Epistemic Uncertainty Quantification For Pre-trained Neural Networks

Supplementary Material

A. Proofs of Propositions

In this section, we will provide the proofs for all the propositions shown in Section 3.

A.1. Proof of Proposition 3.1

Proof. This proof is based on the Bernstein-von Mises theorem [10, 15]. Under mild regularity conditions, such as continuity of the likelihood function of θ and the maximum likelihood estimate θ^* not being on the parameter space boundary, the posterior distribution of θ converges in distribution to a multivariate Gaussian distribution as the sample size $|\mathcal{D}| \to \infty$. Specifically, we have:

$$\begin{aligned} &\sup_{\theta} |p(\theta|\mathcal{D}) - \mathcal{N}(\theta; \theta^*, |\mathcal{D}|^{-1} I(\theta^*)^{-1}))| \to 0 \\ &\text{where} \quad \theta^* = \arg\max_{\theta} \log p(\mathcal{D}|\theta) \end{aligned} \tag{14}$$

where $I(\theta^*)$ is the Fisher information matrix at θ^*. Assuming θ^* is bounded and the likelihood function at θ^* is differentiable and continuous, the Fisher information matrix is also bounded. This is because the bounded θ^* and the continuity of the likelihood function preclude unbounded derivatives. As $|\mathcal{D}| \to \infty$, we observe:

$$\| |\mathcal{D}|^{-1} I(\theta^*)^{-1} - \sigma^2 I \| \to 0 \tag{15}$$

for a sufficiently small σ approaching 0. Therefore, we conclude:

$$\sup_{\theta} |p(\theta|\mathcal{D}) - \mathcal{N}(\theta; \theta^*, \sigma^2 I)| \to 0 \tag{16}$$

as $|\mathcal{D}| \to \infty$ and $\sigma \to 0$. □

A.2. Proof of Proposition 3.2

Proof. We first introduce Lemma A.1, which is central to our proof.

Lemma A.1. *Denote* $v(\theta) = -\frac{1}{|\mathcal{D}|} \log p(\theta) - \frac{1}{|\mathcal{D}|} \sum_{(x,y)\in\mathcal{D}} \log p(y|x,\theta)$ *with* $p(\theta)$ *as a pre-defined prior distribution. Under the regularity constraints on* $v(\theta)$ *from Sec. 2.2 of [14], the total variation distance* $D_{TV}(p(\theta|\mathcal{D}), \mathcal{N}(\theta; \theta^*, -H^{-1}))$ *between the posterior distribution* $p(\theta|\mathcal{D})$ *and its Laplacian approximation* $\mathcal{N}(\theta; \theta^*, -H^{-1})$ *fulfills*

$$\begin{aligned} &|D_{TV}(p(\theta|\mathcal{D}), \mathcal{N}(\theta; \theta^*, -H^{-1})) - L| \\ &\leq c(c_3^2(v) + c_4(v)) \frac{d^2}{|\mathcal{D}|} \end{aligned} \tag{17}$$

$H(\theta^*) = \nabla^2_{\theta^*} \log p(\theta^*|\mathcal{D})$ represents Hessian matrix. $L \in [0, c_3(v)\frac{d}{\sqrt{|\mathcal{D}|}}]$ is an explicit function of $|\mathcal{D}|$. d is the dimension of θ, c is an absolute constant, and $c_3(v), c_4(v)$ are constants computed from third/fourth-order derivatives of v.

The definition of $c, c_3(v), c_4(v)$ can be directly found in [14]. This lemma is directly obtained from [14]. It is worth noting that $\mathcal{N}(\theta; \theta^*, -H^{-1})$ is the Laplacian approximation of $p(\theta|\mathcal{D})$. According to Sec. 3.3 of [34], the Hessian matrix $-H$ can be well-approximated by the dataset fisher $|\mathcal{D}|F^{\mathcal{D}}_{\theta^*}$. For a detailed discussion, see [34]. This shows that $-H^{-1} \to \mathbf{0}$ when $|\mathcal{D}| \to \infty$. Lemma A.1 further enables:

$$\begin{aligned} &\mathrm{D_{TV}}(p(\theta|\mathcal{D}), \mathcal{N}(\theta; \theta^*, -H^{-1})) \\ &\leq c(c_3^2(v) + c_4(v)) \frac{d^2}{|\mathcal{D}|} + |c_3(v)| \frac{d}{\sqrt{|\mathcal{D}|}} \end{aligned} \tag{18}$$

Then, based on Lemma A.1, we have

$$\begin{aligned} &\mathrm{D_{TV}}(p(\theta|\mathcal{D}), N(\theta; \theta^*, \sigma^2 I)) = \sup_{\theta} |p(\theta|\mathcal{D}) - \mathcal{N}(\theta; \theta^*, \sigma^2 I)| \\ &\leq \sup_{\theta} \left| p(\theta|\mathcal{D}) - \mathcal{N}(\theta^*, -H^{-1}) \right| \\ &+ \sup_{\theta} \left| \mathcal{N}(\theta^*, -H^{-1}) - \mathcal{N}(\theta^*, \sigma^2 I) \right| \\ &\leq c(c_3^2(v) + c_4(v)) \frac{d^2}{|\mathcal{D}|} + |c_3(v)| \frac{d}{\sqrt{|\mathcal{D}|}} \text{(Lemma A.1)} \\ &+ \sqrt{\frac{1}{2} \mathrm{KL}(\mathcal{N}(\theta^*, \sigma^2 I) || \mathcal{N}(\theta^*, -H^{-1})} \text{(Pinsker's inequality [3])} \end{aligned} \tag{19}$$

□

A.3. Proof of Proposition 3.3

Proof. When perturbations $\Delta\theta$ and Δx are both small, we can perform the first-order Taylor expansion as follows:

$$\begin{aligned} f(x, \theta + \Delta\theta) &= f(x, \theta) + \left(\frac{\partial f(x, \theta)}{\partial \theta} \right)^T \Delta\theta \\ f(x + \Delta x, \theta) &= f(x, \theta) + \left(\frac{\partial f(x, \theta)}{\partial x} \right)^T \Delta x. \end{aligned} \tag{20}$$

It is worth noting that $f(x, \theta + \Delta\theta) = f(x + \Delta x, \theta)$ requires

$$\left(\frac{\partial f(x, \theta)}{\partial \theta} \right)^T \Delta\theta = \left(\frac{\partial f(x, \theta)}{\partial x} \right)^T \Delta x \tag{21}$$

which generally holds true when $\Delta\theta \to 0$ and $\Delta x \to 0$. Furthermore, for any small $\Delta\theta$ that is not near zero, there

also likely exists a solution to Eq. (21) when Δx is treated as an unknown. However, this solution may not be unique. This situation arises because the matrix $\left(\frac{\partial f(x,\theta)}{\partial x}\right)$ is often of full rank, given that the dimension of f is usually much lower than that of x and θ. Additionally, the gradient $\frac{\partial f(x,\theta)}{\partial x}$ tends to be noisy within the context of neural networks.

Similarly, for any perturbation Δx, we can always find a corresponding $\Delta\theta$ to satisfy Eq. (21). More specifically, we only need to find a perturbation on the first layer parameters.

Let's denote $\theta = \{\theta^l\}$ where θ^l represents the parameters of the lth layer of the neural network. Without loss of generality, we can assume θ^1 is linearly connected with x without the bias variable, i.e.,

$$h_j^1 = \text{ReLU}((\theta^1)^T x) = \text{ReLU}(\sum_k \theta_{kj}^1 x_k) \tag{22}$$

where h_j^1 is the jth neuron in the first hidden layer. x_k is the kth element of x and θ_{kj}^1 is the corresponding weight linearly connecting x_k to h_j^1. Applying corresponding perturbations $\Delta\theta_{kj}^1$ and Δx_k leads to

$$\sum_k (\theta_{kj}^1 + \Delta\theta_{kj}^1)x_k = \sum_k \theta_{kj}^1 (x_k + \Delta x_k). \tag{23}$$

The sufficient and necessary condition is

$$\Delta\theta_{kj}^1 x_k = \theta_{kj}^1 \Delta x_k \tag{24}$$

which shows that we can find the corresponding first-layer parameter perturbations to mimic the perturbations on the input space. □

A.4. Proof of Proposition 3.4

Proof. **1. Global optimality of θ^***

Based on the Bernstein-von Mises theorem [10, 15] and Proposition 1, we know that with a sufficiently large number of training samples, the posterior distribution of parameters approximates a Gaussian distribution. The mean of this Gaussian distribution is the maximum likelihood estimate θ^*, and the covariance matrix approaches $\sigma^2 I$ with σ sufficiently small and approaching 0. It's important to note that this proposition primarily focuses on the neighborhood of an in-distribution x rather than the entire training set $\mathcal{D}$. In contrast, our focus is on an unlimited number of training samples in the neighborhood of x, denoted as $\mathcal{D}(x)$, which is well-represented by the classification model θ^*. By applying Proposition 1 directly, we obtain:

$$\sup_\theta |p(\theta|\mathcal{D}(x)) - \mathcal{N}(\theta; \theta^*, \sigma^2 I)| \to 0 \tag{25}$$

where $\theta^* = \arg\max_\theta \log p(\mathcal{D}(x)|\theta)$. As the posterior distribution $p(\theta|\mathcal{D}(x))$ is a Gaussian distribution with a single mode at θ^*, it follows that θ^* is the global optimum.

2. The gradients are close to 0

Given $f(x, \theta^*) = \log p(y|x, \theta^*)$ for classification problems, we aim to prove:

$$\begin{aligned} &\frac{\partial \log p(y|x,\theta^*)}{\partial \theta^*} = 0 \\ &\frac{\partial \log p(y|x+\Delta x,\theta^*)}{\partial \theta^*} = 0 \qquad \forall x + \Delta x \in \mathcal{N}(x). \end{aligned} \tag{26}$$

However, we only know that θ^* is the maximum likelihood estimate (MLE) given all the samples in the neighborhood of x. Since θ^* is the MLE, we have:

$$\frac{\partial \log p(D(x)|\theta^*)}{\partial \theta^*} = \frac{\partial \sum_{\Delta x} \log p(y|x+\Delta x,\theta^*)}{\partial \theta^*} = 0. \tag{27}$$

We assume the neighborhood of x is small (Δx is small) and they share the same label y. Without loss of generality, we also consider that the neighborhood of x is a ball centered at x and $D(x)$ is sufficiently large to cover all possible Δx in the ball. By performing the first-order Taylor expansion on $\log p(y|x+\Delta x, \theta^*)$ at x when Δx is small, we have

$$\begin{aligned} \log p(y|x+\Delta x,\theta^*) = &\log p(y|x,\theta^*) \\ &+ \left(\frac{\partial \log p(y|x,\theta^*)}{\partial x}\right)^T \Delta x. \end{aligned} \tag{28}$$

Combining Eq. (27) and Eq. (28), we can derive that

$$\begin{aligned} &\frac{\partial \sum_{\Delta x} \log p(y|x+\Delta x,\theta^*)}{\partial \theta^*} \\ &= \sum_{\Delta x} \frac{\partial \left(\log p(y|x,\theta^*) + \left(\frac{\partial \log p(y|x,\theta^*)}{\partial x}\right)^T \Delta x\right)}{\partial \theta^*} \\ &= \sum_{\Delta x} \frac{\partial \log p(y|x,\theta^*)}{\partial \theta^*} + \sum_{\Delta x} \frac{\partial \log p(y|x,\theta^*)}{\partial x \partial \theta^*} \Delta x \\ &= |D(x)| \frac{\partial \log p(y|x,\theta^*)}{\partial \theta^*} \\ &\quad + \frac{\partial \log p(y|x,\theta^*)}{\partial x \partial \theta^*} \left(\sum_{\Delta x} \Delta x\right) \\ &\xrightarrow{|\mathcal{D}(x)|\to\infty} |D(x)| \frac{\partial \log p(y|x,\theta^*)}{\partial \theta^*} = 0. \end{aligned} \tag{29}$$

It is important to note that the matrix $\frac{\partial \log p(y|x,\theta^*)}{\partial x \partial \theta^*}$ has a dimension $|\theta| \times |x|$, and $\sum_{\Delta x} \Delta x = 0$ since positive and negative perturbations negate each other. Eq. (29) indicates that

$$\frac{\partial \log p(y|x,\theta^*)}{\partial \theta^*} = 0. \tag{30}$$

Similarly,

$$\begin{aligned}
&\frac{\partial \log p(y|x+\Delta x,\theta^*)}{\partial \theta^*} \\
&= \frac{\partial \log p(y|x,\theta^*)}{\partial \theta^*} + \frac{\partial \log p(y|x,\theta^*)}{\partial x \partial \theta^*}\Delta x \\
&= \frac{\partial \log p(y|x,\theta^*)}{\partial \theta^* \partial x}\Delta x = \frac{\partial \frac{\partial \log p(y|x,\theta^*)}{\partial \theta^*}}{\partial x}\Delta x \\
&= \frac{\partial 0}{\partial x}\Delta x = 0.
\end{aligned} \tag{31}$$

This derivation relies on the assumption that the second partial derivatives of $\log p(y|x,\theta^*)$ are continuous, leading to the equality $\frac{\partial \log p(y|x,\theta^*)}{\partial \theta^* \partial x} = \frac{\partial \log p(y|x,\theta^*)}{\partial x \partial \theta^*}$ as per Clairaut's theorem. Clairaut's theorem states that for a function with continuous second partial derivatives, the differentiation order is immaterial. □

A.5. Proof of Proposition 3.5

Proof. When $\Delta\theta \to 0$ and by using Taylor expansion for $\log p(y|x,\theta^* + \Delta\theta)$ at θ^*, we can derive

$$\begin{aligned}
&\mathbb{E}_{p(\Delta\theta)}\left[\mathrm{KL}(p(y|x,\theta^*)||p(y|x,\theta^*+\Delta\theta))\right] \\
&= \mathbb{E}_{p(\Delta\theta)}\left[\sum_{c=1}^{C} p(y=c|x,\theta^*)\log\frac{p(y=c|x,\theta^*)}{p(y=c|x,\theta^*+\Delta\theta)}\right] \\
&= \mathbb{E}_{p(\Delta\theta)}\left[-\sum_{c=1}^{C} p(y=c|x,\theta^*)\left(\frac{\partial \log p(y=c|x,\theta^*)}{\partial \theta^*}\right)^T \Delta\theta\right] \\
&= \mathbb{E}_{p(\Delta\theta)}\left[\left\|-\sum_{c=1}^{C} p(y=c|x,\theta^*)\left(\frac{\partial \log p(y=c|x,\theta^*)}{\partial \theta^*}\right)^T \Delta\theta\right\|\right] \\
&= \mathbb{E}_{p(\Delta\theta)}\left[\left\|\sum_{c=1}^{C} p(y=c|x,\theta^*)\left(\frac{\partial \log p(y=c|x,\theta^*)}{\partial \theta^*}\right)^T \Delta\theta\right\|\right] \\
&\leq \mathbb{E}_{p(\Delta\theta)}\left[\sum_{c=1}^{C} p(y=c|x,\theta^*)\left\|\left(\frac{\partial \log p(y=c|x,\theta^*)}{\partial \theta^*}\right)^T\right\| \|\Delta\theta\|\right] \\
&= \sum_{c=1}^{C} p(y=c|x,\theta^*)\left\|\frac{\partial \log p(y=c|x,\theta^*)}{\partial \theta^*}\right\| \mathbb{E}_{p(\Delta\theta)}[||\Delta\theta||] \\
&\propto \mathbb{E}_{y\sim p(y|x,\theta^*)}\left[\left\|\frac{\partial \log p(y|x,\theta^*)}{\partial \theta^*}\right\|\right]
\end{aligned} \tag{32}$$

□

B. Experiment Settings and Implementation Details

B.1. The Pre-trained Model

While the pre-trained model could be more complex, we conduct experiments under the assumption of using a single deterministic neural network. Our goal is to demonstrate the effectiveness of our method in quantifying epistemic uncertainty, specifically within such a deterministic pre-trained network. We conducted experiments on four datasets: MNIST, C10, SVHN, and C100. For all training sessions, we randomly allocate 10% of the training data as validation data for model selection. We utilize an RTX2080Ti GPU to perform all the experiments. Below, we detail the training procedures for the pre-trained models corresponding to each of the four datasets.

- MNIST: We employ a simple CNN architecture: Conv2D-ReLU-Conv2D-ReLU-MaxPool2D-Dense-ReLU-Dense-Softmax. Each convolutional layer includes 32 filters with a 4×4 kernel size. We utilize a max-pooling layer with a 2×2 kernel and dense layers comprising 128 units. The SGD optimizer is used with a learning rate of 1e-2 and a momentum of 0.9. We set the maximum number of epochs at 30 and the weight decay coefficient at 5e-4. The batch size is 128.
- C10: We utilize ResNet18 for feature extraction, connected to a fully-connected layer for classification. The SGD optimizer is employed with an initial learning rate of 1e-1, decreasing to 1e-2, 1e-3, and 1e-4 at the 30th, 60th, and 90th epochs, respectively. The momentum is set at 0.9, with a maximum of 100 epochs and a weight decay coefficient of 5e-4. Standard data augmentation techniques, such as random cropping, horizontal flipping, and random rotation, are applied. The batch size is 128.
- C100: ResNet152 is used for feature extraction, connected to a fully-connected layer for classification. The SGD optimizer starts with an initial learning rate of 1e-1, decreasing to 1e-2, 1e-3, and 1e-4 at the 40th, 70th, and 100th epochs, respectively. We keep the momentum at 0.9 and set the maximum number of epochs to 120, with a weight decay coefficient of 5e-4. Similar to C10, standard data augmentation techniques are employed. The batch size for C100 is 64.
- SVHN: We use a CNN architecture: Conv2D-ReLU-Conv2D-ReLU-MaxPool2D-Dense-ReLU-Dense-Dense-ReLU-Softmax. Each convolutional layer has 32 filters with a 4×4 kernel size. A max-pooling layer with a 2×2 kernel is used, along with dense layers having 128 units. The SGD optimizer starts with an initial learning rate of 0.05, reduced to 0.005, and 0.0005 at the 15th and 30th epochs. We set the maximum number of epochs at 50 and the weight decay coefficient at 5e-4. The batch size is 64.

B.2. Implementation of Our Methods

- Class-specific Gradient Weighting: We compute gradients using torch.autograd, which facilitates straightforward backpropagation from the model outputs to the model parameters.
- Layer-selective Gradients: The hyperparameter λ was adjusted within the range of [0.1, 0.5], using increments of 0.05. Ultimately, the selected values of λ were de-

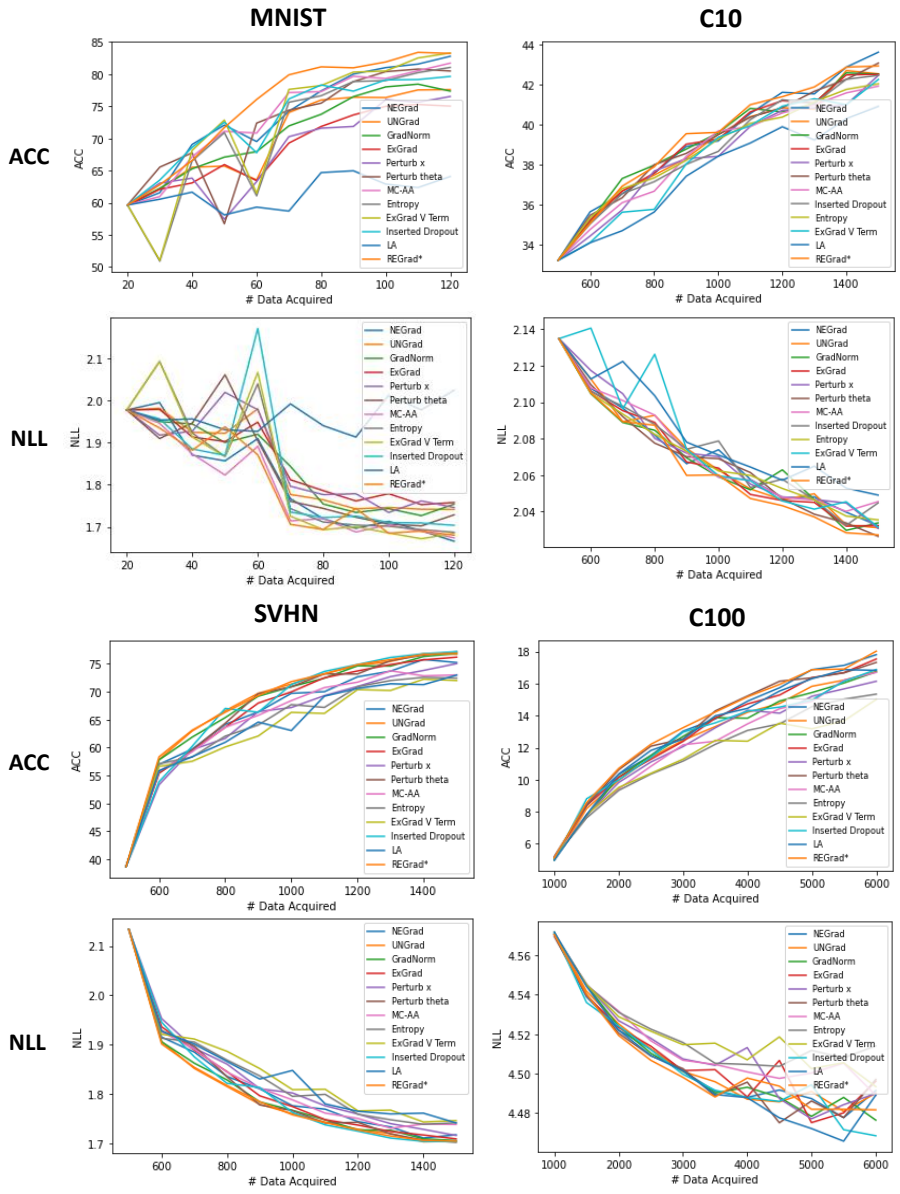

Figure 1. ACC and NLL for the MNIST, C10, SVHN, and C100 datasets across 10 active learning acquisition cycles are presented. In each figure, the x-axis represents the number of data samples acquired for training the model, while the y-axis shows either the accuracy (ACC) or the negative log-likelihood (NLL) on the testing data. The results are averaged over three independent runs.

termined as 0.3 for the MNIST dataset, 0.2 for the C10 dataset, 0.2 for the SVHN dataset, and 0.4 for the C100 dataset.

- Perturbation-Gradient Integration: Following the guidelines of Proposition 3.1, we selected a small value for the hyperparameter σ. It was tuned within the range [0.01, 0.05], in increments of 0.01. As a result, the value of σ was set at 0.02 for all four datasets. Additionally, to generate perturbed inputs, we utilized 100 samples.

B.3. Implementation of Baselines

The mathematical formulation of all baseline methods is discussed in Section 5 of the main paper. Below, we provide detailed descriptions of their implementations.

- NEGrad, UNGrad, GradNorm, ExGrad: These gradient-based methods are implemented using torch.autograd, and notably, they do not require any hyperparameter tuning. It's noteworthy that we experimented with both L1 and L2 norms for computing gradient norms, recording the best performing norm for each method.

- Perturb x: We introduce perturbations to the input x by adding the Gaussian noise $\mathcal{N}(0, \sigma^2 I)$. The hyperparameter σ is tuned in the range of [0.005,0.05]. Specifically, the values of σ are determined to be 0.008 for the MNIST, C10, and SVHN datasets, and 0.02 for the C100 dataset. The number of perturbed inputs is set to 100.
- Perturb θ: Perturbations are applied to all model parameters θ by adding the Gaussian noise from $\mathcal{N}(0, \sigma^2 I)$. The hyperparameter σ is tuned within the range of [0.0001,0.01]. For the MNIST, C10, and SVHN datasets, the optimal σ is found to be 0.008, while for C100 dataset, σ is set to be 0.0003. The number of perturbed inputs is set to 100.
- MC-AA: The MC-AA approach involves introducing perturbations to the input using adversarial attacks. These attacks are performed through the Fast Gradient Sign Method (FGSM), as described in the following equation:

$$x_{\text{adv}} = x + \epsilon \cdot \text{sign}(\nabla_x L(\theta^*, x, y)) \tag{33}$$

  In this equation, L represents the negative log-likelihood loss, utilizing the predicted class as the label. The hyperparameter ϵ indicates the perturbation level, where sign(u) is a function that outputs 1 if $u \geq 0$ and -1 if $u < 0$. To create multiple samples, we uniformly select ϵ from the range $[-a, a]$ with a being tuned in the range of [0.0001, 0.01]. Based on this tuning process, we set a at 0.0001 for MNIST, C10, and SVHN datasets, and at 0.0005 for the C100 dataset. The number of perturbations is set to 100.
- Inserted Dropout: In all pre-trained models, we incorporate an extra dropout layer just before the final fully connected layer. The dropout rate was subject to tuning, for which we explore values within the range [0.1, 0.5], using increments of 0.1. Ultimately, we set the dropout probability at 0.4 for all datasets.
- Entropy, ExGrad V Term: These can be directly calculated from the softmax output probabilities.
- LA: we employ the last-layer LA [16] with full Hessian matrix computation. We use the existing software proposed by [5], which is available at `https://github.com/AlexImmer/Laplace`. All hyperparameters are kept at their default settings as specified in the software.

B.4. Additional Information on Experiments

Uncertainty Calibration. The implementation of metric rAULC is available at `https://github.com/janisgp/practicality_deterministic_epistemic_uncertainty`.

Active Learning.

- Model Details: We utilize a simple CNN architecture for active learning across MNIST, C10, SVHN, and C100 datasets: Conv2D-ReLU-Conv2D-ReLU-MaxPool2D-Dense-ReLU-Dense-Softmax. Each convolutional layer includes 32 filters with a 4×4 kernel size. We utilize a max-pooling layer with a 2×2 kernel and dense layers comprising 128 units.
- Training Details: We employ the SGD optimizer, configured with a learning rate of 0.01 and a momentum of 0.9. The weight decay is set at 0.0005, and we use a batch size of 128. In each active learning cycle, the model is trained for up to 200 epochs to ensure convergence. This training approach is consistently applied across all four datasets. Additionally, during each active learning cycle, the best-performing model on the validation data is saved.
- Splitting Training and Validation Data: Initially, we randomly select the first m_1 training samples from indices 1 to 10,000 in the training dataset. We ensure a balanced selection where each class is represented by $\frac{m_1}{C}$ samples. For validation purposes, we use the data samples indexed from 10,001 to 20,000. This validation set is crucial for model selection during the 200 training epochs of each active learning cycle. Samples indexed above 20,000 in the training dataset are reserved in an "unused training pool", which serves as a source for acquiring new samples for subsequent retraining. In each cycle, we iteratively select m_2 samples from this pool based on the highest epistemic uncertainty. Upon completion of each training cycle, we assess the model's accuracy (ACC) and negative log-likelihood (NLL) using the original test dataset. We set $m_1 = 20, 500, 500, 1000$ and $m_2 = 20, 100, 100, 500$ for MNIST, C10, SVHN, and C100 datasets, respectively.

C. Additional Active Learning Results

This section presents the visualizations of ACC and NLL progression for all four datasets shown in Figure 1. We arrive at the same conclusion as indicated in Table 3: our proposed method consistently surpasses various baselines in achieving the highest accuracy and the lowest negative log-likelihood. For the MNIST dataset, the noticeable variance in the lines is likely attributed to the process starting with only 20 samples and acquiring just 10 more samples each time. This large variance could be due to the randomness inherent in the limited sample size and the training process itself. For averaged ACCs and NLLs across the four datasets, please refer to Table 3 in the main paper.

D. Additional Ablation Studies

D.1. Effectiveness of Component Contributions

Besides Table 4, we also provide additional results for OOD detection and uncertainty calibration to demonstrate the effectiveness of each component. These results, detailed in Tables 6, support similar conclusions to those in Section 5.4 of the main paper. Each component independently enhances performance, with the combination of all components yielding the best results. Although all

Table 6. The following two tables illustrate the effectiveness of component contributions for our method. The first table presents additional OOD detection results using AUROC (%) ↑ and AUPR (%) ↑. The second table displays uncertainty calibration results measured by rAULC. All experiments are aggregated over three independent runs.

Method	MNIST → Omniglot	C10 → LSUN	SVHN→ C100	C100→ LSUN
	AUROC/AUPR	AUROC/AUPR	AUROC/AUPR	AUROC/AUPR
ExGrad	97.55/96.99	88.23/82.26	88.57/85.55	76.93/71.00
REGrad	97.80/97.54	90.15/87.16	89.76/87.69	78.95/73.71
REGrad + layer-selective	98.01/97.86	90.79/88.86	90.07/88.11	79.01/**74.61**
REGrad + layer-selective + perturb	**98.19/97.95**	**91.11/89.93**	**90.37/88.83**	**79.11**/74.39

Method	MNIST	C10	SVHN	C100
	rAULC	rAULC	rAULC	rAULC
ExGrad	0.985	0.893	0.868	0.841
REGrad	0.985	0.893	0.869	0.846
REGrad + layer-selective	0.985	0.896	0.870	0.854
REGrad + layer-selective + perturb	**0.985**	**0.898**	**0.873**	**0.858**

Table 7. This table presents additional OOD detection results, highlighting the combination of layer-selective gradients and perturbation-integrated gradients with UNGrad and ExGrad. For UNGrad, the hyperparameters are set to $\lambda = 0.2, \sigma = 0.1$ for the SVHN dataset and $\lambda = 0.7, \sigma = 0.05$ for the C100 dataset. For ExGrad, $\lambda = 0.4, \sigma = 0.02$ are used for both the SVHN and C100 datasets. All experiments are aggregated over three independent runs.

Method	SVHN → C10	SVHN → C100	C100→ SVHN	C100→ LSUN
	AUROC/AUPR	AUROC/AUPR	AUROC/AUPR	AUROC/AUPR
UNGrad	68.28/69.21	68.34/68.95	44.34/43.44	47.46/45.41
UNGrad + layer-selective	75.41/75.52	75.49/**75.11**	82.44/75.87	73.47/65.53
UNGrad + layer-selective + perturb	**78.71/76.54**	**77.48**/74.98	**85.08/79.84**	**75.49/70.03**
ExGrad	88.57/85.55	88.25/85.21	79.99/72.86	76.93/71.00
ExGrad + layer-selective	89.57/87.76	88.54/86.66	80.91/73.90	76.79/70.74
ExGrad + layer-selective + perturb	**90.57/89.64**	**89.64/88.58**	**81.09/74.39**	**78.23/73.43**

components significantly improve OOD detection, the impact on uncertainty calibration becomes marginal. This may be partly due to the fact that a well-calibrated pre-trained model typically outputs predictive probabilities that closely match actual performance. By incorporating probability information through class-specific gradient weighting, our gradient-based method achieves good calibration performance, even without employing layer-selective and perturbation-integrated gradient strategies. This scenario presents a challenge in further enhancing calibration, particularly when focusing on pre-trained models. However, as model complexity increases, these models tend to be less calibrated. In such instances, employing layer-selective and perturbation-integrated gradient strategies becomes more advantageous for significant improvements. This is especially evident in the case of the C100 dataset.

We also demonstrate the effectiveness of layer-selective gradients and perturbation-integrated gradients when combined with other gradient-based methods such as ExGrad and UNGrad. The OOD detection results for the MNIST and C10 datasets are presented in Table 7. From these results, we can conclude that the two techniques we propose can effectively enhance the performance of other gradient-based methods.

D.2. Hyperparameter Analysis.

In addition to the analysis presented in Table 5, we further evaluate the impact of the coefficient λ in the layer-selective gradients and σ in the perturbation-integrated gradients for the MNIST, C10, and C100 datasets. These additional results are shown in Table 8, where we observe similar conclusions: the performance is not overly sensitive to the hyperparameters within certain ranges. It's also important to note that these hyperparameters were not determined through an exhaustive brute-force search of all possible parameter combinations. Furthermore, they were chosen based on their overall performance across various experimental tasks. While it's possible that some hyperparameters might perform better in specific cases, the current settings already provide the best performance in comparison to various baselines.

Table 8. OOD detection for hyperparameter analysis is presented using AUROC(%)/AUPR(%). A "***" symbol indicates the hyperparameters used in the main paper.

Method	MNIST→ Omniglot	MNIST→ FMNIST
REGrad + layer-selective ($\lambda \to 0$)	97.80/97.54	98.42/98.45
REGrad + layer-selective ($\lambda \to \infty$)	97.71/97.17	97.70/97.27
REGrad + layer-selective ($\lambda = 0.25$)	98.01/97.86	98.38/98.40
REGrad + layer-selective ($\lambda = 0.3$)	98.01/97.86	98.51/98.55
REGrad + layer-selective ($\lambda = 0.35$)	98.01/97.86	98.39/98.41
REGrad + layer-selective ($\lambda = 0.3$) + perturb ($\sigma = .015$)	98.13/97.91	98.74/98.76
REGrad + layer-selective ($\lambda = 0.3$) + perturb ($\sigma = .02$)***	**98.19/97.95**	98.78/98.79
REGrad + layer-selective ($\lambda = 0.3$) + perturb ($\sigma = .025$)	98.18/97.95	**98.87/98.88**

Method	C10→ SVHN	C10→ LSUN
REGrad + layer-selective ($\lambda \to 0$)	89.64/85.75	90.15/87.16
REGrad + layer-selective ($\lambda \to \infty$)	89.87/86.76	90.65/88.71
REGrad + layer-selective ($\lambda = 0.15$)	90.29/86.51	90.94/88.11
REGrad + layer-selective ($\lambda = 0.2$)	90.33/87.45	90.79/88.86
REGrad + layer-selective ($\lambda = 0.25$)	90.45/86.92	91.02/88.30
REGrad + layer-selective ($\lambda = 0.2$) + perturb ($\sigma = .01$)	**92.50**/89.08	91.01/89.03
REGrad + layer-selective ($\lambda = 0.2$) + perturb ($\sigma = .015$)	92.45/89.08	91.06/89.37
REGrad + layer-selective ($\lambda = 0.2$) + perturb ($\sigma = .02$)***	92.32/**89.42**	**91.11/89.93**

Method	C100→ SVHN	C100→ LSUN
REGrad + layer-selective ($\lambda \to 0$)	81.43/81.12	78.95/73.71
REGrad + layer-selective ($\lambda \to \infty$)	86.74/82.15	76.93/72.40
REGrad + layer-selective ($\lambda = 0.35$)	87.22/83.34	78.66/74.12
REGrad + layer-selective ($\lambda = 0.4$)	87.32/83.39	79.01/**74.61**
REGrad + layer-selective ($\lambda = 0.45$)	87.38/83.25	78.91/74.57
REGrad + layer-selective ($\lambda = 0.4$) + perturb ($\sigma = .015$)	87.89/84.93	78.98/74.45
REGrad + layer-selective ($\lambda = 0.4$) + perturb ($\sigma = .02$)***	**88.06/85.74**	**79.11**/74.39
REGrad + layer-selective ($\lambda = 0.4$) + perturb ($\sigma = .025$)	88.01/85.52	78.91/74.23

Table 9. Runtime analysis for UQ in seconds (s).

Method	C10 UQ Runtime
NEGrad	18.73s
UNGrad	33.45s
GradNorm	18.54s
ExGrad	34.72s
Perturb x	32.56s
Perturb θ	43.16s
MC-AA	32.56s
Inserted Dropout	16.15s
Entropy	3.05s
ExGrad V Term	2.52s
LA	22.61s
REGrad	33.28s
REGrad + layer-selective	29.27s
REGrad + layer-selective + perturb	75.87s

D.3. Efficiency Analysis for UQ.

While the complexity of both layer-selective gradients and perturbation-integrated gradients is discussed in Section 4, Table 9 provides their empirical runtime, specifically for computing the uncertainty of 1000 C10 images. Entropy-based methods are the most efficient, requiring only a single forward pass for uncertainty calculations. In comparison, gradient-based and perturbation-based methods demonstrate similar efficiency levels. Perturbation-based methods necessitate multiple forward passes, whereas gradient-based methods require both forward and backward passes. However, the efficiency of gradient-based methods can be enhanced through parallel computing. Due to the limitation of torch.autograd, which can only compute gradients of scalar functions of the input, we compute the gradients for each test sample sequentially (with a batch size of 1). For other baselines, except for gradient-based methods, we use a batch size of 32. Methods like ExGrad, UNGrad, and REGrad are more time-consuming than NEGrad and GradNorm because they require sequential computation of gradients for each class's probability. Although certain computations in class-wise gradient calculations can be reused, torch.autograd does not support parallel computation of gradients. Moreover, as discussed in Section 4 of the main

Table 10. ViT Results with AUROC(%)/AUPR(%).

Method	C10 → SVHN	C10 → LSUN	C100 → SVHN	C100 → LSUN
UNGrad	78.77/72.51	84.83/76.61	70.20/60.06	72.08/60.19
GradNorm	53.03/58.91	72.84/71.04	56.30/54.84	63.76/57.20
ExGrad	92.41/88.06	89.06/86.55	85.47/78.91	85.55/79.68
Perturb θ	89.13/81.26	90.70/81.59	84.35/77.37	85.70/77.74
Ours	**96.62/95.69**	**94.99/94.06**	**90.22/88.61**	**88.38/85.10**

paper, employing layer-wise gradient strategies does not add to the complexity. However, the perturbation-integrated gradient strategy requires additional time, mainly due to the need for extra forward propagation compared to other gradient-based methods.

D.4. Varying Model Architecture

We further train the model using a Vision Transformer (ViT) for the C10 and C100 datasets. The training procedure adheres to the guidelines provided in the GitHub repository at `https://github.com/kentaroy47/vision-transformers-cifar10`. We utilize the ViT-timm model, following the default hyperparameters. As shown in Table 10, the OOD detection results for the C10 and C100 datasets demonstrate improved performance using ViT compared to ResNet. Our method continues to outperform baseline approaches.