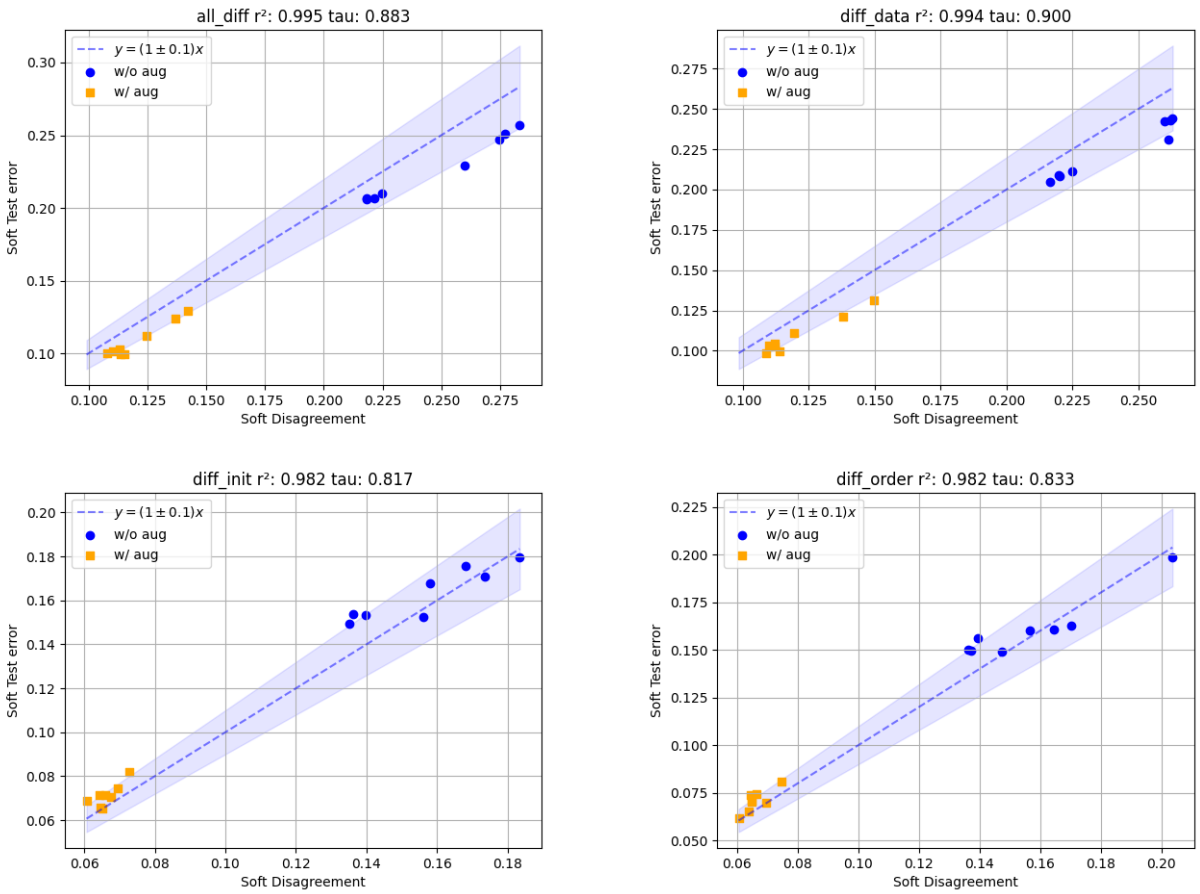


Figure 4.2: Resnet18-Cifar10

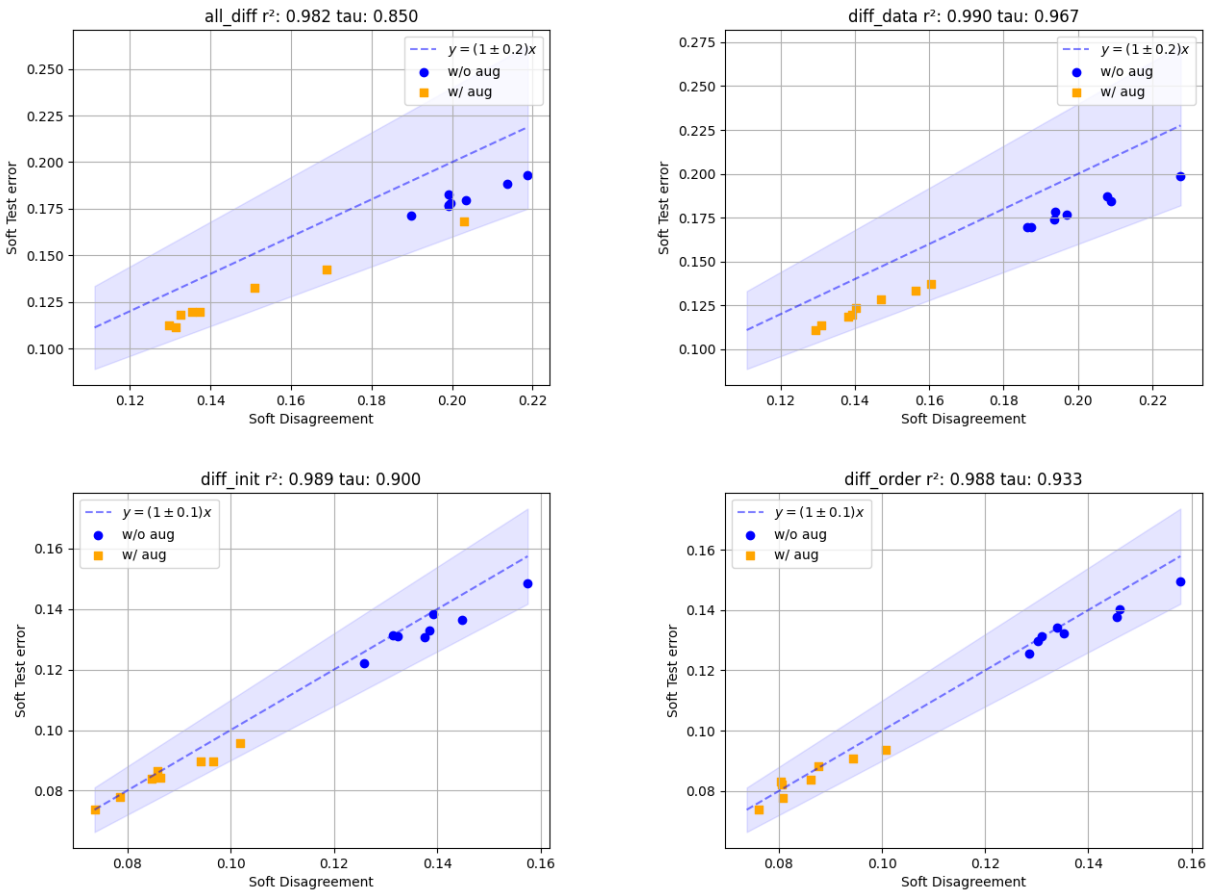


Figure 4.3: VGG16-Cifar10

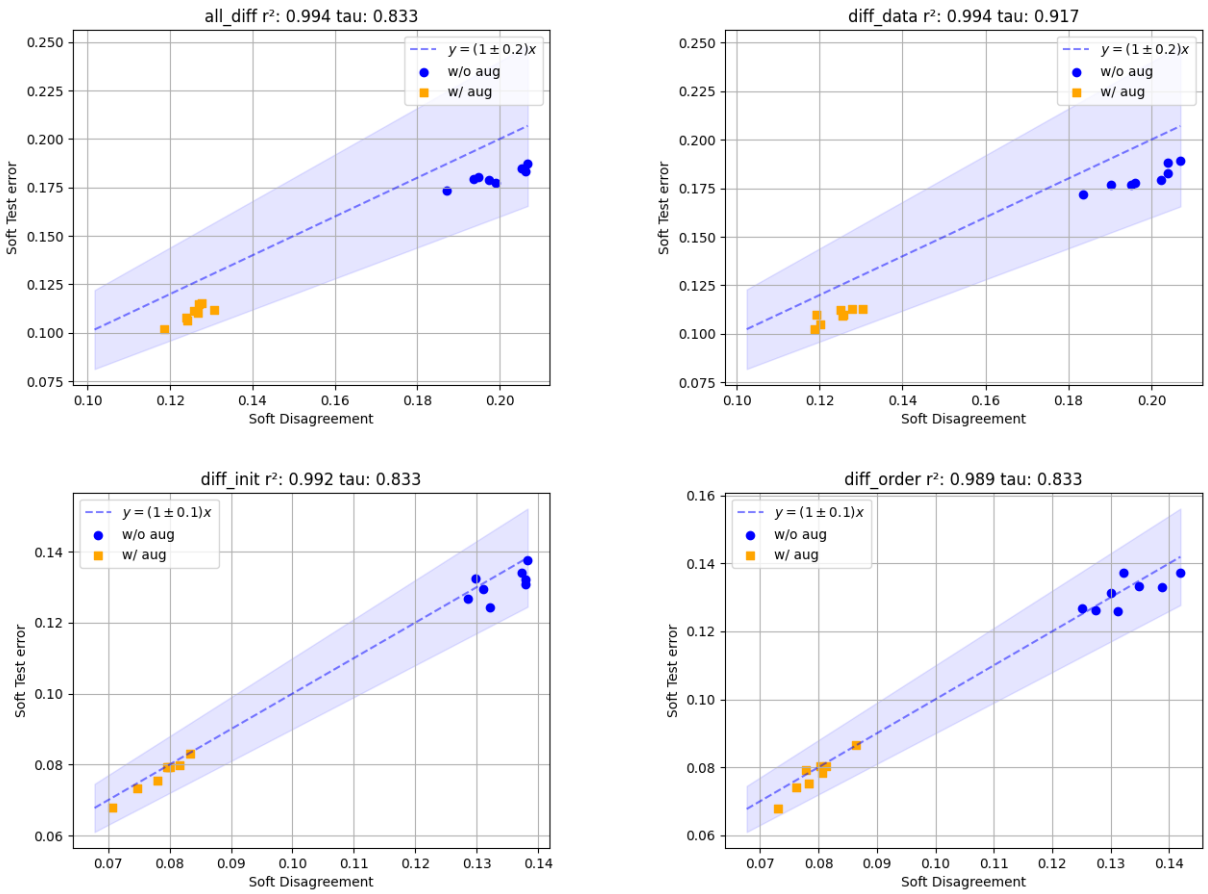


Figure 4.4: WRN(22-2)-Cifar10

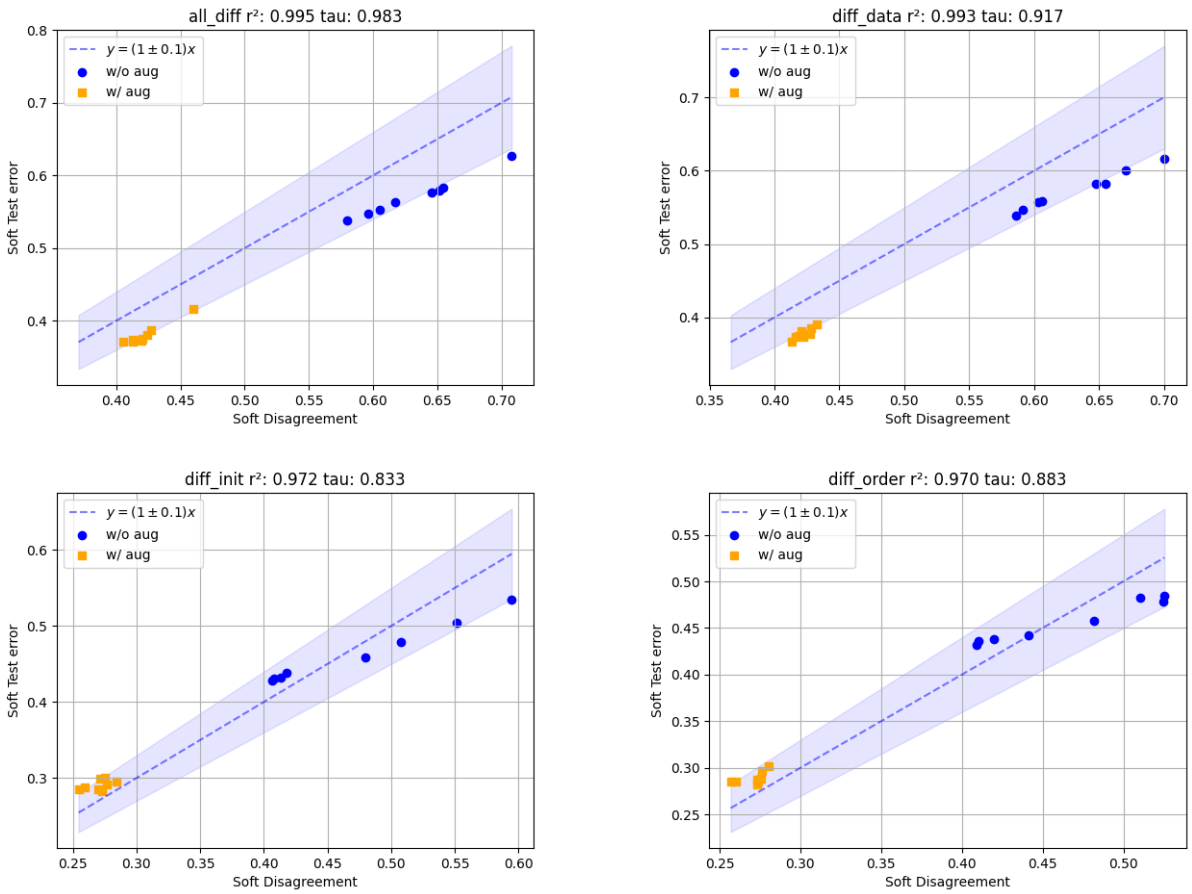


Figure 4.5: Resnet18-Cifar100

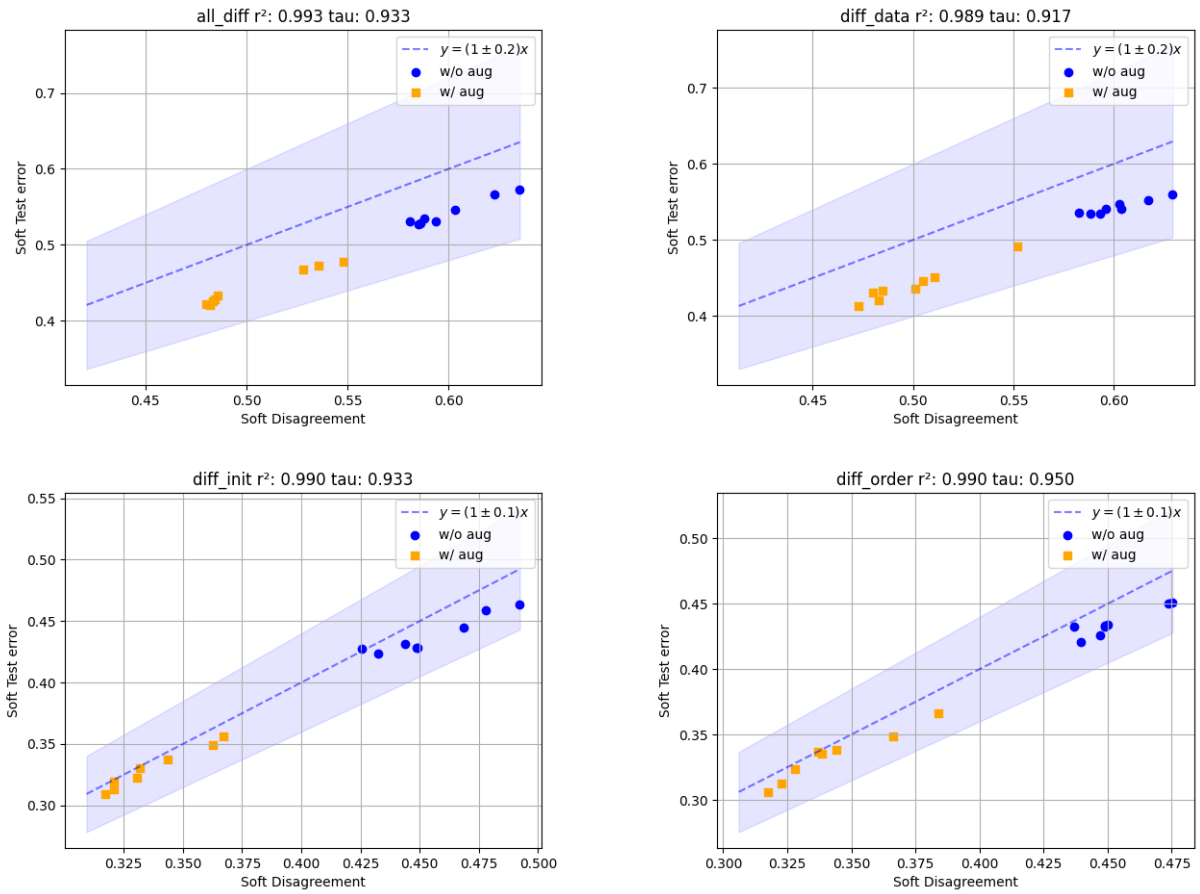


Figure 4.6: VGG16-Cifar100

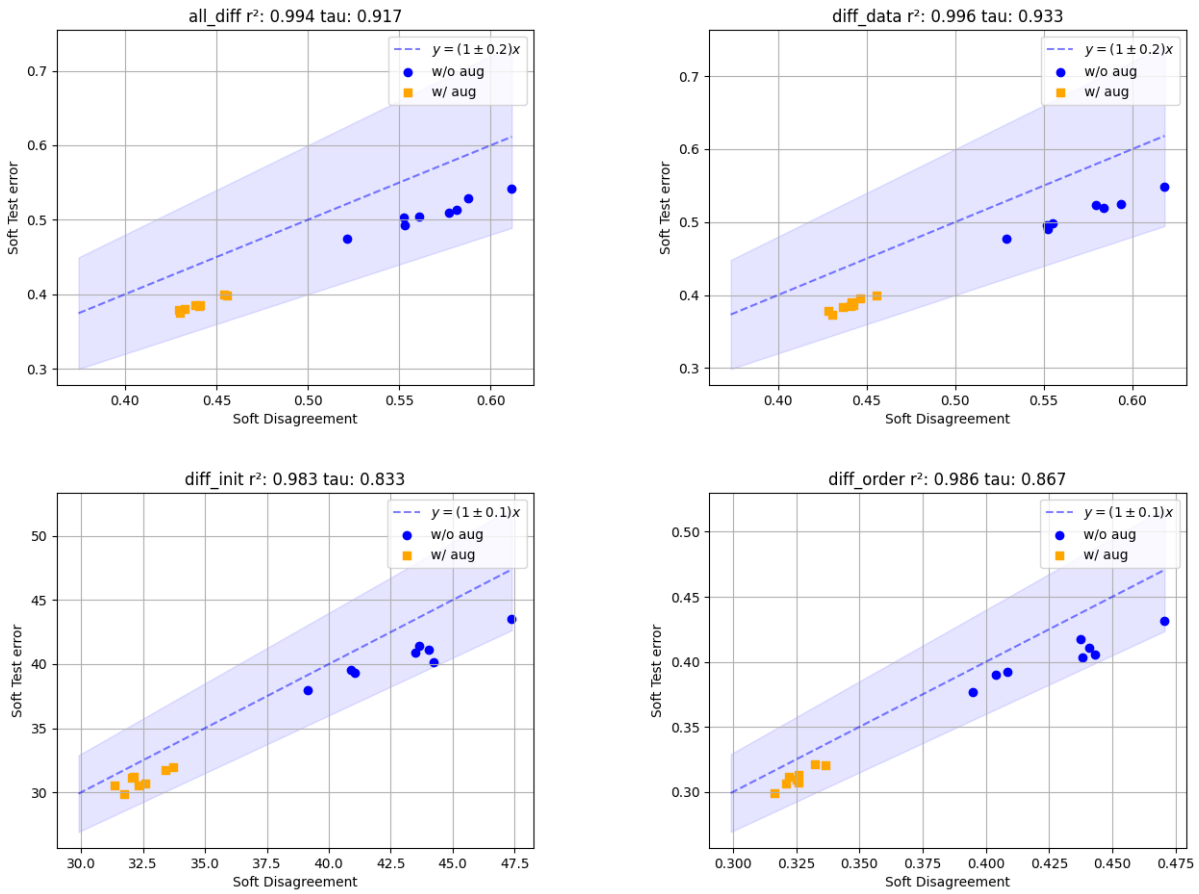


Figure 4.7: WRN(22-2) CIFAR-100

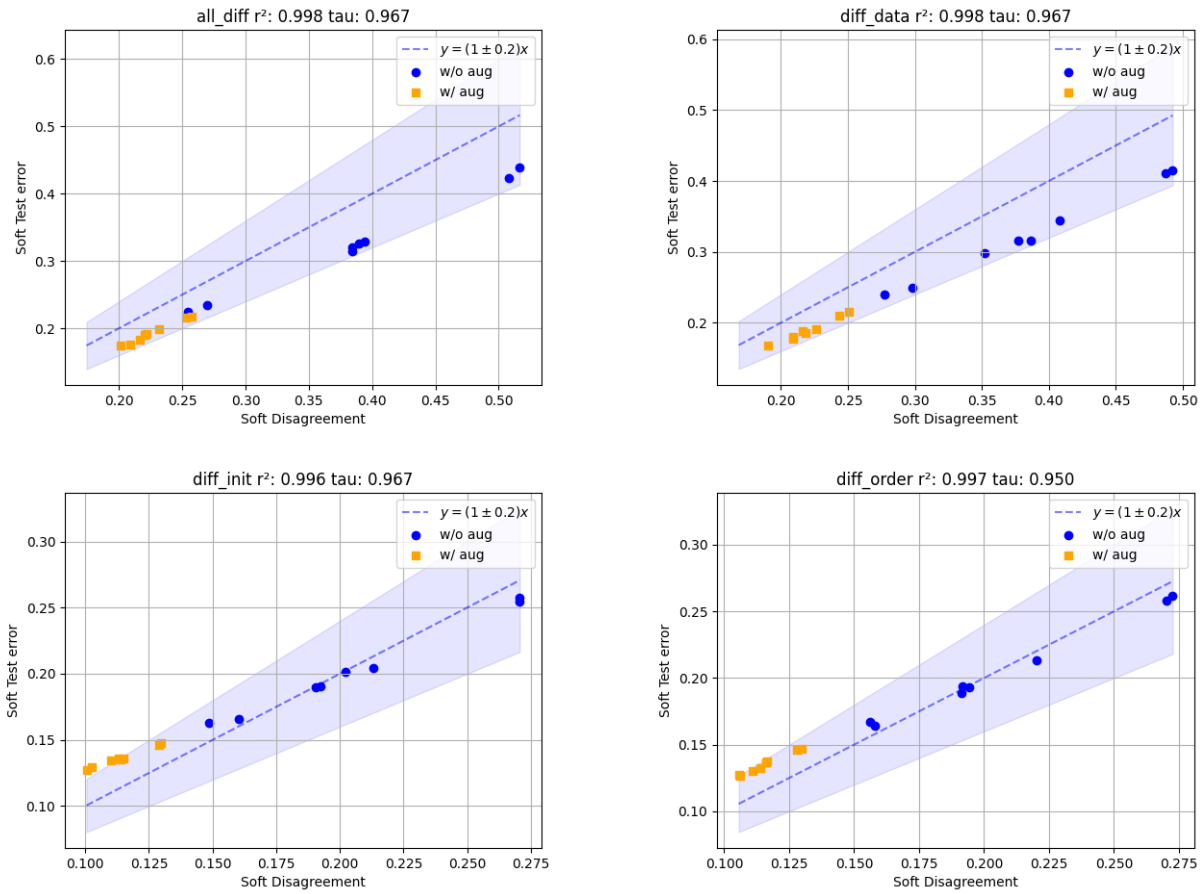


Figure 4.8: Resnet18-SVHN(5%)

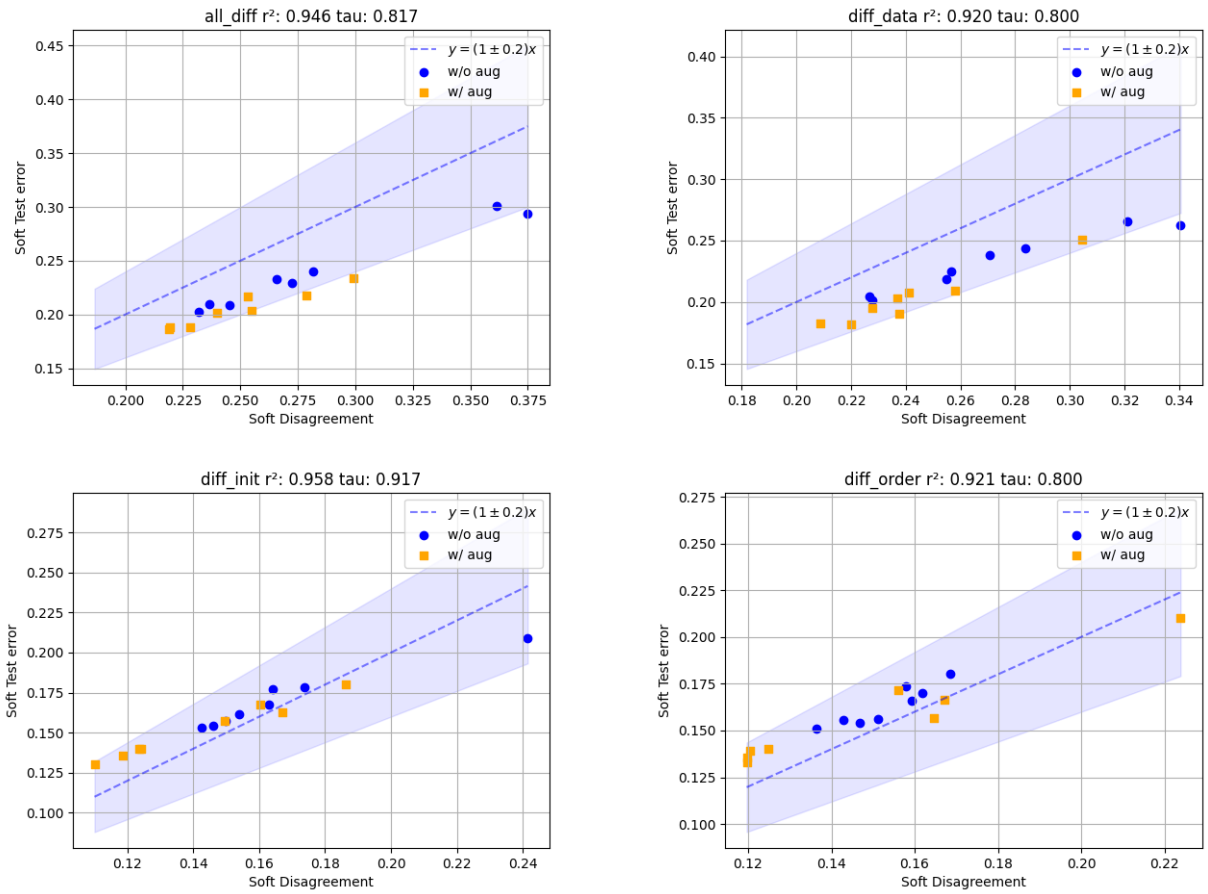


Figure 4.9: VGG16-SVHN(5%)

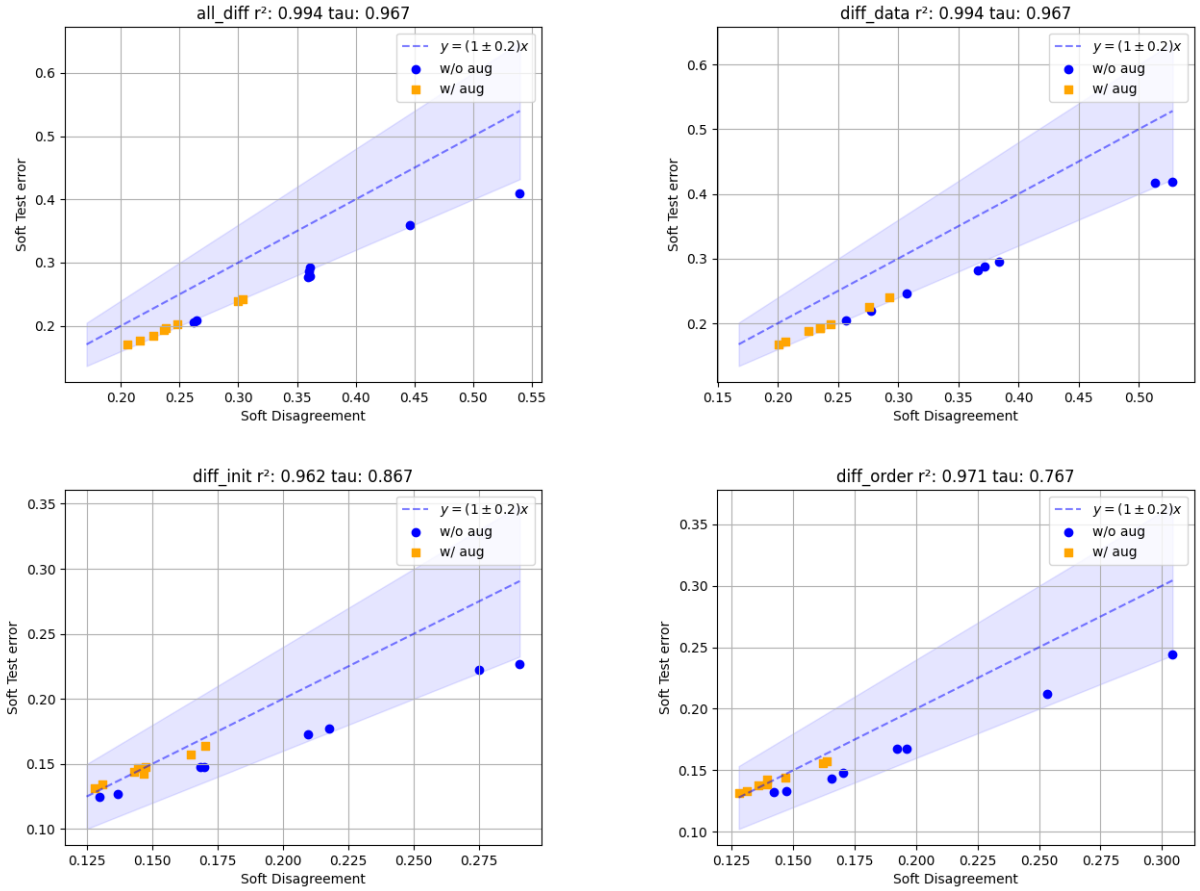


Figure 4.10: WRN(22-2)-SVHN(5%)

4.6 Relation between Agreement and Accuracy

Now that we have seen the GDE and GGDE phenomenons, it is natural to question the relation between the agreement and accuracy of an ensemble. Moreover, this relation will help us discern some properties of the ensemble if GGDE were to hold instance-wise. Let's redefine soft and hard accuracy of a classifier h for an instance $x \in \mathcal{X}$.

$$\text{SoftAcc}(h; x) := \mathbb{P}_{Y \sim f(x), \hat{Y} \sim h(x)}[Y = \hat{Y} | X = x]$$

$$\text{HardAcc}(h;x) := \mathbb{P}_{Y\sim f(x),\hat{Y}\sim h_{\text{H}}(x)}[Y=\hat{Y}|X=x]$$

Now, to realize this relationship we divided the the samples into two sets X_C and X_W where

$$X_C = \{x \in \mathcal{X} : \text{HardAcc}(h;x) = 1\}$$
$$X_W = X - X_C$$

In other words, X_C is the set containing all the instances on which the hard version of the classifier makes the correct prediction, and X_W is the set containing all instances on which it makes the wrong prediction. It is also apparent that for any $x \in X_C$ the value of $\text{SoftAcc}(h;x)$ would be the highest value inside the vector $h(x)$. Taking this into account, we can now formulate the upper and lower bound of agreement for any $x \in X_C$ as:

$$\frac{K\text{SoftAcc}(h;x)^2 - 2\text{SoftAcc}(h;x) + 1}{K-1} \leq \text{Agree}(h(x),h(x)) \tag{4.5a}$$

$$\text{Agree}(h(x),h(x)) \leq \lfloor\frac{1}{\text{SoftAcc}(h;x)}\rfloor\text{SoftAcc}(h;x)^2 + \left(1 - \lfloor\frac{1}{\text{SoftAcc}(h;x)}\rfloor\text{SoftAcc}(h;x)\right)^2 \tag{4.5b}$$

where $\lfloor x \rfloor$ represent the floor of x and the equation 4.5a is obtained after splitting $1-\text{SoftAcc}(h;x)$ into $K-1$ equal parts, and equation 4.5b is obtained as a result of splitting $1-\text{SoftAcc}(h;x)$ into $\lfloor 1/\text{SoftAcc}(h;x)\rfloor$ equal parts. Simplifying the above equation and subtracting $\text{SoftAcc}(h;x)$ from both sides we get

$$\frac{K\text{SoftAcc}(h;x)^2 - (K+1)\text{SoftAcc}(h;x) + 1}{K-1} \leq \text{Agree}(h(x),h(x)) - \text{SoftAcc}(h;x) \tag{4.6}$$

$$\begin{aligned}\text{Agree}(h(x),h(x)) - \text{SoftAcc}(h;x) \leq &\left(\lfloor\frac{1}{\text{SoftAcc}(h;x)}\rfloor^2 + \lfloor\frac{1}{\text{SoftAcc}(h;x)}\rfloor\right)\text{SoftAcc}(h;x)^2 \\ &- \text{SoftAcc}(h;x)\left(1 + 2\lfloor\frac{1}{\text{SoftAcc}(h;x)}\rfloor\right) + 1\end{aligned} \tag{4.7}$$

These bounds can be visualized through Figure 4.11. Similarly, Figure 4.12 represents

the bound on the absolute value of the difference between the soft accuracy and agreement on X_C.

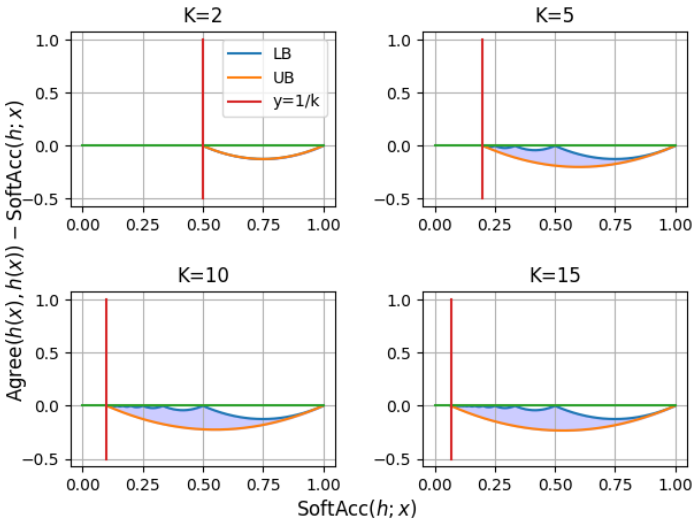


Figure 4.11: Bounds on $\text{Agree}(h(x), h(x)) - \text{SoftAcc}(h; x)$ given $\text{SoftAcc}(h; x)$ on X_C

For any $x \in X_W$ we know that $\text{SoftAcc}(h; x) \in [0, 0.5]$. To obtain the bounds for this case we first define a function $v(\cdot, \cdot)$ as:

$$v(a, b) = \begin{cases} 2, & \text{if} \quad a > \frac{1}{b} \\ 1, & \text{if} \quad a \leq \frac{1}{b} \end{cases} \tag{4.8}$$

Now we formulate the bounds for any $x \in X_W$:

$$v(\text{SoftAcc}(h; x), K)\text{SoftAcc}(h; x)^2 + \frac{(1 - v(\text{SoftAcc}(h; x), K)\text{SoftAcc}(h; x))^2}{K - v(\text{SoftAcc}(h; x), K)} \leq \text{Agree}(h(x), h(x)) \tag{4.9a}$$

$$\text{Agree}(h(x), h(x)) \leq 2 \cdot \text{SoftAcc}(h; x)^2 - 2 \cdot \text{SoftAcc}(h; x) + 1 \tag{4.9b}$$

where the upper bound (eq. 4.9b) is obtained through $\text{SoftAcc}(h; x)^2 + (1 - \text{SoftAcc}(h; x))^2$, and the lower bound is obtained through splitting $1 - \text{SoftAcc}(h; x)$ into equal $K - 1$ equal parts but making sure that $\text{SoftAcc}(h; x)$ is not greater than the highest value inside vector $h(x)$. By subtracting $\text{SoftAcc}(h; x)$ through equation 4.9, we get

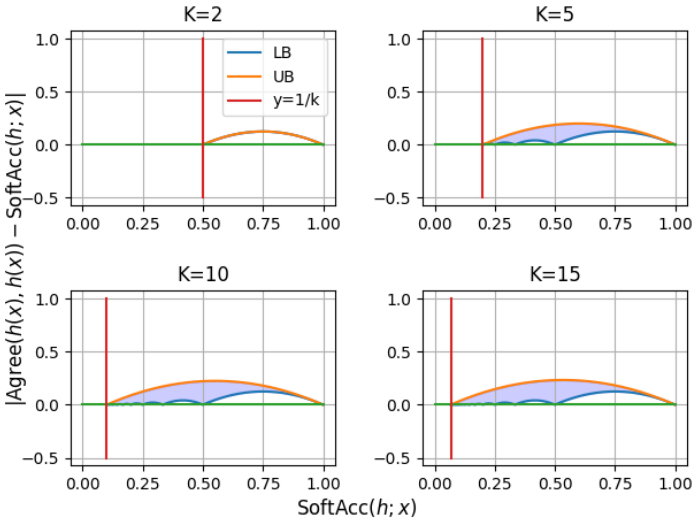


Figure 4.12: Bounds on $|\text{Agree}(h(x), h(x)) - \text{SoftAcc}(h; x)|$ given $\text{SoftAcc}(h; x)$ on X_C

$$\text{Agree}(h(x),h(x)) - \text{SoftAcc}(h; x) \geq \frac{(K\text{SoftAcc}(h; x)^2 - 2\text{SoftAcc}(h; x))(v(\text{SoftAcc}(h; x), K)) + 1}{K - v(\text{SoftAcc}(h; x), K)} - \text{SoftAcc}(h; x) \tag{4.10}$$

$$\text{Agree}(h(x), h(x)) - \text{SoftAcc}(h; x) \leq 2 \cdot \text{SoftAcc}(h; x)^2 - 3 \cdot \text{SoftAcc}(h; x) + 1 \tag{4.11}$$

These bounds can be visualized through Figure 4.13. Similarly, Figure 4.14 represents the bound on the absolute value of the difference between the soft accuracy and agreement on X_W.

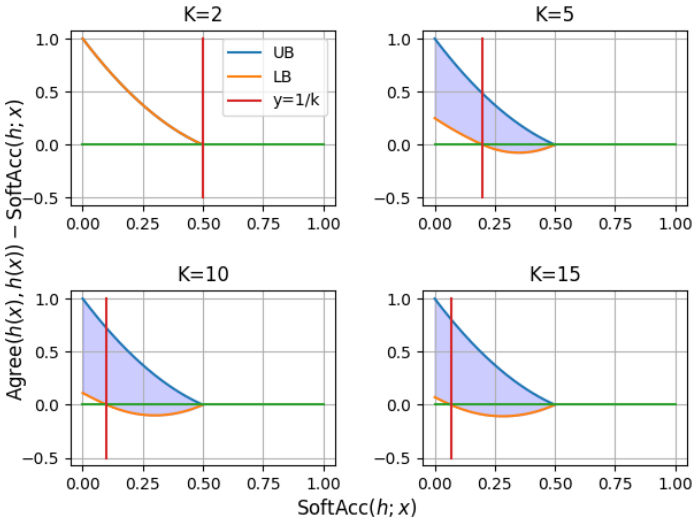


Figure 4.13: Bounds on $\text{Agree}(h(x), h(x)) - \text{SoftAcc}(h; x)$ given $\text{SoftAcc}(h; x)$ on X_W

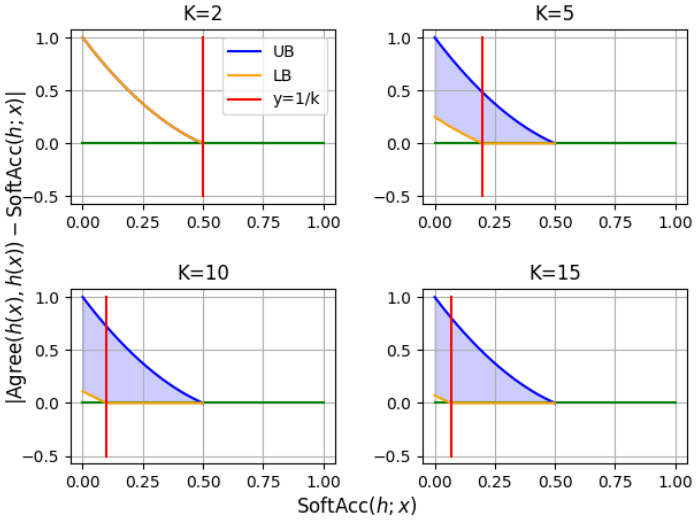


Figure 4.14: Bounds on $|\text{Agree}(h(x), h(x)) - \text{SoftAcc}(h; x)|$ given $\text{SoftAcc}(h; x)$ on X_W

Now that we have bounds on both X_C and X_W we can now present the bound on X by taking the maximum lower bound and minimum upper bounds from X_C and X_W combined. Figure 4.15 and 4.16 represent these bounds.

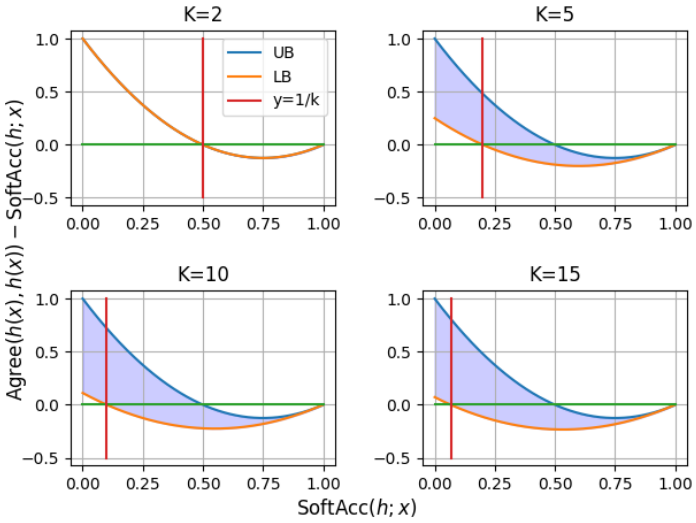


Figure 4.15: Bounds on $\text{Agree}(h(x), h(x)) - \text{SoftAcc}(h; x)$ given $\text{SoftAcc}(h; x)$ on X

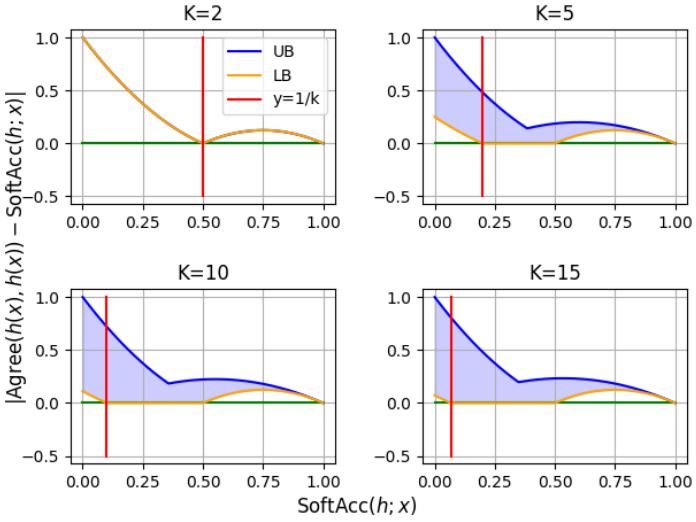


Figure 4.16: Bounds on $|\text{Agree}(h(x), h(x)) - \text{SoftAcc}(h; x)|$ given $\text{SoftAcc}(h; x)$ on X

From Figure 4.16, we can conclude that, an ensemble model can only achieve instance-wise GGDE if it is a perfect classifier ($\text{SoftAcc}(h; x) = 1$ for all $x \in \mathcal{X}$) or when its soft accuracy is less than or equal to 0.5 for all $x \in calX$.

We answer another intriguing question: what calibration could lead to instance-wise GGDE, and what would be the form of a *prefect* confidence function for a classifier following such condition? Firstly, we know that GCAC over μ implies GGDE over μ. Extending on this fact we define the notion of **Universal GCAC (UGCAC)** and a prefect confidence function, which will help us answer the questions posed above.

Definition 4.3. *We say that the ensemble $\mathcal{H}_{\mathcal{A}}$ satisfies Universal Generalized Class Aggregated Calibration (UGCAC) if it satisfies GCAC on all possible distributions μ of $\mathcal{X}$.*

Definition 4.4. *A confidence function $g_h : \mathcal{X} \to [0, 1]$ is said to be a perfect confidence function for classifier h if for all $x \in \mathcal{X}$,*

$$g_h(x) = \text{SoftAcc}(h; x)$$

From the two definitions above the following lemmas are easy to realize

Lemma 4.1. *UGCAC implies instance-wise GGDE.*

Lemma 4.2. *If the ensemble $\mathcal{H}_{\mathcal{A}}$ satisfies UGCAC, then its perfect confidence function $g_{\tilde{h}}$ must take the form:*

$$g_{\tilde{h}}(x) = \text{Agree}(\tilde{h}(x), \tilde{h}(x)) \qquad \forall x \in \mathcal{X}$$

4.7 Inferring Selective Classification

Observing the GDE or GGDE phenomenon and studying the relation between agreement and accuracy, raises a natural question: How good is Agreement as a selection function in a Selective Classification setting?

A selective classifier [12] is a pair (h, g), where h is a classifier and $g : \mathcal{X} \to \{0, 1\}$ is a selection function (or a confidence score function), which serves as a binary classifier for h for a fixed threshold τ as follows:

$$(h, g)(x) = \begin{cases} h(x), & \text{if} g(x) \geq \tau \\ \text{abstain}, & \text{otherwise} \end{cases} \tag{4.12}$$

Area Under Risk-Coverage (AURC) curve is a popular evaluation metric for such selective classifiers where coverage is the fraction of samples covered using a threshold τ, and

generally, the coverage decreases with an increase in the threshold value. To plot a Risk-Coverage (RC) curve, each instance x in the validation/test set V, is assigned a score using the selection function g, then we scan through all possible values of threshold $\tau \in [0, 1]$ and choose those samples whose score is greater than or equal to the selected value of τ. The ratio of the number of sample chosen to the total number of samples in V is called the coverage of τ, and the average risk/error of these samples is plotted against the coverage obtained. This process when performed for all possible values of τ, results in a RC curve.

These RC curves can vary depending upon the prediction technique used i.e., a "hard" RC curve is generated when the classifier used is a hard classifier, whereas a "soft" RC curve is generated when the classifier used is a soft-classifier. The lower the AURC, the better the confidence score function.

It is important to take into account that for any classifier (hard or soft), there exists an "ideal" RC curve (lowest AURC possible), which can be realized when the confidence score function is equal to the accuracy of the classifier h, i.e.

$$g_h^{\text{ideal}}(x) = \mathbb{P}_{Y \sim f(x), \hat{Y} \sim h(x)} \left[Y = \hat{Y} \right]$$

It is easy to understand this statement as for a given value of coverage we would want the average risk of the samples in that coverage to be minimal, that will only happen when the τ used to obtain the coverage perfectly reflects the risk of that coverage.

In other words, for a given classifier h and a confidence score function $g(.)$, the ideal RC-curve is obtained when the following property is satisfied for all $x \in \mathcal{X}$:

$$\mathbb{P}_{Y \sim f(x), \hat{Y} \sim h(x)} \left[Y = \hat{Y} \middle| g(x) = q \right] = q$$

Another glance to this property would help us realize that this property is the definition of a top-class calibrated model (eq. 3.1).

Furthermore, given the context of RC-curve and selective classification, the GGDE phenomenon poses a natural question: How good is Agreement ($\text{Agree}(\cdot, \cdot)$) as a confidence-score function for $\tilde{h}$ in the context of Selective Classification?

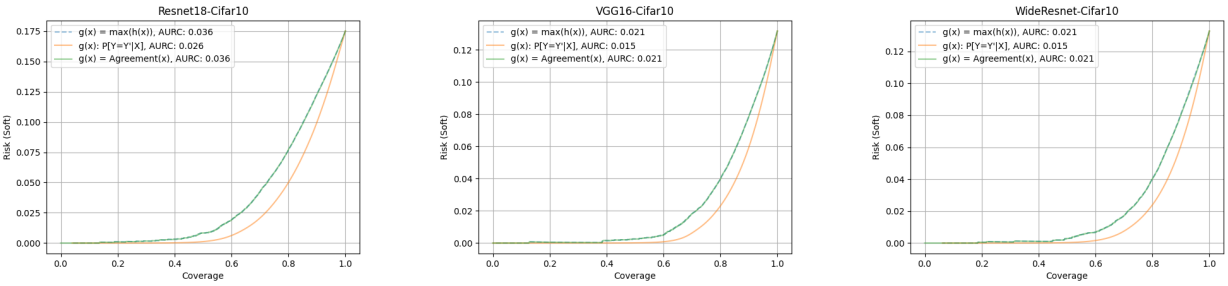


Figure 4.17: RC curves on Cifar10

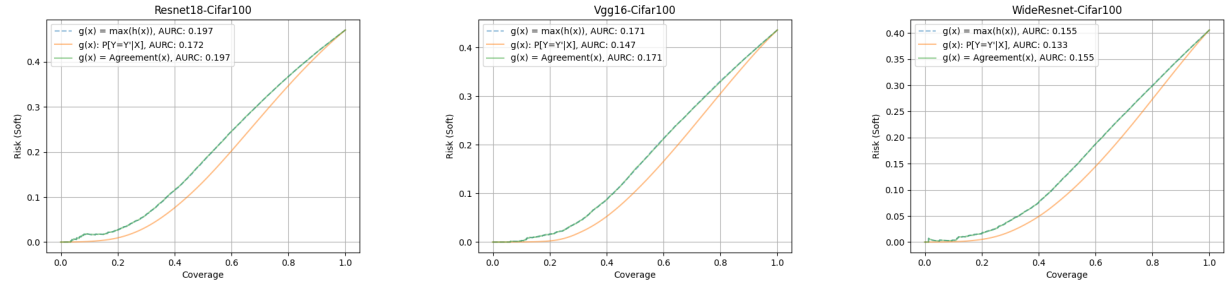


Figure 4.18: RC curves on Cifar100

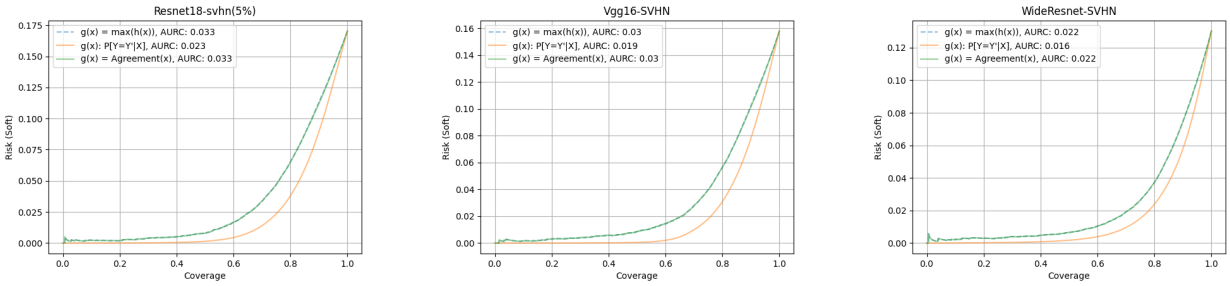


Figure 4.19: RC curves on SVHN(5%)

Generally, a natural choice of confidence-score function for a classifier would be $\|\tilde{h}(x)\|_\infty$, which is nothing but the highest probability value inside the output probability vector $\tilde{h}(x)$. We compare agreement as a confidence-score function with this confidence-score function, and found out that in-spite of GGDE phenomenon, Agreement is only as good as the highest-probability when used as a confidence-score function. Figures 4.17, 4.18 and 4.19 show RC curves for different model-dataset matchup that compares agreement (green) and $\|\tilde{h}(x)\|_\infty$ (blue) as confidence selection function with the ideal selection function (orange), for an ensemble of size 10.

Chapter 5

Confidence Calibration

5.1 Introduction

So far, we have discussed calibration, its notions, its properties (GDE and GGDE), various methods to achieve calibration, and the need for calibration. In general, calibration refers to a condition where the confidence of predictions matches their accuracy. Moving onto practical scenarios, the standard way to generate a prediction from a classifier h is to transform the predictions of h into one-hot labels h_{H} and then sample the labels from it. In such cases, the most natural way to assign the confidence to a prediction $h_{\mathrm{H}}(x)$ would be to choose the largest value in the probability vector $h(x)$, i.e. $\|h(x)\|_\infty$.

We now define the notions of Confidence Calibration, a central concept of this thesis, as a type of top-class calibration with the confidence function $G_h(x) = \|h(x)\|_\infty$.

For any subset $U \subseteq \mathcal{X}$, let μ_U denote the conditional distribution of X drawn from μ but conditioned on $X \in U$.

Definition 5.1. *A classifier h is said to satisfy **Average Confidence Calibration (ACC)** if for any subset $U \subseteq \mathcal{X}$ with $\mathbb{E}_{X\sim\mu_U}\|h(x)\|_\infty = q$, we have*

$$\mathbb{P}_{X\sim\mu_U, Y\sim f(X), \hat{Y}\sim h(X)}\left[Y = \hat{Y}\right] = q \tag{5.1}$$

for every $q \in [0, 1]$ for which the probability on the left-hand side is well defined.

It is easy to verify that such q must live in $[1/K, 1]$. Notably, ACC defined here is a special case of "perfect calibration" defined in [13].

Definition 5.2. *A classifier h is said to satisfy* ***Instance-Wise Confidence Calibration (ICC)*** *if for every $x \in \mathcal{X}$*

$$\mathbb{P}_{Y \sim f(x), \hat{Y} \sim h(x)} \left[Y = \hat{Y} \right] = \|h(x)\|_\infty \tag{5.2}$$

This particular choice of confidence function reveals that the ACC and ICC are equivalent.

Lemma 5.1. *ACC and ICC are equivalent.*

The lemma is easy to verify. First, from their definitions, it is obvious that ICC implies ACC. In the other direction, for any classifier h, it can be shown that for all $x \in \mathcal{X}$, $\|h(x)\|_\infty \geq \mathbb{P}_{Y \sim f(x), \hat{Y} \sim h(x)} \left[Y = \hat{Y} \right]$. Then if ACC holds, ICC must hold.

As the two calibration conditions are equivalent; moving forward we will only talk about ICC and may refer to it simply as confidence calibration.

5.2 Properties

Now that we have formulated the notion of Confidence Calibration, we now show that this calibration condition reveals an interesting property. First, we define another pragmatical property of a classifier which will help us explore properties of confidence calibration.

Definition 5.3. *A classifier h is said to satisfy* ***Unique-Max*** *property if for every $x \in \mathcal{X}$ there exists a unique label $i \in [K]$ such that $h(x)_i > h(x)_j \quad \forall j \neq i$, and $j \in [K]$.*

Arguably more than one value in the vector $h(x)$ tie in achieving its highest value has probability zero. Thus the Unique-Max condition is satisfied. Through this property, we have the following theorem.

Theorem 5.1. *If a classifier h satisfies ICC and the Unique-Max property, then, its hard version h_{H} is a perfect classifier, i.e. $\forall x \in \mathcal{X}$*

$$h_{\mathrm{H}}(x) = f(x) \tag{5.3}$$

Proof: If a classifier h satisfies ICC, we know that for any $x \in \mathcal{X}$, $\mathbb{P}_{Y \sim f(x), \hat{Y} \sim h(x)} \left[Y = \hat{Y} \right] = \|h(x)\|_\infty$. As it also the Unique-Max property, then for any $x \in \mathcal{X}$, $\mathbb{P}_{Y \sim f(x), \hat{Y} \sim h_{\mathrm{H}}(x)} \left[Y = \hat{Y} \right] = 1$. Therefore $\forall x \in \mathcal{X}$, $h_{\mathrm{H}}(x) = f(x)$. □

Theorem 5.1 signifies the importance of the ICC, and justifies the pursuit of this property.

In practice, achieving ICC perfectly is clearly non-realistic, since obtaining a perfect classifier is usually impossible. Thus we wish to achieve such a notion approximately, and hence define the notion measuring how closely ICC is satisfied.

We define the **Confidence Calibration Error(CCE)** of a classifier h on a given instance $x \in \mathcal{X}$ as:

$$e(h(x), f(x)) := \|h(x)\|_\infty - \mathbb{P}_{Y \sim f(x), \hat{Y} \sim h(x)}[Y = \hat{Y} | X = x] \tag{5.4}$$

Similarly, for any distribution Q over $\mathcal{X}$, define **Expected Confidence Calibration Error(ECCE)** of a classifier h as:

$$E_Q(h, f) = \mathbb{E}_{x \sim Q}[e(h(x), f(x))] \tag{5.5}$$

When the ECCE is computed over the true distribution μ of $\mathcal{X}$, we call it true-ECCE. It is important to note that Expected Calibration Error(ECE) as defined in equation 3.4 [29], is an approximation of 5.5.

Proof:

Expected Calibration Error(ECE) ([29]) is defined as:

$$ECE = \sum_{m=1}^{M} \frac{|B_m|}{n} \Big| acc(B_m) - conf(B_m) \Big|$$

where n predictions are partitioned into M equally-spaced bins. As stated by [13], it is clear that ECE is an approximation of :

$$\mathbb{E}_{q \sim P_G} \left[\Big| \mathbb{P}(Y = \hat{Y} | g(X) = q) - q \Big| \right]$$

where $G = g(X)$ is a random variable of X with probability distribution P_G and $g(x)$ denotes a confidence function which maps $\mathcal{X} \to [0, 1]$. ECCE can also be expressed in such

form, where $g(x) = \|h(x)\|_\infty$:

$$E_\mu(h, f) = \mathbb{E}_{x\sim\mu}\left[\|h(x)\|_\infty - \mathbb{P}_{Y\sim f(x),\hat{Y}\sim h(x)}[Y = \hat{Y}|X = x]\right] \tag{5.6}$$

$$= \mathbb{E}_{q\sim P_G}\left[q - \mathbb{P}_{Y\sim f(x),\hat{Y}\sim h(x)}[Y = \hat{Y}|g(X) = q]\right] \tag{5.7}$$

$$= \mathbb{E}_{q\sim P_G}\left[\left|\mathbb{P}_{Y\sim f(x),\hat{Y}\sim h(x)}[Y = \hat{Y}|g(X) = q] - q\right|\right] \tag{5.8}$$

The last step in the above equation is valid because of the fact that $\forall q \in [1/K, 1]$, $q \geq \mathbb{P}_{Y\sim f(x),\hat{Y}\sim h(x)}[Y = \hat{Y}|g(X) = q]$ when $g(x) = \|h(x)\|_\infty$. □

The formulation of ECCE gives rise to the following lemma

Lemma 5.2. *A classifier h satisfies ICC when $E_\mu(h, f) = 0$.*

Proof:

When $E_\mu(h, f) = 0$ for a classifier h we know that,

$$\mathbb{E}_{x\sim\mu}[\|h(x)\|_\infty] = \mathbb{E}_{x\sim\mu}\left[\mathbb{P}_{Y\sim f(x),\hat{Y}\sim h(x)}[Y = \hat{Y}]\right]$$

and since for every $x \in \mathcal{X}$, $\|h(x)\|_\infty \geq \mathbb{P}_{Y\sim f(x),\hat{Y}\sim h(x)}[Y = \hat{Y}]$, the above equation is satisfied only when for any $x \in \mathcal{X}$,

$$\|h(x)\|_\infty = \mathbb{P}_{Y\sim f(x),\hat{Y}\sim h(x)}[Y = \hat{Y}]$$

which is the definition of ICC. □

Note that if such a classifier h satisfies the Unique-Max property, then its hard version h_{H} is perfect, according to Theorem 5.1.

5.3 Connection to Label Smoothing

Now that we have highlighted the need for the ICC, we show its connection to label smoothing and reveal its gravity as a theoretical foundation for label smoothing.

Lemma 5.2 implies that minimizing $E_\mu(h, f)$ will drive the classifier h towards ICC, and hence drive h_{H} closer to the ground truth classifier f. Since this error function is

non-negative, $E_\mu(h, f)$ reaches minimum when

$$\mathbb{E}_{x\sim\mu}[\|h(x)\|_\infty] = \mathbb{E}_{x\sim\mu}\left[\mathbb{P}_{Y\sim f(x),\hat{Y}\sim h(x)}[Y = \hat{Y}|X = x]\right]$$

The above equation helps bring label-smoothing ([34]) into the picture, as it suggests the model should not be over-confident and assign a label with confidence 1, rather its confidence should reflect its soft accuracy. [34] defines label smoothing as a technique that modifies the output of the ground-truth classifier f based on a given hyperparameter ϵ during training. The label smoothed ground truth $f^{\mathrm{LS}}(x)$ for any instance $x \in \mathcal{X}$ is obtained by:

$$f^{\mathrm{LS}}(x) = (1 - \epsilon)f(x) + \frac{\epsilon}{K} \tag{5.9}$$

It is important to note that, when a classifier achieves 100% train accuracy over an ϵ-label-smoothed ground truth, it is apparent that $\mathbb{E}_{x\sim S}[\|h(x)\|_\infty] = 1 - \frac{\epsilon(K-1)}{K}$. Similarly, we can manipulate ϵ to achieve a desired value of $\mathbb{E}_{x\sim S}[\|h(x)\|_\infty]$ through label smooth training. We indulge deeper into the use of this information in order to be able to obtain a good hyperparameter(ϵ) for label smoothing in the next section.

Now that we have established a connection between ICC and label smoothing, we show that on the label-smoothed loss, a model tends to generalize better.

Let $\mathcal{H}$ be a family of classifiers (each $h \in \mathcal{H}$ maps x to a distribution in $\Delta([K])$). Let g_h be the non-negative function on $\mathcal{X}$ defined by

$$g_h(x) := e(h(x), f(x)).$$

Define $\mathcal{G} = \{g_h : h \in \mathcal{H}\}$. We will use $\mathfrak{R}(\mathcal{G})$ to denote the Rademacher complexity ([4]) of $\mathcal{G}$.

Theorem 5.2. *Let $\Delta \in (0, 1)$. With probability at least $1 - \Delta$, for all $h \in \mathcal{H}$*

$$E_\mu(h, f) \le E_S(h, f) + 2\mathfrak{R}_m(\mathcal{G}) + \sqrt{\frac{\log(1/\Delta)}{2m}} \tag{5.10}$$

$$\begin{aligned} E_\mu(h, f^{\mathrm{LS}}) \le E_S(h, f^{\mathrm{LS}}) + 2(1 - \epsilon)\mathfrak{R}_m(\mathcal{G}) \\ + (1 - \frac{\epsilon}{K})\sqrt{\frac{\log(2/\Delta)}{2m}} \end{aligned} \tag{5.11}$$

Proof:

With the Rademacher bound on the maximal deviation between true risk and empirical risk, with probability at least $1-\Delta$ we have,

$$E_\mu(h,f) \leq E_S(h,f) + 2\mathfrak{R}_m(\mathcal{G}) + \sqrt{\frac{\log(1/\Delta)}{2m}}$$

Moreover, we know that,

$$E_\mu(h,f^{\text{LS}}) = \mathbb{E}_{x\sim\mu}[\|h(x)\|_\infty] - \mathbb{E}_{x\sim\mu}\left[\mathbb{P}_{Y\sim f^{\text{LS}}(x),\hat{Y}\sim h(x)}[Y=\hat{Y}]\right]$$

$$E_\mu(h,f^{\text{LS}}) = \mathbb{E}_{x\sim\mu}[\|h(x)\|_\infty] - \mathbb{E}_{x\sim\mu} < h(x), f^{\text{LS}}(x) >$$

where $< \cdot,\cdot >$ represents the dot product of two vectors and given the label smoothing parameter ϵ,

$$f^{\text{LS}}(x) = (1-\epsilon).f(x) + \frac{\epsilon}{K}$$

Therefore,

$$E_\mu(h,f^{\text{LS}}) = \epsilon.\mathbb{E}_{x\sim\mu}\left[\|h(x)\|_\infty\right] + (1-\epsilon).E_\mu(h,f) - \frac{\epsilon}{K}$$

$$E_\mu(h,f) = \frac{E_\mu(h,f^{\text{LS}}) + \frac{\epsilon}{K} - \epsilon.\mathbb{E}_{x\sim\mu}[\|h(x)\|_\infty]}{(1-\epsilon)}$$

Similarly,

$$E_S(h,f) = \frac{E_S(h,f^{\text{LS}}) + \frac{\epsilon}{K} - \epsilon.\mathbb{E}_{x\sim S}[\|h(x)\|_\infty]}{(1-\epsilon)}$$

Substituting above in 5.10, with probability at least $1-\Delta$ we have:

$$E_\mu(h,f^{\text{LS}}) \leq E_S(h,f^{\text{LS}}) + 2(1-\epsilon)\mathfrak{R}_m(\mathcal{G}) + (1-\epsilon)\sqrt{\frac{\log(1/\Delta)}{2m}} + \epsilon.\left[\mathbb{E}_{x\sim\mu}[\|h(x)\|_\infty] - \mathbb{E}_{x\sim S}[\|h(x)\|_\infty]\right] \tag{5.12}$$

Using McDiarmid's Inequality ([24]) we can derive a bound on $\mathbb{E}_{x\sim\mu}[\|h(x)\|_\infty] - \mathbb{E}_{x\sim S}[\|h(x)\|_\infty]$, as we know that $\|h(x)\|_\infty$ lies in the range $[1/K, 1]$, i.e.

$$\mathbb{P}\left[\mathbb{E}_{x\sim\mu}[\|h(x)\|_\infty] - \mathbb{E}_{x\sim S}[\|h(x)\|_\infty] \leq \delta\right] \geq 1 - \exp\left(\frac{-2\delta^2\mathrm{K}^2\mathrm{m}}{(\mathrm{K}-1)^2}\right)$$

Or with probability at least $1-\Delta$,

$$\mathbb{E}_{x\sim\mu}[\|h(x)\|_\infty] - \mathbb{E}_{x\sim S}[\|h(x)\|_\infty] \leq \frac{K-1}{K}.\sqrt{\frac{\log(1/\Delta)}{2m}} \tag{5.13}$$

Using union bound to combine inequalities 5.12 and 5.13 yields, with probability at least $1-\Delta$:

$$E_\mu(h, f^{\text{LS}}) \leq E_S(h, f^{\text{LS}}) + 2(1-\epsilon)\mathfrak{R}_m(\mathcal{G}) + (1-\epsilon)\sqrt{\frac{\log(2/\Delta)}{2m}} + \epsilon.\frac{K-1}{K}.\sqrt{\frac{\log(2/\Delta)}{2m}}$$

Or,

$$E_\mu(h, f^{\text{LS}}) \leq E_S(h, f^{\text{LS}}) + 2(1-\epsilon)\mathfrak{R}_m(\mathcal{G}) + (1-\frac{\epsilon}{K})\sqrt{\frac{\log(2/\Delta)}{2m}}$$

Hence proved. □

Comparing the two bounds in this theorem, we see that for the same hypothesis class $\mathcal{H}$, the upper bound of generalization gap with respect to f^{LS} is smaller than that with respect to f, particularly for large m or large model capacity.

As a corollary, the gap between the empirical label-smoothed ECCE and true standard ECCE can be upper bound.

Corollary 5.1. *Let $\Delta \in (0,1)$. With probability at least $1-\Delta$, for all $h \in \mathcal{H}$,*

$$\begin{aligned} E_\mu(h, f) \leq \frac{E_S(h, f^{\text{LS}})}{(1-\epsilon)} + 2\mathfrak{R}_m(\mathcal{G}) \\ + \frac{(1-\epsilon/K)}{(1-\epsilon)}\sqrt{\frac{\log(2/\Delta)}{2m}} + \frac{\epsilon}{K(1-\epsilon)} \end{aligned} \tag{5.14}$$

Proof:

From equation 5.11, we know that with probability at least $1-\Delta$:

$$E_\mu(h, f^{LS}) \leq E_S(h, f^{LS}) + 2(1-\epsilon)\mathfrak{R}_m(\mathcal{G}) + (1-\frac{\epsilon}{K})\sqrt{\frac{\log(2/\Delta)}{2m}}$$

We also know that

$$E_\mu(h, f^{LS}) = \epsilon.\mathbb{E}_{x\sim\mu}\left[\|h(x)\|_\infty\right] + (1-\epsilon).E_\mu(h, f) - \frac{\epsilon}{K}$$

Combining the two equations above gives, with probability at least $1 - \Delta$:

$$\epsilon.\mathbb{E}_{x\sim\mu}\left[\|h(x)\|_\infty\right]+(1-\epsilon).E_\mu(h,f)-\frac{\epsilon}{K} \leq E_S(h,f^{LS})+2(1-\epsilon)\mathfrak{R}_m(\mathcal{G})+(1-\frac{\epsilon}{K})\sqrt{\frac{\log(2/\Delta)}{2m}}$$

$$E_\mu(h,f) \leq \frac{E_S(h,f^{LS})}{(1-\epsilon)} + 2\mathfrak{R}_m(\mathcal{G}) + \frac{(1-\epsilon/K)}{(1-\epsilon)}\sqrt{\frac{\log(2/\Delta)}{2m}} + \frac{\epsilon}{K(1-\epsilon)} - \frac{\epsilon.\mathbb{E}_{x\sim\mu}\left[\|h(x)\|_\infty\right]}{(1-\epsilon)}$$

$$E_\mu(h,f) \leq \frac{E_S(h,f^{LS})}{(1-\epsilon)} + 2\mathfrak{R}_m(\mathcal{G}) + \frac{(1-\epsilon/K)}{(1-\epsilon)}\sqrt{\frac{\log(2/\Delta)}{2m}} + \frac{\epsilon}{K(1-\epsilon)}$$

Hence Proved. □

Chapter 6

Agreement Guided Label Smoothing

6.1 Formulation and Justification

In this section, we present a training algorithm that capitalizes on several of the results discussed till now.

We first define two quantities related to $E_\mu(\tilde{h}, f)$.

$$\tilde{E}(\tilde{h}) := \mathbb{E}_{x\sim\mathcal{S}}[\|\tilde{h}(x)\|_\infty] - \text{SoftAcc}_\mu(\tilde{h}) \tag{6.1}$$

$$\hat{E}(\tilde{h}) := -\mathbb{E}_{x\sim S}\left[\mathbb{P}_{Y\sim f(x),\hat{Y}\sim\tilde{h}(x)}[Y = \hat{Y}|X = x]\right] \tag{6.2}$$

It is apparent that $\tilde{E}(\tilde{h})$ is essentially $E_\mu(\tilde{h}, f)$ with the expected confidence over population replaced with the expected confidence over the training set and that $\hat{E}(\tilde{h})$ is the negative of the soft accuracy of $\tilde{h}$ over the train set S.

Let Δ be a small positive number close to 0. Define

$$B := \left\{\tilde{h} : \tilde{E}(\tilde{h}) \leq \frac{K-1}{K}.\sqrt{\frac{\log(1/\Delta)}{2m}}\right\} \tag{6.3}$$

For relatively large m, $\frac{K-1}{K}.\sqrt{\frac{\log(1/\Delta)}{2m}} \approx 0$, then B contains classifiers h with $\mathbb{E}_{x\sim\mathcal{S}}[\|\tilde{h}(x)\|_\infty] \approx$

$\text{SoftAcc}_\mu(\tilde{h})$. If GGDE holds, then each member $h \in B$ satisfy

$$\mathbb{E}_{x\sim\mathcal{S}}[\|\tilde{h}(x)\|_\infty] \approx \mathbb{E}_{x\sim\mu}\text{Agree}(\tilde{h}(x), \tilde{h}(x)). \tag{6.4}$$

As we have empirically verified GGDE for neural network models trained with SGD, we consider that (6.4) holds. Note that it is possible to measure expected agreement $\mathbb{E}_{x\sim\mu}\text{Agree}(\tilde{h}(x), \tilde{h}(x))$ using an unlabelled validation set. Further, if the expected agreement is δ, when we apply label-smoothed training with smoothing parameter $\epsilon = \frac{K(1-\delta)}{K-1}$, we will reach a classifier in B.

Further, define

$$C := \{\tilde{h} : E_\mu(\tilde{h}, f) = 0\} \tag{6.5}$$

$$D := \{\tilde{h} : \hat{E}(\tilde{h}) = -1\} \tag{6.6}$$

Note that the set C is the set of all classifiers with true-ECCE zero; implied by lemma 5.2 and theorem 5.1, these classifiers, when hardened, agree with the ground-truth classifier f. Therefore, obtaining a classifier in set C is the ultimate goal of the learning task. The set D contains all the classifiers whose soft accuracy on the train set S is 1. Standard training (that minimizes the cross-entropy loss) converges to a classifier in D.

The following theorem can be proved, where the second half is due to McDiarmid Inequality ([24]).

Theorem 6.1. *$C \cap D \neq \emptyset$ and with probability at least $1 - \Delta$, $C \subseteq B$.*

Proof:

We have, for $\Delta \approx 0$:

$$B := \left\{\tilde{h} : \tilde{E}(\tilde{h}) \leq \frac{K-1}{K}.\sqrt{\frac{\log(1/\Delta)}{2m}}\right\} \tag{6.7}$$

$$C := \{\tilde{h} : E_\mu(\tilde{h}, f) = 0\} \tag{6.8}$$

$$D := \{\tilde{h} : \hat{E}(\tilde{h}) = -1\} \tag{6.9}$$

and

$$\tilde{E}(\tilde{h}) := \mathbb{E}_{x\sim\mathcal{S}}[\|\tilde{h}(x)\|_\infty] - \text{SoftAcc}_\mu(\tilde{h}) \tag{6.10}$$

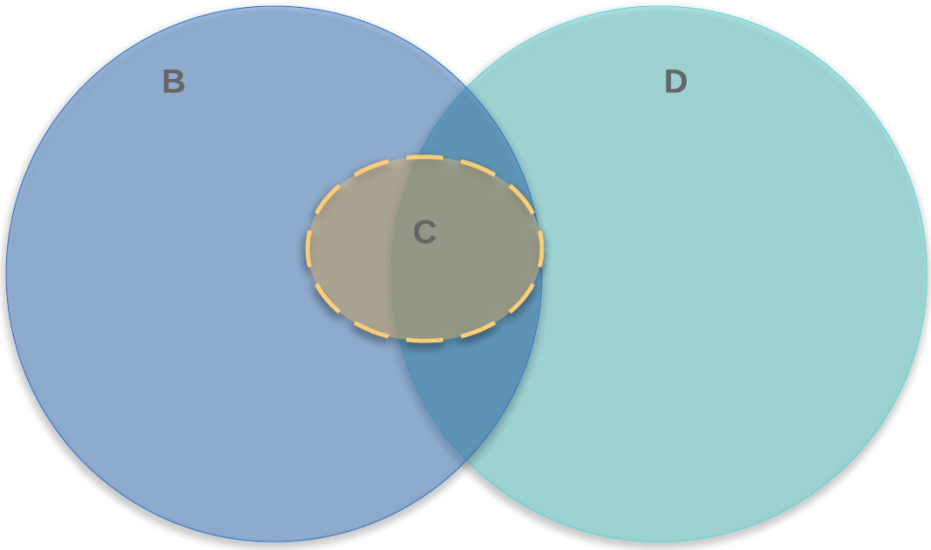


Figure 6.1: Venn Diagram of Sets B, C and D

$$\hat{E}(\tilde{h}) := -\mathbb{E}_{x\sim S}\left[\mathbb{P}_{Y\sim f(x),\hat{Y}\sim\tilde{h}(x)}[Y=\hat{Y}|X=x]\right] \tag{6.11}$$

It is apparent that C and D intersect non-trivially, i.e. $C \cap D \neq \emptyset$ and we know that for any $\tilde{h} \in C$:

$$\mathbb{E}_{x\sim\mu}[\|\tilde{h}(x)\|_\infty] = \text{SoftAcc}_\mu(\tilde{h})$$
$$\tilde{E}(\tilde{h}) := \mathbb{E}_{x\sim\mathcal{S}}[\|\tilde{h}(x)\|_\infty] - \mathbb{E}_{x\sim\mu}[\|\tilde{h}(x)\|_\infty]$$

Also, since the function $\|h(x)\|_\infty$ is bounded between $[1/K, 1]$ for any $x \in \mathcal{X}$, then through McDiarmid inequality ([24]), with probability at least $1-\Delta$ we have:

$$\mathbb{E}_{x\sim S}[\|\tilde{h}(x)\|_\infty] - \mathbb{E}_{x\sim\mu}[\|\tilde{h}(x)\|_\infty] \leq \frac{K-1}{K}\cdot\sqrt{\frac{\log(1/\Delta)}{2m}}$$

or,

$$\tilde{E}(\tilde{h}) \leq \frac{K-1}{K}\cdot\sqrt{\frac{\log(1/\Delta)}{2m}}$$

which is true for any $\tilde{h} \in B$. Therefore, with probability at least $1-\Delta$, $C \subseteq B$ $\square$

By this theorem, the Venn Diagram given in Figure 6.1 showing the relationship between B, C, and D is correct with probability at least $1-\Delta$. This relationship motivates a training strategy that attempts to find a classifier in $B \cap D$. Such a strategy arguably increases the chances of finding a classifier closer to the ground truth classifier f, compared with the standard training, which merely searches in D.

We now present a novel training algorithm, **Agreement Guided Label Smoothing**, or **AGLS**, implementing such a strategy. In AGLS, a set of M models with the same architecture are trained in parallel using SGD. In addition to the training set S, an unlabelled validation set V is also required. AGLS alternates between a standard training round and a label-smooth training round. At the end of each standard training round, expected agreement δ is estimated using V, then the following label-smooth training round continues to train the models using label-smooth parameter $\epsilon = \frac{K(1-\delta)}{K-1}$. Remarkably, if B and D are convex sets and if each training round corresponds to projecting the model parameter to set B or D respectively, the algorithm can be regarded as an instance of the Projection Onto Convex Sets (POCS) algorithm ([5]), which guarantees to converge to a solution in $B \cap D$. In AGLS, such a guarantee can not be argued rigorously. Nonetheless, if we instead restrict the definition of D and B to a neighborhood of a pre-trained $\tilde{h}$, one may argue that these sets are closer to being convex and that the algorithm is more likely to reach $B \cap D$.

Algorithm 6.1 Agreement Guided Label Smoothing (AGLS)

Input:

Training set S
Validation set V
Pre-Trained ensemble model $\tilde{h}$
Rounds r
Number of classes K

Output:

$\tilde{h}_{\text{opt}}$

$t \leftarrow 0$
best_agreement $\leftarrow 0$
$\tilde{h}_{\text{opt}} \leftarrow \tilde{h}$
while $t \leq r$ **do**
    $t \leftarrow t + 1$
    if t is odd **then**
        Compute soft agreement δ

$$\delta = \mathbb{E}_{x \sim V} \left[\text{Agree}(\tilde{h}(x), \tilde{h}(x)) \right]$$

        Perform label-smoothing($\epsilon = \frac{K(1-\delta)}{K-1}$) on $\tilde{h}$
    else if t is even **then**
        Perform standard training on $\tilde{h}$
    end if
    Compute hard agreement a_{H}:

$$a_{\text{H}} = \mathbb{E}_{x \sim V} \left[\text{Agree}(\tilde{h}_{\text{H}}(x), \tilde{h}_{\text{H}}(x)) \right]$$

    if $a_{\text{H}} >$ best_agreement **then**
        $\tilde{h}_{\text{opt}} \leftarrow \tilde{h}$
        best_agreement $\leftarrow a_{\text{H}}$
    end if
end while
return $\tilde{h}_{\text{opt}}$

6.2 Experimental Details

Datasets Three datasets are used in our experiments. The **CIFAR-10** and **CIFAR-100** [17] dataset consists of 32 $\times$ 32 color images which represent 10 and 100 classes respectively. The train set contains 50,000 images while the test set contains 10,000 images for both of these datasets. The **SVHN** dataset [30] is a widely-used real-world image dataset generally used for object recognition tasks. Similar to the CIFAR-10/100 dataset, SVHN consists of 32 $\times$ 32 color images, each associated with a label ranging from 0 to 9 (10 classes). For the training of the ensemble model, we only use (5%) 3,663 images out of 73,257 images available as the train set, whereas the full test set is used which consists of 26,032 images. We believe that the sparsity of SVHN train set may help unveil some properties of the AGLS algorithm. The Top-1 classification accuracy (hard accuracy) is used as the evaluation metric. While looking at the outputs of AGLS algorithm we focus our attention on comparing 4 quantities: baseline (Pre-train) Top-1 classification (hard) accuracy, optimal label-smooth (hard) accuracy achieved on the pre-trained model, (hard) accuracy of the output model of AGLS algorithm and the best possible (hard) accuracy of the AGLS algorithm.

Networks We report our observations using three widely-used networks namely ResNet18 (ImgNet version) [14], WideResNet WRN-22(depth)-2(widen-factor) with 0.0 dropout [42] and CIFAR-VGG16-BN [23]. Table 6.1 shows the number of parameters of these networks on CIFAR-100 dataset.

Table 6.1: Number of parameters on CIFAR-100 dataset

WRN 22-2	ResNet18	CIFAR-VGG16-BN
1.1M	11M	15M

The choice of these specific hyperparameters is inspired from the GDE experiments done by [16], and these specific hyperparameters do not affect the results as long as the 100% training accuracy is achieved.

An ensemble of size 2 is generated using these models with the implementation details discussed below.

Implementation Details We implement all networks and training procedures in PyTorch and conduct all experiments on an NVIDIA V100 GPU. For CIFAR-10/100 we use Momentum SGD and set the initial learning rate to 0.1, momentum to 0.9, mini-batch size to 200, and weight decay to 0.0001. The learning rate is dropped by 0.1 every 50

epochs. We train to 200 epochs or 100% train accuracy whichever is achieved first. For SVHN dataset we use momentum SGD with a learning rate of 0.01, a momentum of 0.9, a mini-batch size of 64, and a weight-decay of 0.0001. The learning rate is dropped by 0.1 every 50 epochs, trained to 200 epochs or 100% train accuracy. Since only 5% SVHN train set is used, we make sure that all the models constituting the ensemble, see the same 5% of the images. For AGLS training, every round uses the same hyperparameter configuration with respect to the dataset as mentioned above with the exception of training until 100 epochs or 100% train accuracy for all network-dataset combinations. To report the optimal label-smooth (hard) accuracy of the ensembles in Table 6.2, we select the highest accuracy obtained after performing the label-smooth training on the pre-trained model 10 times over a set of hyperparameter ϵ used, let's call this set as eps-set. For CIFAR-10 and SVHN eps-set= $\{0.02, 0.04, \ldots, 0.30\}$. For CIFAR-100 eps-set= $\{0.05, 0.10, \ldots, 0.75\}$. The reason behind the choice of this set is due to the fact that for datasets CIFAR10 and SVHN, the hyperparameter ϵ for first round of AGLS is approximately in the range $[0.2, 0.3]$ for all the networks considered. In contrast, the range for CIFAR100 is $[0.4, 0.6]$, this range of ϵ can be attributed to the soft accuracy or the soft agreement of the pre-trained models.‘

6.3 Results and Observation

Table 6.2: Top-1 accuracy (%) on CIFAR-10, SVHN(5%) and CIFAR-100 dataset. Baseline refers to the accuracy of pre-trained models, Opt. LS refers to the optimal Label smoothing accuracy achieved, AGLS and Best-AGLS refer to the accuracy of the output model and highest accuracy achieved through the AGLS algorithm respectively, with $r = 200$ for CIFAR-10 and SVHN, whereas $r = 50$ for CIFAR-100.

Networks	CIFAR-10				SVHN(5%)				CIFAR-100			
	Baseline	Opt. LS	AGLS	Best-AGLS	Baseline	Opt. LS	AGLS	Best-AGLS	Baseline	Opt. LS	AGLS	Best-AGLS
Resnet18	85.80 ± 0.93	87.25 ± 0.07	89.57 ± 0.22	89.87 ± 0.17	86.32 ± 0.19	86.77 ± 0.16	87.27 ± 0.27	87.73 ± 0.07	62.19 ± 0.74	65.02 ± 0.25	65.62 ± 0.33	65.73 ± 0.16
WRN 22-2	89.44 ± 0.11	89.58 ± 0.10	91.27 ± 0.15	91.54 ± 0.08	90.70 ± 0.33	90.56 ± 0.18	91.22 ± 0.27	91.75 ± 0.17	66.59 ± 0.17	66.19 ± 0.22	66.79 ± 0.36	67.85 ± 0.16
CIFAR-VGG16-BN	88.37 ± 0.25	89.67 ± 0.07	90.77 ± 0.11	91.12 ± 0.08	86.89 ± 0.82	87.66 ± 0.40	88.07 ± 0.44	88.20 ± 0.43	62.20 ± 0.52	65.08 ± 0.25	66.36 ± 0.50	66.88 ± 0.32

Table 6.2 and Figures 6.4, 6.5, 6.6 shows how well AGLS fares when compared to normal training and label-smooth training. AGLS refers to the output of the AGLS algorithm whereas Best-AGLS refers to the best possible outcome of AGLS, we report Best-AGLS along with AGLS as it is obtainable if a validation set is used to measure the hard accuracy instead of using hard-agreement like in AGLS.

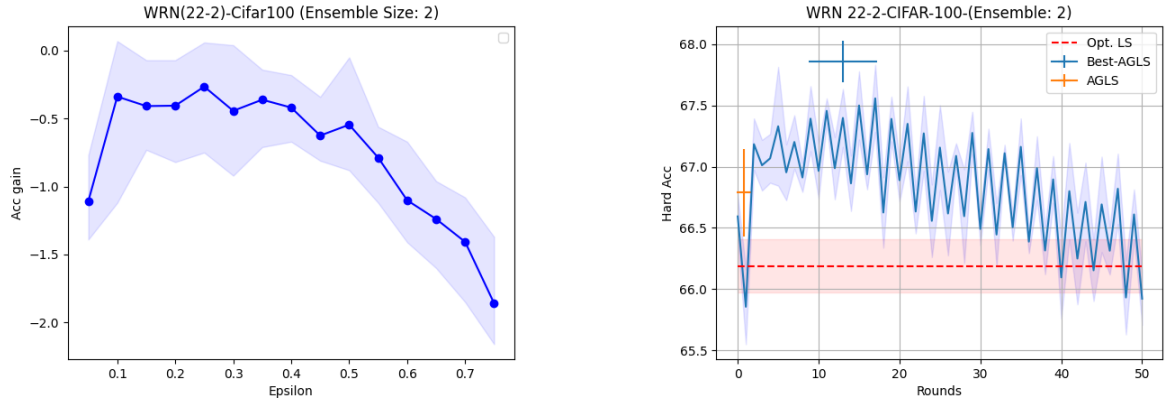
Observations (i) From Table 6.2 we find that AGLS demonstrates a superior performance in comparison to baseline and label smoothing across all network-dataset settings

Table 6.3: True-ECCE (lower is better) on different network-dataset configurations

Networks	CIFAR-10			SVHN(5%)			CIFAR-100		
	Baseline	AGLS	Best-AGLS	Baseline	AGLS	Best-AGLS	Baseline	AGLS	Best-AGLS
ResNet18	0.0731	0.0532	0.0317	0.0723	0.0485	0.0458	0.1449	0.0183	0.0183
WRN 22-2	0.0545	0.0450	0.0438	0.0441	0.0281	0.0118	0.1388	0.1353	0.0180
CIFAR-VGG16-BN	0.0559	0.0517	0.0379	0.0683	0.0515	0.0333	0.1679	0.0747	0.0707

investigated in this paper. Specifically, we observe that AGLS improves the ensemble model's accuracy by at least 1% over the baseline, except in the case of WRN 22-2 on the CIFAR-100 dataset, where the increase is only 0.35%. This small increase can be speculated as a result of either a low parameter and complex dataset matchup or due to the fact that GDE holds loosely in this setting.

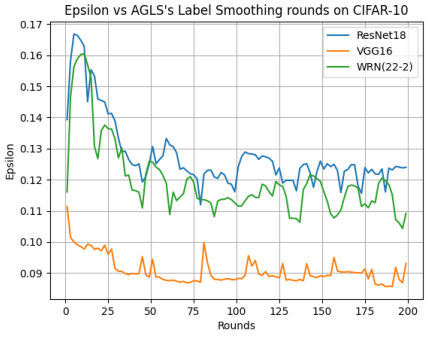
(ii) It can also be seen that AGLS's performance is at least as good as label smoothing and if label smoothing improves over baseline significantly then we can expect the same with AGLS.



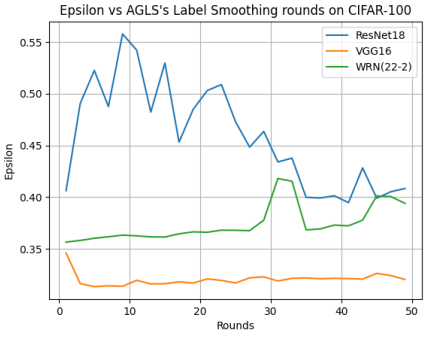
(a) (b)

Figure 6.2: (a) Shows gain in hard accuracy vs different LS hyperparameter ϵ; (b) Shows hard accuracy vs AGLS rounds in AGLS

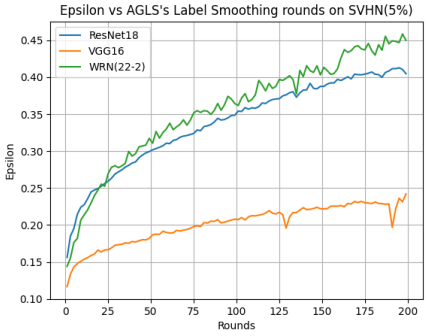
(iii) Following, this logic one might think that in case label smoothing worsens the performance (which is usually observed in network-dataset configuration with low baseline), AGLS may perform even worse. However, Figure 6.2 shows that although label smoothing worsens the baseline accuracy, AGLS still shows improved performance in comparison.



(a) CIFAR-10



(b) CIFAR-100



(c) SVHN(5%)

Figure 6.3: Epsilon used for Label smoothing in AGLS rounds.

(iv) In Figure 6.3 we observe that the hyperparameter ϵ used for label smooth training round in AGLS, decreases over rounds for CIFAR-10 and CIFAR-100, whereas it increases for SVHN over rounds. Increasing epsilon indicates that the model is driven towards being a random classifier. It is to be noted that this rise in ϵ is prominently observed in the SVHN dataset only, which can be accredited to the sparsity of the training set (5%) used for the dataset.

(v) Figure 6.5 shows that even though the model performance worsens over rounds, hard agreement as a proxy for hard accuracy helps AGLS to return the better-performing model through the course of its runtime.

(vi) From Table 6.3 we can see that AGLS outputs a model with reduced true-ECCE, combined with the fact that it also improves hard accuracy (Table 6.2) justifies the AGLS

strategy empirically i.e., it improves the Confidence Calibration of the classifier in conjunction with improving its hard accuracy.

(vii) From Figures 6.7, 6.8 and 6.9 we see that hard agreement holds as a proxy for hard accuracy throughout AGLS training. Arguably, it captures the trend in hard accuracy as we alternate between the standard and label smooth training, therefore, justifying its use as a checkpoint/stopping criterion for AGLS.

(viii) From Table 6.2 we see that although AGLS outperforms optimal LS in low data regime (SVHN), it does so by a smaller margin when compared to other datasets. We believe that this can be attributed to the fact that GGDE doesn't hold as closely in SVHN-5% when compared to other datasets, and thus the looseness of GGDE might appear as a potential limitation for AGLS in some scenarios.

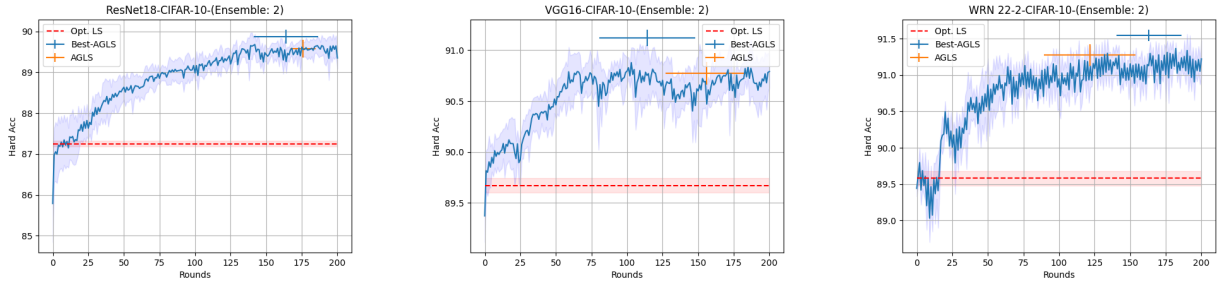


Figure 6.4: AGLS on Cifar10

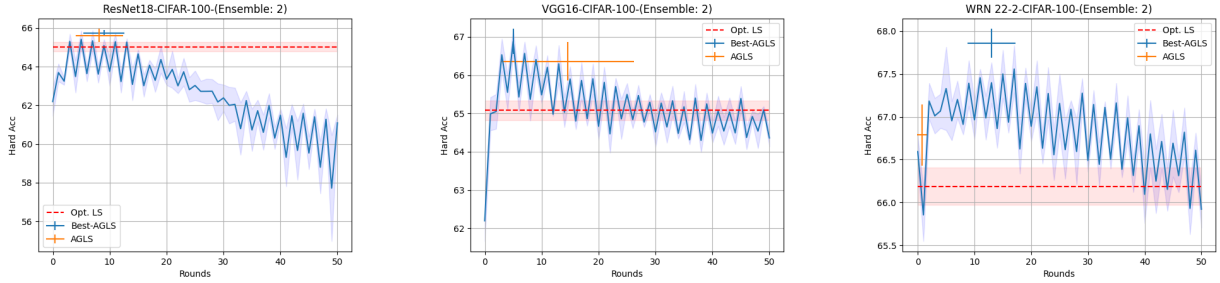


Figure 6.5: AGLS on Cifar100

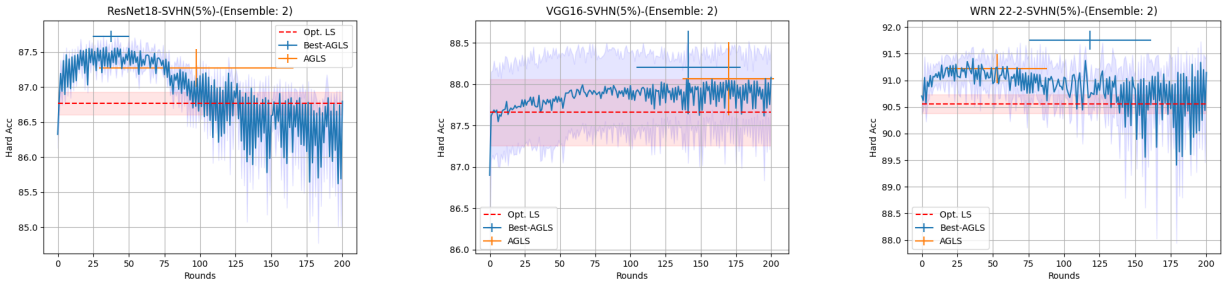


Figure 6.6: AGLS on SVHN(5%)

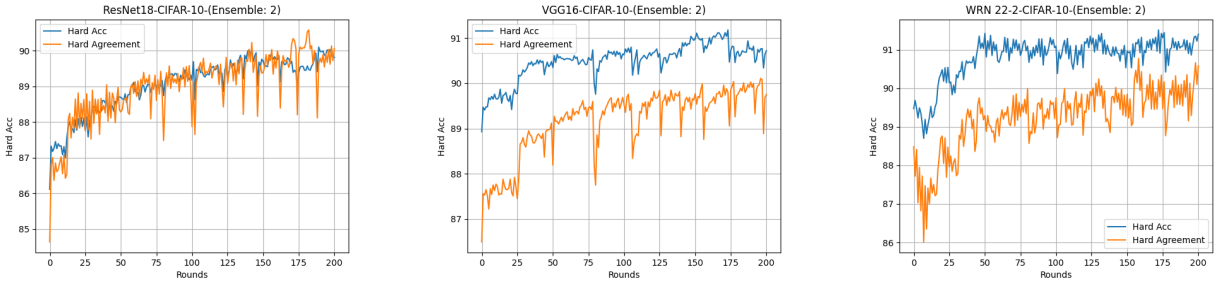


Figure 6.7: Hard Accuracy vs Hard Agreement for AGLS on Cifar10

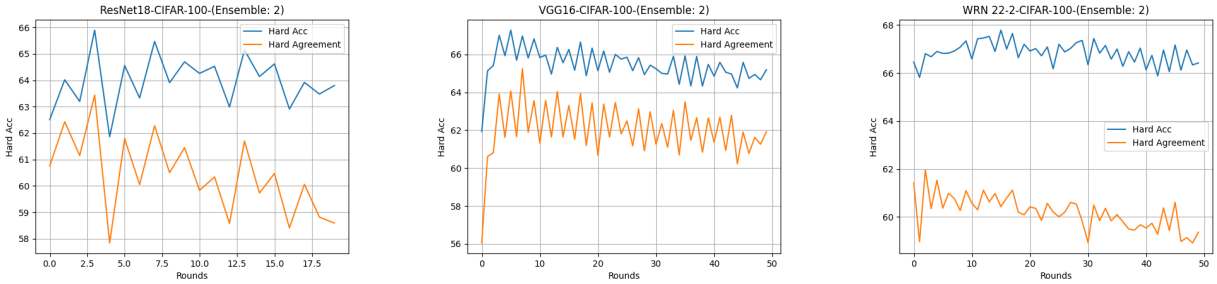


Figure 6.8: Hard Accuracy vs Hard Agreement for AGLS on Cifar100

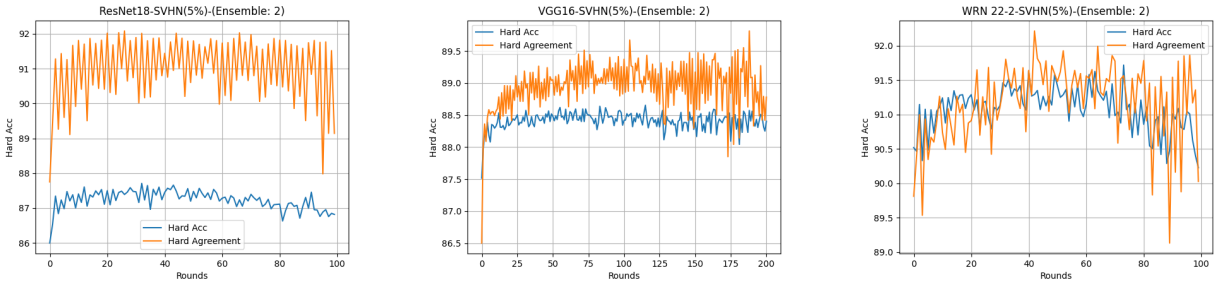


Figure 6.9: Hard Accuracy vs Hard Agreement for AGLS on SVHN(5%)

Chapter 7

Conclusions and Future Work

In this thesis, we embarked on a comprehensive exploration of the critical concepts of calibration, agreement, and confidence in the context of deep neural networks (DNNs) and ensembles. Our journey began with the recognition of the remarkable success of DNNs in various machine-learning tasks, which has been tempered by concerns regarding their reliability and robustness in real-world applications. The need for well-calibrated confidence estimates emerged as a central challenge, leading us to delve into the intricate interplay between these fundamental concepts.

We started by investigating the concept of calibration, unraveling its various notions, techniques, and methods of evaluation. Our examination revealed that calibration can take on different forms, each with unique characteristics and purposes, from top-class calibration to class-aggregated calibration. This understanding laid the groundwork for further exploration.

Our thesis extends on the phenomenon known as Generalization Disagreement Equality (GDE), which demonstrates that even a pair of models within an ensemble could be well-calibrated. We extended upon GDE to introduce the concept of Generalized Generalization Disagreement Equality (GGDE), showing its applicability and validity through empirical evidence. This extension offers greater flexibility in the ensemble hypothesis space, transcending the confines of hard classifiers.

Building on the foundations laid by GDE and GGDE, we formulate the intriguing concept of Confidence Calibration, which aligns with top-class calibration but introduces novel and desirable properties. This notion's connection with Label Smoothing added a new dimension to our understanding of the interplay between calibration and regularization techniques.

Our research culminated in the development of the Agreement Guided Label Smoothing (AGLS) training algorithm, leveraging the insights gained from GGDE, Confidence Calibration, and Label Smoothing. AGLS, utilizing an **unlabeled validation set**, provides an effective and practical means of enhancing the reliability and generalization performance of DNNs, eliminating the need for tedious hyperparameter tuning of the label smoothing hyperparameter.

The contributions of this thesis can be summarized in three key points:

- We extended and validated the concept of GGDE, offering a more flexible approach to ensemble calibration.
- We introduced Confidence Calibration, shedding new light on the relationship between calibration and regularization techniques.
- We presented AGLS as a practical training algorithm, utilizing Confidence Calibration and GGDE to improve DNN reliability and generalization. AGLS while only requiring an unlabelled validation set outperforms hyperparameter-tuned LS that requires a labeled validation set.

This thesis sheds light on some important questions and provides a foundation for future works. Firstly, the phenomenon that GGDE holds closely for SGD-trained neural networks remains intriguing and requires further investigation. Secondly, exploring the potential exploitation of GGDE phenomenon in existing training techniques that use labelled validation data. Finally, to investigate if GDE/GGDE is data dependent, specifically investigating whether there exists a threshold for the amount of data (training set size) below which GDE/GGDE ceases to manifest – essentially, determining the minimum amount of data (training set size) required to observe GDE/GGDE.

Our thesis not only advances the understanding of Confidence Calibration but also contributes to the broader goal of making deep learning models more trustworthy and applicable in various domains. By providing a structured exploration of calibration, agreement, and confidence, this thesis enhances the foundations of deep ensemble techniques, promoting their robustness and reliability in real-world applications.

As we conclude this journey, we envision the continued evolution of deep ensembles and calibration techniques, which will play a pivotal role in ensuring the trustworthiness and applicability of deep learning models in the ever-expanding landscape of machine learning.

References

[1] Fareed Ahmad, Amjad Farooq, Muhammad Usman Ghani, et al. Deep ensemble model for classification of novel coronavirus in chest x-ray images. *Computational intelligence and neuroscience*, 2021, 2021.

[2] Babak Alipanahi, Andrew Delong, Matthew T Weirauch, and Brendan J Frey. Predicting the sequence specificities of dna-and rna-binding proteins by deep learning. *Nature biotechnology*, 33(8):831–838, 2015.

[3] Yu Bai, Song Mei, Huan Wang, and Caiming Xiong. Don't just blame over-parametrization for over-confidence: Theoretical analysis of calibration in binary classification. In *International Conference on Machine Learning*, pages 566–576. PMLR, 2021.

[4] Peter L Bartlett and Shahar Mendelson. Rademacher and gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research*, 3(Nov):463–482, 2002.

[5] Heinz H. Bauschke and Jonathan M. Borwein. On projection algorithms for solving convex feasibility problems. *SIAM Review*, 38(3):367–426, 1996.

[6] 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*, pages 9368–9377, 2018.

[7] Luís Felipe P Cattelan and Danilo Silva. Improving selective classification performance of deep neural networks through post-hoc logit normalization and temperature scaling. *arXiv preprint arXiv:2305.15508*, 2023.

[8] A Philip Dawid. The well-calibrated bayesian. *Journal of the American Statistical Association*, 77(379):605–610, 1982.

[9] Stanislav Fort, Huiyi Hu, and Balaji Lakshminarayanan. Deep ensembles: A loss landscape perspective. *arXiv preprint arXiv:1912.02757*, 2019.

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

[11] Mudasir A Ganaie, Minghui Hu, AK Malik, M Tanveer, and PN Suganthan. Ensemble deep learning: A review. *Engineering Applications of Artificial Intelligence*, 115:105151, 2022.

[12] Yonatan Geifman and Ran El-Yaniv. Selective classification for deep neural networks. *Advances in neural information processing systems*, 30, 2017.

[13] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International conference on machine learning*, pages 1321–1330. PMLR, 2017.

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

[15] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.

[16] Yiding Jiang, Vaishnavh Nagarajan, Christina Baek, and J Zico Kolter. Assessing generalization of SGD via disagreement. In *International Conference on Learning Representations*, 2022.

[17] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

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

[19] Anders Krogh. What are artificial neural networks? *Nature biotechnology*, 26(2):195–197, 2008.

[20] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

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

[22] Alexander LeNail. Nn-svg: Publication-ready neural network architecture schematics. *Journal of Open Source Software*, 4(33):747, 2019.

[23] Shuying Liu and Weihong Deng. Very deep convolutional neural network based image classification using small training sample size. In *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, pages 730–734, 2015.

[24] Colin McDiarmid et al. On the method of bounded differences. *Surveys in combinatorics*, 141(1):148–188, 1989.

[25] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[26] Jishnu Mukhoti, Andreas Kirsch, Joost van Amersfoort, Philip HS Torr, and Yarin Gal. Deep deterministic uncertainty: A simple baseline. *arXiv preprint arXiv:2102.11582*, 2021.

[27] Jishnu Mukhoti, Viveka Kulharia, Amartya Sanyal, Stuart Golodetz, Philip Torr, and Puneet Dokania. Calibrating deep neural networks using focal loss. *Advances in Neural Information Processing Systems*, 33:15288–15299, 2020.

[28] Allan H Murphy and Edward S Epstein. Verification of probabilistic predictions: A brief review. *Journal of Applied Meteorology and Climatology*, 6(5):748–755, 1967.

[29] Mahdi Pakdaman Naeini, Gregory Cooper, and Milos Hauskrecht. Obtaining well calibrated probabilities using bayesian binning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 29, 2015.

[30] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[31] 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*, pages 427–436, 2015.

[32] Jeremy Nixon, Michael W Dusenberry, Linchuan Zhang, Ghassen Jerfel, and Dustin Tran. Measuring calibration in deep learning. In *CVPR workshops*, volume 2, 2019.

[33] John Platt et al. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74, 1999.

[34] 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*, pages 2818–2826, 2016.

[35] TagX. Data augmentation for computer vision, Jun 2022.

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

[37] David Widmann, Fredrik Lindsten, and Dave Zachariah. Calibration tests in multi-class classification: A unifying framework. *Advances in neural information processing systems*, 32, 2019.

[38] Xixin Wu and Mark Gales. Should ensemble members be calibrated? *arXiv preprint arXiv:2101.05397*, 2021.

[39] Joseph Awoamim Yacim and Douw Gert Brand Boshoff. Impact of artificial neural networks training algorithms on accurate prediction of property values. *Journal of Real Estate Research*, 40(3):375–418, 2018.

[40] Bianca Zadrozny and Charles Elkan. Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers. In *Icml*, volume 1, pages 609–616, 2001.

[41] Bianca Zadrozny and Charles Elkan. Transforming classifier scores into accurate multiclass probability estimates. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 694–699, 2002.

[42] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[43] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

[44] Dengyong Zhou, Olivier Bousquet, Thomas Lal, Jason Weston, and Bernhard Schölkopf. Learning with local and global consistency. *Advances in neural information processing systems*, 16, 2003.

[45] Jian Zhou and Olga G Troyanskaya. Predicting effects of noncoding variants with deep learning–based sequence model. *Nature methods*, 12(10):931–934, 2015.