UPPSALA
UNIVERSITET

Estimating Uncertainty of Deep Learning Models

Implementation and Performance Analysis

Markus Sagen, Jiahao Lu, Jianbo Li

Project in Computational Science: Report

January 2020

PROJECT REPORT

Department of Information Technology

Abstract

Although modern deep learning models are often highly e□ective, such e□ectiveness comes at the cost of being overconfident of its own predictions. Quantifying and calibrating models' uncertainty is critical for real-world applications, such as medical imaging or self-driving cars. Calibration of predictive uncertainty for deep learning has been an active research area in recent years.

In this paper, we implement a framework to intensively evaluate five state-of-the-art approaches, including Label Smoothing, Monte Carlo Dropout, Concrete Dropout, Variational Inference, Temperature Scaling, and some of their variants and combinations to reach well-calibrated deep learning models for image classification task. Two models, LeNet and ResNet, are trained on MNIST and an oral cancer dataset. The evaluation metrics include calibration error, classification accuracy, processing time and memory usage. Our results indicate that the combination of Label Smoothing and Temperature Scaling performs best on both accuracy and calibration, with very little extra cost in time and memory. Our code is available as open source[1].

Keywords: Bayesian deep learning, certainty, calibration

Acknowledgement

We would like to thank our supervisors Joakim Lindblad and Nataša Sladoje for their valuable feedback and support.

Contact Details

Anyone interested in our work, results or source code is welcome to contact us on one of the following e-mails addresses:

Markus Sagen: Markus.John.Sagen@gmail.com
Jiahao Lu: Jiahao.Lu.2199@student.uu.se
Jianbo Li: Jianbo.Li.4196@student.uu.se

[1] https://github.com/Noodles-321/Certainty

Contents

1 Introduction

In today's world, an increasing number of systems guide and automate our decisions based on algorithms of varying complexity. Many of these systems require high accuracy and performance, where the so-called deep neural networks (DNNs) have excelled over other models and have proven effective in a vast number of distinct areas, such as object detection [1], image classification [2], speech recognition [3], automated driving [4], and medical diagnosis [5]. In particular, deep learning (DL) has become an integral part of many decision-making pipelines. However, when making decisions in the real world for use in critical applications, accurate and well-calibrated measures of models' uncertainty is crucial. Being able to reliably measure the uncertainty of model's decisions, such that the predicted accuracy of the models always correlates with the confidence in the models' prediction is called a well-calibrated model [6, 7].

In recent years, calibration of predictive uncertainty for DNNs has been an active research area. The probabilistic approach uses Bayesian statistics for estimating the predictive posterior distribution of the model. But since the true posterior is often intractable due to the complexity of modern neural networks, approximation methods have been proposed. Variational inference (VI) methods were some of the earliest successful ones for computing an approximate posterior for Bayesian neural networks (BNNs). Stochastic variational inference (SVI) [8] improved stochastic gradient descent (SGD) by introducing natural gradient from information geometry. However, this can greatly increase the number of parameters and time for training. Recently, the so-called Flipout approach is proposed ([9]) to efficiently decorrelate the gradients within a mini-batch. By implicitly sampling pseudo-independent sample-wise weight perturbations, it claims to achieve high speedups in the training of fully connected networks and convolutional networks.

On the other hand, Monte Carlo Dropout (Drop) [10] shows that dropout layers [11] staying active at testing phase can approximate the complicated sampling in variational inference methods. By ensemble of the prediction results under multiple dropout masks, uncertainty can be estimated without sacrificing accuracy or computational complexity. But to obtain well-calibrated uncertainty estimates, dropout rate must be properly decided. Concrete Dropout (CDrop) [12], as a variant of Drop, addresses this problem by a continuous relaxation of dropout's discrete masks. It allows automatic tuning of the dropout rate towards better calibration.

The calibration of confidence can also be a post-processing step after a model is trained. Temperature scaling (TS) [13] learns a single corrective multiplicative scalar from the logits on the validation set, and applies it at testing phase. It achieves better-calibrated predictions without changing test accuracy, as shown in many evaluation experiments [14–16]. Furthermore, Attended-NLL [17] is proposed as an improved loss function for TS. Recently, utilising unlabelled test samples, Unsupervised Temperature Scaling (UTS) [18] allows the calibration done in an unsupervised manner, and the prediction adjusted towards the distribution of test data.

Since the methods to achieve calibrated certainty estimates are conducted at different components of the DL models, as far as we know, there is currently no work on a rigorous and empirical comparison of these methods. In this paper, we implement a framework to evaluate different methods. Five state-of-the-art methods are implemented within this framework, together with several evaluation metrics. They are evaluated with two network architectures on two datasets for DL-based image classification task. Furthermore, we also show that the methods can be easily combined within the framework. The performance of their combinations are also evaluated.

2 Background and Related Work

2.1 Introducing Uncertainty into a Deep Learning Model

An inherent limitation with deep conventional neural networks, is the deterministic approach to the model prediction. Each weight and bias in the network is asserted to have a specific value, meaning much information about the parameters and subsequently the model is not utilised. A natural approach to introducing uncertainty into a neural network is to view the model from a Bayesian approach. In Bayesian statistics, all parameters in a model are viewed, not as deterministic, but rather as random variables drawn from an underlying probability distribution [19]. This means that the model parameters can be one of many possible values and that the parameters describe an inherent measure of uncertainty.

Instead of inferring point estimates of the parameters, Bayesian statistics attempts to infer the full posterior distribution of the parameters. This means that all weights in a Bayesian deep neural network are modelled as a probability density function. The uncertainty of an approximate Bayesian model, given a prior distribution over its parameters, is given by calculating the variance and mean of the predictive posterior distribution $p(\mathbf{y}^* \mid \mathbf{x}^*)$ [20]. However, since the posterior distribution is intractable, approximate methods, such as variational inference or Markov Chain Monte Carlo (MCMC) methods are used [21].

Calibration is the process of making a model well calibrated. Measuring calibration is to measure the reliability of the model's confidence in its own prediction. For instance, given 100 predictions, if the model assigns a class with 70% probability, then we expect that class to appear in the prediction 70% of the time. More formally, for a classification model with class labels $y \in \{0, 1, \cdots, K-1\}$ and a predicted class probability $\hat{p}$ (confidence) for each labels [22], then the model is said to be calibrated if

$$\mathbb{P}\Big(\hat{Y} = y \mid \hat{p} = p\Big) = p, \quad \forall p \in [0, 1], \tag{1}$$

where $\hat{Y}$ is the class label prediction and $\hat{p}$ is the confidence in that label being correctly classified. Their difference indicates the calibration error. If there is no calibration error for any class label y and corresponding class probability $\hat{p}$, then the model is considered perfectly calibrated [23].

2.2 Label Smoothing (LS)

Labels represent the ground truth of classes in classification tasks. Label smoothing [24] is a way to soften labels and extend one-hot encoding. In one-hot encoding, for each training data x, we have its ground-truth label y, and the probability which calculated by the model for each label is $k \in \{1 \dots K\}$, where the ground-truth distribution over labels is $q(k|x)$. Hence, we have $\sum_k q(k|x) = 1$, this means that the sum of the probabilities for all labels is 1. We want to ensure that for all labels, only the label probability corresponding to the ground truth label y is active and all others are inactive:

$$\begin{cases} q(k) = 1, \text{ for all } k = y \\ q(k) = 0, \text{ for all } k \neq y \end{cases} \tag{2}$$

The equation above corresponds to a Dirac delta function $\delta_{k,y}$, such that $q(k) = \delta_{k,y}$. The label distribution can then be expressed as [25]:

$$q(k|x) = \delta_{k,y}. \tag{3}$$

However, this formulation only allows the value of a given label to have a probability of ones and zeros. This restriction leads to several problems:

1. The model is more prone to overfit, since the assumption that the samples from training and testing sets are independent and identically distributed does not reflect the reality. If the model learns to assign full probability to the ground-truth label for each training example, it is not guaranteed to generalise [25].

2. The model also learns inherent noise.

3. For some fuzzy examples, the model can not make clear distinctions, since much information about label probabilities are lost. For instance, assuming a model is trained to distinguish between the length of people as being either $short$ or $tall$, if the network is given as an input the image of a average height person, the current model would classify it as either a short or tall person.

To mitigate these issues, we can use label smoothing. Introducing a smoothing parameter ϵ with $\epsilon \in [0, 1]$, the label distribution is reformulated as:

$$q'(k|x) = (1 - \epsilon)\delta_{k,y} + \epsilon u(k), \tag{4}$$

where $u(k)$ is a fixed distribution over labels. If we use the uniform distribution $u(k) = 1/K$, then the label smoothing formulation is equivalent to a regularisation factor. Note that a label smoothing factor of $\epsilon = 0$ is equivalent to using regular label smoothing. When using label smoothing,

$$q'(k) = (1 - \epsilon)\delta_{k,y} + \frac{\epsilon}{K}, \tag{5}$$

even if $k \neq y$, $q(k)$, the value is no longer zeros, but rather a value determined by the number of labels K, and the smoothing parameter ϵ. This is equivalent to adding noise to label y, which can prevent the model from making over-concentrating predictions on the label with higher probabilities. Empirically, it has been shown that label smoothing can e□ectively prevent the model from becoming overconfident, e.g., [24].

2.3 Monte Carlo Dropout (Drop)

Monte Carlo Dropout [10] is a method to approximate a BNN, using a regular NN with dropout active during testing. It has become a commonly used baseline method when measuring model uncertainty. The reason for this is because the method does not require a fully BNN-model structure, but rather approximates a Bayesian network; since minimising a network with a cross-entropy loss function and dropout applied after each layer is equivalent to minimising the KL-divergence [26, 27].

Because the methods allow for training on a deterministic DL model, and can be extended to an approximate BNN model with a few extra steps, MC Dropout has since its inception become one of the most commonly used calibration methods in comparative tests [23, 28].

The uncertainty estimate for the model, can be decomposed and calculated from the posterior distribution. The uncertainty, using approximate dropout method is thus given by using Equations (6) - (8):

$$\mathrm{Var}_{q(\mathbf{y}^*|\mathbf{x}^*)}\left[\mathbf{y}^*\right] = \underbrace{\mathbb{E}_{q(\mathbf{y}^*|\mathbf{x}^*)}[(\mathbf{y}^*)^T(\mathbf{y}^*)]}_{\text{Second raw momentum}} - \underbrace{\mathbb{E}_{q(\mathbf{y}^*|\mathbf{x}^*)}[\mathbf{y}^*]^T}_{\text{MC Dropout}^T}\underbrace{\mathbb{E}_{q(\mathbf{y}^*|\mathbf{x}^*)}[\mathbf{y}^*]}_{\text{MC Dropout}}. \tag{6}$$

The Monte Carlo averaging (MC dropout) is defined as the averaging of the model prediction over the weights L and the total number of samples T:

$$\mathbb{E}_{q(\mathbf{y}^*|\mathbf{x}^*)}\left[\mathbf{y}^*\right] \approx \frac{1}{T}\sum_{t=1}^{T}\hat{\mathbf{y}}_t^*, \tag{7}$$

and the second raw momentum for the uncertainty estimate is defined as

$$\mathbb{E}_{q(\mathbf{y}^*|\mathbf{x}^*)}[(\mathbf{y}^*)^T(\mathbf{y}^*)] \approx \tau^{-1}\mathbf{I}_D + \frac{1}{T}\sum_{t=1}^{T}(\hat{\mathbf{y}}_t^*)^T(\hat{\mathbf{y}}_t^*). \tag{8}$$

At the evaluation phase, random nodes are dropped with a fixed dropout rate and vary from each time the model is testing. By setting each weight in the network as either active (1) or dropped (0), the drawn samples of active or dropped weights over several samples, model a Bernoulli distribution over each weight [10]. Thus, Monte Carlo dropout or dropout during the test phase can be seen as an approximate Bayesian method.

2.4 Concrete Dropout (CDrop)

Concrete dropout [12] is an extension of the Monte Carlo dropout method [10]. In the original model, MC dropout uses a fixed dropout rate, but to obtain well-calibrated uncertainty estimates, a grid-search over all possible dropout probabilities are commonly used. Concrete dropout, however, uses a continual relaxation of the dropout's discrete masks to obtain an optimal dropout rate for each layer [12, 29]. This allows for a dynamically updated dropout rate, which can be especially useful when training models, where the amount of available training data changes in each iteration. Empirically, it has been shown that concrete dropout used on when trained on MNIST performs equally well as hand-tunes parameters or MC dropout with grid-search [12].

2.5 Variational Inference with Flipout (VI)

Variational Inference methods use a proposal distribution q, which is easier to calculate than a target distribution p, which usually is the posterior distribution. When the difference between q and p is small, q can be used as the approximate distribution of p. To measure the difference between these two distributions, we use the Kullback-Leibler (KL) divergence. The KL divergence is a non-negative measure between two distributions. When two distributions are equivalent, the KL divergence is equal to 0. The KL divergence can be defined as [30]:

$$\begin{aligned}\theta^\star &= \arg\min_\theta \mathrm{KL}[q(\mathbf{w}|\theta)\|P(\mathbf{w}|\mathcal{D})] \\ &= \arg\min_\theta \int q(\mathbf{w}|\theta)\log\frac{q(\mathbf{w}|\theta)}{P(\mathbf{w})P(\mathcal{D}|\mathbf{w})}\mathrm{d}\mathbf{w} \\ &= \arg\min_\theta \mathrm{KL}[q(\mathbf{w}|\theta)\|P(\mathbf{w})] - \mathbb{E}_{q(\mathbf{w}|\theta)}[\log P(\mathcal{D}|\mathbf{w})],\end{aligned} \tag{9}$$

where $P(w|D)$ is the posterior distribution of the weights w given the training data D. Our goal is to finds the parameters θ in the proposal distribution for the weights, called the variational parameter $q(w|\theta)$, such that it minimises the KL divergence. Through the above steps, we turned

the problem of finding the posterior distribution into an optimisation problem. To make this tractable and feasible to optimise in neural networks, an alternative function is often used called evidence lower bound (ELBO, $\mathcal{L}$) [31]. Maximising the ELBO is equivalent to minimising the KL divergence. Though, KL divergence is not the only measure between distributions that can be used, it is the most common because of the speed of computation [32].

Usually, when training a neural network, updating of the weights is done using weight perturbation. This technique, however, has an inherent high variance for the gradient estimates, since all training examples in a mini-batch share the same perturbation and can lead to correlations between gradients. Flipout [9] is a methods proposed to mitigate this problem. Flipout is a gradient estimator, which applied to variational inference tasks that can minimise the KL divergence. The biggest advantage of using Flipout with VI methods, is that it can greatly decrease the training time, while guaranteeing a reduce variance.

2.6 Temperature Scaling (TS)

Temperature scaling [23] is a post-processing calibration method, which adds a new scalar parameter T to the original SoftMax function, called temperature. The temperature scaling function is defined as

$$\begin{aligned} \sigma(\mathbf{z}_i)^{(k)} &= \frac{e^{z_i^{(k)}}}{\sum_{j=1}^{K} e^{z_i^{(j)}}}, \\ \hat{q}_i &= \max_k \sigma(\mathbf{z}_i/T)^{(k)}, \end{aligned} \tag{10}$$

where $\hat{q}_i$ is a calibrated probability based on the network's predicted probability $\sigma\left(z_i\right)$. σ is a sigmoid function, z is the logits (inputs to the SoftMax function), and T is the learned scalar. The index i represents the sample and k represents the class. In Equation (10), if the temperature T is equal to 1, the temperature scaling function equals a regular SoftMax. Depending on the value of the scalar T, the temperature scaling will either yield a smoother SoftMax output (if T > 1) or sharper (if 1 > T > 0) [33], meaning it raises or lowers the output entropy. If $T \to \infty$, then $\hat{q}$ equals $\frac{1}{K}$ and a maximum uncertainty of the labels is obtained. We learn the scalar parameter T during training on a validation set, by minimising the NLL. Once the scalar T is calculated, it is used in the temperature scaling function during training instead of SoftMax.

Since the parameter T in temperature scaling does not change the maximum of a SoftMax function, the original class prediction is still the same, thus evaluating with temperature scaling will yield the same accuracy [23] as evaluating with SoftMax.

One of the benefits of temperature scaling is that it provides reliable model calibration ([23]) or knowledge distillation ([33]) and is also a so called post-processing methods. This means it can be used in combination with other methods – achieving calibration with just one extra parameter and much reduced training time, compared to other calibration methods.

3 Data and Evaluation Methods

We compare several existing state-of-the-art methods for calibrating predictive uncertainty, by building a DL based framework for achieving calibrated probabilities and varying each component in the DL framework. This process is made in order to verify which components in combination yield the most well-calibrated models and most reliably. A model is considered well-calibrated when model accuracy equals the model confidence for all possible values. The components of the DL framework and its various combinations are built upon one of two neural network architectures: LeNet or ResNet50. The framework is divided into four subcategories, as depicted in Figure 1. A fifth step is also used for evaluating the model, which is not depicted in the figure.

1. Datasets used (see Section 3.1)
2. Pre-processing methods (see Section 2.2)
3. BNN models or probabilistic calibration methods (see Sections 2.3 - 2.5)
4. Post-processing calibration methods (see Section 2.6)
5. Plotting and calibration metrics (see Section 3.2)

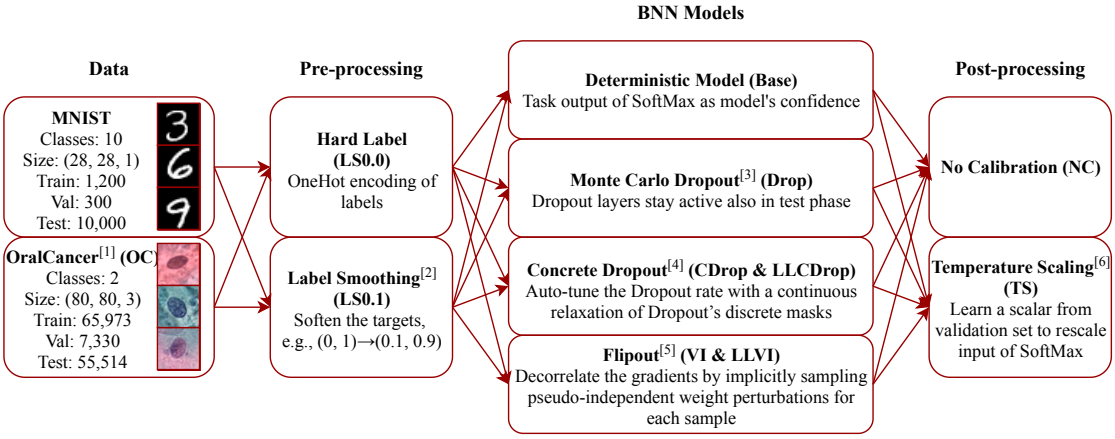


Figure 1: The structure of our DL framework

3.1 Datasets

This section describes the two datasets used in the DL framework. The first dataset, MNIST is used as a baseline dataset to test the calibration methods. The second dataset, the oral cancer dataset, is the dataset of primary importance to train and evaluate our model against.

3.1.1 MNIST

The MNIST dataset is a broadly used image dataset of grey scale handwritten digits, commonly utilised for testing image classifiers. The input data has a dimension of 28x28 pixels and a total of ten output labels of numerical integers between 0 and 9. The dataset is subdivided into

60,000 training images with labels and 10,000 test images with labels. This dataset is considered deeply understood and state-of-the-art deep CNNs achieves accuracy scores of 99% or above.

For the MNIST dataset, to restrain the accuracy of the models, we only use a subset of 1500 images, randomly sampled from the original training set. 300 images from the 1500 are randomly chosen as a validation set, and the remaining 1200 images are used for training. The test set contains the original 10000 images.

3.1.2 Oral Cancer Data (OC)

The oral cancer dataset is a dataset of brush scraped samples from patient's oral cavities, scanned in and digitised by Folktandvården Stockholms län AB. The oral cancer dataset consists of 128,817, 80x80 colorized images (3 channels for RGB), split into training and test set of size 73,303 and 55,514 respectively. The dataset used is based on the **dataset 3**, described in [34]. The label of this dataset is described on the "glass level", i.e., if a glass is sampled from a patient with tumor, all cells from this glass are labeled as cancer cells.

For the OC dataset, 10% of the training set is randomly chosen as validation set. Since it is shown from previous studies that the nucleus texture is the key feature for classification [35], we convert the input RGB images to greyscale to avoid the influence of colors, $L = 0.229R + 0.587G + 0.114B$. In addition, the data is augmented without interpolation. Each image is reflected with 50% probability and rotated by angle $\phi \in \{0°, 90°, 180°, 270°\}$.

3.2 Plotting and Measuring Calibration

In order to quantify and compare calibration methods, we utilise various calibration metrics and methods to visualise calibration.

3.2.1 Reliability Diagrams

To visualise the relationship between the average accuracy and the average confidence, the so-called reliability diagram is often used. The difference between accuracy and confidence is called the accuracy gap. The x-axis represents confidence of the model and corresponds to the probability of the SoftMax output. The y-axis corresponds to the model accuracy. In a perfectly calibrated model, the confidence will always be the same as it's accuracy [23]. In our comparison, reliability diagrams are used to visually display the effect of different calibration methods.

In Figure 2, two binning approaches are displayed, uniform to the left and adaptive to the right. The red-coloured gap in the figure is the positive gap, which means our model has higher accuracy than confidence, in other words: it's an overconfident model. On the other hand, the yellow gap represents a negative gap, which means that model is under-confident. Figure 2 also illustrates some of the potential problems with using a uniform binning compared to an adaptive binning. From the calibration metrics we present below, both ECE and $AECE$ uses the reliability diagram for its binning strategy. ECE uses an uniform binning and $AECE$ and adaptive binning.

3.2.2 Calibration Metrics

NLL

Negative Log-Likelihood or Cross Entropy Loss is the most commonly used loss function for

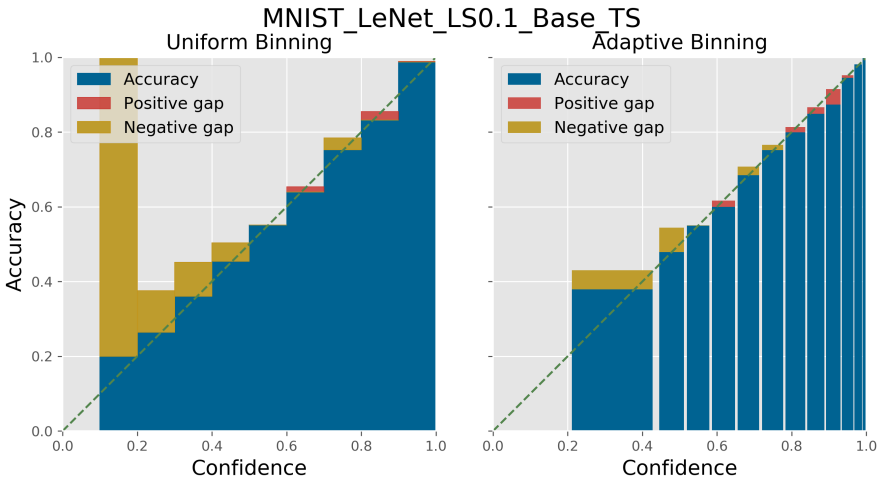


Figure 2: Reliability Diagram using uniform binning (left) and adaptive binning (right)

multi-class classification problems and measures the miss-classification, based on entropy. NLL extends BCE, by assuming that the number of classes K can be any integer value greater than one. Importantly, Negative Log Likelihood heavily penalises predictions that are confident, but wrong:

$$\text{NLL} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} y_{i,k}\ log(\hat{p}_{i,k}), \tag{11}$$

where y is the ground true class output, $\hat{p}$ the predicted probability of the label given by the model and N the number of training samples.

Brier Score
Calculating the Brier score is similar to calculating the NLL, with the difference being that is taking the mean squared error [36]. It is defined as

$$\text{Brier Score} = \frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K}\left(y_{i,k} - \hat{p}_{i,k}\right)^2. \tag{12}$$

ECE
In order to evaluate calibration, we partition predictions into M equally-spaced bins. The predicted value of each instance falls into one of these bins. We calculate the difference between accuracy and confidence for all bins and take the weighted average. This calibration method is called Expected Calibration Error (ECE) [23] and is formulated as

$$\text{ECE} = \sum_{m=1}^{M}\frac{|B_m|}{N}\left|\text{acc}\left(B_m\right) - \text{conf}\left(B_m\right)\right|. \tag{13}$$

MCE
In some cases, it is not as important to minimize Expected Calibration Error, but rather minimize

the worst-case deviation. That type calibration method is called Maximum Calibration Error (MCE) and is similar to ECE, but calculates the maximum calibration error instead of expected calibration error [23]. ECE is normally defined as

$$\mathrm{MCE} = \max_{m \in \{1,\ldots,M\}} \left|\mathrm{acc}\left(B_m\right) - \mathrm{conf}\left(B_m\right)\right|. \tag{14}$$

The reason we use ECE and MCE is because these metrics are very commonly used, which allows our experimental results to be compared with other researchers. These measures have some inherent problems, which is why we primarily use AECE and AMCE as our default calibration metrics.

AECE and AMCE

The accuracy of ECE and MCE are highly depending on the binning strategy and there are three main problems with conventional binning methods [37]:

1. Undetectable accuracy gap: Due to the uniform distribution of confidence on a large bin range, an undetectable accuracy gap may appear within a bin. For example, the total accuracy gap between [0.9,1] is small, but inside this bin the accuracy gap between [0.9,0.91] might be very large.

2. Internal compensation: Within a bin, the accuracy gap may have di□erent signs, that is, the confidence may be bigger than or less than the accuracy.

3. Inaccurate accuracy estimation: This maybe encountered when proposing solutions for the first two problems by increasing the number of bins. When our bins are many and there are not enough samples, then some bins may not have enough samples or no samples at all. This leads to inaccurate accuracy estimation. An example of this phenomena can be seen in Figure 2, where a uniform binning is used, but to few samples within that range can be obtained.

We use adaptive ECE (AECE) or adaptive MCE (AMCE), to tackle these issues. The process of assigning the correct number of samples in each bin is made according to the distribution of the samples and the specified confidence region. The number of samples n needed to estimate the accuracy in a given bin is given as

$$n = 0.25\left(\frac{Z_{\alpha/2}}{\epsilon}\right)^2, \tag{15}$$

where ϵ is the error margin and 1 - α is the confidence interval. This method, in comparison to other previously discussed binning methods, is highly robust with respect to the choice of hyper-parameters. During testing, the confidence range is initially set to 80%, yielding a value for $Z_{\alpha/2}$ as 1.645 according to the settings in [37]. The error margin ϵ is set to the width of the bins confidence range.

4 Experiment Setting

This section describes the tools used in the model framework, environmental settings and hyper-parameters. For an overview of the framework and its components, see Section 3 - Material and Methods.

4.1 Experiment Environment

The models are implemented using Tensorflow GPU version 1.14. The high-level API Keras, incorporated into Tensorflow, has been used for building most of the models [38]. Two base libraries Tensorflow and Tensorflow Probability are used to access low-level features and interact with the main models, such as inscribing probability distributions over weights or storing variables from previous iterations.

The configuration of our environment is as follows:

- CPU: Intel(R) Core(TM) i9-7980XE @ 2.60GHz
- GPU: NVIDIA GeForce RTX 2080 Ti
- OS: Linux 5.4.8-gentoo.

To run the current implementation of our code, a CUDA enabled GPU is required.

4.2 Implementation Details

The dataset available to run the framework for is either the MNIST or Oral cancer dataset. Depending on the dataset chosen, the hyperparameters used in the model di□er because of complexity, input dimension and available data for each dataset.

Models are trained with a mini-batch size of 512 for 40 epochs on MNIST, and with a mini-batch size of 256 for 50 epochs on the OC dataset. If the validation loss has not decreased for 5 epochs, the learning rate is scaled by a factor of 0.1. Early stopping was implemented to stop the training if no improvement has been made after 10 epochs. The checkpoint with minimum validation loss is saved for testing.

For all BNN models other than the deterministic model, the inference methods are run 20 times to sample from the distributions. The predicted logits for both validation and test set are the mean values of the 20 runs. Since the Base methods with LeNet and ResNet architectures both include a dropout layer with dropout rate $p = 0.5$ before the last dense layer, for Drop, the dropout rate p is also set to 0.5. For CDrop and VI, the methods are also implemented only on the last layer of the models to reduce computational and memory complexity, similar to [16], named LLCDrop and LLVI. For LLCDrop, the dropout layer in the deterministic base models is replaced with the CDrop layer. For LS, the smoothing factor is set to 0.1, as used in [24].

Categorical cross-entropy is used as the loss function for all models except for VI and LLVI, where KL divergence is also summed with the cross-entropy, as described in Section 2.5.

Evaluating the framework is done by running a bash script for the desired dataset (MNIST or Oral cancer). It then executes the framework by exchanging all possible permutations of network architectures, pre-processing methods, BNN models and post-processing methods (as shown in Figure 1). The command line options of the framework are shown in the Appendix.

5 Results and Discussion

In this section, we evaluate the results of each combination in Figure 1. However, since the total number of possible combinations is large, only a subset of interesting or best-performing methods are displayed. Though all models have been evaluated using the calibration metrics described in Section 3.2, only AECE is displayed in the tables below. The other evaluation metrics such as NLL, Brier, ECE and more are included in an extensive table of results, available in the Appendix. For each combination in the framework, classification accuracy, training time and testing time are also measured using the settings described in Section 4.

5.1 Sole Method Comparison

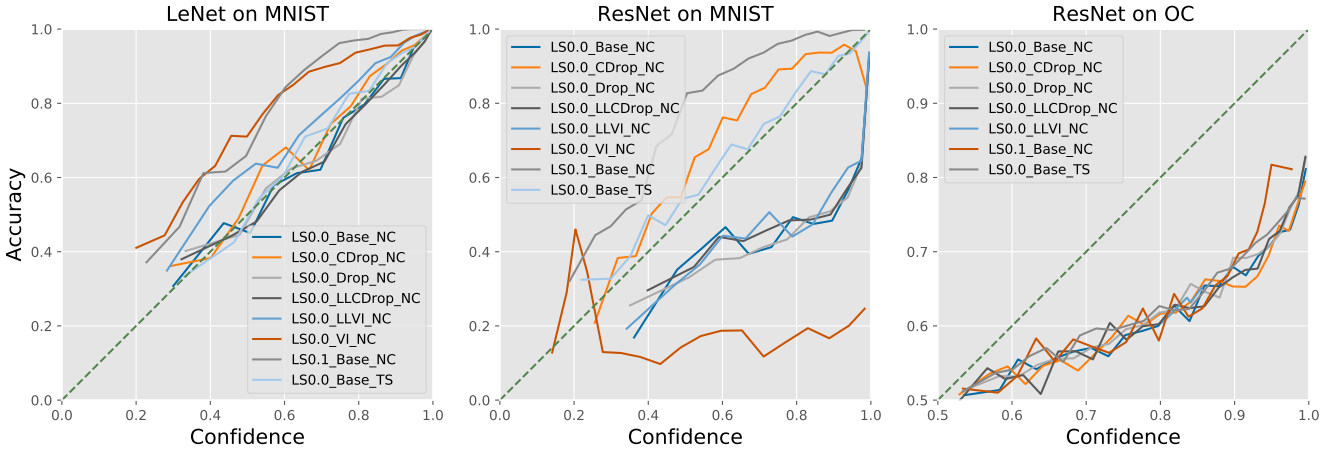


Figure 3: Sole method comparison. LLCDrop/LLVI denotes CDrop/VI applied only on the last layer.

Results of each method working alone are presented in Figure 3, including LeNet on MNIST, ResNet on MNIST, and ResNet on OC. Since LeNet appears to be too simple for the OC dataset, only ResNet is applied. From Figure 3a and Figure 3b, we can see that BNN models do not improve calibration significantly. Since LeNet appears to be initially well calibrated enough, it is not easy to see how much the methods can improve from Figure 3a.

Comparing Figure 3a with Figure 3b, we can also see that ResNet tends to be worse calibrated than LeNet. This is consistent with the conclusion in [23], where the model's calibration is suggested to be correlated to its width, depth and the type of regularisation techniques used.

To better illustrate Figure 3b, Table 1 is presented to show the quantitative results of ResNet on MNIST. Although CDrop and VI tend to mitigate the overconfidence problem, they increase the number of parameters and make the model harder to training. As we can see from Table 1, the training of VI even failed, resulting in unacceptably low accuracy. Compared with using full CDrop and VI, applying these methods only on the last dense layer reduces training and testing time by around 50%.

As Table 1 shows, TS achieves the lowest AECE 0.022, suggesting the best-calibrated certainty

Label Smoothing	Method	Calibration	Accuracy	AECE	Train time (s/epoch)	Test time (s)
LS0.0	Base	NC	0.801	0.145	**21.61**	**1.14**
LS0.1	Base	NC	**0.841**	0.170	**21.57**	**1.13**
LS0.0	Base	**TS**	0.809	**0.022**	**21.35**	**1.14**
LS0.0	Drop	NC	0.823	0.133	**20.42**	10.97
LS0.0	CDrop	NC	0.778	0.096	41.73	22.77
LS0.0	LLCDrop	NC	0.821	0.134	21.95	11.72
LS0.0	VI	NC	0.211	0.232	48.86	23.51
LS0.0	LLVI	NC	0.822	0.125	23.05	11.57

Table 1: Sole method comparison of ResNet on MNIST

estimates. LS reaches the highest accuracy 0.841, improving the Base model by 0.04. Both TS and LS show very little difference in processing time. We consider these two methods promising and continue to evaluate how they work when combined with other BNN methods.

As we can see in Figure 3c, methods show little difference on OC. This might due to the domain shift in OC dataset, i.e., the training set and testing set are not exactly from the same distribution. Taking TS as an example, since the temperature scalar is trained on the validation set, which is randomly sampled from the training set, the calibration quality of testing set is unlikely to improve. Considering this, the remaining results on OC are only included in the Appendix.

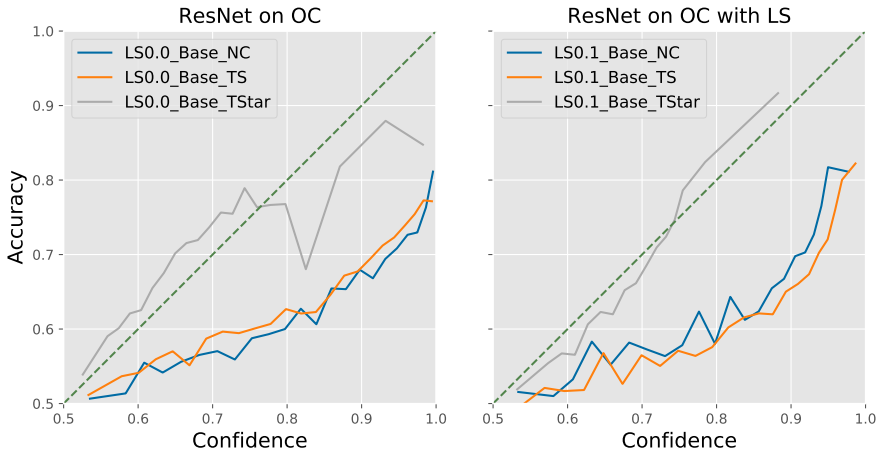
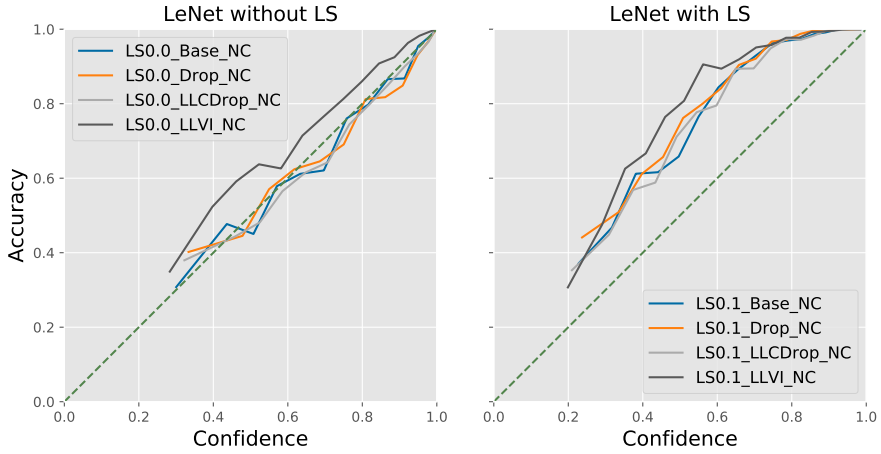
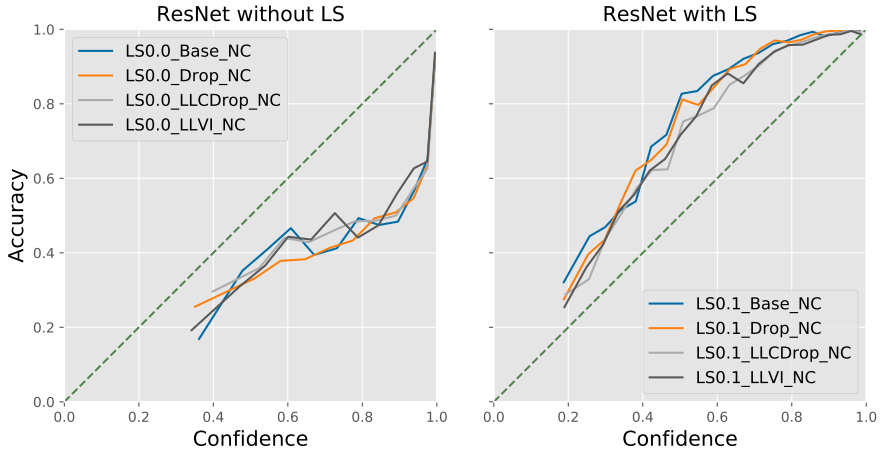


Figure 4: Possible improvements on OC. TStar denotes TS trained on the testing set.

We present Figure 4 to verify the domain shift problem mentioned above. “TStar” in the legends means the temperature scalar is trained on the target domain, i.e., testing set. In this case, as is shown in Figure 4, the lines of TStar become close to the diagonal ideal line again. This indicates the domain shift problem is the cause of ill-calibrated models on the OC dataset.



(a) LS combined with LeNet



(b) LS combined with ResNet

Figure 5: E□ect of combining LS with BNN models on MNIST

5.2 Combining LS with Other Methods

Based on the results from the previous Section 5.1, we choose four methods: Base, Drop, LLCDrop, and LLVI to combine with LS. CDrop and VI are not assessed further, considering their relatively low accuracy and long training/testing time. The results are tested on the models using LeNet and ResNet on MINST and are displayed in Figure 5 and Table 2.

As Table 2 shows, acting as a regularisation, applying LS improves accuracy. As Figure 5 shows, both LeNet and ResNet models become under-confident with LS applied, but whether it improves calibration or not depends on how well the models are initially calibrated. More specifically, the calibration became worse for LeNet, but better for ResNet.

Architecture	Label Smoothing	Method	Accuracy	AECE	Train time (s/epoch)	Test time (s)
LeNet	No	Base	0.932	0.008	1.04	0.17
		Drop	0.939	0.011	0.94	0.67
		LLCDrop	0.924	0.009	1.70	1.06
		LLVI	0.930	0.026	1.61	1.01
	Yes	Base	**0.943**	0.105	1.02	0.17
		Drop	**0.943**	0.120	0.92	0.67
		LLCDrop	**0.942**	0.102	1.66	1.02
		LLVI	**0.943**	0.135	1.59	0.98
ResNet	No	Base	0.801	0.145	21.61	1.14
		Drop	0.823	0.133	20.42	10.97
		LLCDrop	0.821	0.134	21.95	11.72
		LLVI	0.822	0.125	23.05	11.57
	Yes	Base	**0.841**	0.170	21.57	1.13
		Drop	**0.847**	0.161	21.47	11.34
		LLCDrop	**0.828**	0.134	21.77	11.53
		LLVI	**0.828**	0.132	22.55	11.64

Table 2: E□ect of combining LS with BNN models on MNIST

5.3 Combining TS with Other Methods

Similarly as above, we combine TS with the Base, Drop, LLCDrop, and LLVI to test their performance using the LeNet and ResNet architectures on MINST. The results are shown in Figure 6 and Table 3.

As we can see from Figure 6 and Table 3, when combined with TS, all calibration lines become close to the diagonal ideal line, and the calibration error AECEs were significantly reduced for all methods. This stands for both LeNet and ResNet architectures.

The little di□erence in accuracy between with/without TS for each BNN model in Table 3 is because that each combination is trained from start. The random initialisation can cause a little perturbation in accuracy. If trained a large number of times, the accuracy of with/without TS should have the same mean and variance values. This applies to results in all other tables.

5.4 Combining LS and TS with Other Methods

Based on the results from Section 5.2 and 5.3, we also evaluated the performance of combining both LS and TS with Base, Drop, LLCDrop, and LLVI. The results are tested by LeNet and ResNet on MINST and shown in Figure 7 and Table 4.

As shown in Figure 7 and Table 4, when TS applied with LS, all calibration lines become close to the diagonal ideal line again, and the calibration error AECEs is significantly reduced for all methods after they become under-confident under the e□ect of LS. This holds true for both LeNet and ResNet architectures. Compared with the base models in Table 3, the training/testing times remain almost unchanged.

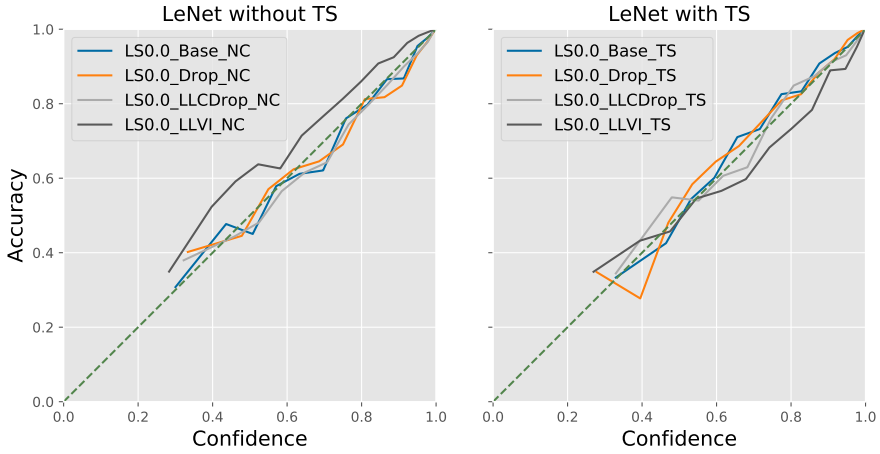
With all these observations, we can conclude that LS and TS can be simply combined with other

Architecture	Calibration	Method	Accuracy	AECE	Train time (s/epoch)	Test time (s)
LeNet	NC	Base	0.932	0.008	1.04	0.17
		Drop	0.939	0.011	0.94	0.67
		LLCDrop	0.924	0.009	1.70	1.06
		LLVI	0.930	0.026	1.61	1.01
	TS	Base	0.936	**0.006**	1.00	0.18
		Drop	0.936	**0.008**	0.94	0.66
		LLCDrop	0.945	**0.007**	1.66	1.02
		LLVI	0.929	**0.017**	1.57	0.98
ResNet	NC	Base	0.801	0.145	21.61	1.14
		Drop	0.823	0.133	20.42	10.97
		LLCDrop	0.821	0.134	21.95	11.72
		LLVI	0.822	0.125	23.05	11.57
	TS	Base	0.809	**0.022**	21.35	1.14
		Drop	0.776	**0.017**	21.80	11.45
		LLCDrop	0.802	**0.014**	22.01	11.67
		LLVI	0.804	**0.021**	22.58	11.60

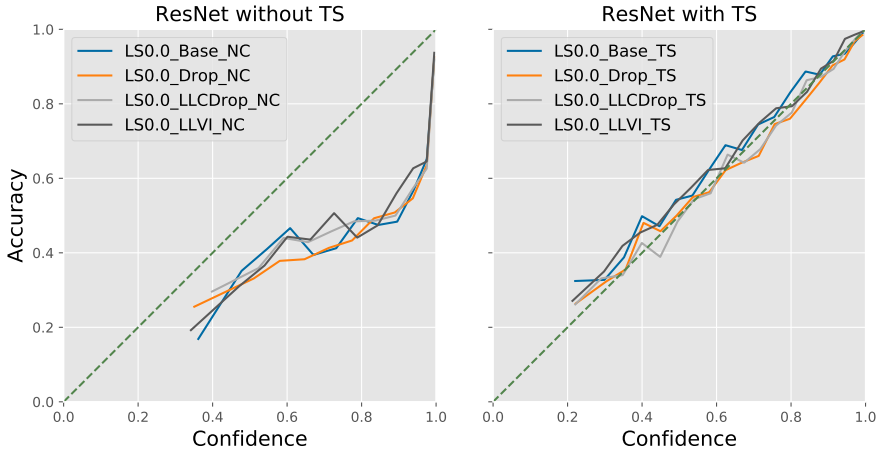
Table 3: E□ect of combining TS with BNN models on MNIST

Architecture	Calibration	Method	Accuracy	AECE	Train time (s/epoch)	Test time (s)
LeNet	NC	Base	0.943	0.105	1.02	0.17
		Drop	0.943	0.120	0.92	0.67
		LLCDrop	0.942	0.102	1.66	1.02
		LLVI	0.943	0.135	1.59	0.98
	TS	Base	0.947	**0.005**	1.01	0.17
		Drop	0.944	**0.007**	0.93	0.66
		LLCDrop	0.936	**0.006**	1.66	1.02
		LLVI	0.938	**0.004**	1.55	0.97
ResNet	NC	Base	0.841	0.170	21.57	1.13
		Drop	0.847	0.161	21.47	11.34
		LLCDrop	0.828	0.134	21.77	11.53
		LLVI	0.828	0.132	22.55	11.64
	TS	Base	0.820	**0.021**	21.78	1.14
		Drop	0.831	**0.016**	21.31	11.31
		LLCDrop	0.841	**0.031**	22.27	11.60
		LLVI	0.842	**0.012**	23.30	12.31

Table 4: E□ect of combining LS and TS with BNN models on MNIST. All methods here are applied with LS0.1.



(a) TS combined with LeNet

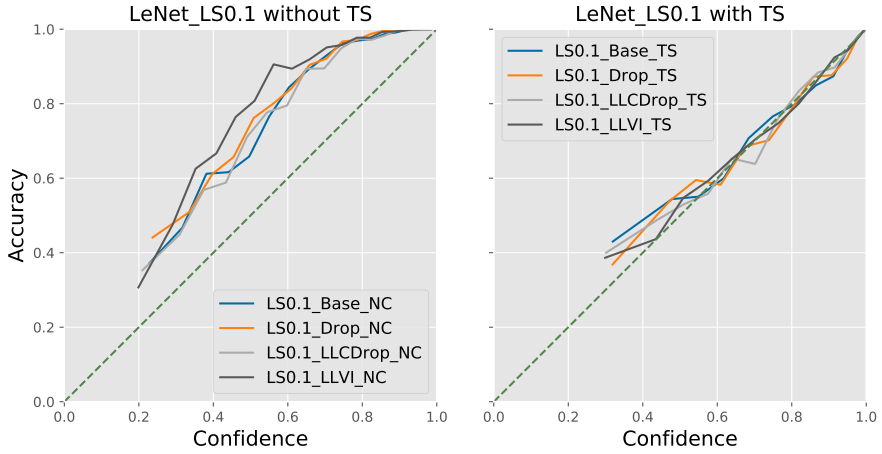


(b) TS combined with ResNet

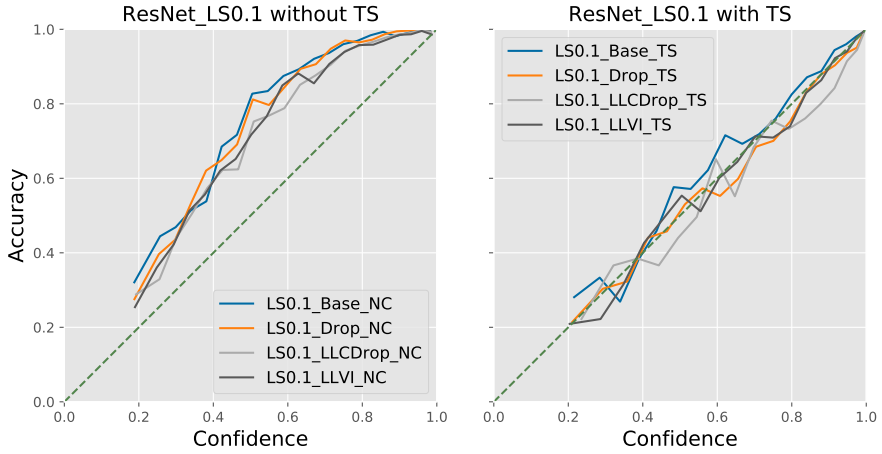
Figure 6: E□ect of combining TS with BNN models on MNIST

BNN models to benefit from both methods, i.e., we can achieve

1) an improvement in accuracy,
2) together with good calibration,
3) on almost every BNN models,
4) at almost no extra computational cost.



(a) LS and TS combined with LeNet



(b) LS and TS combined with ResNet

Figure 7: E□ect of combining LS and TS with BNN models on MNIST. All methods here are applied with LS0.1.

6 Future Work

The evaluated methods tested in this study did not show significant improvement on the calibration of OC dataset. Figure 4 in Section 5.1 indicates the direction of solving this problem, although training on the testing data is not applicable for real-world applications. The UTS [18] (proposed later than our implementation) showed in their results that using unlabelled test data to adjust the prediction towards the distribution of the target domain, the calibration line can result between the regular TS and TS trained on testing set. This can be used to improve the calibration on OC dataset.

On the other hand, some papers have suggested increased robustness and certainty calibration by training using adversarial examples [39], and others have shown plausible methods for detect adversarial examples with Bayesian neural networks [40, 41]. We believe both these aspects in

conjunction could make automatic classification safer and more robust for possible real-world usage [42].

Furthermore, we would like to extend the number of network architectures used and vary the depth of the CNNs, to ensure a better estimate of how well calibration methods generalise for different networks and depth. Finally, we would like to launch all permutations of pre-trained models as an interactive cloud-based solution with a web interface, where each feature can be toggled on or off. This would make the resulting interactions easier to compare and share with others.

7 Conclusion

We implemented a framework to intensively evaluate five state-of-the-art approaches and some of their variants to reach well-calibrated DL models for image classification task. The five methods are evaluated on two datasets, MNIST and OC, with two network architectures, LeNet and ResNet. Accuracy, ECE, MCE, AECE, AMCE, NLL, Brier score, training/testing time, and number of parameters are used as evaluation metrics. In addition, since the five methods apply on different stages of the model, we also explored combinations of these methods.

According to our results, TS reaches the best calibration results with very little computation cost. Three BNN models do not improve calibration significantly. LS can improve accuracy and meanwhile mitigate the overconfidence problem, but not guarantee well-calibrated models. LS and TS can be easily combined to benefit from both.

Our implemented framework enables modular replacement of DL models' calibration evaluation, including datasets, pre-processing methods, network architectures, post-processing methods, evaluation metrics and plotting options. Future new methods can be easily and fairly compared with the implemented ones.

References

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

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[3] G. Hinton, L. Deng, D. Yu, G. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, B. Kingsbury, et al. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine*, 29, 2012.

[4] B. Huval, T. Wang, S. Tandon, J. Kiske, W. Song, J. Pazhayampallil, M. Andriluka, P. Rajpurkar, T. Migimatsu, R. Cheng-Yue, et al. An empirical evaluation of deep learning on highway driving. *arXiv preprint arXiv:1504.01716*, 2015.

[5] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. Van Der Laak, B. Van Ginneken, and C. I. Sánchez. A survey on deep learning in medical image analysis. *Medical image analysis*, 42:60–88, 2017.

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

[7] M. P. Naeini, G. Cooper, and M. Hauskrecht. Obtaining well calibrated probabilities using bayesian binning. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[8] M. Ho□man, D. M. Blei, C. Wang, and J. Paisley. Stochastic Variational Inference. *arXiv:1206.7051 [cs, stat]*, Apr. 2013.

[9] Y. Wen, P. Vicol, J. Ba, D. Tran, and R. Grosse. Flipout: E□cient Pseudo-Independent Weight Perturbations on Mini-Batches. *arXiv:1803.04386 [cs, stat]*, Apr. 2018.

[10] Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning*, pages 1050–1059, 2016.

[11] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15:1929–1958, 2014.

[12] Y. Gal, J. Hron, and A. Kendall. Concrete dropout. In *Advances in Neural Information Processing Systems*, pages 3581–3590, 2017.

[13] J. Maroñas, R. Paredes, and D. Ramos. Calibration of deep probabilistic models with decoupled bayesian neural networks. *arXiv preprint arXiv:1908.08972*, Aug. 2019.

[14] W. Maddox, T. Garipov, P. Izmailov, D. Vetrov, and A. G. Wilson. A simple baseline for bayesian uncertainty in deep learning. *arXiv preprint arXiv:1902.02476*, 2019.

[15] S. AngelosFilos, A. Gomez, and T. J. Rudner. Benchmarking bayesian deep learning with diabetic retinopathy diagnosis, 2019.

[16] Y. Ovadia, E. Fertig, J. Ren, Z. Nado, D. Sculley, S. Nowozin, J. V. Dillon, B. Lakshminarayanan, and J. Snoek. Can You Trust Your Model's Uncertainty? Evaluating Predictive Uncertainty Under Dataset Shift. *arXiv preprint arXiv:1906.02530*, 2019.

[17] A. S. Mozafari, H. S. Gomes, W. Leão, S. Janny, and C. Gagné. Attended Temperature Scaling: A Practical Approach for Calibrating Deep Neural Networks. *arXiv:1810.11586 [cs, stat]*, May 2019.

[18] A. S. Mozafari, H. S. Gomes, and C. Gagne. A Novel Unsupervised Post-Processing Calibration Method for DNNS with Robustness to Domain Shift. *arXiv:1911.11195 [cs, stat]*, Nov. 2019.

[19] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015.

[20] K. Shridhar, F. Laumann, and M. Liwicki. A Comprehensive guide to Bayesian Convolutional Neural Network with Variational Inference. *arXiv:1901.02731 [cs, stat]*, Jan. 2019.

[21] Y. Kwon, J.-H. Won, B. J. Kim, and M. C. Paik. Uncertainty quantification using bayesian neural networks in classification: Application to biomedical image segmentation. *Computational Statistics & Data Analysis*, 142:106816, 2020.

[22] J. Nixon, M. Dusenberry, L. Zhang, G. Jerfel, and D. Tran. Measuring calibration in deep learning. *arXiv preprint arXiv:1904.01685*, 2019.

[23] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1321–1330. JMLR. org, 2017.

[24] R. Müller, S. Kornblith, and G. Hinton. When Does Label Smoothing Help? *arXiv:1906.02629 [cs, stat]*, June 2019.

[25] C. Szegedy, V. Vanhoucke, S. Io□e, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[26] Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning, 2015.

[27] T. Nair, D. Precup, D. L. Arnold, and T. Arbel. Exploring uncertainty measures in deep networks for multiple sclerosis lesion detection and segmentation. *Medical image analysis*, 59:101557, 2020.

[28] A. Kendall and Y. Gal. What uncertainties do we need in bayesian deep learning for computer vision? In *Advances in Neural Information Processing Systems*, pages 5574–5584, 2017.

[29] D. Molchanov, A. Ashukha, and D. Vetrov. Variational dropout sparsifies deep neural networks, 2017.

[30] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015.

[31] D. M. Blei, A. Kucukelbir, and J. D. McAuli□e. Variational Inference: A Review for Statisticians. *Journal of the American Statistical Association*, 112(518):859–877, Apr. 2017.

[32] G. Hinton and D. Van Camp. Keeping neural networks simple by minimizing the description length of the weights. In *in Proc. of the 6th Ann. ACM Conf. on Computational Learning Theory*. Citeseer, 1993.

[33] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[34] J. Lu, N. Sladoje, C. R. Stark, E. D. Ramqvist, J.-M. Hirsch, and J. Lindblad. A deep learning based pipeline for e□cient oral cancer screening on whole slide images, 2019.

[35] J. Jabalee, A. Carraro, T. Ng, E. Prisman, C. Garnis, and M. Guillaud. Identification of Malignancy-Associated Changes in Histologically Normal Tumor-Adjacent Epithelium of Patients with HPV-Positive Oropharyngeal Cancer, 2018.

[36] F. Pan, X. Ao, P. Tang, M. Lu, D. Liu, and Q. He. Towards reliable and fair probabilistic predictions: field-aware calibration with neural networks. *arXiv preprint arXiv:1905.10713*, 2019.

[37] Y. Ding, J. Liu, J. Xiong, and Y. Shi. Evaluation of neural network uncertainty estimation with application to resource-constrained platforms. *arXiv preprint arXiv:1903.02050*, 2019.

[38] F. Chollet et al. Keras. `https://keras.io`, 2015. , (Last Accessed on 21/11/2019).

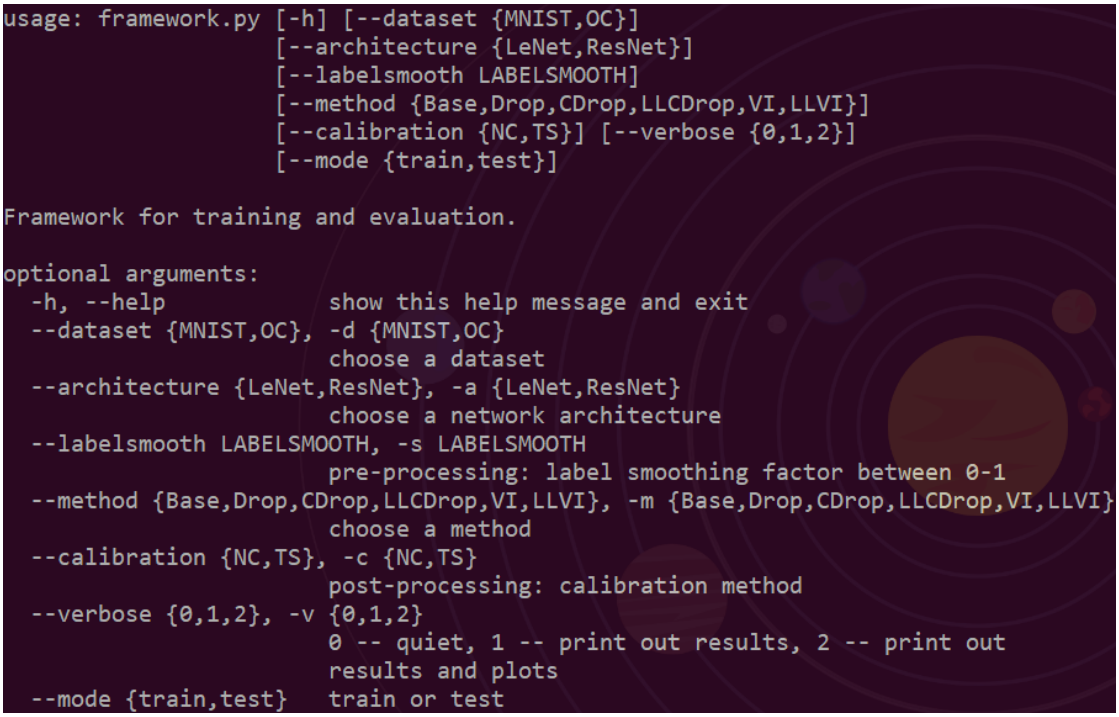
[39] D. Stutz, M. Hein, and B. Schiele. Confidence-calibrated adversarial training and detection: More robust models generalizing beyond the attack used during training, 2019.

[40] L. Cardelli, M. Kwiatkowska, L. Laurenti, N. Paoletti, A. Patane, and M. Wicker. Statistical guarantees for the robustness of bayesian neural networks. *arXiv preprint arXiv:1903.01980*, 2019.

[41] A. Rawat, M. Wistuba, and M.-I. Nicolae. Adversarial phenomenon in the eyes of bayesian deep learning. *arXiv preprint arXiv:1711.08244*, 2017.

[42] R. Michelmore, M. Wicker, L. Laurenti, L. Cardelli, Y. Gal, and M. Kwiatkowska. Uncertainty quantification with statistical guarantees in end-to-end autonomous driving control. *arXiv preprint arXiv:1909.09884*, 2019.

Appendix

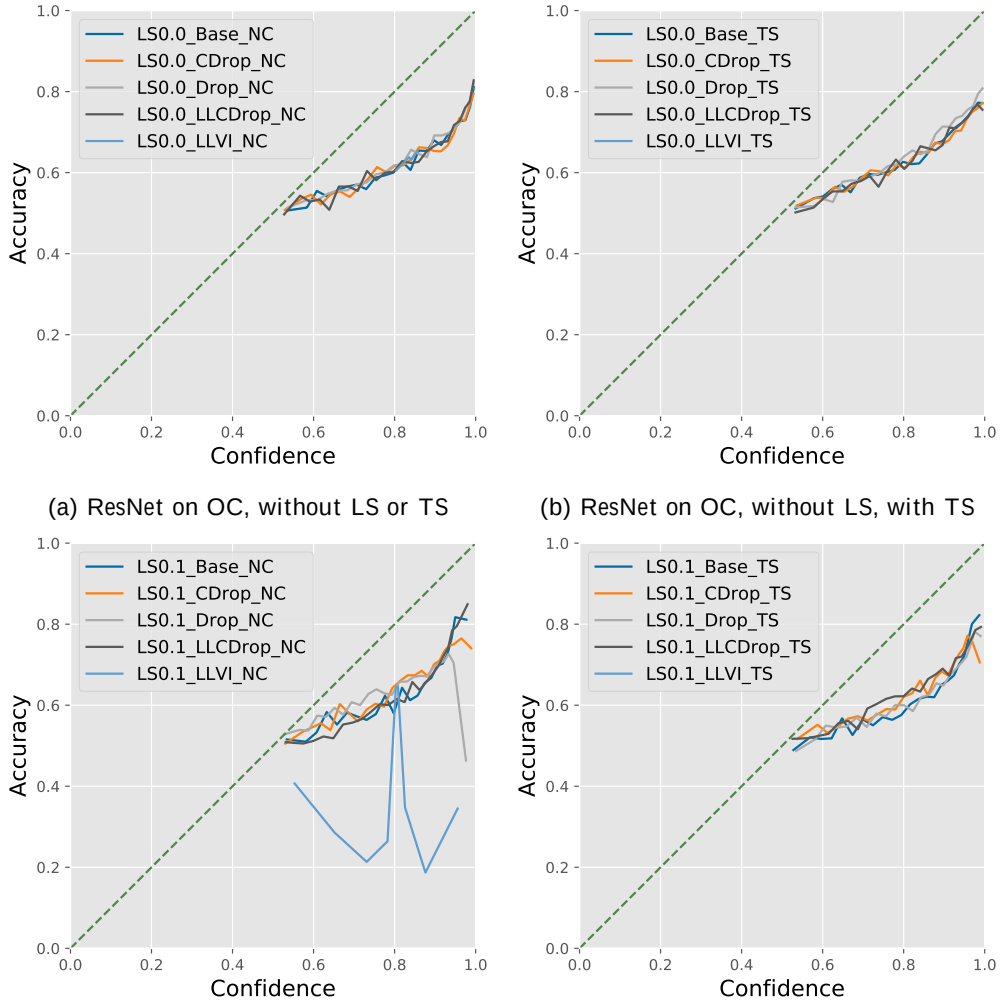


```
usage: framework.py [-h] [--dataset {MNIST,OC}]
                    [--architecture {LeNet,ResNet}]
                    [--labelsmooth LABELSMOOTH]
                    [--method {Base,Drop,CDrop,LLCDrop,VI,LLVI}]
                    [--calibration {NC,TS}] [--verbose {0,1,2}]
                    [--mode {train,test}]

Framework for training and evaluation.

optional arguments:
  -h, --help            show this help message and exit
  --dataset {MNIST,OC}, -d {MNIST,OC}
                        choose a dataset
  --architecture {LeNet,ResNet}, -a {LeNet,ResNet}
                        choose a network architecture
  --labelsmooth LABELSMOOTH, -s LABELSMOOTH
                        pre-processing: label smoothing factor between 0-1
  --method {Base,Drop,CDrop,LLCDrop,VI,LLVI}, -m {Base,Drop,CDrop,LLCDrop,VI,LLVI}
                        choose a method
  --calibration {NC,TS}, -c {NC,TS}
                        post-processing: calibration method
  --verbose {0,1,2}, -v {0,1,2}
                        0 -- quiet, 1 -- print out results, 2 -- print out
                        results and plots
  --mode {train,test}   train or test
```

Figure 8: Usage instruction of the framework.



(a) ResNet on OC, without LS or TS

(b) ResNet on OC, without LS, with TS

(c) ResNet on OC, with LS, without TS

(d) ResNet on OC, with LS and TS

Figure 9: E□ect of combining LS and TS with BNN models on OC. All methods here are based on the ResNet50 architecture.

Architecture	LabelSmooth	Method	Calibration	Accuracy	ECE	MCE	AECE	AMCE	NLL	Brier	t_train (s/epoch)	t_cal (s)	t_test (s)	Parameters (1e6)
LeNet	LS0.0	Base	NC	0.932	0.007	0.090	0.008	0.076	0.225	0.868	1.04	0.00	0.17	0.164
LeNet	LS0.0	Drop	NC	0.939	0.011	0.108	0.011	0.062	0.202	0.883	0.94	0.00	0.67	0.164
LeNet	LS0.0	CDrop	NC	0.936	0.009	0.101	0.009	0.090	0.215	0.855	2.73	0.00	1.58	0.164
LeNet	LS0.0	LLCDrop	NC	0.924	0.010	0.094	0.009	0.066	0.238	0.855	1.70	0.00	1.06	0.164
LeNet	LS0.0	VI	NC	0.864	0.121	0.276	0.121	0.257	0.509	0.579	2.81	0.00	1.65	0.261
LeNet	LS0.0	LLVI	NC	0.930	0.026	0.144	0.026	0.131	0.242	0.815	1.61	0.00	1.01	0.169
LeNet	LS0.0	Base	TS	0.936	0.006	0.142	0.006	0.056	0.202	0.860	1.00	0.07	0.18	0.164
LeNet	LS0.0	Drop	TS	0.936	0.008	0.092	0.008	0.120	0.199	0.857	0.94	0.08	0.66	0.164
LeNet	LS0.0	LLCDrop	TS	0.945	0.008	0.074	0.007	0.063	0.178	0.888	1.66	0.05	1.02	0.164
LeNet	LS0.0	LLVI	TS	0.929	0.017	0.193	0.017	0.085	0.234	0.875	1.57	0.07	0.98	0.169
LeNet	LS0.0	CDrop	TS	0.942	0.006	0.186	0.007	0.053	0.191	0.867	2.65	0.06	1.55	0.164
LeNet	LS0.1	Base	NC	0.943	0.105	0.241	0.105	0.242	0.266	0.720	1.02	0.00	0.17	0.164
LeNet	LS0.1	Drop	NC	0.943	0.120	0.241	0.120	0.254	0.283	0.696	0.92	0.00	0.67	0.164
LeNet	LS0.1	LLCDrop	NC	0.942	0.102	0.228	0.102	0.242	0.269	0.722	1.66	0.00	1.02	0.164
LeNet	LS0.1	LLVI	NC	0.943	0.136	0.324	0.135	0.343	0.312	0.675	1.59	0.00	0.98	0.169
LeNet	LS0.1	CDrop	NC	0.943	0.129	0.270	0.129	0.283	0.298	0.684	2.70	0.00	1.53	0.164
LeNet	LS0.1	Base	TS	0.947	0.005	0.801	0.005	0.065	0.174	0.888	1.01	0.05	0.17	0.164
LeNet	LS0.1	Drop	TS	0.944	0.006	0.274	0.007	0.059	0.182	0.885	0.93	0.07	0.66	0.164
LeNet	LS0.1	CDrop	TS	0.943	0.015	0.073	0.016	0.118	0.184	0.859	2.67	0.07	1.52	0.164
LeNet	LS0.1	LLCDrop	TS	0.936	0.005	0.199	0.006	0.064	0.204	0.866	1.66	0.06	1.02	0.164
LeNet	LS0.1	LLVI	TS	0.938	0.004	0.072	0.004	0.048	0.200	0.870	1.55	0.06	0.97	0.169
ResNet	LS0.0	Base	NC	0.801	0.145	0.373	0.145	0.414	1.190	0.768	21.61	0.00	1.14	23.575
ResNet	LS0.0	Drop	NC	0.823	0.133	0.334	0.133	0.393	1.176	0.796	20.42	0.00	10.97	23.575
ResNet	LS0.0	CDrop	NC	0.778	0.091	0.138	0.096	0.161	1.009	0.487	41.73	0.00	22.77	23.575
ResNet	LS0.0	LLCDrop	NC	0.821	0.134	0.362	0.134	0.393	1.127	0.792	21.95	0.00	11.72	23.575
ResNet	LS0.0	VI	NC	0.211	0.210	0.748	0.232	0.745	3.263	0.048	48.86	0.00	23.51	47.068
ResNet	LS0.0	LLVI	NC	0.822	0.125	0.371	0.125	0.370	0.998	0.788	23.05	0.00	11.57	23.596
ResNet	LS0.0	Base	TS	0.809	0.019	0.061	0.022	0.098	0.598	0.617	21.35	0.05	1.14	23.575
ResNet	LS0.0	Drop	TS	0.776	0.015	0.144	0.017	0.077	0.701	0.591	21.80	0.05	11.45	23.575
ResNet	LS0.0	LLCDrop	TS	0.802	0.008	0.185	0.014	0.061	0.612	0.629	22.01	0.07	11.67	23.575
ResNet	LS0.0	LLVI	TS	0.804	0.019	0.219	0.021	0.070	0.634	0.611	22.58	0.06	11.60	23.596
ResNet	LS0.0	CDrop	TS	0.839	0.056	0.153	0.060	0.153	0.644	0.615	44.97	0.06	22.91	23.575
ResNet	LS0.1	Base	NC	0.841	0.170	0.292	0.170	0.324	0.646	0.482	21.57	0.00	1.13	23.575
ResNet	LS0.1	Drop	NC	0.847	0.161	0.265	0.161	0.305	0.616	0.503	21.47	0.00	11.34	23.575
ResNet	LS0.1	LLCDrop	NC	0.828	0.134	0.218	0.134	0.244	0.643	0.507	21.77	0.00	11.53	23.575
ResNet	LS0.1	LLVI	NC	0.828	0.132	0.237	0.132	0.263	0.660	0.512	22.55	0.00	11.64	23.596
ResNet	LS0.1	CDrop	NC	0.459	0.219	0.671	0.219	0.677	1.872	0.058	44.27	0.00	22.44	23.575
ResNet	LS0.1	Base	TS	0.820	0.019	0.067	0.021	0.094	0.586	0.640	21.78	0.04	1.14	23.575
ResNet	LS0.1	Drop	TS	0.831	0.014	0.149	0.016	0.056	0.552	0.690	21.31	0.08	11.31	23.575
ResNet	LS0.1	CDrop	TS	0.782	0.050	0.812	0.057	0.103	0.743	0.526	44.09	0.06	22.31	23.575
ResNet	LS0.1	LLCDrop	TS	0.841	0.028	0.811	0.031	0.096	0.512	0.722	22.27	0.06	11.60	23.575
ResNet	LS0.1	LLVI	TS	0.842	0.009	0.177	0.012	0.068	0.510	0.704	23.30	0.06	12.31	23.596

Table 5: All results on MNIST using both LeNet and ResNet50

LabelSmooth	Method	Calibration	Accuracy	ECE	MCE	AECE	AMCE	NLL	Brier	t_train (s/epoch)	t_cal (s)	t_test (s)	Parameters (1e6)
LS0.0	Base	NC	0.682	0.190	0.220	0.190	0.247	0.907	0.568	93.14	0.00	11.04	23.559
LS0.0	CDrop	NC	0.667	0.191	0.232	0.191	0.264	0.932	0.542	136.95	0.00	46.51	23.559
LS0.0	Drop	NC	0.691	0.181	0.209	0.181	0.242	0.894	0.577	94.83	0.00	43.35	23.559
LS0.0	LLCDrop	NC	0.679	0.183	0.219	0.183	0.254	0.858	0.558	95.60	0.00	43.88	23.559
LS0.0	LLVI	NC	0.634	0.204	0.204	0.204	0.227	0.778	0.455	78.85	0.00	23.91	23.563
LS0.0	Base	TS	0.667	0.174	0.218	0.174	0.223	0.864	0.527	82.42	0.07	12.18	23.559
LS0.0	CDrop	TS	0.662	0.178	0.226	0.178	0.239	0.889	0.518	134.69	0.07	45.42	23.559
LS0.0	Drop	TS	0.686	0.162	0.197	0.162	0.208	0.812	0.547	82.30	0.07	27.55	23.559
LS0.0	LLCDrop	TS	0.668	0.180	0.221	0.180	0.239	0.877	0.532	82.80	0.05	27.72	23.559
LS0.0	LLVI	TS	0.634	0.056	0.770	0.057	0.476	0.665	0.337	77.95	0.07	22.75	23.563
LS0.1	Base	NC	0.684	0.168	0.219	0.168	0.233	0.736	0.538	96.68	0.00	11.15	23.559
LS0.1	CDrop	NC	0.665	0.160	0.197	0.160	0.254	0.769	0.500	137.23	0.00	46.61	23.559
LS0.1	Drop	NC	0.648	0.153	0.219	0.153	0.500	0.756	0.463	82.85	0.00	28.23	23.559
LS0.1	LLCDrop	NC	0.671	0.169	0.212	0.169	0.225	0.746	0.519	85.23	0.00	28.78	23.559
LS0.1	LLVI	NC	0.635	0.171	0.609	0.171	0.679	0.739	0.426	81.21	0.00	24.77	23.563
LS0.1	Base	TS	0.673	0.195	0.235	0.195	0.256	0.800	0.550	82.17	0.05	12.24	23.559
LS0.1	CDrop	TS	0.671	0.181	0.211	0.181	0.281	0.790	0.532	134.91	0.05	45.53	23.559
LS0.1	Drop	TS	0.668	0.198	0.233	0.198	0.250	0.833	0.546	82.19	0.09	27.49	23.559
LS0.1	LLCDrop	TS	0.683	0.177	0.208	0.177	0.241	0.796	0.552	82.57	0.08	27.75	23.559

Table 6: All results with ResNet50 on OC