Mixtures of Laplace Approximations for Improved *Post-Hoc* Uncertainty in Deep Learning

Runa Eschenhagen[*,t] **Erik Daxberger**[c,m] **Philipp Hennig**[t,m] **Agustinus Kristiadi**[t]

[t]University of Tübingen
[c]University of Cambridge
[m]MPI for Intelligent Systems, Tübingen

Abstract

Deep neural networks are prone to overconfident predictions on outliers. Bayesian neural networks and deep ensembles have both been shown to mitigate this problem to some extent. In this work, we aim to combine the benefits of the two approaches by proposing to predict with a Gaussian mixture model posterior that consists of a weighted sum of Laplace approximations of independently trained deep neural networks. The method can be used *post hoc* with any set of pre-trained networks and only requires a small computational and memory overhead compared to regular ensembles. We theoretically validate that our approach mitigates overconfidence "far away" from the training data and empirically compare against state-of-the-art baselines on standard uncertainty quantification benchmarks.

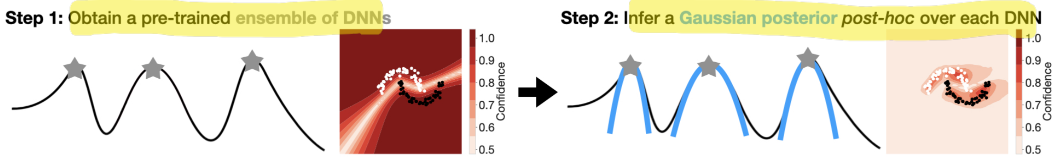
1 Introduction

While deep neural networks (DNNs) have achieved impressive results in a wide range of domains, they are prone to make overconfident predictions on outliers, such as *out-of-distribution* (OOD) data and data under *distribution shift* [45, 39]; this is especially harmful in safety-critical applications [1].

Bayesian inference of the DNN weights, resulting in a class of methods called *Bayesian neural networks* (BNNs), is a principled approach for quantifying predictive uncertainty. Commonly, a Gaussian distribution is used to approximate the true posterior. Many methods have been proposed to infer the parameters of this distribution, such as Laplace approximations [34, 48], variational inference (VI) [15, 4], and sampling-based approaches [35, 55]. While BNNs have been demonstrated to scale to large networks and problems like ImageNet [41, 35], recent work [42, 3] has shown that they tend to be outperformed by a simpler method called Deep Ensemble [28], which averages the predictions of multiple identical networks, independently trained with different random initializations.

In this work, we combine—intuitively speaking—the *global* uncertainty of Deep Ensembles with the *local* uncertainty around a single mode of BNNs with Gaussian approximate posterior, in a *post-hoc* way (Figure 1); this idea has been proposed previously [52]. We apply Laplace approximations to multiple independent pre-trained DNNs; specifically, we focus on last-layer Laplace approximations [49, 27]: this makes our method fast (see Table 2), scalable, and easy to implement without sacrificing predictive and uncertainty quantification performance. We call the resulting method *Mixtures of Laplace Approximations* (MoLA). It is trivial to implement in PyTorch, using the recently published `laplace` library[1] [8]. Besides empirically studying our method, we also extend the analysis of Kristiadi et al. [27] on single-Gaussian posteriors to the multi-class classification setting and subsequently show that MoG-based posteriors also avoid overconfidence "far away" from the training data.

[*]Correspondence to: `runa.eschenhagen@student.uni-tuebingen.de`.
[1]Available at `https://github.com/AlexImmer/Laplace`.



Step 1: Obtain a pre-trained ensemble of DNNs
Step 2: Infer a Gaussian posterior *post-hoc* over each DNN
Confidence
1.0
0.9
0.8
0.7
0.6
0.5

Figure 1: A Deep Ensemble captures *global* uncertainty by using only a small number of point estimates at modes (represented by stars) of the posterior, resulting in overconfident predictions away from the training data. Here we construct *post-hoc local* uncertainty (blue curves) around each mode, resulting in improved predictive uncertainty. Figure adapted from Fort et al. [13].

2 Background

We focus on the multi-class classification setting of a dataset $\mathcal{D} = \{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$ in C classes, with i.i.d. data points $\boldsymbol{x}_i \in \mathbb{R}^D$ and the corresponding one-hot encoded labels $\boldsymbol{y}_i \in \{\boldsymbol{e}_1, ..., \boldsymbol{e}_C\}$, where $\boldsymbol{e}_i$ is the i-th $C-$dimensional standard unit vector. Consider a neural network $\mathbf{f}_{\boldsymbol{\theta}} : \mathbb{R}^D \to \mathbb{R}^C$ parameterized by $\boldsymbol{\theta} \in \mathbb{R}^M$ with posterior distribution $p(\boldsymbol{\theta}|\mathcal{D}) \propto p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})$. A common way to train such a network is via *maximum a posteriori* (MAP) inference, i.e. we find the MAP estimate $\boldsymbol{\theta}_{\text{MAP}} = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta}|\mathcal{D}) = \arg\max_{\boldsymbol{\theta}} \log p(\mathcal{D}|\boldsymbol{\theta}) + \log p(\boldsymbol{\theta})$. The log-likelihood $\log p(\mathcal{D}|\boldsymbol{\theta})$ is equivalent to the negative loss function, e.g. the cross-entropy loss for classification, and the prior distribution $p(\boldsymbol{\theta})$ over model parameters $\boldsymbol{\theta}$ is usually a simple isotropic Gaussian $\mathcal{N}(\boldsymbol{\theta}|\mathbf{0}, \lambda^{-1}\mathbf{I})$ with prior precision $\lambda \in \mathbb{R}_+$, which is closely related to L_2 regularization and weight decay.

2.1 (Last-Layer) Laplace Approximation

A standard way to define the approximate posterior $q(\boldsymbol{\theta}|\mathcal{D})$ is via a simple Gaussian approximation $\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$. One option is to employ the Laplace approximation [34]. In this technique, we apply a second-order Taylor expansion at $\boldsymbol{\theta}_{\text{MAP}}$ to the log joint distribution, and then choose $\boldsymbol{\mu} = \boldsymbol{\theta}_{\text{MAP}}$ and $\boldsymbol{\Sigma} = (\boldsymbol{H}_{\boldsymbol{\theta}} + \lambda\mathbf{I})^{-1} \in \mathbb{R}^{M \times M}$, where $\boldsymbol{H}_{\boldsymbol{\theta}}$ is the Hessian of $-\log p(\mathcal{D}|\boldsymbol{\theta})$ w.r.t. $\boldsymbol{\theta}$ evaluated at $\boldsymbol{\theta}_{\text{MAP}}$.

It has been shown that last-layer Bayesian approximations yield a competitive performance compared to all-layer ones [49, 40, 6]. In this case, we consider the network $\mathbf{f}_{\boldsymbol{\theta}}$ as a linear function in the weight matrix $\boldsymbol{W} \in \mathbb{R}^{C \times P}$ of the last layer, i.e. $\mathbf{f}_{\boldsymbol{W}}(\boldsymbol{x}_*) = \boldsymbol{W}\boldsymbol{\phi}(\boldsymbol{x}_*)$, with *fixed* features $\boldsymbol{\phi}(\boldsymbol{x}_*) =: \boldsymbol{\phi}_* \in \mathbb{R}^P$, which are simply the output of the penultimate layer of the neural network given an arbitrary input $\boldsymbol{x}_*$. Here, we only need to do a Laplace approximation on the weight matrix $\boldsymbol{W}$: we obtain a Gaussian approximation $\mathcal{N}(\text{vec}(\boldsymbol{W})|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ with $\boldsymbol{\mu} = \text{vec}(\boldsymbol{W}_{\text{MAP}}) \in \mathbb{R}^{CP}$ and $\boldsymbol{\Sigma} = (\boldsymbol{H}_{\boldsymbol{W}} + \lambda\mathbf{I})^{-1} \in \mathbb{R}^{CP \times CP}$, where $\boldsymbol{H}_{\boldsymbol{W}}$ is the Hessian of $-\log p(\mathcal{D}|\boldsymbol{W})$ w.r.t. $\text{vec}(\boldsymbol{W})$ evaluated at $\text{vec}(\boldsymbol{W}_{\text{MAP}})$. This method is called *last-layer Laplace approximation* [LLLA, 27]. Since $\mathbf{f}_{\boldsymbol{W}}$ is linear in $\boldsymbol{W}$, we also have a Gaussian distribution $p(\mathbf{f}_*|\boldsymbol{x}_*, \mathcal{D}) = \mathcal{N}(\mathbf{f}_*|\boldsymbol{m}_*, \boldsymbol{C}_*)$ over the network outputs $\mathbf{f}_* := \mathbf{f}(\boldsymbol{x}_*)$, where $\boldsymbol{m}_* = \boldsymbol{W}_{\text{MAP}}\boldsymbol{\phi}_* \in \mathbb{R}^C$ and $\boldsymbol{C}_* = (\boldsymbol{\phi}_*^\top \otimes \mathbf{I})\boldsymbol{\Sigma}(\boldsymbol{\phi}_* \otimes \mathbf{I}) \in \mathbb{R}^{C \times C}$.[2]

3 Mixtures of Laplace Approximations

Consider DE's MAP estimates $(\boldsymbol{\theta}_{\text{MAP}}^{(k)})_{k=1}^K$ of an L-layer network. For each $k = 1 \dots, K$, we treat the first $L-1$ layers as a fixed feature map $\boldsymbol{\phi}^{(k)}$ and construct a Gaussian approximate posterior $\mathcal{N}(\text{vec}(\boldsymbol{W})|\boldsymbol{\mu}^{(k)}, \boldsymbol{\Sigma}^{(k)})$ over the last-layer weights via LLLA. That is, we set $\boldsymbol{\mu}^{(k)} = \text{vec}(\boldsymbol{W}_{\text{MAP}}^{(k)})$ and $\boldsymbol{\Sigma}^{(k)}$ to be the inverse Hessian of the negative log-posterior w.r.t. $\text{vec}(\boldsymbol{W})$ at $\boldsymbol{\mu}^{(k)}$. Given a sequence $\boldsymbol{\pi} := (\pi^{(k)})_{k=1}^K$ of non-negative real numbers with $\sum_{k=1}^K \pi^{(k)} = 1$, we define the approximate posterior as[3] $p_{\text{MoLA}}(\text{vec}(\boldsymbol{W})|\mathcal{D}) := \sum_{k=1}^K \pi^{(k)} \mathcal{N}(\text{vec}(\boldsymbol{W})|\boldsymbol{\mu}^{(k)}, \boldsymbol{\Sigma}^{(k)})$. By employing the *multi-class probit approximation* [MPA, 14, 33], MoLA's predictive distribution takes a particularly simple form:

$$p_{\text{MoLA}}(\boldsymbol{y} = \boldsymbol{e}_c|\boldsymbol{x}_*, \mathcal{D}) \approx \sum_{k=1}^K \pi^{(k)} \, \sigma(\boldsymbol{z}_*^{(k)})_c, \tag{1}$$

[2] $\otimes$ denotes the Kronecker product.
[3] The choice of $\boldsymbol{\pi}$ shall be discussed in Appendix B.2.

Table 1: In-distribution: CIFAR-10 → OOD. Values are means along with their standard errors over five runs with models trained with different random initalizations.

Method	In-Dist. MMC	SVHN MMC ↓	SVHN AUROC ↑	LSUN MMC ↓	LSUN AUROC ↑	CIFAR-100 MMC ↓	CIFAR-100 AUROC ↑
MAP	97.2 ± 0.0	77.5 ± 2.9	91.7 ± 1.2	71.7 ± 0.8	94.3 ± 0.3	79.2 ± 0.1	90.0 ± 0.1
DE	96.1 ± 0.0	62.8 ± 0.7	95.4 ± 0.2	59.2 ± 0.5	96.0 ± 0.1	70.7 ± 0.1	91.3 ± 0.1
SWAG	95.1 ± 0.4	69.3 ± 4.0	91.6 ± 1.3	62.2 ± 2.3	94.0 ± 0.7	73.0 ± 0.4	88.2 ± 0.5
MSWAG	94.5 ± 0.2	57.0 ± 1.2	95.6 ± 0.5	56.3 ± 1.0	95.6 ± 0.3	65.5 ± 0.5	91.1 ± 0.1
LLLA	94.1 ± 0.2	60.5 ± 4.0	93.6 ± 1.1	54.5 ± 1.5	95.4 ± 0.3	64.8 ± 0.6	90.8 ± 0.1
MoLA	93.8 ± 0.1	**52.3** $\pm$ **0.7**	**96.2** $\pm$ **0.2**	**48.3** $\pm$ **0.5**	**96.9** $\pm$ **0.1**	**61.2** $\pm$ **0.2**	**92.0** $\pm$ **0.0**

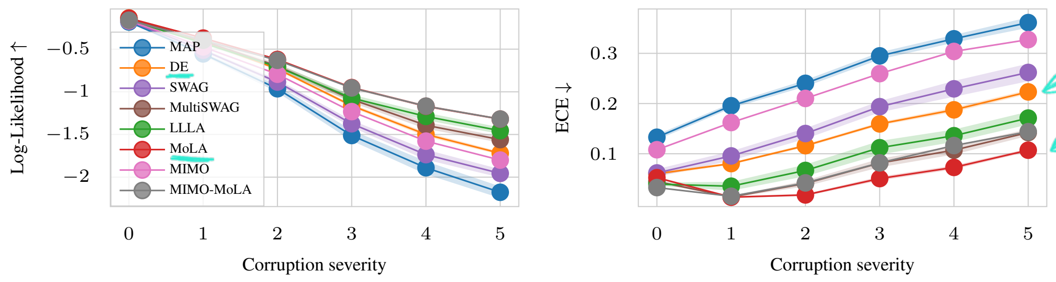


Figure 2: All methods on corrupted CIFAR-10, in terms of the log-likelihood (**left**, higher is better) and expected calibration error (**right**, lower is better) metrics. Dots represent means while shades represent standard errors over five runs with models trained with different random initalizations.

where $z_*^{(k)}$ is a vector with i-th component $z_{*i}^{(k)} = m_{*i}^{(k)} / \sqrt{1 + (\pi/8) C_{*ii}^{(k)}}$ for each $k = 1, \ldots, K$ (see Appendix E for the derivation), and $\sigma(\cdot)$ is the softmax function. The MPA enables fast predictions with a single forward pass per component. While we focus on MoLA applied to a regular DE, recent work of Havasi et al. [17] allows us to apply MoLA to a *single* DNN (MIMO-MoLA). This further improves the efficiency of MoLA without sacrificing much performance (see Figure 2). For more details on practical considerations when applying MoLA, please refer to Appendix B.2.

In Appendix B.1 we show how MoLA can mitigate overconfident predictions of DNNs with ReLU nonlinearities "far away" from the training data, in the sense that a training input x is scaled with a scalar $\delta > 0$, and as $\delta \to \infty$ [18]; all proofs are in Appendix E.

4 Experiments

We conduct extensive experiments on multiple image classification benchmarks, such as rotated MNIST, corrupted CIFAR-10 (Figure 2), and OOD detection tasks with MNIST and CIFAR-10 (Table 1) as in-distribution datasets. Moreover, we consider corrupted ImageNet to demonstrate the scalability of our approach. Generally, MoLA matches or outperforms all other considered methods, including Multi-SWAG, despite being cheaper. See Appendix D for the results and a detailed discussion.

5 Conclusion

We propose to combine two complementary forms of approximate Bayesian inference to infer and predict with a mixture of Gaussians constructed from *post-hoc* Laplace approximations. Empirically, the method compares favorably against state-of-the-art baselines on image classification benchmarks, in terms of performance as well as cost. The method can also be used with a single-model thanks to MIMO [17], which is especially attractive in the case where no pre-trained ensemble is available.

Acknowledgments and Disclosure of Funding

We thank Alexander Immer for helpful discussions on using the marginal likelihood for model weighting.

R.E., P.H., and A.K. gratefully acknowledge financial support by the European Research Council through ERC StG Action 757275 / PANAMA; the DFG Cluster of Excellence "Machine Learning - New Perspectives for Science", EXC 2064/1, project number 390727645; the German Federal Ministry of Education and Research (BMBF) through the Tübingen AI Center (FKZ: 01IS18039A); and funds from the Ministry of Science, Research and Arts of the State of Baden-Württemberg. E.D. acknowledges funding from the EPSRC and Qualcomm. A.K. is grateful to the International Max Planck Research School for Intelligent Systems (IMPRS-IS) for support.

References

[1] D. Amodei, C. Olah, J. Steinhardt, P. F. Christiano, J. Schulman, and D. Mané. Concrete problems in AI safety. *ArXiv*, abs/1606.06565, 2016.

[2] R. Arora, A. Basu, P. Mianjy, and A. Mukherjee. Understanding deep neural networks with rectified linear units. In *ICLR*, 2018.

[3] A. Ashukha, A. Lyzhov, D. Molchanov, and D. P. Vetrov. Pitfalls of in-domain uncertainty estimation and ensembling in deep learning. In *ICLR*, 2020.

[4] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight uncertainty in neural network. In *ICML*, 2015.

[5] G. W. Brier. Verification of forecasts expressed in terms of probability. *Monthly weather review*, 78(1): 1–3, 1950.

[6] N. Brosse, C. Riquelme, A. Martin, S. Gelly, and É. Moulines. On last-layer algorithms for classification: Decoupling representation from uncertainty estimation. *arXiv preprint arXiv:2001.08049*, 2020.

[7] F. Dangel, F. Kunstner, and P. Hennig. BackPACK: Packing more into backprop. In *ICLR*, 2020.

[8] E. Daxberger, A. Kristiadi, A. Immer, R. Eschenhagen, M. Bauer, and P. Hennig. Laplace redux - effortless Bayesian deep learning. In *NeurIPS*, 2021.

[9] E. Daxberger, E. Nalisnick, J. U. Allingham, J. Antoran, and J. M. Hernández-Lobato. Bayesian deep learning via subnetwork inference. In *ICML*, 2021.

[10] M. Dusenberry, G. Jerfel, Y. Wen, Y. Ma, J. Snoek, K. A. Heller, B. Lakshminarayanan, and D. Tran. Efficient and scalable Bayesian neural nets with rank-1 factors. In *ICML*, 2020.

[11] A. Filos, S. Farquhar, A. N. Gomez, T. G. J. Rudner, Z. Kenton, L. Smith, M. Alizadeh, A. D. Kroon, and Y. Gal. A systematic comparison of bayesian deep learning robustness in diabetic retinopathy tasks. *ArXiv*, abs/1912.10481, 2019.

[12] A. Y. K. Foong, Y. Li, J. M. Hernández-Lobato, and R. Turner. 'in-between' uncertainty in Bayesian neural networks. *arXiv*, abs/1906.11537, 2019.

[13] S. Fort, H. Hu, and B. Lakshminarayanan. Deep ensembles: A loss landscape perspective. *ArXiv*, abs/1912.02757, 2019.

[14] M. N. Gibbs. Bayesian Gaussian processes for regression and classification. *PhD thesis, University of Cambridge*, 1998.

[15] A. Graves. Practical variational inference for neural networks. In *NIPS*, 2011.

[16] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger. On calibration of modern neural networks. In *ICML*, 2017.

[17] M. Havasi, R. Jenatton, S. Fort, J. Z. Liu, J. Snoek, B. Lakshminarayanan, A. M. Dai, and D. Tran. Training independent subnetworks for robust prediction. In *ICLR*, 2021.

[18] M. Hein, M. Andriushchenko, and J. Bitterwolf. Why ReLU networks yield high-confidence predictions far away from the training data and how to mitigate the problem. In *CVPR*, 2019.

[19] D. Hendrycks and T. G. Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In *ICLR*, 2019.

[20] D. Hendrycks and K. Gimpel. A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks. In *ICLR*, 2017.

[21] D. Hendrycks, M. Mazeika, and T. Dietterich. Deep anomaly detection with outlier exposure. *ICLR*, 2019.

[22] A. Immer, M. Korzepa, and M. Bauer. Improving predictions of Bayesian neural networks via local linearization. In *AISTATS*, 2020.

[23] A. Immer, M. Bauer, V. Fortuin, G. Rätsch, and M. E. Khan. Scalable marginal likelihood estimation for model selection in deep learning. In *ICML*, 2021.

[24] M. E. Khan, D. Nielsen, V. Tangkaratt, W. Lin, Y. Gal, and A. Srivastava. Fast and scalable Bayesian deep learning by weight-perturbation in adam. In *ICML*, 2018.

[25] M. E. Khan, A. Immer, E. Abedi, and M. Korzepa. Approximate inference turns deep networks into Gaussian processes. In *NeurIPS*, 2019.

[26] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[27] A. Kristiadi, M. Hein, and P. Hennig. Being Bayesian, even just a bit, fixes overconfidence in ReLU networks. In *ICML*, 2020.

[28] B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *NIPS*, 2017.

[29] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989. doi: 10.1162/neco.1989.1.4.541.

[30] J. Z. Liu, Z. Lin, S. Padhy, D. Tran, T. Bedrax-Weiss, and B. Lakshminarayanan. Simple and principled uncertainty estimation with deterministic deep learning via distance awareness. *ArXiv*, abs/2006.10108, 2020.

[31] E. Lobacheva, N. Chirkova, M. Kodryan, and D. Vetrov. On power laws in deep ensembles, 2020.

[32] I. Loshchilov and F. Hutter. SGDR: stochastic gradient descent with warm restarts. In *ICLR*, 2017.

[33] Z. Lu, E. Ie, and F. Sha. Mean-field approximation to gaussian-softmax integral with application to uncertainty estimation, 2021.

[34] D. J. MacKay. A practical Bayesian framework for backpropagation networks. *Neural computation*, 4(3): 448–472, 1992.

[35] W. J. Maddox, P. Izmailov, T. Garipov, D. P. Vetrov, and A. G. Wilson. A simple baseline for Bayesian uncertainty in deep learning. In *NeurIPS*, 2019.

[36] J. Martens and R. B. Grosse. Optimizing neural networks with Kronecker-factored approximate curvature. In *ICML*, 2015.

[37] M. P. Naeini, G. F. Cooper, and M. Hauskrecht. Obtaining well calibrated probabilities using Bayesian binning. In *AAAI*, 2015.

[38] R. M. Neal. *Bayesian Learning for Neural Networks*. Springer-Verlag, Berlin, Heidelberg, 1996. ISBN 0387947248.

[39] A. M. Nguyen, J. Yosinski, and J. Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *CVPR*, pages 427–436. IEEE Computer Society, 2015.

[40] S. Ober and C. Rasmussen. Benchmarking the neural linear model for regression. *ArXiv*, abs/1912.08416, 2019.

[41] K. Osawa, S. Swaroop, M. E. Khan, A. Jain, R. Eschenhagen, R. E. Turner, and R. Yokota. Practical deep learning with Bayesian principles. In *NeurIPS*, 2019.

[42] 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. In *NeurIPS*, 2019.

[43] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019.

[44] R. Pinsler, J. Gordon, E. T. Nalisnick, and J. M. Hernández-Lobato. Bayesian batch active learning as sparse subset approximation. In *NeurIPS*, pages 6356–6367, 2019.

[45] J. Quionero-Candela, M. Sugiyama, A. Schwaighofer, and N. D. Lawrence. *Dataset Shift in Machine Learning*. The MIT Press, 2009. ISBN 0262170051.

[46] R. Rahaman and A. H. Thiery. Uncertainty quantification and deep ensembles. *ArXiv*, abs/2007.08792, 2020.

[47] C. Riquelme, G. Tucker, and J. Snoek. Deep Bayesian bandits showdown: An empirical comparison of Bayesian deep networks for Thompson sampling. In *ICLR*, 2018.

[48] H. Ritter, A. Botev, and D. Barber. A scalable Laplace approximation for neural networks. In *ICLR*, 2018.

[49] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. M. A. Patwary, Prabhat, and R. P. Adams. Scalable Bayesian optimization using deep neural networks. In *ICML*, 2015.

[50] M. Welling and Y. W. Teh. Bayesian learning via stochastic gradient Langevin dynamics. In L. Getoor and T. Scheffer, editors, *ICML*, pages 681–688, 2011.

[51] Y. Wen, G. Jerfel, R. Muller, M. W. Dusenberry, J. Snoek, B. Lakshminarayanan, and D. Tran. Combining ensembles and data augmentation can harm your calibration. *ArXiv*, abs/2010.09875, 2020.

[52] A. G. Wilson and P. Izmailov. Bayesian deep learning and a probabilistic perspective of generalization. In *NeurIPS*, 2020.

[53] S. Zagoruyko and N. Komodakis. Wide residual networks. In R. C. Wilson, E. R. Hancock, and W. A. P. Smith, editors, *BMVC*, 2016.

[54] G. Zhang, S. Sun, D. Duvenaud, and R. B. Grosse. Noisy natural gradient as variational inference. In *ICML*, 2018.

[55] R. Zhang, C. Li, J. Zhang, C. Chen, and A. G. Wilson. Cyclical stochastic gradient MCMC for Bayesian deep learning. In *ICLR*, 2020.

A Background

A.1 Bayesian Deep Learning

We focus on the multi-class classification setting of a dataset $\mathcal{D} = \{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$ in C classes, with i.i.d. data points $\boldsymbol{x}_i \in \mathbb{R}^D$ and the corresponding one-hot encoded labels $\boldsymbol{y}_i \in \{\boldsymbol{e}_1, ..., \boldsymbol{e}_C\}$, where $\boldsymbol{e}_i$ is the i-th $C-$dimensional standard unit vector. Consider a neural network $\mathbf{f}_{\boldsymbol{\theta}} : \mathbb{R}^D \to \mathbb{R}^C$ parameterized by $\boldsymbol{\theta} \in \mathbb{R}^M$ with posterior distribution $p(\boldsymbol{\theta}|\mathcal{D}) \propto p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})$. A common way to train such a network is via *maximum a posteriori* (MAP) inference, i.e. we find the MAP estimate

$$\boldsymbol{\theta}_{\text{MAP}} = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta}|\mathcal{D}) = \arg\max_{\boldsymbol{\theta}} \log p(\mathcal{D}|\boldsymbol{\theta}) + \log p(\boldsymbol{\theta}). \tag{2}$$

In (2), the log-likelihood $\log p(\mathcal{D}|\boldsymbol{\theta})$ is equivalent to the negative loss function, e.g. the cross-entropy loss for classification, and the prior distribution $p(\boldsymbol{\theta})$ over model parameters $\boldsymbol{\theta}$ is usually a simple isotropic Gaussian $\mathcal{N}(\boldsymbol{\theta}|\mathbf{0}, \lambda^{-1}\mathbf{I})$ with prior precision $\lambda \in \mathbb{R}_+$, which is closely related to L_2 regularization and weight decay. Note that $\boldsymbol{\theta}_{\text{MAP}}$ corresponds to a mode of the posterior distribution $p(\boldsymbol{\theta}|\mathcal{D})$ and thus it is not unique since $p(\boldsymbol{\theta}|\mathcal{D})$ is generally multi-modal—different training procedures, or even different random initializations of $\boldsymbol{\theta}$, could yield different solutions to (2).

Exploiting the randomness within MAP estimation, Deep Ensembles [DE, 28] aggregate the predictions of multiple MAP estimates arising from different random parameter initializations. More formally, given a sequence of K distinct MAP estimates $(\boldsymbol{\theta}_{\text{MAP}}^{(k)})_{k=1}^K$, a DE simply averages (i.e. with uniform weights) the predictions of the induced networks to obtain the predictive distribution

$$p_{\text{DE}}(\boldsymbol{y}|\boldsymbol{x}, \mathcal{D}) := \frac{1}{K}\sum_{k=1}^K p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}_{\text{MAP}}^{(k)}). \tag{3}$$

The number of ensemble members K is typically chosen to be small, e.g. $K = 5$. Despite their simplicity, DE has been shown to yield state-of-the-art results in uncertainty quantification [42].

The MAP estimate $\boldsymbol{\theta}_{\text{MAP}}$ represents a single point estimate in the parameter space and hence it ignores the uncertainty inherent in the parameters of the network. A Bayesian treatment of $\mathbf{f}_{\boldsymbol{\theta}}$, which results in a Bayesian neural network (BNN), attempts to capture this uncertainty by inferring the full posterior distribution $p(\boldsymbol{\theta}|\mathcal{D})$. Alas, this is computationally intractable due to the nonlinear nature of the network $\mathbf{f}_{\boldsymbol{\theta}}$, requiring the use of approximate inference techniques. Given an approximate, easy-to-sample-from posterior $q(\boldsymbol{\theta}|\mathcal{D})$, predictions can then be made via Monte Carlo (MC) integration:

$$p_{\text{BNN}}(\boldsymbol{y}|\boldsymbol{x}, \mathcal{D}) = \int p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta})\, q(\boldsymbol{\theta}|\mathcal{D})\, d\boldsymbol{\theta} \approx \frac{1}{S}\sum_{s=1}^S p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}^{(s)}); \quad \boldsymbol{\theta}^{(s)} \sim q(\boldsymbol{\theta}|\mathcal{D}). \tag{4}$$

A.2 (Last-Layer) Laplace Approximation

A standard way to define the approximate posterior $q(\boldsymbol{\theta}|\mathcal{D})$ is via a simple Gaussian approximation $\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$. One option is to employ the Laplace approximation [34]. In this technique, we apply a second-order Taylor expansion at $\boldsymbol{\theta}_{\text{MAP}}$ to the log joint distribution, and then choose $\boldsymbol{\mu} = \boldsymbol{\theta}_{\text{MAP}}$ and $\boldsymbol{\Sigma} = (\boldsymbol{H}_{\boldsymbol{\theta}} + \lambda\mathbf{I})^{-1} \in \mathbb{R}^{M\times M}$, where $\boldsymbol{H}_{\boldsymbol{\theta}}$ is the Hessian of $-\log p(\mathcal{D}|\boldsymbol{\theta})$ w.r.t. $\boldsymbol{\theta}$ evaluated at $\boldsymbol{\theta}_{\text{MAP}}$.

Recently it has been shown that last-layer Bayesian approximations yield a competitive performance compared to all-layer ones [49, 40, 6]. In this case, we consider the network $\mathbf{f}_{\boldsymbol{\theta}}$ as a linear function in the weight matrix $\boldsymbol{W} \in \mathbb{R}^{C\times P}$ of the last layer, i.e. $\mathbf{f}_{\boldsymbol{W}}(\boldsymbol{x}_*) = \boldsymbol{W}\boldsymbol{\phi}(\boldsymbol{x}_*)$, with *fixed* features $\boldsymbol{\phi}(\boldsymbol{x}_*) =: \boldsymbol{\phi}_* \in \mathbb{R}^P$, which are simply the output of the penultimate layer of the neural network given an input $\boldsymbol{x}_*$. Note, that this formulation includes the case where we have a bias parameter, by simply employing the standard bias trick. In this setting, we therefore only need to do a Laplace approximation on a single weight matrix $\boldsymbol{W}$: we obtain a Gaussian approximation $\mathcal{N}(\text{vec}(\boldsymbol{W})|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ with $\boldsymbol{\mu} = \text{vec}(\boldsymbol{W}_{\text{MAP}}) \in \mathbb{R}^{CP}$ and $\boldsymbol{\Sigma} = (\boldsymbol{H}_{\boldsymbol{W}} + \lambda\mathbf{I})^{-1} \in \mathbb{R}^{CP\times CP}$, where $\boldsymbol{H}_{\boldsymbol{W}}$ is the Hessian of $-\log p(\mathcal{D}|\boldsymbol{W})$ w.r.t. $\text{vec}(\boldsymbol{W})$ evaluated at $\text{vec}(\boldsymbol{W}_{\text{MAP}})$. This method is called the *last-layer Laplace approximation* [LLLA, 27].

Let $\boldsymbol{x}_* \in \mathbb{R}^D$ be an arbitrary test point. Since $\mathbf{f}_{\boldsymbol{W}}$ is linear in $\boldsymbol{W}$, we also have a Gaussian distribution $p(\mathbf{f}_*|\boldsymbol{x}_*, \mathcal{D}) = \mathcal{N}(\mathbf{f}_*|\boldsymbol{m}_*, \boldsymbol{C}_*)$ over the marginal network outputs $\mathbf{f}_* := \mathbf{f}(\boldsymbol{x}_*)$, where

$\boldsymbol{m}_* = \boldsymbol{W}_{\text{MAP}}\boldsymbol{\phi}_* \in \mathbb{R}^C$ and $\boldsymbol{C}_* = (\boldsymbol{\phi}_*^\top \otimes \mathbf{I})\boldsymbol{\Sigma}(\boldsymbol{\phi}_* \otimes \mathbf{I}) \in \mathbb{R}^{C\times C}$.[4] For multi-class classification, the predictive distribution is therefore given by

$$p(\boldsymbol{y} = \boldsymbol{e}_c|\boldsymbol{x}_*, \mathcal{D}) = \int \sigma(\mathbf{f}_*)_c \, p(\mathbf{f}_*|\mathcal{D}) \, d\mathbf{f}_*, \tag{5}$$

where $\sigma(\mathbf{f}_*)_c = \exp(\mathbf{f}_{*c}) / \sum_{i=1}^{C} \exp(\mathbf{f}_{*i})$ is the softmax function. This integral does not have an analytic solution, but it can be approximated via the *multi-class probit approximation* [MPA, 14, 33]:

$$p(\boldsymbol{y} = \boldsymbol{e}_c|\boldsymbol{x}_*, \mathcal{D}) \approx \sigma(\boldsymbol{z}_*)_c, \tag{6}$$

where $\boldsymbol{z}_*$ is a vector with i-th component $\boldsymbol{z}_{*i} = \boldsymbol{m}_{*i}/\sqrt{1 + (\pi/8)\boldsymbol{C}_{*ii}}$. Intuitively, the output mean $\boldsymbol{m}_{*i}$ is scaled by a factor which depends on the (non-negative) output variance $\boldsymbol{C}_{*ii}$, thus $\boldsymbol{z}_{*i}$ can only stay the same or decrease. This is conceptually similar to temperature scaling [16], but with a different temperature parameter $T \geq 1$ for each class and data point.

B Mixtures of Laplace Approximations

Comparing (3) and (4), one can notice that DE approximate the BNN predictive distribution. Why, then, do BNNs perform worse than DE? One hypothesis is that each MAP estimate within DE constitutes a mode of $p(\boldsymbol{\theta}|\mathcal{D})$—therefore, DE can capture K modes of the posterior, whereas Gaussian-based BNNs can only capture one mode. DE can thus be seen as capturing *global* uncertainty of the posterior, whereas Gaussian-based BNNs capture *local* uncertainty [13, 52]. In this section, we build on top of the MAP estimates provided by a DE to construct a BNN with a MoG posterior in a *post-hoc* manner via Laplace approximations. We call the resulting method *Mixtures of Laplace Approximations* (MoLA). While MoLA can be applied using any type of Laplace approximation, in this work we focus on a lightweight variant, where LLLA is employed. We henceforth assume LLLA by default when referring to MoLA.

Consider DE's MAP estimates $(\boldsymbol{\theta}_{\text{MAP}}^{(k)})_{k=1}^K$ of an L-layer network. For each $k = 1 \ldots, K$, we treat the first $L-1$ layers as a fixed feature map $\boldsymbol{\phi}^{(k)}$ and construct a Gaussian approximate posterior $\mathcal{N}(\text{vec}(\boldsymbol{W})|\boldsymbol{\mu}^{(k)}, \boldsymbol{\Sigma}^{(k)})$ over the last-layer weights via LLLA. That is, we set $\boldsymbol{\mu}^{(k)} = \text{vec}(\boldsymbol{W}_{\text{MAP}}^{(k)})$ and $\boldsymbol{\Sigma}^{(k)}$ to be the inverse Hessian of the negative log-posterior w.r.t. $\text{vec}(\boldsymbol{W})$ at $\boldsymbol{\mu}^{(k)}$. Given a sequence $\boldsymbol{\pi} := (\pi^{(k)})_{k=1}^K$ of non-negative real numbers with $\sum_{k=1}^K \pi^{(k)} = 1$, we define the approximate posterior as[5]

$$p_{\text{MoLA}}(\text{vec}(\boldsymbol{W})|\mathcal{D}) := \sum_{k=1}^{K} \pi^{(k)} \, \mathcal{N}(\text{vec}(\boldsymbol{W})|\boldsymbol{\mu}^{(k)}, \boldsymbol{\Sigma}^{(k)}). \tag{7}$$

This posterior can be seen intuitively as endowing each of DE's modes with a sense of local uncertainty, cf. Figure 1 for an illustration. MoLA's posterior also induces a MoG distribution over the network outputs due to the linearity of $\mathbf{f}_{\boldsymbol{W}}$ in $\boldsymbol{W}$. That is, given any input $\boldsymbol{x}_* \in \mathbb{R}^D$, we have

$$p_{\text{MoLA}}(\mathbf{f}_*|\boldsymbol{x}_*, \mathcal{D}) := \sum_{k=1}^{K} \pi^{(k)} \, \mathcal{N}(\mathbf{f}_*|\boldsymbol{m}_*^{(k)}, \boldsymbol{C}_*^{(k)}). \tag{8}$$

As a consequence, by employing the approximation from (6), MoLA's predictive distribution takes a particularly simple form:

$$p_{\text{MoLA}}(\boldsymbol{y} = \boldsymbol{e}_c|\boldsymbol{x}_*, \mathcal{D}) \approx \sum_{k=1}^{K} \pi^{(k)} \, \sigma(\boldsymbol{z}_*^{(k)})_c, \tag{9}$$

where $\boldsymbol{z}_*^{(k)}$ is as defined in (6) for each $k = 1, \ldots, K$ (see Appendix E for the derivation). We note that (9) is simply the weighted sum of the predictive distributions (6) induced by MoLA's mixture components. Due to the LLLA used, this can be computed at low overhead—cheaper than standard MC-integration. While other kinds of Laplace approximations can also satisfy this, they require a costly linearization over the entire network's parameters. LLLA is thus a favorable practical choice.

[4]$\otimes$ denotes the Kronecker product.

[5]The choice of $\boldsymbol{\pi}$ shall be discussed in Appendix B.2.

Table 2: The memory and computation costs of all methods after training in $\mathcal{O}$ notation. We also measure the wall-clock time (in seconds) and memory (in megabyte) for a Wide-ResNet on the CIFAR-10 test set, on a single NVIDIA RTX 2080Ti GPU. N, M, K, C, P are defined in Section A. For SWAG and MultiSWAG (MSWAG), S denotes the number of MC samples used to approximate the predictive distribution, and R denotes the number of model snapshots used to approximate the posterior. For prediction, the theoretical complexity is for a single test point.

	Inference		Prediction			
Method	Computation		Computation		Memory	
MAP	-	-	M	[1.17s]	M	[11MB]
LLLA	$NM+C^3+P^3$	[42.58s]	M	[1.14s]	$M+C^2+P^2$	[12MB]
SWAG	RNM	[1310.81s]	SRM	[21.90s]	RM	[440MB]
DE	-	-	KM	[3.80s]	KM	[55MB]
MoLA	$K(NM+C^3+P^3)$	[155.70s]	KM	[4.04s]	$K(M+C^2+P^2)$	[58MB]
MSWAG	$KRNM$	[6718.42s]	$KSRM$	[114.48s]	KRM	[2200MB]

B.1 Analysis

Feed-forward DNNs with ReLU nonlinearity—the so-called *ReLU networks*—are provably overconfident "far away" from the training data, in the sense that a training input $\boldsymbol{x}$ is scaled with a scalar $\delta > 0$, and as $\delta \to \infty$ [18]. For the binary classification setting it has been shown that an approximate (Gaussian-form) Bayesian treatment can mitigate this issue [27]. We are thus interested to know whether this desirable property of single-Gaussian BNNs also holds for MoLA.[6] Our analysis omits the bias parameters of the network and we employ the multi-class probit approximation (6) for analytical tractability. All proofs are in Appendix E.

As a preliminary, we define the *confidence* of a prediction for an input $\boldsymbol{x}_* \in \mathbb{R}^D$ by $\max_{i \in \{1,\dots,C\}} p(\boldsymbol{y} = \boldsymbol{e}_i | \boldsymbol{x}_*, \mathcal{D})$, i.e. it is the probability associated with the predicted class label. Also, note that the multi-class probit approximation only uses the diagonal elements $\boldsymbol{C}_{*ii}$ of the output covariance matrix $\boldsymbol{C}_*$. Hence, without loss of generality, for our analysis we only need to consider the posterior distributions $\mathcal{N}(\boldsymbol{w}_{(i)} | \boldsymbol{\mu}_{(i)}, \boldsymbol{\Sigma}_{(i)})$ over each row $\boldsymbol{w}_{(i)}$ of the weight matrix $\boldsymbol{W}$, instead of the full posterior over $\mathrm{vec}(\boldsymbol{W})$.

First, we present the following result which holds for single-Gaussian BNNs, as an extension of Kristiadi et al. [27]'s analysis to the multi-class classification case. It shows that asymptotically, the confidence of any input point is bounded away from the maximum confidence of 1 and the tightness of this bound depends on the uncertainty encoded in the covariance matrix of the approximate posterior.

Lemma 1. *Suppose that $\mathbf{f}_{\boldsymbol{W}} : \mathbb{R}^D \to \mathbb{R}^C$ is a ReLU network without bias. For any non-zero $\boldsymbol{x}_* \in \mathbb{R}^D$, there exists $\alpha > 0$ such that for all $\delta \geq \alpha$, we have under the multi-class probit approximation* (6)

$$p_{\text{BNN}}(\boldsymbol{y} = \boldsymbol{e}_{c_*} | \delta \boldsymbol{x}_*, \mathcal{D}) \leq \frac{1}{1 + \sum_{i \neq c_*} \exp(-(b_i + b_{c_*}))},$$

where $c_* = \arg\max_{i \in \{1,\dots,C\}} p_{\text{BNN}}(\boldsymbol{y} = \boldsymbol{e}_i | \delta \boldsymbol{x}_*, \mathcal{D})$ is the predicted class label and for each $i = 1, \dots, C$, and we define $b_i := \|\boldsymbol{\mu}_{(i)}\|_2 / \sqrt{(\pi/8)\lambda_{\text{min}}(\boldsymbol{\Sigma}_{(i)})}$, with $\lambda_{\text{min}}(\boldsymbol{\Sigma}_{(i)})$ being the smallest eigenvalue of $\boldsymbol{\Sigma}_{(i)}$.

To get an intuition for the behavior of the bound, we can consider two cases: When $\lambda_{\text{min}}(\boldsymbol{\Sigma}_{(i)}) \to 0$ for all $i = 1, \dots, C$ the upper bound on the confidence approaches one, i.e. minimal uncertainty. Conversely, if $\forall i \in \{1, ..., C\} : \lambda_{\text{min}}(\boldsymbol{\Sigma}_{(i)}) \to \infty$, the upper bound approaches $1/C$, i.e. maximal uncertainty. This confirms the intuition that increased uncertainty in the weight space results in increased uncertainty in the function space.

Now we can easily use the upper bound provided by Lemma 1 to obtain an asymptotic confidence bound for MoLA. As in Lemma 1, we only need to consider the posterior distributions

[6]The results in this section also hold for general last-layer Gaussian- (Lemma 1) and MoG-based (Theorem 2) BNNs. We focus on MoLA for clarity.

$p_{\text{MoLA}}(\boldsymbol{w}_{(i)}|\mathcal{D}) = \sum_{k=1}^{K} \pi^{(k)} \mathcal{N}(\boldsymbol{w}_{(i)}|\boldsymbol{\mu}_{(i)}^{(k)}, \boldsymbol{\Sigma}_{(i)}^{(k)})$ over each row $\boldsymbol{w}_{(i)}$ of the weight matrix $\boldsymbol{W}$, instead of the full MoLA posterior in (7).

Theorem 2. *Suppose that $\mathbf{f}_{\boldsymbol{W}} : \mathbb{R}^D \to \mathbb{R}^C$ is a ReLU network without bias, equipped with a MoLA posterior $p_{\text{MoLA}}(\boldsymbol{w}_{(i)}|\mathcal{D})$ for each $i = 1, \dots, C$. Using the approximation in (9), for any non-zero $\boldsymbol{x}_* \in \mathbb{R}^D$ there exists $\alpha > 0$ such that for all $\delta \geq \alpha$, we have*

$$p_{\text{MoLA}}(\boldsymbol{y} = \boldsymbol{e}_{c_*}|\delta\boldsymbol{x}_*, \mathcal{D}) \leq \sum_{k=1}^{K} \frac{\pi^{(k)}}{1 + \sum_{i \neq c_*} \exp(-(b_i^{(k)} + b_{c_*}^{(k)}))},$$

where for each $k = 1, \dots, K$, the integer $c_*^{(k)}$ is the predicted class label according to (6) and $b_i^{(k)}$ is defined as in Lemma 1, both under the k-th mixture component of p_{MoLA}.

Note that when all mixture components of p_{MoLA} are identical and $\boldsymbol{\pi}$ is a uniform probability vector, then we recover the result in Lemma 1. However, this is unlikely under the assumption that each MAP estimate on which MoLA is built upon is obtained via a random initialization. In fact, MoLA's bound can be tighter than that of Lemma 1 since intuitively it seems unlikely that the bounds on all components are worse than the one of any single randomly chosen component, given that they are all obtained via random initialization.

B.2 Practical Considerations

Alternative Laplace approximation for MoLA. While Laplace approximations over all weights might not be feasible for very large networks, one can also use lightweight variants of Laplace approximations, such as a Laplace approximation over a *subset* of the weights [9], instead of LLLA.

Kronecker-factored approximation of the Hessian. If the output dimension of the DNN, i.e. the number of classes C, is sufficiently small, one can usually use the full Hessian. However, for problems with many classes, like ImageNet ($C = 1000$), this becomes infeasible for even the LLLA. To ensure general applicability, we use a Kronecker factored (K-FAC) generalized Gauss-Newton (GGN) approximation [36, 48] which can be efficiently computed using automatic differentiation [7]. Note that for LLLA the GGN and the Hessian coincide due to the linearity of the output layer. Under the K-FAC approximation, we only have to compute, invert, and store two matrices of size $C \times C$ and $P \times P$, instead of one $CP \times CP$ matrix. Moreover, we empirically find that the difference in performance between a K-FAC and full GGN is relatively small (cf. Figure 3 for example); hence, we choose the K-FAC approximation in all our experiments.

Choice of mixture weights. So far we have yet to discuss the choice of the mixture weights $\boldsymbol{\pi}$ of the MoLA posterior. Intuitively, one way to pick a mixture coefficient is by picking it proportional to the importance of the corresponding mixture component. The marginal likelihood of each the mixture component can be used to do so [34, 23], leading to the following mixture coefficient

$$\pi^{(k)} := \frac{p(\mathcal{D}|\mathcal{M}_k)}{\sum_{k'=1}^{K} p(\mathcal{D}|\mathcal{M}_{k'})} \qquad \text{for all } k = 1, \dots, K, \tag{10}$$

where $\mathcal{M}_k$ is the model (i.e. the choice of architecture and hyperparameters) of the kth mixture component. See Appendix E.1 for a more formal derivation. In our setting, where the only difference between the mixture components is the random initialization of the DNN weights, all models $\mathcal{M}_k$ are identical. Hence, this approach should lead to uniform mixture weights $\pi^{(k)} = 1/K$ for all $k = 1, \dots, K$. We confirm this empirically and find that the marginal likelihoods of the components only vary minimally, consistent with what Immer et al. [23] report. Therefore, in all our experiments, we use this uniform weighting.

***Post-hoc* tuning of prior precision.** To be able to apply the Laplace approximation *post-hoc* on an arbitrary pre-trained DNN, the prior precision λ can usually not be set to the value used for L_2 regularization or weight decay during training [48, 22]. Thus, we tune the prior precision for MoLA, assuming a single prior precision for all mixture components—consistent with what Rahaman and Thiery [46] report for temperature scaling on a regular ensemble—of $p_{\text{MoLA}}(\text{vec}(\boldsymbol{W})|\mathcal{D})$. For our experiments we use thresholds on the validation set's average confidence and Brier score, choosing the smallest prior precision which results in meeting the thresholds (cf. Appendix F). Alternatively, the prior precision can also be tuned by optimizing the components' marginal likelihoods [23], w.r.t. to a proper scoring rule like the validation Brier score or log-likelihood, or, using OOD data [21].

Approximating the predictive distribution. While one can use the standard MC-integral (4) to approximate the predictive distribution $p_{\text{MoLA}}(\boldsymbol{y} = \boldsymbol{e}_c | \boldsymbol{x}_*, \mathcal{D})$, in our last-layer setting the closed-form MPA (6) is not only theoretically useful, but also computationally efficient (cf. Table 2). Moreover, it achieves competitive uncertainty calibration in terms of log-likelihood (Figure 3).

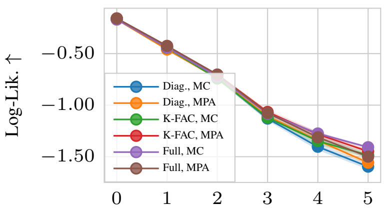


Figure 3: Comparisons of LLLAs on CIFAR-10-C [19] in terms of log-likelihood.

Efficient ensembling techniques. When no set of pre-trained DNNs is available, MoLA is more expensive than single-model methods. In any case, MoLA requires K forward passes for prediction. Leveraging recent work by Havasi et al. [17], we can leverage their multi-input multi-output (MIMO) method to implement MoLA within a *single* DNN. The resulting method MIMO-MoLA applies MoLA to multiple independent subnetworks within one DNN. We tested MIMO-MoLA on CIFAR-10-C as a proof-of-concept and find very competitive performance compared to the other methods, especially for a single-model method (see Figure 4).

Limitations. As opposed to other common Bayesian deep learning methods, MoLA cannot be directly applied to other interesting problem domains besides predictive uncertainty quantification, such as continual learning – we leave this for future work.

C Related Work

Bayesian deep learning has a long history and many methods besides the Laplace approximation have been proposed to infer an approximate posterior, such as VI [15, 4, 24, 54] and Markov Chain Monte Carlo [38, 50, 55] methods. The idea of capturing multiple posterior modes with a BNN has previously been explored. Wilson and Izmailov [52] proposed a method called MultiSWAG which takes multiple samples around several MAP estimates of a DNN to construct a MoG approximation. Other methods that also sample around a single mode have been combined with Deep Ensembles, such as subspace sampling [13, similar to SWAG] and MC dropout [11]. Meanwhile, Dusenberry et al. [10] proposed to construct a MoG posterior via VI with an efficient rank-one parameterization. Their method requires full training (thus non *post-hoc*) but is more efficient compared to training multiple DNNs from scratch.

The efficacy of last-layer Bayesian approximations have been shown previously [49, 47, 44, 42, 30, 40, 6]. Especially for the Gaussian-based variants, their properties have theoretically been studied by Kristiadi et al. [27]. However *mixtures* of last-layer approximate posteriors have not been studied—both theoretically and empirically—so far.

Recently, the linearization of DNNs in the context of Laplace approximations has been explored [25, 12, 22, 27]. While this can also lead to different approximations to the predictive distribution, it requires the computation of Jacobian matrices which is generally expensive. In contrast, MoLA can make use of the multi-class probit approximation at a low overhead due to its last-layer nature.

D Extended Experiments

D.1 Setup

We focus on evaluating the predictive uncertainty of MoLA on image data. Specifically, we consider the benchmark datasets of Hendrycks and Dietterich [19], Ovadia et al. [42]: (i) the rotated MNIST (MNIST-R) dataset, which consists of transformed MNIST images, rotated with increasing angle up to 180 degrees, (ii) the corrupted CIFAR-10 (CIFAR-10-C), and (iii) corrupted ImageNet (ImageNet-C) datasets, both consisting of 5 severity levels of 16/19 different perturbations of the respective original

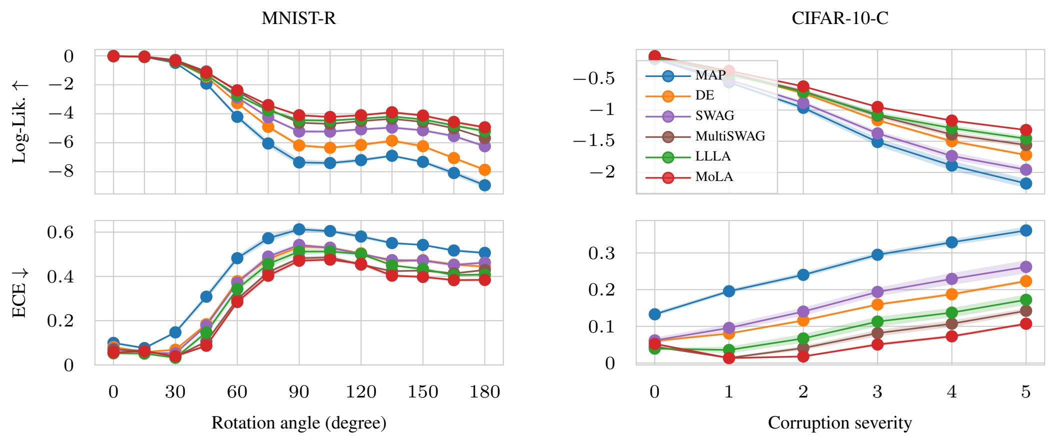


Figure 4: Comparison of all methods on MNIST-R (**left column**) and corrupted CIFAR-10-C (**right column**), in terms of the log-likelihood (**top row**, higher is better) and expected calibration error (**bottom row**, lower is better) metrics. Lines and dots represent means while shades represent standard error of five runs with models trained with different random initalizations.

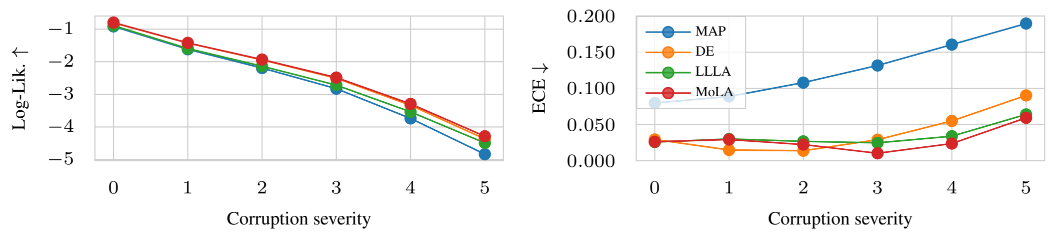


Figure 5: Comparison of all methods on ImageNet-C, in terms of the log-likelihood (**left**, higher is better) and ECE (**right**, lower is better) metrics. Lines and dots are averages over all corruption types.

dataset (cf. Appendix F). Moreover, we also study the performance of MoLA in OOD detection tasks with MNIST and CIFAR-10 as the in-distribution datasets.

We compare MoLA against the following baselines: (i) the standard MAP-trained network (MAP), (ii) Deep Ensemble [DE, 28], (iii) Stochastic Weight Averaging Gaussian [SWAG, 35], (iv) MultiSWAG [MSWAG, 52], (v) and the single-Gaussian-based LLLA.

Note in particular that MultiSWAG represents the state-of-the-art *post-hoc* MoG-based Bayesian methods. While the method of Dusenberry et al. [10] also constructs MoG posteriors, its performance is similar to Deep Ensemble. Moreover, Deep Ensemble has been shown to perform better than other methods like VI, MC-Dropout, and temperature scaling—at least on larger benchmarks like CIFAR-10-C and especially ImageNet-C [42]. Thus, the Deep Ensemble baseline is already representative of these methods. We do not compare against SWAG and MultiSWAG on ImageNet-C, due to their high computational and memory cost, see Appendix D.3 for context. We use standard DNN architectures (LeNet, ResNet, and WideResNet) and training methods. For the MNIST and CIFAR-10 experiments, we use five mixture components, while for ImageNet we use three. Full detail in Appendix F.

To evaluate the distribution shift tasks, we use the log-likelihood (LL) and expected calibration error [ECE, 37] metrics. We also provide results for accuracy, Brier score [5], mean confidence (MMC), and maximum calibration error [MCE, 37] in Appendix H. For OOD detection, on top of MMC, we use the area under the ROC curve (AUROC) metric [20].

D.2 Benchmarks

MNIST-R and CIFAR-10-C. The results for MNIST-R are in the first row of Figure 4. MoLA achieves the best results in both LL and ECE metrics, albeit by a small margin to LLLA on LL and to MultiSWAG on ECE. Consistent with Ovadia et al. [42], Deep Ensemble is outperformed by

Table 3: In-distribution: CIFAR-10 $\rightarrow$ OOD. Values are means along with their standard errors over five runs with models trained with different random initalizations.

	In-Dist.	SVHN		LSUN		CIFAR-100	
Method	MMC	MMC $\downarrow$	AUROC $\uparrow$	MMC $\downarrow$	AUROC $\uparrow$	MMC $\downarrow$	AUROC $\uparrow$
MAP	97.2 ± 0.0	77.5 ± 2.9	91.7 ± 1.2	71.7 ± 0.8	94.3 ± 0.3	79.2 ± 0.1	90.0 ± 0.1
DE	96.1 ± 0.0	62.8 ± 0.7	95.4 ± 0.2	59.2 ± 0.5	96.0 ± 0.1	70.7 ± 0.1	91.3 ± 0.1
SWAG	95.1 ± 0.4	69.3 ± 4.0	91.6 ± 1.3	62.2 ± 2.3	94.0 ± 0.7	73.0 ± 0.4	88.2 ± 0.5
MSWAG	94.5 ± 0.2	57.0 ± 1.2	95.6 ± 0.5	56.3 ± 1.0	95.6 ± 0.3	65.5 ± 0.5	91.1 ± 0.1
LLLA	94.1 ± 0.2	60.5 ± 4.0	93.6 ± 1.1	54.5 ± 1.5	95.4 ± 0.3	64.8 ± 0.6	90.8 ± 0.1
MoLA	93.8 ± 0.1	$\mathbf{52.3 \pm 0.7}$	$\mathbf{96.2 \pm 0.2}$	$\mathbf{48.3 \pm 0.5}$	$\mathbf{96.9 \pm 0.1}$	$\mathbf{61.2 \pm 0.2}$	$\mathbf{92.0 \pm 0.0}$

single-DNN Bayesian methods. On CIFAR-10-C, similar to our observation on MNIST-R, we observe that while all methods' performances degrade as the corruption severity increases, MoLA yields the best results on both metrics across all severity levels (Figure 4, second row). MultiSWAG performs similarly to MoLA for low corruption levels but worsens as the corruption levels increases. While Deep Ensemble improves the vanilla MAP model, it performs worse than MoLA and MultiSWAG. Among the single-model methods, LLLA yields the best results and is only beaten by MoLA and MultiSWAG, as expected. In contrast to Ovadia et al. [42], a single-model BNN, namely LLLA, outperforms DE. One likely explanation is that in contrast to the last-layer VI method used in Ovadia et al. [42], LLLA does not change the underlying optimization process since it is based on the MAP estimate. This indicates that capturing multiple modes is not the sole reason why DE typically outperforms methods based on only a single model. However, capturing multiple modes further improves performance, as can be seen from the improvement of MoLA compared to LLLA.

ImageNet-C. In Figure 5 we observe that LLLA improves MAP in terms of LL and significantly so in terms of ECE, especially in higher severity levels. The results for DE and MoLA on LL are similar—though, as expected, both methods perform better than MAP and LLLA. MoLA achieves better calibration in terms of ECE compared to DE in more challenging scenarios, i.e. at higher severity levels. Nevertheless, DE is more calibrated than LLLA and MoLA for low corruption severity. One potential explanation for these results is that DE is already well-calibrated in itself and that MoLA is decreasing the confidence, resulting in underconfident predictions for low corruption levels. These results are reminiscent of the work by Wen et al. [51] who observed underconfident predictions as a result of combining DE with a method to improve calibration, in this case, data augmentation.

OOD detection. We present the OOD detection results for CIFAR-10 in Table 3. We observe that MoLA achieves significantly the best results across all OOD test sets and all metrics considered, without sacrificing its in-distribution confidence estimates.[7] Unlike in the previous dataset shift experiments, here we observe that all ensemble methods (Deep Ensemble, MultiSWAG, and MoLA) yield better results than single-model methods (MAP, SWAG, LLLA) in terms of the AUROC metric. This signifies that considering multiple modes is beneficial for discriminating in- against out-of-distribution data. Endowing each of the ensemble members with local uncertainty via MoLA and MultiSWAG further improves this. The MNIST OOD experiment follows a similar trend; we present the full results in Table 4.

D.3 Computational and Memory Cost

We measure the wall-clock time for inference and prediction on the CIFAR-10 test set (cf. Table 2). Inference only has to be done once—for LLLA and MoLA it includes the computation and inversion of the K-FAC GGN and the tuning of the prior precision. We can see that inference is more than an order of magnitude faster for LLLA and MoLA than for SWAG and MultiSWAG respectively. For prediction, LLLA and MoLA are about as fast as MAP and DE respectively, due to our use of the multi-class probit approximation. Moreover, they only require small additional memory overhead. In contrast, SWAG and MultiSWAG require multiple forward passes which results in slower prediction speed; also, they require multiple snapshots of the model which results in higher memory requirements.

[7]Significances are established by comparing means and error bars.

Table 4: In-distribution: MNIST $\rightarrow$ OOD. Values are means along with their standard errors over five random initialization.

Method	In-Dist. MMC	FMNIST MMC ↓	FMNIST AUROC ↑	EMNIST MMC ↓	EMNIST AUROC ↑	KMNIST MMC ↓	KMNIST AUROC ↑
MAP	99.4 ± 0.0	64.1 ± 0.5	99.0 ± 0.0	83.6 ± 0.3	93.7 ± 0.3	77.4 ± 0.3	97.1 ± 0.1
DE	99.2 ± 0.0	55.2 ± 0.4	99.3 ± 0.0	75.8 ± 0.2	95.2 ± 0.0	66.0 ± 0.3	98.4 ± 0.0
SWAG	99.4 ± 0.0	64.6 ± 0.2	99.0 ± 0.0	84.2 ± 0.2	93.6 ± 0.2	78.6 ± 0.3	97.1 ± 0.1
MSWAG	99.3 ± 0.0	55.6 ± 0.4	99.3 ± 0.0	76.1 ± 0.3	95.2 ± 0.0	66.3 ± 0.3	**98.5 ± 0.0**
LLLA	98.2 ± 0.0	48.2 ± 0.5	99.2 ± 0.0	71.5 ± 0.3	94.4 ± 0.2	63.3 ± 0.4	97.5 ± 0.1
MoLA	98.3 ± 0.0	**44.0 ± 0.5**	**99.5 ± 0.0**	**67.4 ± 0.2**	**95.7 ± 0.1**	**56.8 ± 0.4**	**98.5 ± 0.0**

E Proofs and Derivations

For completeness, we first derive Equation (9). Notice that the summation in the definition of $p_{\text{MoLA}}(\mathbf{f}_* | \boldsymbol{x}_*, \mathcal{D})$ in (8) is finite and thus we can safely interchange it with integrals. Now, using (5) and (8), we have

$$\begin{aligned} p_{\text{MoLA}}(\boldsymbol{y} = \boldsymbol{e}_c | \boldsymbol{x}_*, \mathcal{D}) &= \int \sigma(\mathbf{f}_*)_c \, p_{\text{MoLA}}(\mathbf{f}_* | \boldsymbol{x}_*, \mathcal{D}) \, d\mathbf{f}_* \\ &= \int \sigma(\mathbf{f}_*)_c \left(\sum_{k=1}^{K} \pi^{(k)} \mathcal{N}(\mathbf{f}_* | \boldsymbol{m}_*^{(k)}, \boldsymbol{C}_*^{(k)}) \right) d\mathbf{f}_* \\ &= \sum_{k=1}^{K} \pi^{(k)} \left(\int \sigma(\mathbf{f}_*)_c \mathcal{N}(\mathbf{f}_* | \boldsymbol{m}_*^{(k)}, \boldsymbol{C}_*^{(k)}) \, d\mathbf{f}_* \right) \\ &\approx \sum_{k=1}^{K} \pi^{(k)} \, \sigma(\boldsymbol{z}_*^{(k)})_c, \end{aligned}$$

as required, where we have used the linearity of the integral in the third and the multi-class probit approximation (6) in the last step.

Lemma 1. *Suppose that $\mathbf{f}_{\boldsymbol{W}} : \mathbb{R}^D \rightarrow \mathbb{R}^C$ is a ReLU network without bias. For any non-zero $\boldsymbol{x}_* \in \mathbb{R}^D$, there exists $\alpha > 0$ such that for all $\delta \geq \alpha$, we have under the multi-class probit approximation* (6)

$$p_{\text{BNN}}(\boldsymbol{y} = \boldsymbol{e}_{c_*} | \delta \boldsymbol{x}_*, \mathcal{D}) \leq \frac{1}{1 + \sum_{i \neq c_*} \exp(-(b_i + b_{c_*}))},$$

where $c_* = \arg\max_{i \in \{1,\dots,C\}} p_{\text{BNN}}(\boldsymbol{y} = \boldsymbol{e}_i | \delta \boldsymbol{x}_*, \mathcal{D})$ is the predicted class label and for each $i = 1, \dots, C$, and we define $b_i := \|\boldsymbol{\mu}_{(i)}\|_2 / \sqrt{(\pi/8)\lambda_{\min}(\boldsymbol{\Sigma}_{(i)})}$, with $\lambda_{\min}(\boldsymbol{\Sigma}_{(i)})$ being the smallest eigenvalue of $\boldsymbol{\Sigma}_{(i)}$.

Proof. Let $\mathbf{f} : \mathbb{R}^D \rightarrow \mathbb{R}^C$ be a neural network with activation functions $\text{ReLU}(x) = \max\{0, x\}$, then $\mathbf{f}$ is a piecewise affine function [2]. A function $\mathbf{f}$ is called piecewise affine if there exists a finite set of polytopes $\{Q_i\}_{i=1}^{I}$ with $\cup_{i=1}^{I} Q_i = \mathbb{R}^D$ and $\mathbf{f}$ is affine when restricted to any single Q_i; we call Q_i a linear region. The following Lemma, which we adopt without proof from Hein et al. [18], says that we can write the neural network as an affine function for data points scaled by a sufficiently large factor.

Lemma 3 (Hein et al. [18])**.** *Let $\{Q_i\}_{i=1}^{I}$ be the set of linear regions associated with the neural network $\mathbf{f} : \mathbb{R}^D \rightarrow \mathbb{R}^C$ with ReLU activations. For any non-zero $\boldsymbol{x} \in \mathbb{R}^D$ there exists $\alpha \in \mathbb{R}$ with $\alpha > 0$ and $t \in \{1, \dots, I\}$ such that $\delta \boldsymbol{x} \in Q_t$ for all $\delta \geq \alpha$.*

In the following, we make the dependence of $\boldsymbol{z}_i$ on the input $\boldsymbol{x}_*$ explicit by writing $\boldsymbol{z}_i(\boldsymbol{x}_*)$. Lemma 3 can be used to derive the following statement.

Lemma 4 (Kristiadi et al. [27])**.** *For any non-zero $\boldsymbol{x}_* \in \mathbb{R}^D$ there exists $\alpha \in \mathbb{R}$ with $\alpha > 0$, such that $\delta \boldsymbol{x}_* \in R$ for all $\delta \geq \alpha$. Also, we assume that the neural network $\mathbf{f}$ has no bias parameters. Then the restriction $|\boldsymbol{z}|_R(\delta \boldsymbol{x}_*)_i|$ is an increasing function in δ for all $i \in \{1, \dots, C\}$.*

To reiterate, the multi-class probit approximation only uses the diagonal elements $\boldsymbol{C}_{*ii}$ of the output covariance matrix $\boldsymbol{C}_*$ and hence, without loss of generality, for our analysis we only need to consider the posterior distributions $\mathcal{N}(\boldsymbol{w}_{(i)}|\boldsymbol{\mu}_{(i)}, \boldsymbol{\Sigma}_{(i)})$ over each row $\boldsymbol{w}_{(i)}$ of the weight matrix $\boldsymbol{W}$, instead of the full posterior over $\operatorname{vec}(\boldsymbol{W})$. With this, Lemma 3 can be used to derive the following bound.

Lemma 5 (Kristiadi et al. [27])**.** *For any non-zero $\boldsymbol{x}_* \in \mathbb{R}^D$ there exists $\alpha \in \mathbb{R}$ with $\alpha > 0$, such that $\delta\boldsymbol{x}_* \in R$ for all $\delta \geq \alpha$. We have*

$$\lim_{\delta\to\infty} |\boldsymbol{z}|_R(\delta\boldsymbol{x}_*)_i| \leq \frac{||\boldsymbol{\mu}_{(i)}||_2}{\sqrt{\pi/8\,\lambda_{min}(\boldsymbol{\Sigma}_{(i)})}} =: b_i, \tag{11}$$

for all $i \in \{1, \dots, C\}$.

Using these results, we can now proof the desired statement. Let $\boldsymbol{x}_* \in \mathbb{R}^D$ be arbitrary but non-zero. Then there exists $\alpha \in \mathbb{R}$ with $\alpha > 0$, such that $\delta\boldsymbol{x}_* \in R$ for all $\delta \geq \alpha$. Using the multi-class probit approximation (6), we have for all $\delta \geq \alpha$

$$\begin{aligned}
p_{\text{BNN}}(\boldsymbol{y} = \boldsymbol{e}_{c_*}|\delta\boldsymbol{x}_*, \mathcal{D}) &\approx \sigma(\boldsymbol{z}(\delta\boldsymbol{x}_*))_{c_*} \\
&= \frac{1}{1 + \sum_{i\neq c_*} \exp(\boldsymbol{z}(\delta\boldsymbol{x}_*)_i - \boldsymbol{z}(\delta\boldsymbol{x}_*)_{c_*})} \\
&\leq \frac{1}{1 + \sum_{i\neq c_*} \exp(-(|\boldsymbol{z}(\delta\boldsymbol{x}_*)_i| + |\boldsymbol{z}(\delta\boldsymbol{x}_*)_{c_*}|))} \\
&\leq \lim_{\delta\to\infty} \frac{1}{1 + \sum_{i\neq c_*} \exp(-(|\boldsymbol{z}(\delta\boldsymbol{x}_*)_i| + |\boldsymbol{z}(\delta\boldsymbol{x}_*)_{c_*}|))} \\
&= \frac{1}{1 + \sum_{i\neq c_*} \exp(-\lim_{\delta\to\infty}(|\boldsymbol{z}(\delta\boldsymbol{x}_*)_i| + |\boldsymbol{z}(\delta\boldsymbol{x}_*)_{c_*}|))} \\
&\leq \frac{1}{1 + \sum_{i\neq c_*} \exp(-(b_i + b_{c_*}))},
\end{aligned}$$

where we have used Lemma 4 in the second and Lemma 5 in the last inequality. This concludes the proof. □

Theorem 2. *Suppose that $\mathbf{f}_{\boldsymbol{W}} : \mathbb{R}^D \to \mathbb{R}^C$ is a ReLU network without bias, equipped with a MoLA posterior $p_{\text{MoLA}}(\boldsymbol{w}_{(i)}|\mathcal{D})$ for each $i = 1, \dots, C$. Using the approximation in (9), for any non-zero $\boldsymbol{x}_* \in \mathbb{R}^D$ there exists $\alpha > 0$ such that for all $\delta \geq \alpha$, we have*

$$p_{\text{MoLA}}(\boldsymbol{y} = \boldsymbol{e}_{c_*}|\delta\boldsymbol{x}_*, \mathcal{D}) \;\leq\; \sum_{k=1}^{K} \frac{\pi^{(k)}}{1 + \sum_{i\neq c_*} \exp(-(b_i^{(k)} + b_{c_*}^{(k)}))},$$

where for each $k = 1, \dots, K$, the integer $c_*^{(k)}$ is the predicted class label according to (6) and $b_i^{(k)}$ is defined as in Lemma 1, both under the k-th mixture component of p_{MoLA}.

Proof. As in Lemma 1, we only need to consider the posterior distributions

$$p_{\text{MoLA}}(\boldsymbol{w}_{(i)}|\mathcal{D}) = \sum_{k=1}^{K} \pi^{(k)} \mathcal{N}(\boldsymbol{w}_{(i)}|\boldsymbol{\mu}_{(i)}^{(k)}, \boldsymbol{\Sigma}_{(i)}^{(k)})$$

over each row $\boldsymbol{w}_{(i)}$ of the weight matrix $\boldsymbol{W}$, instead of the full MoLA posterior in (7). Using the approximation from Equation (9), we have

$$\begin{aligned}
p_{MoLA}(\boldsymbol{y} = \boldsymbol{e}_{c_*}|\delta\boldsymbol{x}_*, \mathcal{D}) &\approx \sum_{k=1}^{K} \pi^{(k)}\, \sigma(\boldsymbol{z}^{(k)}(\delta\boldsymbol{x}_*))_{c_*} \\
&\leq \sum_{k=1}^{K} \frac{\pi^{(k)}}{1 + \sum_{i\neq c_*} \exp(-(b_i^{(k)} + b_{c_*}^{(k)}))},
\end{aligned}$$

where we have applied Lemma 1 to each mixture component to get the inequality. □

E.1 Mixture Weights via Model Selection

The goal is to not only infer the neural network's last-layer weight matrix $\boldsymbol{W}$, but also include model selection into the inference problem. Hence, consider the models $\mathcal{M}_k$ of the K components, where the model typically consists of the DNN architecture and hyperparameters. We now want to infer the joint posterior

$$\begin{aligned} p(\boldsymbol{W}, \mathcal{M}|\mathcal{D}) &= \frac{p(\mathcal{D}|\boldsymbol{W}, \mathcal{M})p(\boldsymbol{W}|\mathcal{M})p(\mathcal{M})}{p(\mathcal{D})} \\ &= \frac{p(\boldsymbol{W}|\mathcal{D}, \mathcal{M})p(\mathcal{D}|\mathcal{M})p(\boldsymbol{W}|\mathcal{M})p(\mathcal{M})}{p(\boldsymbol{W}|\mathcal{M})p(\mathcal{D})} \\ &= p(\boldsymbol{W}|\mathcal{D}, \mathcal{M})p(\mathcal{M}|\mathcal{D}). \end{aligned} \tag{12}$$

We can see that $p(\boldsymbol{W}|\mathcal{D}, \mathcal{M})$ is our regular posterior over the weights, given a model $\mathcal{M}$. We consider the model to have a categorical distribution with K categories and impose a uniform prior on them, i.e. $p(\mathcal{M}) = \mathcal{U}(1, \ldots, K)$. The posterior probability of the kth model $\mathcal{M}_k$ is then given by

$$\begin{aligned} p(\mathcal{M}_k|\mathcal{D}) &= \frac{p(\mathcal{D}|\mathcal{M}_k)p(\mathcal{M}_k)}{p(\mathcal{D})} \\ &= \frac{p(\mathcal{D}|\mathcal{M}_k)p(\mathcal{M}_k)}{\sum_{k'=1}^{K} p(\mathcal{D}|\mathcal{M}_{k'})p(\mathcal{M}_{k'})} \\ &= \frac{p(\mathcal{D}|\mathcal{M}_k)\frac{1}{K}}{\sum_{k'=1}^{K} p(\mathcal{D}|\mathcal{M}_{k'})\frac{1}{K}} \\ &= \frac{p(\mathcal{D}|\mathcal{M}_k)}{\sum_{k'=1}^{K} p(\mathcal{D}|\mathcal{M}_{k'})}, \end{aligned} \tag{13}$$

where we can recognize $p(\mathcal{D}|\mathcal{M}_k)$ as the marginal likelihood corresponding to the posterior over the weights $\boldsymbol{W}$ given the model $\mathcal{M}_k$. Using the posterior over weights and models, the predictive distribution under the Laplace approximation becomes

$$\begin{aligned} p(\boldsymbol{y} = \boldsymbol{e}_c|\boldsymbol{x}_*, \mathcal{D}) &= \int \sum_{k=1}^{K} p(\boldsymbol{y} = \boldsymbol{e}_c|\boldsymbol{x}_*, \boldsymbol{W}, \mathcal{M}_k)p(\boldsymbol{W}|\mathcal{D}, \mathcal{M}_k)p(\mathcal{M}_k|\mathcal{D})\, d\boldsymbol{W} \\ &= \sum_{k=1}^{K} \underbrace{p(\mathcal{M}_k|\mathcal{D})}_{=:\pi^{(k)}} \int p(\boldsymbol{y} = \boldsymbol{e}_c|\boldsymbol{x}_*, \boldsymbol{W}, \mathcal{M}_k)p(\boldsymbol{W}|\mathcal{D}, \mathcal{M}_k)\, d\boldsymbol{W} \\ &= \sum_{k=1}^{K} \pi^{(k)} \int \sigma(\mathbf{f}_*)_c\, p(\mathbf{f}_*|\mathcal{D}, \mathcal{M}_k)\, d\mathbf{f}_* \\ &= p_{\text{MoLA}}(\boldsymbol{y} = \boldsymbol{e}_c|\boldsymbol{x}_*, \mathcal{D}), \end{aligned} \tag{14}$$

where we have used the predictive distribution of the LLLA from Equation (5) in the third step. The mixture weights are therefore the normalized marginal likelihoods of the K different models. Since we apply the Laplace approximation to each DNN, each model's weight posterior has Gaussian form, and therefore, the marginal likelihood of each model $\mathcal{M}_k$ can be estimated in closed form.

F Experimental Details

F.1 Algorithm

We present the algorithm for inference and prediction with MoLA using LLLA and a K-FAC approximation of the Hessian in Algorithm 1. Note that the inference function only needs to be called once. The algorithm for LLLA with K-FAC is adopted from Kristiadi et al. [27].

F.2 Datasets

The rotated MNIST benchmark uses the standard MNIST dataset [29] which consists of black and white images of handwritten digits, labeled with 10 classes. The images are roatated by increasing

Algorithm 1 MoLA inference and prediction functions with LLLA and K-FAC approximation of the Hessian.

```
function Inference($\{\mathbf{f}^{(k)}\}_{k=1}^K, \mathcal{D}_{\text{train}}, \mathcal{D}_{\text{val}}$)
    Input: Set of $K$ pre-trained neural networks $\mathbf{f}(\cdot)^{(k)} = \boldsymbol{W}_{\text{MAP}}^{(k)}\boldsymbol{\phi}(\cdot)^{(k)}$, training dataset $\mathcal{D}_{\text{train}}$, validation dataset $\mathcal{D}_{\text{val}}$, mini-batch size $n$, running average parameter $\beta$
    Initialize $\mathcal{A} = \emptyset$ and $\mathcal{B} = \emptyset$.
    for $k = 1$ to $K$ do
        Initialize $\boldsymbol{A}^{(k)} = \mathbf{0} \in \mathbb{R}^{C \times C}$ and $\boldsymbol{B}^{(k)} = \mathbf{0} \in \mathbb{R}^{P \times P}$.
        for $\boldsymbol{X}, \boldsymbol{Y}$ in sampleMiniBatch($\mathcal{D}_{\text{train}}, n$) do
            $\hat{\boldsymbol{A}}^{(k)}, \hat{\boldsymbol{B}}^{(k)} = \text{computeK-FAC}(\mathcal{L}(\mathbf{f}(\boldsymbol{X})^{(k)}, \boldsymbol{Y}), \boldsymbol{W}_{\text{MAP}})$
            $\boldsymbol{A}^{(k)} = \beta\boldsymbol{A}^{(k)} + (1-\beta)\hat{\boldsymbol{A}}^{(k)}$
            $\boldsymbol{B}^{(k)} = \beta\boldsymbol{B}^{(k)} + (1-\beta)\hat{\boldsymbol{B}}^{(k)}$
        end for
        $\mathcal{A} = \mathcal{A} \cup \{\boldsymbol{A}^{(k)}\}$
        $\mathcal{B} = \mathcal{B} \cup \{\boldsymbol{B}^{(k)}\}$
    end for
    $\boldsymbol{\pi} = \text{chooseMixtureWeights}(\cdot)$
    $\lambda = \text{tunePriorPrecision}(\{\mathbf{f}^{(k)}\}_{k=1}^K, \mathcal{A}, \mathcal{B}, \boldsymbol{\pi}, \mathcal{D}_{\text{val}})$
    Initialize $\mathcal{U} = \emptyset$ and $\mathcal{V} = \emptyset$.
    for $k = 1$ to $K$ do
        $\boldsymbol{U}^{(k)} = (\sqrt{|\mathcal{D}_{\text{train}}|}\boldsymbol{A}^{(k)} + \sqrt{\lambda}\mathbf{I})^{-1}$
        $\boldsymbol{V}^{(k)} = (\sqrt{|\mathcal{D}_{\text{train}}|}\boldsymbol{B}^{(k)} + \sqrt{\lambda}\mathbf{I})^{-1}$
        $\mathcal{U} = \mathcal{U} \cup \{\boldsymbol{U}^{(k)}\}$
        $\mathcal{V} = \mathcal{V} \cup \{\boldsymbol{V}^{(k)}\}$
    end for
    Output: $\mathcal{U}, \mathcal{V}, \boldsymbol{\pi}$
end function
function Prediction($\{\mathbf{f}^{(k)}\}_{k=1}^K, \mathcal{U}, \mathcal{V}, \boldsymbol{\pi}, \mathcal{D}_{\text{test}}$)
    Input: Set of $K$ pre-trained neural networks $\mathbf{f}(\cdot)^{(k)} = \boldsymbol{W}_{\text{MAP}}^{(k)}\boldsymbol{\phi}(\cdot)^{(k)}$, sets $\mathcal{U}$ and $\mathcal{V}$ with the corresponding Kronecker factored covariances, mixture weights $\boldsymbol{\pi} = \{\pi^{(k)}\}_{k=1}^K$, test dataset $\mathcal{D}_{\text{test}}$
    Initialize $\mathcal{Y} = \emptyset$.
    for $\boldsymbol{x}_*$ in $\mathcal{D}_{\text{test}}$ do
        Initialize $\boldsymbol{y} = \mathbf{0} \in \mathbb{R}^C$.
        for $k = 1$ to $K$ do
            $\boldsymbol{m}_*^{(k)} = \boldsymbol{W}_{\text{MAP}}^{(k)}\boldsymbol{\phi}_*^{(k)}$
            $\boldsymbol{C}_*^{(k)} = \langle \boldsymbol{\phi}_*^{(k)}, \boldsymbol{V}^{(k)}\boldsymbol{\phi}_*^{(k)} \rangle \boldsymbol{U}^{(k)}$
            $\boldsymbol{z}_*^{(k)} = \boldsymbol{m}_*^{(k)} / \sqrt{1 + (\pi/8)\,\text{diag}(\boldsymbol{C}_*^{(k)})}$
            $\boldsymbol{y} = \boldsymbol{y} + \pi^{(k)}\sigma(\boldsymbol{z}_*^{(k)})$
        end for
        $\mathcal{Y} = \mathcal{Y} \cup \{\boldsymbol{y}\}$
    end for
    Output: $\mathcal{Y}$
end function
```

angle up to 180 degrees. The corrupted CIFAR-10 benchmark consists of the the standard CIFAR-10 dataset and 5 corruption levels with 16 different corruptions types each. Examples for the corruption types are Gaussian noise and changed brightness. The corrupted ImageNet benchmark consists of the standard ImageNet2012 dataset and 5 corruption levels with 19 corruption types each. The types of corruptions are the same as for corrupted CIFAR-10. Both corrupted datasets were proposed in Hendrycks and Dietterich [19]; see Figure 1 in their paper for example images of the different corruption types.

As OOD datasets for the OOD experiment we use SVHN, LSUN-Classroom, and CIFAR-100. All datasets are available through the PyTorch `torchvision` package [43].

F.3 Training

For MNIST we use a five-layer LeNet [29] and Adam [26] and initial learning rate of 1e-3. For CIFAR-10 we use a Wide ResNet (WRN) 16-4 [53] with dropout rate of 0.3 and stochastic gradient descent (SGD) with Nesterov momentum of 0.9 and inital learning rate of 0.1; we also use standard data augmentation, i.e. random cropping and flipping. For both, we train for 100 epochs with mini-batch size of 128, use a cosine learning rate schedule [32], and weight decay with factor 5e-4. For ImageNet we use a WRN 50-2 and follow the PyTorch example training script [8]: we optimize for 90 epochs with SGD with initial learning rate of 0.1 and decrease it by 90% at epoch 30 and epoch 60; we also use weight decay of 1e-4 and random cropping and flipping as data augmentation. The LeNet, WRN 16-14, and WRN 50-2 achieve 99.2%, 94.8%, and 77.6% accuracy, respectively.

F.4 Hyperparameter Tuning

We only tune one hyperparameter for LLLA and MoLA, namely the prior precision. For MoLA we tune a single scalar prior precision for all components. We want to choose it high enough to avoid underconfidence on in-distribution data while keeping it low enough that the behaviour does not becomes too similar to MAP or DE. Hence, we perform a simple grid search on a validation set of size 2000, going over a range of up to 100 values for the prior precision. The validation set is a randomly sampled subset of the test dataset. We start with a prior precision of 1e-4 and go up to at most 1e3 and check if the mean confidence and Brier score on the validation set fulfill a threshold. We then choose the first, i.e. smallest, prior precision which fulfills these thresholds. We set the thresholds for the mean confidence to be slightly below the accuracy of MAP on the validation set. For MNIST the threshold is 0.98 and for CIFAR-10 it is 0.94. While we have also set a threshold on the Brier score for our experiments, just the threshold on the average confidence is sufficient by itself. Note that we could also choose some other method to tune the prior precision; however, in practice our heuristic method seems sufficient.

For SWAG and MultiSWAG, we follow Maddox et al. [35] and run stochastic gradient descent with a constant learning rate on the pre-trained models to collect one model snapshot per epoch, for a total of 40 snapshots. At test time, we then make predictions by using 30 Monte Carlo samples from the posterior distribution; we correct the batch normalization statistics of each sample as described in Maddox et al. [35]. To tune the constant learning rate, we used the same approach as for tuning the prior precision of LLLA and MoLA described above, combining a grid search with a threshold on the mean confidence. For MNIST, we defined the grid to be the set { 1e-1, 5e-2, 1e-2, 5e-3, 1e-3 }, yielding an optimal value of 1e-2. For CIFAR-10, searching over the same grid suggested that the optimal value lies between 5e-3 and 1e-3; another, finer-grained grid search over the set { 5e-3, 4e-3, 3e-3, 2e-3, 1e-3 } then revealed the best value to be 2e-3.

F.5 Implementation Details of MIMO and MIMO-MoLA

For MIMO and MIMO-MoLA, we only use $K = 3$ ensemble components due to the limited capacity of the model. We use input repetition $\rho = 0$ and batch repetition of 4; see Havasi et al. [17] for more details on these hyperparameters. As for MoLA, we only tune a single scalar prior precision for MIMO-MoLA, using the heuristic described in Appendix F.4. In contrast to LLLA and MoLA, we use the full GGN of each head of the last layer for MIMO-MoLA, in favor of implementational convenience. While the results on corrupted CIFAR-10 are promising, a more thorough study of MIMO-MoLA is necessary.

F.6 Computing Resources

All experiments were conducted on a cluster with multiple NVIDIA RTX 2080Ti and Tesla V100 GPUs.

[8] https://github.com/pytorch/examples/tree/master/imagenet

G Additional Results

G.1 Varying the Number of Mixture Components

To check the behavior of MoLA with increasing number of mixture components, we compare DE and MoLA with one to ten mixture components on CIFAR-10-C. For each number of mixture components, we average each metric over all corruption levels and types. We observe that MoLA follows a similar power-law-like behaviour as DE with increasing number of mixture components; see Lobacheva et al. [31] for a detailed investigation in these power laws in DEs. Interestingly, just one component of MoLA, i.e. LLLA, performs better than DE with ten components regarding log-likelihood and ECE; it also achieves lower mean confidence. The only metric where DE performs as well as MoLA is accuracy; DE is even better, albeit by a very small margin (note the scale of the y-axis).

H Full Results

H.1 Variations of LLLA

To explore the effect of the K-FAC approximation of the Hessian (Appendix B.2) and the multi-class probit approximation (MPA) to the predictive distribution on performance, we compare different variations of LLLA on CIFAR-10-C. We test all six combinations of diagonal/K-FAC approximation/full Hessian and the MC integral (4)/MPA (6). We use 100 MC samples for the MC integral. The prior precision is tuned by choosing the smallest from a grid with 100 steps which leads to the mean confidence on the validation set being larger than 0.94. Regarding accuracy and Brier score, the least accurate approximation, i.e. the diagonal/MPA combination, performs worst. The most accurate approximation, i.e. the full/MC combination performs best regarding log-likelihood (by a small margin and with overlapping error bars), ECE (tied with K-FAC/MC), and Brier score. While we prefer the combination of K-FAC, which is even feasible for models with a large last-layer, e.g. WRN 50-2 for ImageNet, and the MPA, which enables predictions with negligible additional cost compared to a regular forward pass, for our experiments, it is of course feasible to use a K-FAC/full covariance and a MC approximation to the predictive distribution. Using one of these two combinations might further increase performance.

H.2 Rotated MNST

On MNIST-R, MoLA is only marginally better than the next best methods regarding log-likelihood, Brier score, and ECE. In terms of accuracy, all methods are very similar, with MultiSWAG being the best by small margin. MoLA achieves the lowest mean confidence. There is no clear trend regarding MCE.

H.3 Corrupted CIFAR-10

On CIFAR-10-C, MoLA performs best regarding log-likelihood, Brier score, and ECE. In terms of accuracy it is similar to MultiSWAG, altough slightly better. There is no clear trend regarding MCE. Interestingly, LLLA performs better than all methods besides MoLA regarding log-likelihood, despite only using a single model; moreover, it outperforms DE regarding ECE.

H.4 Corrupted ImageNet

On ImageNet-C, MoLA performs more or less the same as DE regarding log-likelihood, accuracy, and Brier score. MoLA performs worse than DE on ECE for low corruption severity but better for high corruption severity. DE is alreay well calibrated for low corrutpion severity and MoLA becomes slightly underconfident. LLLA outperforms MAP on log-likelihood and Brier score, but only for higher corruption severity. On ECE, LLLA performs much better than MAP and almost the same as DE and MoLA (even better than DE for high corruption severity). In terms of accuracy, LLLA performs about the same as MAP. Regarding MCE, MAP is outperformed by all other methods.

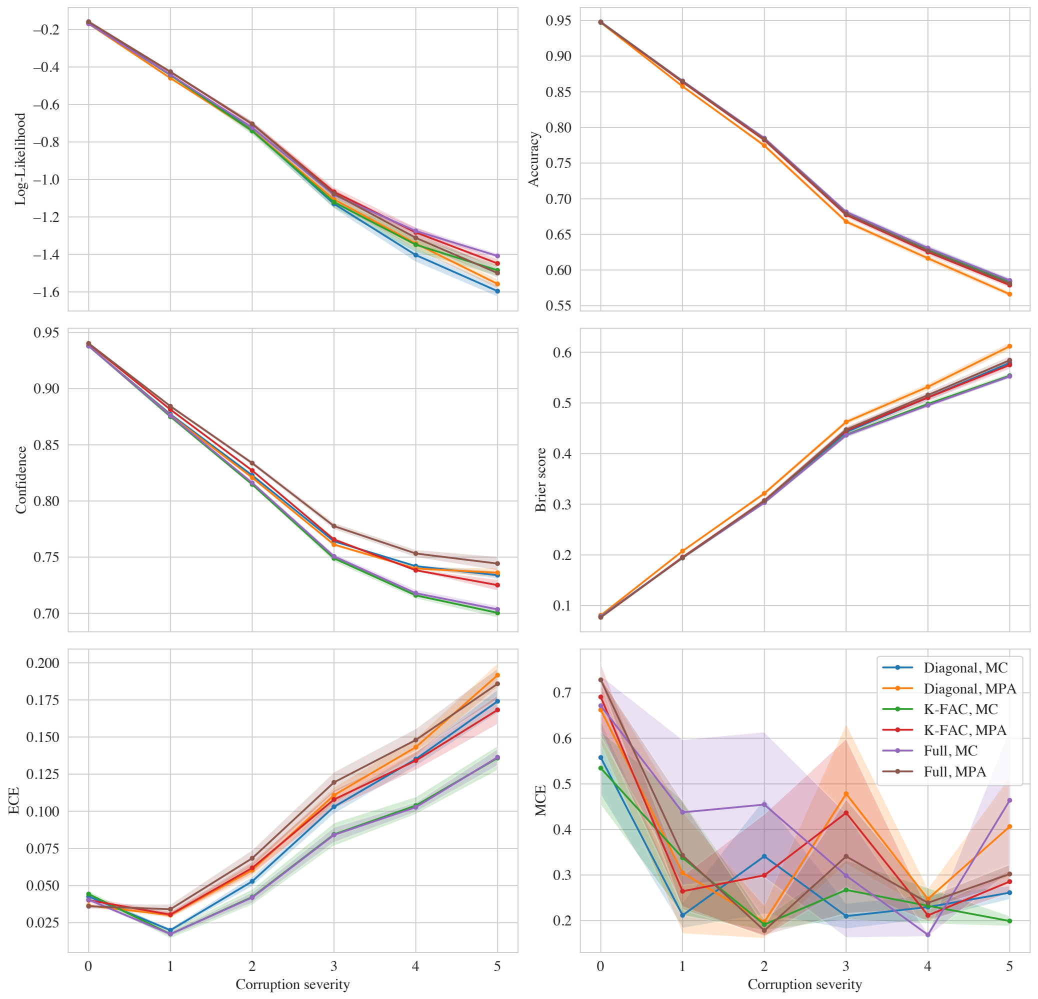


Figure 6: Comparison of different variations of LLLA on the corrupted CIFAR-10 dataset. Lines and dots are means over all corruption types at a particular severity level while shades represent standard errors of five runs. MC stands for "Monte Carlo" and MPA for "multi-class probit approximation".

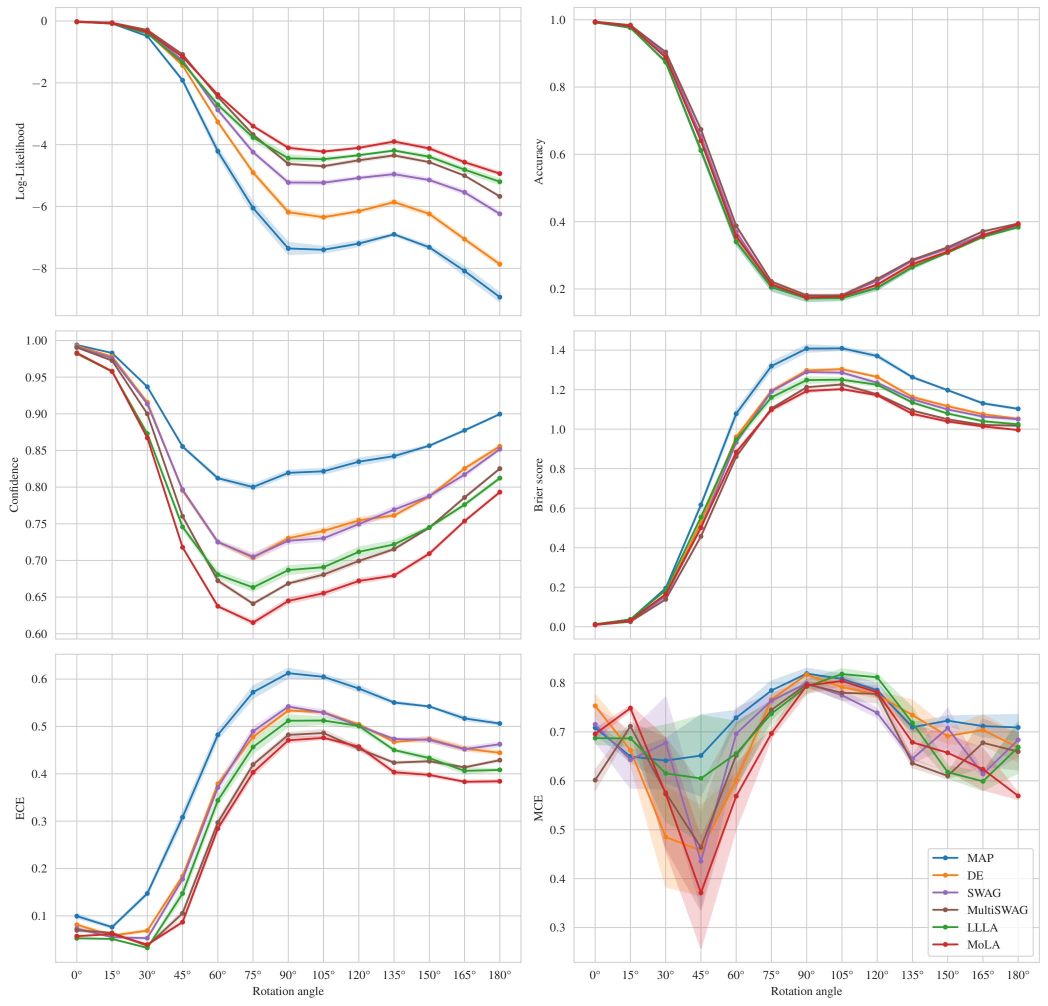


Figure 7: Comparison of all methods on rotated MNIST. Metrics are averaged over all corruption types for each corruption severity. Lines and dots represent means while shades represent standard errors of five runs with models trained with different random initalizations.

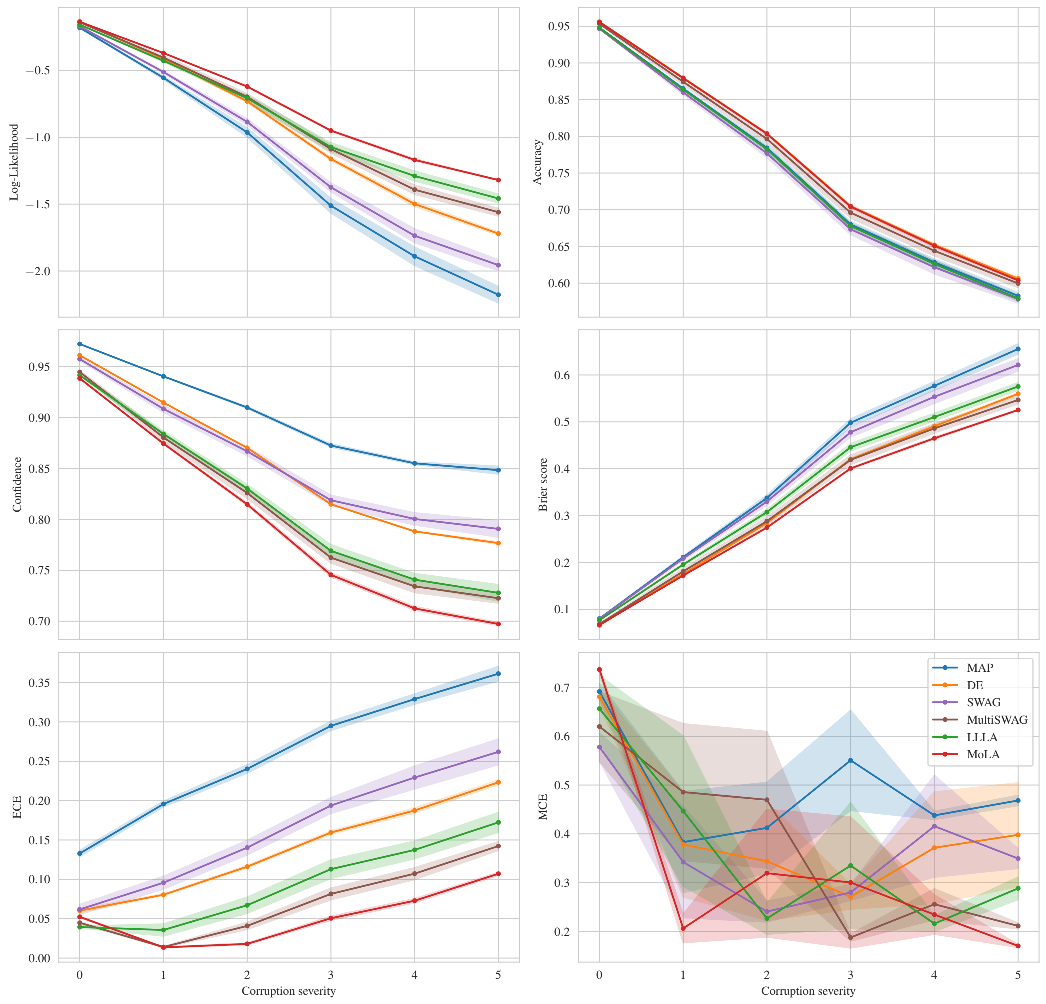


Figure 8: Comparison of all methods on corrupted CIFAR-10. Metrics are averaged over all corruption types for each corruption severity. Lines and dots represent means while shades represent standard errors of five runs with models trained with different random initalizations.

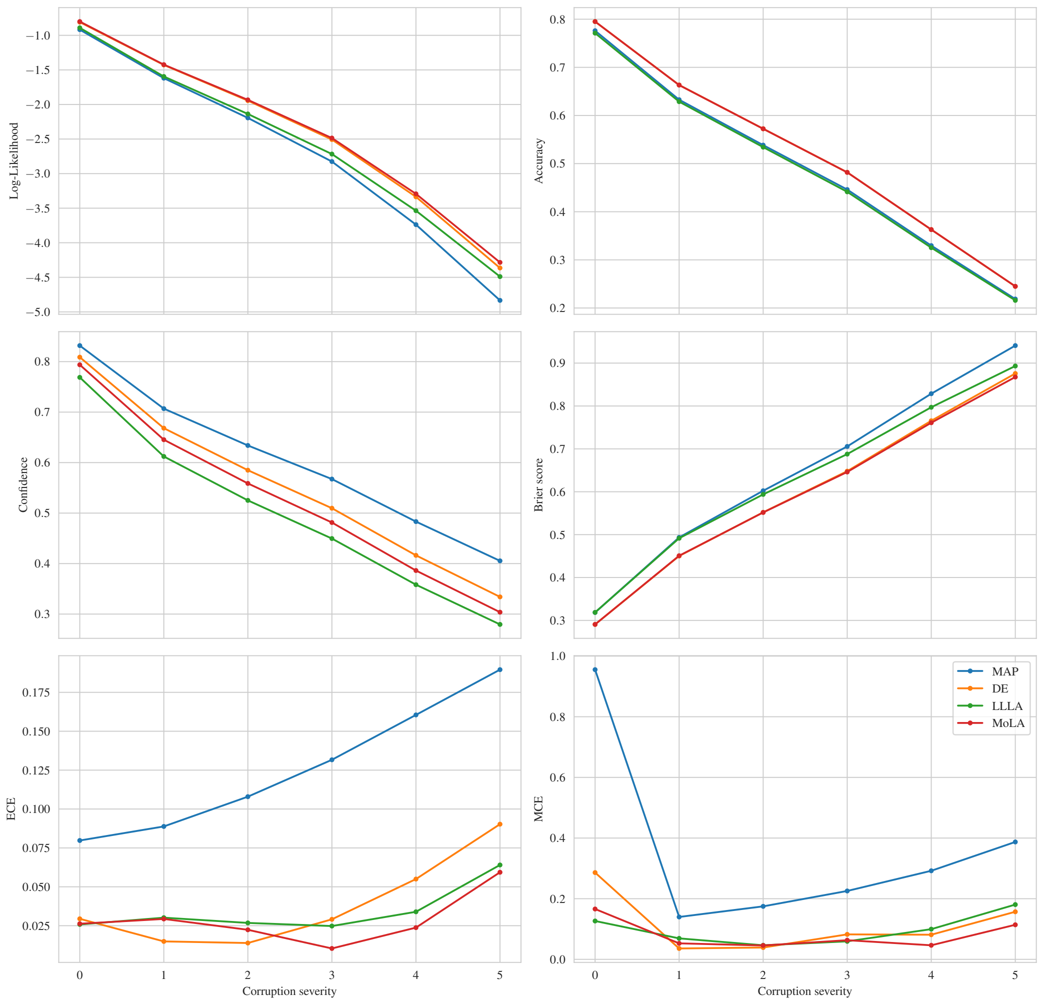


Figure 9: Comparison of MAP, DE, LLLA, and MoLA on corrupted ImageNet. Lines and dots represent the mean over all corruption types for each corruption severity.

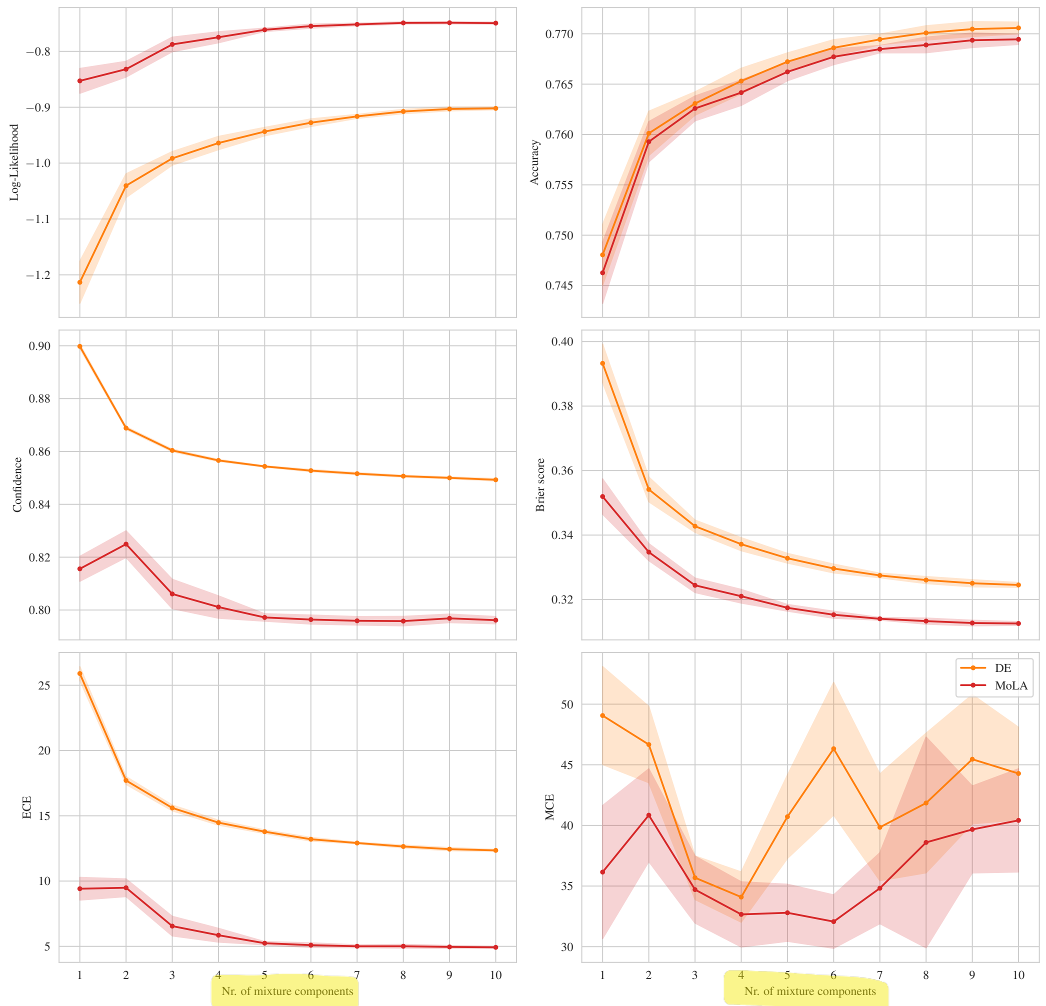


Figure 10: Comparison of different numbers of mixture components on corrupted CIFAR-10. Metrics are averaged over all corruption severities and types. Lines and dots represent means while shades represent standard error of five runs with models trained with different random initalizations.