Evaluating Prediction-Time Batch Normalization for Robustness under Covariate Shift

Zachary Nado [1] **Shreyas Padhy** [2 1] **D. Sculley** [1] **Alexander D'Amour** [1] **Balaji Lakshminarayanan** [1] **Jasper Snoek** [1]

Abstract

Covariate shift has been shown to sharply degrade both predictive accuracy and the calibration of uncertainty estimates for deep learning models. This is worrying, because covariate shift is prevalent in a wide range of real world deployment settings. However, in this paper, we note that frequently there exists the potential to access small unlabeled batches of the shifted data just before prediction time. This interesting observation enables a simple but surprisingly effective method which we call prediction-time batch normalization, which significantly improves model accuracy and calibration under covariate shift. Using this one line code change, we achieve state-of-the-art on recent covariate shift benchmarks and an mCE of 60.28% on the challenging ImageNet-C dataset; to our knowledge, this is the best result for any model that does not incorporate additional data augmentation or modification of the training pipeline. However, the method has mixed results when used alongside pre-training, and does not seem to perform as well under more natural types of dataset shift, and is therefore worthy of additional study. We include links to the data in our figures to improve reproducibility, including a Python notebooks that can be run to easily modify our analysis at this url.

1. Introduction

Covariate shift is one of the key problems facing modern machine learning. Informally defined as situations in which training data differs from the data seen at final prediction time, covariate shift breaks the traditional i.i.d. assumptions used to underpin supervised machine learning (Vapnik, 1995). For deep models in particular, covariate shift has been shown to not only cause incorrect predictions, but to do so with disproportionately high levels of confidence (Ovadia et al., 2019). This is potentially worrying, because covariate shift often occurs in practical settings such as real-world deployment of ML systems (McMahan et al., 2013), for reasons that may include non-stationarity over time or differences between local and global distributions. Furthermore, traditional training-time methods for covariate shift correction may be impractical in such settings.

We propose one such method, which we call prediction-time batch normalization, as an extension of the widely adopted (training time) batch normalization method proposed by (Ioffe & Szegedy, 2015). We also carefully analyze the model performance, teasing apart several factors to help understand why this method works well, identifying key factors via ablation studies. Finally, we explore the limits of the method under more natural types of dataset shift and examine its potential failure modes, notably its lackluster performance when combined with pre-training, which indicates that the method is worthy of additional study.. Together the results in this paper lay out an interesting and highly practical methodology which could be used, with caveats, to correct for covariate shift in real-world deployment settings.

2. Setup

Here, we formalize our prediction setting. We observe feature-label pairs $\{(_i, y_i)\}_{i=1}^{N}$ drawn i.i.d. from some training distribution $p(, y)$, and wish to predict the labels of unlabeled test examples $\{_j\}_{j=1}^{T}$. We assume that the test examples are drawn i.i.d. from a potentially distinct, unknown target distribution $q()$, with an accompanying conditional label distribution $q(y \mid)$.

Unlike the standard supervised learning setting, we assume that predictions can be made in batches. Specifically, at prediction time, we obtain batches of $t < T$ examples, $^{(b)} = (^{(b)}_i)_{i=1}^{t}$, and make predictions for these examples simultaneously, $\hat{}^{(b)} = (\hat{y}_1^{(b)}, \ldots, \hat{y}_t^{(b)}) = f_{(}^{(b)})$. We express our goal as minimizing a predictive risk that is evaluated batch-wise. Specifically, let $\ell(^{(b)}, f_{(}^{(b)})) := \sum_{i=1}^{t} \ell(y_i^{(b)}, \hat{y}_i^{(b)})$ be a batch-wise loss function that decomposes additively across the points in a test batch. Our goal is to minimize

[1]Google Research, Brain Team [2]Work done as an AI Resident. Correspondence to: Zachary Nado <znado@google.com>.

Presented at the ICML 2020 Workshop on Uncertainty and Robustness in Deep Learning. Copyright 2020 by the author(s).

the expected loss, or risk, over i.i.d. test batches $(,)$ drawn from the product distribution $q(, y)^t := \prod_{i=1}^{t} q(_i, y_i)$:

$$\min_{\mathbb{E}_{(,)\sim q(,y)^t}\left[\ell(,f_{()})\right].} \tag{1}$$

Conveniently, because the loss function ℓ decomposes linearly, the empirical analogue of this risk can be computed in the standard way, as the mean loss across examples $\ell(y_i, \hat{y}_i)$. Thus, empirical evaluation in this setting can be done using standard pipelines.[1]

2.1. Reliable Uncertainty Quantification Under Covariate Shift

Formally, we quantify the quality of the uncertainty of a predictive distribution using two measures, *calibration error* and *Brier score*. **Expected Calibration Error** (ECE) (Guo et al., 2017) is the difference between the confidence and accuracy of a model, binned by confidence. We define confidence as the max predicted class probability for a given example. Let B_i be the elements in each confidence bin, then ECE $= \sum_i \frac{B_i}{N} |\operatorname{acc}(B_i) - \operatorname{conf}(B_i)|$, where $\operatorname{acc}(B_i)$ is the accuracy of elements in B_i and $\operatorname{conf}(B_i)$ is the confidence of elements in B_i. **Brier Score** (Brier, 1950) is defined as the squared distance between a model output distribution and the one-hot target labels. It is a proper scoring rule (Gneiting & Raftery, 2007), and as such decreases to zero monotonically as the predictive distribution approaches the true underlying distribution.

3. Related work

Wilson & Cook (2018) survey different methods in unsupervised domain adaptation. These include methods that learn mappings between domains (Fernando et al., 2013; Sener et al., 2016), match means and covariances across feature vectors (Sun et al., 2017), or match moments of the distributions directly (Peng et al., 2019) or through kernel embeddings (Long et al., 2015; Gong et al., 2012). Importance weighting methods are also a common approach, where training examples are reweighted to minimize an estimate of the predictive risk in the target domain (Sugiyama et al., 2007). A common theme in these approaches is that they assume access to a set of unlabeled samples from the test set at training time, whereas our focus is to apply an intervention at test time, after a model has been trained. In addition, these methods can become brittle when the training and test distributions are highly distinct (Johansson et al., 2019).

There has been work in the domain adaptation literature that specifically focuses on using batch normalization as a means of mapping between domains, and Li et al. (2016) make the claim that the batch norm statistics in deep networks learn domain-specific knowledge. They propose AdaBN, which calculates domain-specific batch norm statistics using the entirety of the target domain at test-time (or an exponential moving average (EMA) in practice). AutoDIAL (Cariucci et al., 2017) mixes data from the source and target domains during training time before passing them through the batch norm layers, whereas TransNorm (Wang et al., 2019) uses source and target domain batch norm statistics during training in an end-to-end fashion to improve transferability across domains.

While Guo et al. (2017) observed that models using traditional batch norm typically have worse calibration on the test set, to our knowledge no one has applied normalization strategies for correcting miscalibration under covariate shift.

The prediction-time batch setting has also been explored for deep generative models, particularly in the context of addressing their failure modes for OOD detection as reported by Nalisnick et al. (2019a); Choi et al. (2019b). Nalisnick et al. (2019b) propose a typicality test for generative models that performs OOD detection using a batch of inputs. Song et al. (2019) use prediction-time batch normalization in deep generative models and show that it improves OOD detection. This is perhaps the closest related work, but is complementary to our method, as we focus on discriminative models and covariate shift rather than OOD detection.

4. Prediction-Time Batch Normalization

We propose a simple protocol to mitigate the effects of covariate shift at prediction time in a batch-prediction setting: use batch normalization (BN) with statistics recalculated on the batch at prediction time. In standard practice, the batch normalization statistics are frozen to particular values after training time that are used to compute predictions, an approach we refer to as **train BN**. On the other hand, our strategy, **prediction-time BN**, recomputes these statistics for each test batch.

To move toward an understanding of when prediction-time BN works, we offer two observations here. We then perform a more thorough exploration of this method with ablation studies in Section A.

Prediction-Time Batch Normalization Repairs Mismatched Supports As $q()$ shifts, the internal activations of a deep model $f_\theta()$ can move outside the ranges encountered during training, as seen in the left column of Figure 1. When this happens, the model layers receive inputs outside of the domain they were trained on, and we can no longer expect well-defined model behavior such as accurate

[1]The within-batch dependence introduced by simultaneous predictions $f^{(b)}_{(}$) will generally alter the concentration properties of this empirical mean, but this effect is modest if the batch size is small relative to the size of the test set. We conjecture that, in the worst case, generalization bounds scale in the number of batches rather than the number of examples.

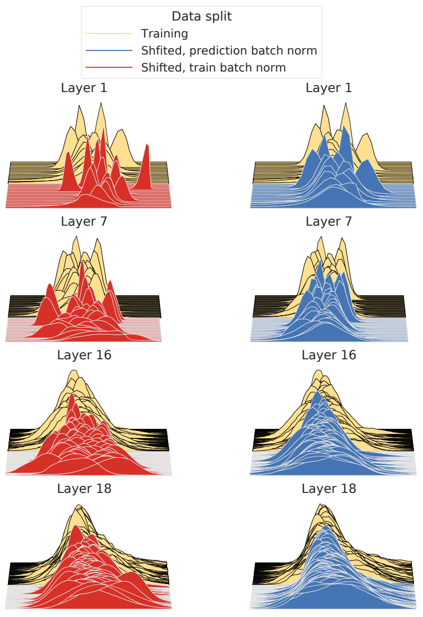

Figure 1. **Empirical distributions for the output of selected normalization layers in Resnet-20 on CIFAR10 and CIFAR10-C.** Activations are averaged over spatial dimensions, resulting in one distribution per output channel. The activations are recorded immediately after the batch normalization layer, before the non-linearity of each layer. The blue and red curves are aggregated across all shifted examples, while the yellow is across all training examples. We can clearly see that prediction-time BN is much more effective at aligning the shifted activations with the training distribution support and shape. These layers were picked as representative examples of activations of all normalization layers in the model, we encourage the reader to check Figures S18, S19 for all layers.

or well-calibrated predictions. The top left panel of Figure 2 demonstrates this behavior, with shifted examples inducing worsening calibrations as measured using Brier Score. Overall there is a trend of decreasing calibration as the distance between test and train activations increases (see Figure S13 for a similar trend in accuracy).

In Figure 2, we visualize the discrepancy between the supports of the empirical training and test activation distributions for the penultimate hidden layer of Resnet-20 on CIFAR10-C. We see that the prediction-time BN correction is effective at bringing the activation distribution supports into alignment (clustered around 0 on the horizontal axis) relative to both train BN and other normalization schemes.

Prediction-Time Batch Normalization Maps Activations to Regions of Uncertainty Aligning the supports of activation distributions is not sufficient to ensure well-calibrated predictive distributions. For example, it would be plausible that a normalization scheme could spuriously map the activations of out-of-support test instances to regions in the activation space that induce highly confident predictions. We find that prediction-time BN avoids this potential failure

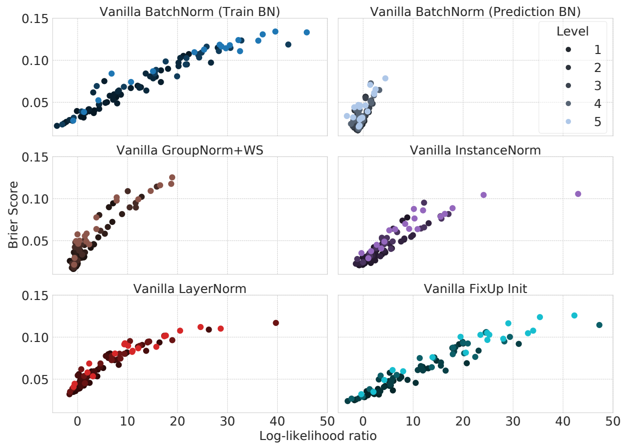

Figure 2. **Brier scores of predictions become higher when the activations from the training and test sets occur in increasingly distinct regions.** Here, we summarize how the distributions of penultimate hidden layer activations $= g()$ on shifted test sets compare to their distributions on the training set, under a number of different normalization schemes. Each point represents a type of shift, where the color indicates the intensity of the shift applied. On the horizontal axis, we plot a measure of the discrepancy between the training and test distributions of activations, $p()$ and $q()$, respectively by approximating $KL(p()\|q()) \approx T^{-1}\sum_{i=1}^{T} \ln \frac{\hat{q}(_i)}{\hat{p}(_i)}$, where the summation is taken over test instances, and $\hat{p}$ and $\hat{q}$ are multivariate normal densities whose means and variances match p and q, respectively. On the vertical axes are the Brier scores for each shifted example, averaged within each split. In addition to the overall trend of increasing discrepancy leading to decreasing performance, we also see that higher shift intensities tend to have higher support mismatch.

mode. As Figure 2 suggests, prediction-time BN seems to map out-of-support activations to regions in the training activation support that induce uncertain predictions, resulting in consistently lower Brier scores when the normalization is applied. This pattern is confirmed in Figure S5.

5. Performance Under Covariate Shift

In our empirical analyses, we consider methods that are test-time only modifications of neural networks. Our method should be applicable to any model with batch normalization. Many sophisticated Bayesian techniques, such as those explored in Ovadia et al. (2019), also require modifications at training time, which is not the setting we consider. In addition to our vanilla baseline we also consider temperature scaling (Guo et al., 2017; Platt, 1999) and ensembles (Lakshminarayanan et al., 2017) with 10 members.

5.1. Prediction Batch Dependence

One potential concern with prediction-time BN is that predictions now depend on other examples and the prediction batch size. Figure 3 shows performance across a range of prediction batch sizes and see we achieve strong performance with a batch of just 100 examples, with larger batch

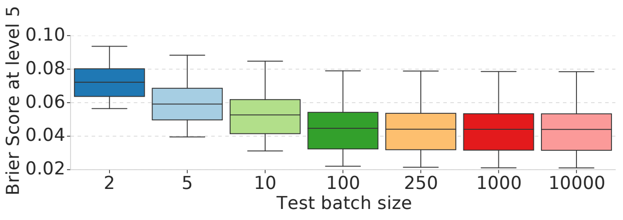

Figure 3. **CIFAR-10-C Brier Score at shift level 5 for different prediction batch sizes.** We see that relatively small batch sizes are required to effectively correct for covariate shift, with only marginal improvements after a 100 examples. See Figure S17 for a similar trend on ImageNet-C.

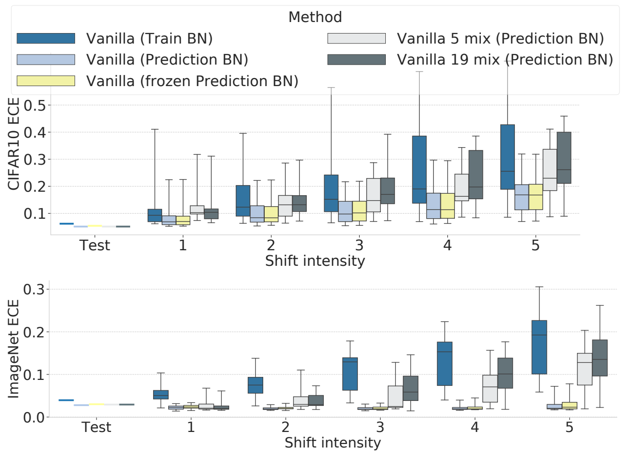

Figure 4. **Calibration on CIFAR-10-C (top) and ImageNet-C (bottom), both with a prediction batch size of 500.** Here we clearly see that just having access to a single batch from each split (frozen Prediction BN) is sufficient to get substantial performance improvements. Also, while prediction-time BN is sensitive to multiple simultaneous types of covariate shift, it still outperforms train BN.

sizes giving marginal improvements.

We also investigate whether the performance gains are from having the statistics of the exact batch we are predicting on, or just from having statistics that are relevant to the prediction distribution's shift. In Figure 4 the statistics from the first batch of each split are stored and reused for all subsequent batches of that split. We only see a marginal performance decrease, meaning we can remove our prediction batch dependence while retaining the performance improvements. Additionally, we test calibration under multiple corruption types and levels at once, and see that even with up to 19 different types of simultaneous shift, prediction-time BN outperforms train BN.

5.2. Limitations

We see negative results when evaluating prediction-time BN on a pre-trained Noisy Student model from (Xie et al., 2019). As seen in Figure S11, prediction-time BN actually does worse than train BN. We believe this is because the statistics used by train BN contain information from the model pre-training. See Appendix F.2 for more details.

In Appendix F.3 we see mixed results on the ImageNet-v2 dataset, which corresponds to a different type of dataset shift. We believe this is because the normalization statistics from ImageNet-v1 are still representative of the v2 dataset, because they are both sets of natural images, whereas the types of shifts in ImageNet-C change the image statistics in characteristic ways.

Exploring the activation distribution alignment in both of these failure modes, as was done in Figures 1, 2, is an area we are pursuing as future work.

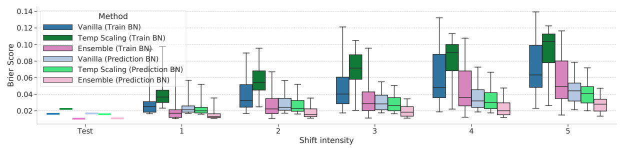

Figure 5. **Calibration under covariate shift for CIFAR-10-C with a prediction batch size of 500 (lower Brier Score is better).** The box plots show the median, quartiles, minimum, and maximum performance per method. Ensembles with prediction batch norm appear minimally affected by the level of shift. See Figure S12 for other metrics. The data and notebook for this plot can be found at https://tensorboard.dev/experiment/IwvHAvuxTZK00Wp76rJwqw/.

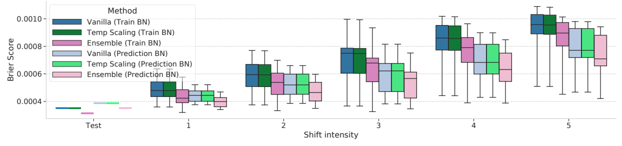

Figure 6. **Calibration under covariate shift for ImageNet-C with a prediction batch size of 100.** The box plots show the median, quartiles, minimum, and maximum performance per method. See Figure S14 for other metrics. The data and notebook for this plot can be found at https://tensorboard.dev/experiment/FRbuxfG5SkaFPQQH4OcpYw/.

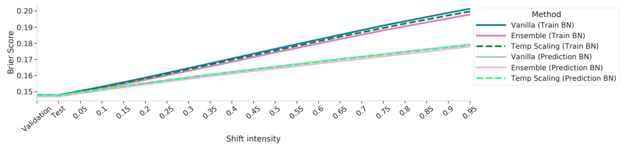

Figure 7. **Calibration under covariate shift for Criteo with a prediction batch size of 500.** See Figure S15 for AUC performance. The data and notebook for this plot can be found at https://tensorboard.dev/experiment/dNxyMRncRgSzozlD1m94Og/.

References

TensorFlow Datasets, a collection of ready-to-use datasets. `https://www.tensorflow.org/datasets`.

Ba, L. J., Kiros, R., and Hinton, G. E. Layer normalization. corr abs/1607.06450 (2016). *arXiv preprint arXiv:1607.06450*, 2016.

Brier, G. W. Verification of forecasts expressed in terms of probability. *Monthly weather review*, 1950.

Cariucci, F. M., Porzi, L., Caputo, B., Ricci, E., and Bulo, S. R. Autodial: Automatic domain alignment layers. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 5077–5085. IEEE, 2017.

Choi, D., Shallue, C. J., Nado, Z., Lee, J., Maddison, C. J., and Dahl, G. E. On empirical comparisons of optimizers for deep learning. *arXiv preprint arXiv:1910.05446*, 2019a.

Choi, H., Jang, E., and Alemi, A. WAIC, but Why?: Generative Ensembles for Robust Anomaly Detection. *ArXiv e-Print arXiv:1810.01392v3*, 2019b.

Fernando, B., Habrard, A., Sebban, M., and Tuytelaars, T. Unsupervised visual domain adaptation using subspace alignment. In *Proceedings of the IEEE international conference on computer vision*, pp. 2960–2967, 2013.

Gneiting, T. and Raftery, A. E. Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 102(477):359–378, 2007.

Gong, B., Shi, Y., Sha, F., and Grauman, K. Geodesic flow kernel for unsupervised domain adaptation. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2066–2073. IEEE, 2012.

Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. Q. On calibration of modern neural networks. In *International Conference on Machine Learning*, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

Hendrycks, D. and Dietterich, T. Benchmarking neural network robustness to common corruptions and perturbations. *arXiv preprint arXiv:1903.12261*, 2019.

Hendrycks, D., Zhao, K., Basart, S., Steinhardt, J., and Song, D. Natural adversarial examples. *arXiv preprint arXiv:1907.07174*, 2019.

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

Huang, G., Liu, Z., and Weinberger, K. Q. Densely connected convolutional networks. corr abs/1608.06993 (2016). *arXiv preprint arXiv:1608.06993*, 2016.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. corr abs/1502.03167. *arXiv preprint arXiv:1502.03167*, 2015.

Johansson, F. D., Sontag, D., and Ranganath, R. Support and invertibility in domain-invariant representations. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 527–536, 2019.

Lakshminarayanan, B., Pritzel, A., and Blundell, C. Simple and Scalable Predictive Uncertainty Estimation Using Deep Ensembles. In *NeurIPS*, 2017.

Li, Y., Wang, N., Shi, J., Liu, J., and Hou, X. Revisiting batch normalization for practical domain adaptation. *arXiv preprint arXiv:1603.04779*, 2016.

Long, M., Cao, Y., Wang, J., and Jordan, M. I. Learning transferable features with deep adaptation networks. *arXiv preprint arXiv:1502.02791*, 2015.

McMahan, H. B., Holt, G., Sculley, D., Young, M., Ebner, D., Grady, J., Nie, L., Phillips, T., Davydov, E., Golovin, D., et al. Ad click prediction: a view from the trenches. In *KDD 2013: Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2013.

Nalisnick, E., Matsukawa, A., Whye Teh, Y., Gorur, D., and Lakshminarayanan, B. Do Deep Generative Models Know What They Don't Know? In *International Conference on Learning Representations (ICLR)*, 2019a.

Nalisnick, E., Matsukawa, A., Whye Teh, Y., Gorur, D., and Lakshminarayanan, B. Do Deep Generative Models Know What They Don't Know? In *ICLR*, 2019b.

Ovadia, Y., Fertig, E., Ren, J., Nado, Z., Sculley, D., Nowozin, S., Dillon, J. V., Lakshminarayanan, B., and Snoek, J. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. In *Advances in Neural Information Processing Systems (NeurIPS 2019)*, 2019.

Peng, X., Bai, Q., Xia, X., Huang, Z., Saenko, K., and Wang, B. Moment matching for multi-source domain adaptation. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1406–1415, 2019.

Platt, J. C. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *Advances in Large Margin Classifiers*, pp. 61–74. MIT Press, 1999.

Qiao, S., Wang, H., Liu, C., Shen, W., and Yuille, A. Weight standardization. *arXiv preprint arXiv:1903.10520*, 2019.

Recht, B., Roelofs, R., Schmidt, L., and Shankar, V. Do imagenet classifiers generalize to imagenet? *arXiv preprint arXiv:1902.10811*, 2019.

Sener, O., Song, H. O., Saxena, A., and Savarese, S. Learning transferrable representations for unsupervised domain adaptation. In *Advances in Neural Information Processing Systems*, pp. 2110–2118, 2016.

Song, J., Song, Y., and Ermon, S. Unsupervised out-of-distribution detection with batch normalization. *arXiv preprint arXiv:1910.09115*, 2019.

Sugiyama, M., Krauledat, M., and Müller, K.-R. Covariate shift adaptation by importance weighted cross validation. *Journal of Machine Learning Research*, 8:985–1005, 05 2007.

Sun, B., Feng, J., and Saenko, K. Correlation alignment for unsupervised domain adaptation. In *Domain Adaptation in Computer Vision Applications*, pp. 153–171. Springer, 2017.

Ulyanov, D., Vedaldi, A., and Lempitsky, V. S. Instance normalization: The missing ingredient for fast stylization. corr abs/1607.08022 (2016). *arXiv preprint arXiv:1607.08022*, 2016.

Vapnik, V. N. *The Nature of Statistical Learning Theory*. Springer-Verlag, Berlin, Heidelberg, 1995. ISBN 0387945598.

Wang, X., Jin, Y., Long, M., Wang, J., and Jordan, M. I. Transferable normalization: Towards improving transferability of deep neural networks. In *Advances in Neural Information Processing Systems*, pp. 1951–1961, 2019.

Wilson, G. and Cook, D. J. A survey of unsupervised deep domain adaptation. *arXiv preprint arXiv:1812.02849*, 2018.

Wu, Y. and He, K. Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 3–19, 2018.

Xie, Q., Hovy, E., Luong, M.-T., and Le, Q. V. Self-training with noisy student improves imagenet classification. *arXiv preprint arXiv:1911.04252*, 2019.

Zhang, H., Dauphin, Y. N., and Ma, T. Fixup initialization: Residual learning without normalization. *arXiv preprint arXiv:1901.09321*, 2019.

A. Ablation Studies

In addition to measuring the model calibration of prediction-time BN, we also run several studies to explore hypotheses about the cause of its performance improvements.

A.1. Sensitivity to ϵ

An often forgotten parameter in batch normalization is the ϵ parameter used in the variance term in the denominator. Originally introduced to avoid division by zero (Ioffe & Szegedy, 2015), it is usually left at its default value[2] of 10^{-3} except for some larger models (such as Resnet-50 on ImageNet) where $\epsilon = 10^{-5}$ is common[3].

In the context of model calibration, the denominator term of batch norm can be viewed as temperature scaling that is adaptable through the variance term added to a fixed temperature $\sqrt{\epsilon}$. The classic temperature scaling method (Guo et al., 2017) selects a temperature that minimizes the negative log-likelihood of the model on the labelled validation set. In contrast, the temperature scaling induced by the batch norm variance is unsupervised and can be adapted per batch. Given that classic temperature scaling with training batch norm does not perform as well as prediction-time BN in Figure 5.2, this adaptive temperature is clearly beneficial. In addition to this, we can follow a similar recipe to Guo et al. (2017) where we re-tune batch norm's ϵ for calibration and accuracy on the in-distribution test set and evaluate it on the shifted data. In Figure S1 we measure the ECE and accuracy for several values of ϵ used at prediction time. Given that deep models are typically overly confident under covariate shift (Ovadia et al., 2019), our intuition is to make ϵ as large as possible to compensate for this. However, for both models we see ϵ can only be increased two orders of magnitude from its default values before accuracy collapses. Nonetheless, we can achieve noticeable calibration performance improvements with these higher values. We did see that ϵ does not have much of an effect for ensemble models; one reason for this could be because they are already smoothing the individual models' output distributions when averaging them together. We use the default values of ϵ for all experiments unless otherwise stated, which is 10^{-3} for CIFAR-10 and Criteo and 10^{-5} for ImageNet (these are also the values used during training).

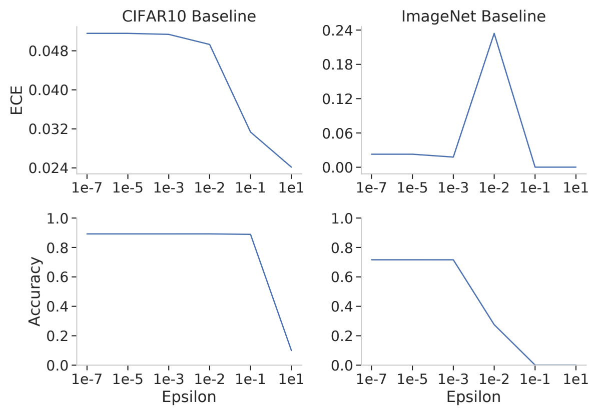

Figure S1. ECE performance across different ϵ values on the **in-distribution test set**. The value that performed the best here was chosen to evaluate on the shifted splits.

A.2. Comparing Normalization Methods

In Figure S2 we evaluate the calibration and accuracy of several normalization methods on CIFAR-10-C: **Instance Normalization** (Ulyanov et al., 2016), a version of Batch Norm which normalizes only along the spatial dimensions per batch element; **Layer Normalization** (Ba et al., 2016), which normalizes along all but the batch dimension; **Group Normalization** (Wu & He, 2018), which is similar to layer norm, but instead of normalizing over all channels at once it splits them into subgroups to compute statistics (we use two). As is commonly done, we use weight Standardization (Qiao et al., 2019) in all experiments with Group Norm. The results echo the distribution mismatch seen in Figure 2. Specifically, using the prediction time batch norm significantly improves both calibration and accuracy under shift. Interestingly, Instance Normalization significantly improves calibration but at the expense of accuracy.

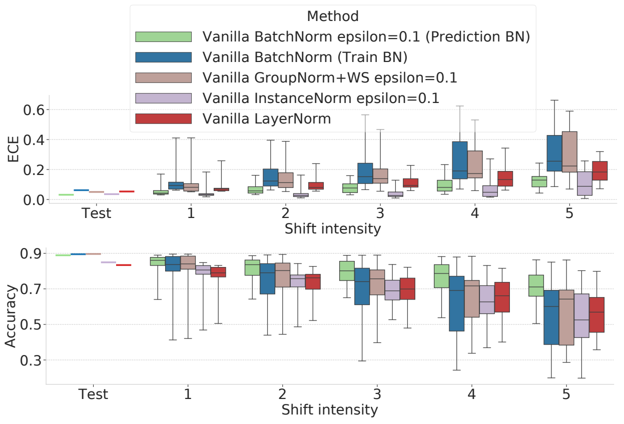

Figure S2. **CIFAR-10-C vanilla model with different normalization methods.** Each method is the same vanilla model but with the normalization layers changed, and all hyperparameters re-tuned for each. Every method was run with $\epsilon \in \{10^{-3}, 10^{-1}\}$ for batch norm at prediction time, and the best performing runs are included here. While instance norm with $\epsilon = 10^{-1}$ has a lower ECE than prediction-time BN, it is notably worse on accuracy. See Figure S16 for more normalization methods.

[2]TensorFlow uses a default of 10^{-3} and defines it as a "small float added to variance to avoid dividing by zero"; https://tensorflow.org/versions/r1.15/api_docs/python/tf/keras/layers/BatchNormalization.

[3]Based on the official TensorFlow model definitions, for example https://git.io/JvsT4.

A.3. Batch Normalization Architectures

While the results in Figure 1 show that prediction-time BN aligns the activation distributions of both the hidden and output layers, we investigate whether or not aligning the layers before the output is necessary to achieve the performance improvements.

In Figure S3 we use the training statistics EMA for all batch norm layers except the last, where we use prediction time batch statistics. The model performance noticeably degrades compared to train BN, implying that the final normalization layer alone cannot compensate for the compounded misalignment of all previous hidden layers. We go a step further in Figure S4 where we remove all normalization layers except for one immediately before the final linear layer. We can recover most but not all of the performance benefits, implying that using prediction-time BN on the internal normalization layers actively helps improve performance. However, normalizing the inputs to the last linear layer of the model significantly improves performance under covariate shift.

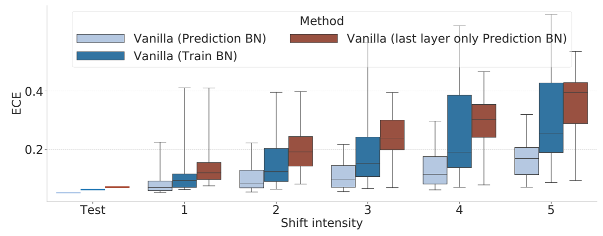

Figure S3. Calibration under covariate shift with the CIFAR-10-C vanilla model where we compare to only using prediction batch statistics on the last batch norm layer.

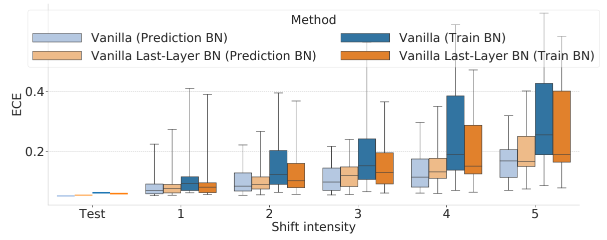

Figure S4. Calibration under covariate shift with the CIFAR-10-C vanilla model compared to an altered Resnet-20 model where we have removed all Batch Norm layers and added one before the final linear layer. We see that we can maintain most, but not all, of the gain in calibration through re-normalizing just the last layer.

B. Understanding Confidence Distributions

Given that prediction-time BN is able to map the shifted activations back into the support of the training distribution, we could expect the model to make predictions with similar accuracy and confidence as during training. However, we still see lower confidence predictions with prediction-time

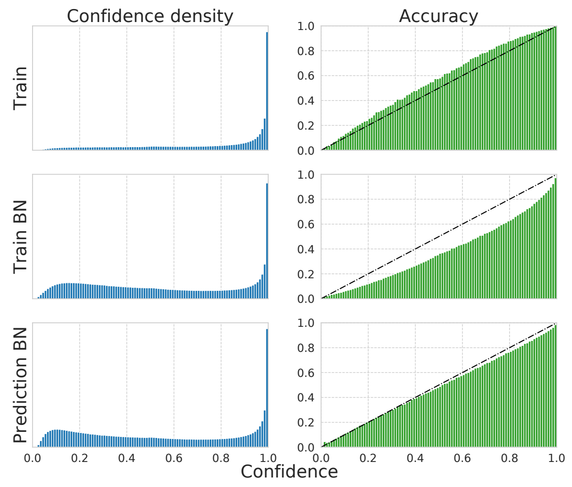

Figure S5. **Resnet-50 on ImageNet-C confidence distribution and accuracies, grouped into 100 equal width confidence bins.** While the model produces lower confidence predictions on the shifted data regardless of batch norm method, prediction-time BN results in slightly lower confidences and higher per-bin accuracies (but not too high as to be underconfident).

BN on ImageNet-C in Figure S5. One possible explanation for this is that even though we are matching the support, we are not precisely matching the distribution densities; while Figure 1 shows that prediction-time BN more closely aligns distribution densities, we see nontrivial misalignment of the eigenspectra of the activation covariance matrices in Figure S6.

C. Datasets

C.1. CIFAR-10

For CIFAR-10 training we applied data augmentation as follows: pad by 4 pixels on all sides with zeros, randomly crop to 32x32 pixels, randomly flip the image, and then rescale to be in [-1, 1]. This can be implemented in TensorFlow with the following Python code:

[language=Python, basicstyle=] image = (tf.image.resize$_{i}mage_{w}ith_{c}rop_{o}r_{p}ad(image, 32 + 4, 32 + 4))image = tf.random_{c}rop(image, [32, 32, 3])image = tf.image.random_{f}lip_{l}eft_{r}ight(image)image = tf.image.convert_{i}mage_{d}type(image, tf.float32)image = 2.0 * (image - 0.5)$ For CIFAR-10-C no pre-processing was applied. In addition to the 15 standard corruption types, we also used the extra corruption defined in Appendix B of (Hendrycks & Dietterich, 2019), which are {`gaussian_blur`, `saturate`, `spatter`, `speckle_noise`}. We used the versions of images as provided by TensorFlow Datasets (TFD).

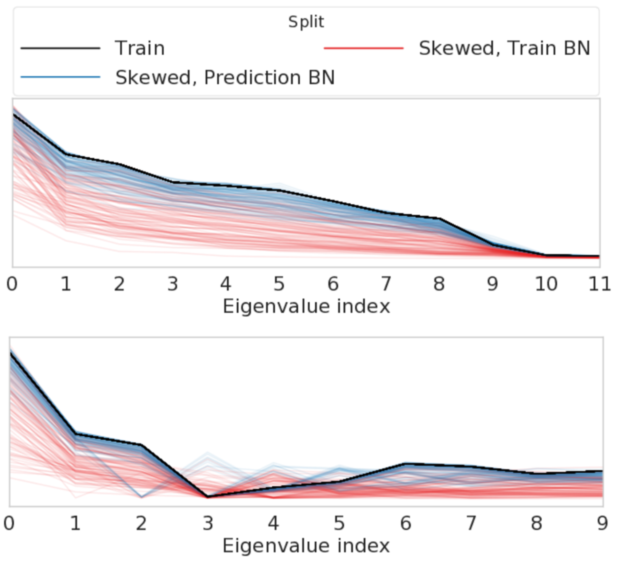

Figure S6. **The eigenvalues of the covariance matrices for the penultimate layer embeddings (top) and logits (bottom) for Resnet-20 on CIFAR-10.** We analyze the eigenspectrum of these covariance matrices to determine how closely the empirical training and test distributions match. While these covariance structures do not indicate anything about aligning distribution supports, they are a proxy for how close the distributions are in shape. We expect there to be some change between the training and shifted activation distribution alignments, because the examples are from similar but not identical data distributions. We see prediction-time BN results in a closer but not identical covariance, which could explain the lower confidence compared to training. Additionally, not being too far from the training covariance could also explain the improved accuracy compared to train BN. Note we truncate the eigenspectra after the top 12 eigenvalues for the embeddings because all values after this index were very close to zero.

C.2. ImageNet

For ImageNet training we used images of size 224x224, and applied standard Inception data augmentation as defined at this url: `https://git.io/JvG6T`.

For ImageNet-C no pre-processing was applied. We used all 19 corruption types, the same as described for CIFAR-10-C.

To avoid tuning on the test set, we hold out 125114 training images as a validation set, and treat the standard "validation" set as a test set.

C.3. Criteo

As done in (Ovadia et al., 2019) we simulate covariate shift in Criteo by randomizing features with increasing probability, ranging from 5% to 95% as seen in Figure S15.

D. Models

CIFAR-10 Our CIFAR-10 model is the standard Resnet-20 v1 (He et al., 2015) with ReLU activations.

ImageNet Our ImageNet model is the standard Resnet-50 v1 (He et al., 2015) with ReLU activations. For ImageNet-A (Hendrycks et al., 2019) we used DenseNet-121 (Huang et al., 2016) as defined in tf.keras.applications: `https://www.tensorflow.org/api_docs/python/tf/keras/applications/DenseNet121`.

Criteo Our Criteo model is the same as in (Ovadia et al., 2019). Summarizing, it encodes each categorical feature into a dense vector which are all then concatenated. This feature vector is then fed into a batch normalization layer followed by three fully connected layers of widths $[2572, 1454, 1596]$, each with a ReLU non-linearity.

E. Hyperparameter Tuning Ranges

Following the recommendations of Choi et al. (2019a), we use random search within the ranges defined in Table E to tune all available hyperparameters for each optimizer. We used 100 random trials for all experiments to tune the learning rate α, one minus the momentum $1-\gamma$, and Adam's ϵ on a logarithmic scale.

For CIFAR-10, we trained for 100 epochs with a batch size of 512. We used a learning rate schedule where the learning rate was reduced at epochs $40, 60, 80, 90$ by $0.1, 0.01, 0.001, 0.0005$.

For ImageNet, we trained for 90 epochs with a batch size of 512. We used a learning rate schedule where the learning rate started at 0 and was linearly increased for the first 5 epochs to α, then reduced at epochs $30, 60, 80$ by $0.1, 0.01, 0.001$.

For Criteo, we trained for 1 epoch with a batch size of 1024 and the same learning rate schedule used for CIFAR-10.

For ECE, we use 10 bins for CIFAR-10-C experiments and 30 bins for ImageNet-C, the same as (Ovadia et al., 2019).

F. Additional Experiments

F.1. Color Filtered ImageNet

In addition to the classic ImageNet dataset, we also ran prediction-time BN on several variants to test its robustness. In Figures S7, S8, S9, S10, we split the ImageNet train and validation splits into two sets based on if the images are dominated by a given color or shade. For example, for ImageNet-Red, we train on images where the green and blue color channels had the highest average pixel values, and evaluate on those that had red as the dominant channel. When we evaluate on red images we refer to this as "out-

Experiment	Optimizer	α	$1-\gamma$	ϵ
CIFAR-10 Resnet-20	Adam	$[10^{-3}, 1]$	$[10^{-2}, 0.15]$	$[10^{-8}, 10^{-5}]$
ImageNet Resnet-50	Nesterov	$[5\times10^{-3}, 5\times10^{-1}]$	$[10^{-3}, 0.15]$	—
Criteo MLP	Adam	$[10^{-4}, 10^{-1}]$	$[10^{-2}, 0.15]$	$[10^{-8}, 10^{-5}]$

Table S1. Tuning ranges for each hyperparameter.

domain", and when we evaluate on green and blue images we call this "in-domain". For dark/light, we simply averaged across all channels and classified dark images as having $mean(image) < 0.5$. The sizes of these splits are shown in Table S2.

Dataset	Train	Validaiton	Test
ImageNet-Red	436103	76376	30832
ImageNet-Green	888422	30889	12186
ImageNet-Blue	987796	17735	6982
ImageNet-Dark	760039	36521	13715

Table S2. Dataset split sizes for our color-filtered ImageNet variations. Note that for our classic ImageNet experiments we hold our 125114 images as a validation set, and treat the standard "validation" set as a test set.

Below we see that prediction-time BN still outperforms train BN on all our ImageNet variants.

F.2. Noisy Student

Using a pre-trained Noisy Student model from (Xie et al., 2019), we evaluate prediction-time BN on ImageNet. As seen in Figure S11, prediction-time BN actually does worse than train BN. We believe this is because the statistics used by train BN contain information from the model pre-training; given that the model was pre-trained on the 300 million images in the JFT (Hinton et al., 2015) dataset, this data likely contains many examples and patterns that resemble the corrupted ImageNet-C data splits. Thus, having this enormous amount of pre-existing information is likely to perform better than using the relatively small amount of shifted data at prediction time. We believe that exploring the relationship between pre-training and model calibration is an exciting area of future work.

F.3. ImageNet-v2 & ImageNet-A Results

While the ImageNet-C benchmark is a popular and challenging dataset, it does not encompass all types of shift typically encountered by a machine learning model in practice. To expand the varieties of covariate shift we evaluate on, we also predict on the ImageNet-v2 test set (Recht et al., 2019) and ImageNet-A (Hendrycks et al., 2019). ImageNet-v2 is a newly curated test dataset drawn from the same test distribution as ImageNet, and we use the Matched Frequency subet of Imagenet-v2, where the images are sampled to match the same class frequency distributions as the original ImageNet validation dataset. Imagenet-A is a dataset of natural images that have been adversarially curated to minimize classifier accuracy when trained on ImageNet. In Table S3, we see that while prediction-time BN performs worse on accuracy, it improves calibration as measured by ECE. This accuracy decrease should be expected, because the training EMA statistics used by train BN still accurately represent the activation statistics for this type of change in $p()$. However, despite using less accurate normalizing statistics, prediction-time BN still performs competitively. In Table S4 prediction-time BN actually outperforms train BN on Imagenet-A, perhaps because the training statistics EMA is not representative of the adversarially constructed test set.

	Vanilla (Train/Pred BN)	Ensemble (Train/Pred BN)	Temp Scaling (Train/Pred BN)
Accuracy	**62.02%** / 58.32%	**65.50%** / 62.08 %	**62.02%** / 58.32%
Brier Score	**0.000 514 /** 0.000 547	**0.000 464 /** 0.000 502	**0.000 513 /** 0.000 546
ECE	0.085 / **0.065**	**0.020** / 0.026	0.080 / **0.060**

Table S3. Resnet-50 results on ImageNet-v2 for train and prediction-time BN. We see similar behavior of prediction-time BN as on the in-distribution test split of ImageNet, where using the prediction-time statistics instead of the training EMA actually degrades accuracy and Brier Score by a small amount, but at the same time improves ECE. See Figure S14 for comparisons to the in-distribution test split of ImageNet.

	Train BN	Prediction BN
Accuracy	1.80%	**1.87%**
Brier Score	0.001 252	**0.001 148**
ECE	0.4070	**0.2895**

Table S4. DenseNet-121 results on the natural adversarial ImageNet-A dataset for train and prediction-time BN. We see that prediction-time BN outperforms train BN, which could potentially be due to the training EMA statistics not being representative of the adversarial nature of ImageNet-A. We do not use Resnet-50 for these results, as ImageNet-A is adversarially curated to obtain 0% accuracy with a Resnet-50 architecture. (Hendrycks et al., 2019)

G. Additional Figures

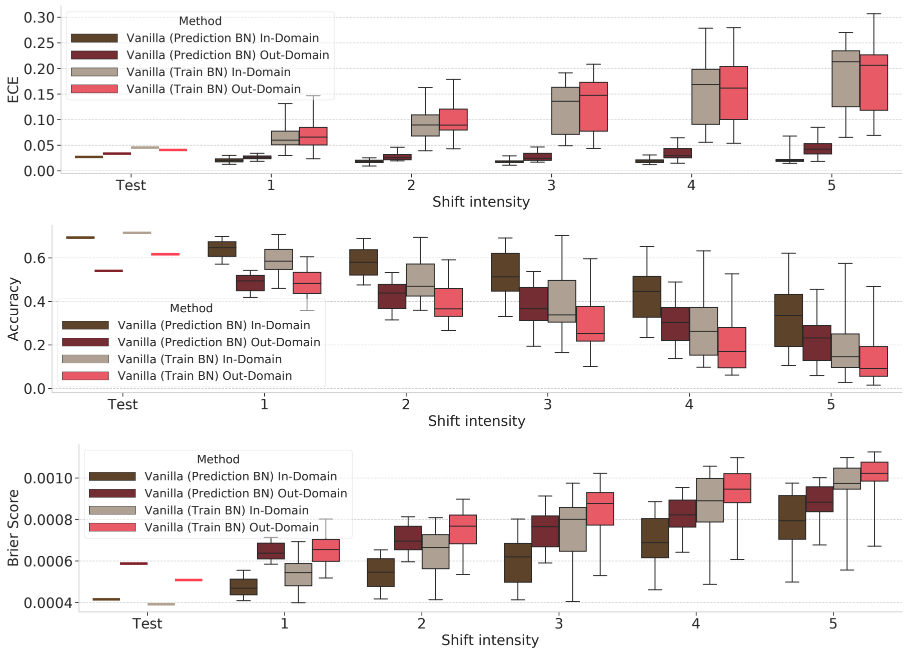

Figure S7. ImageNet-Red ResNet-50 model performance and calibration across increasing levels of dataset shift. Here we train on images where the green and blue color channels have the highest average pixel values, and consider those that have red as the dominant channel to be out-domain.

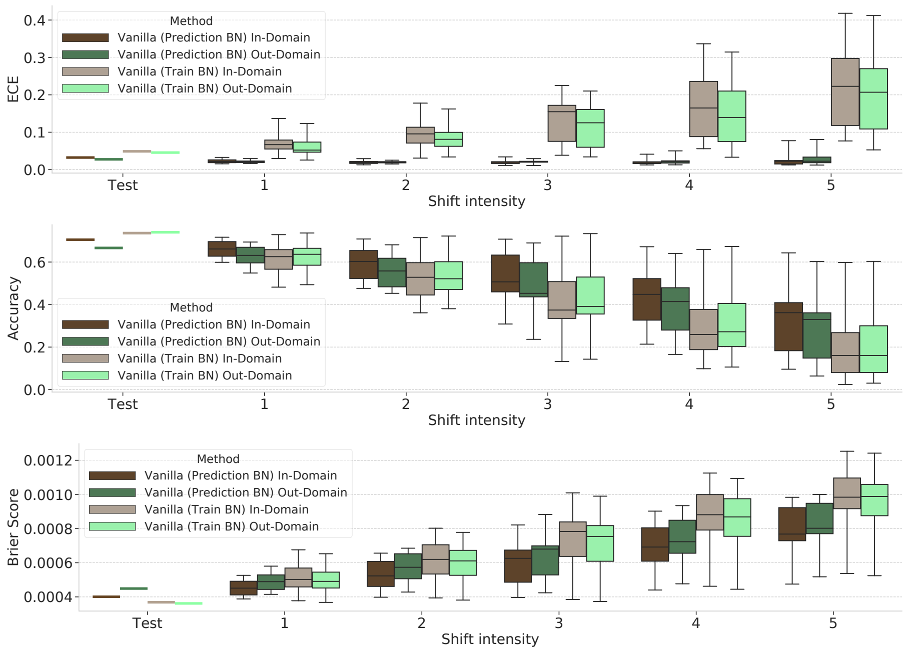

Figure S8. ImageNet-Green ResNet-50 model performance and calibration across increasing levels of dataset shift. Here we train on images where the red and blue color channels have the highest average pixel values, and consider those that have green as the dominant channel to be out-domain.

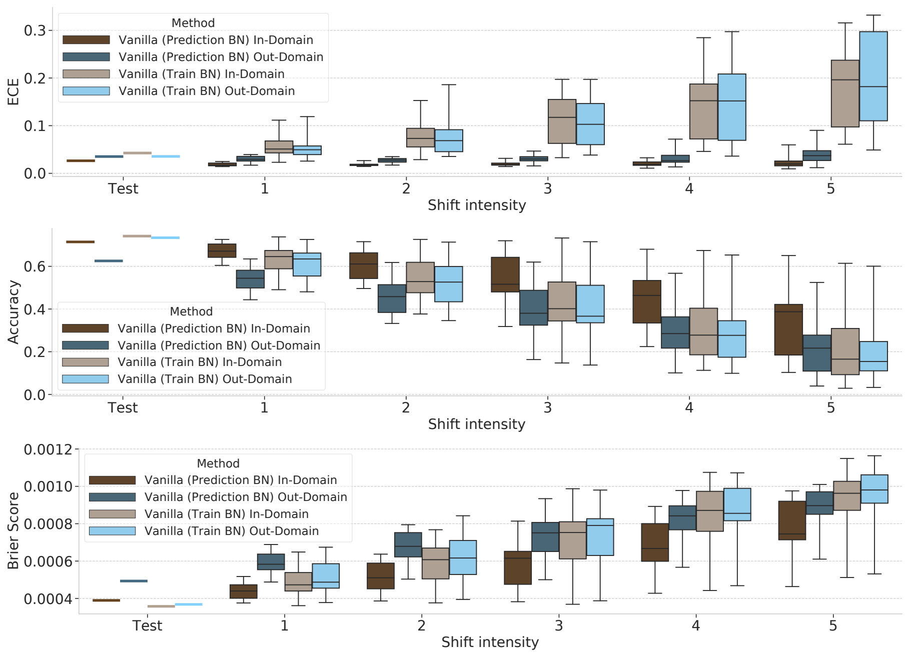

Figure S9. ImageNet-Blue ResNet-50 model performance and calibration across increasing levels of dataset shift. Here we train on images where the red and green color channels have the highest average pixel values, and consider those that have blue as the dominant channel to be out-domain.

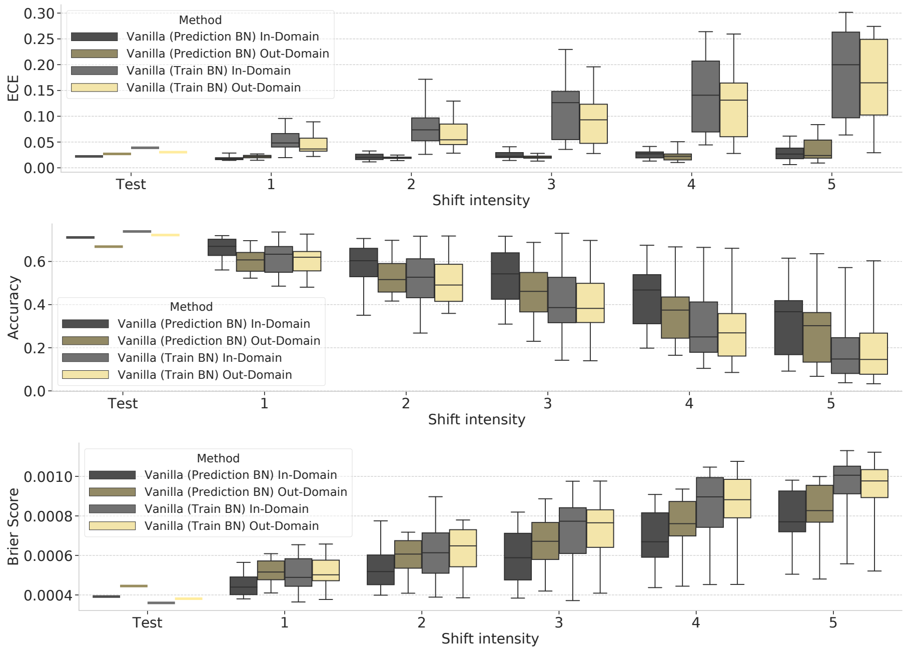

Figure S10. ImageNet-Dark ResNet-50 model performance and calibration across increasing levels of dataset shift. Here we train on images where the average pixel value was greater than 0.5, and consider those whose average pixel value is less than 0.5 to be out-domain.

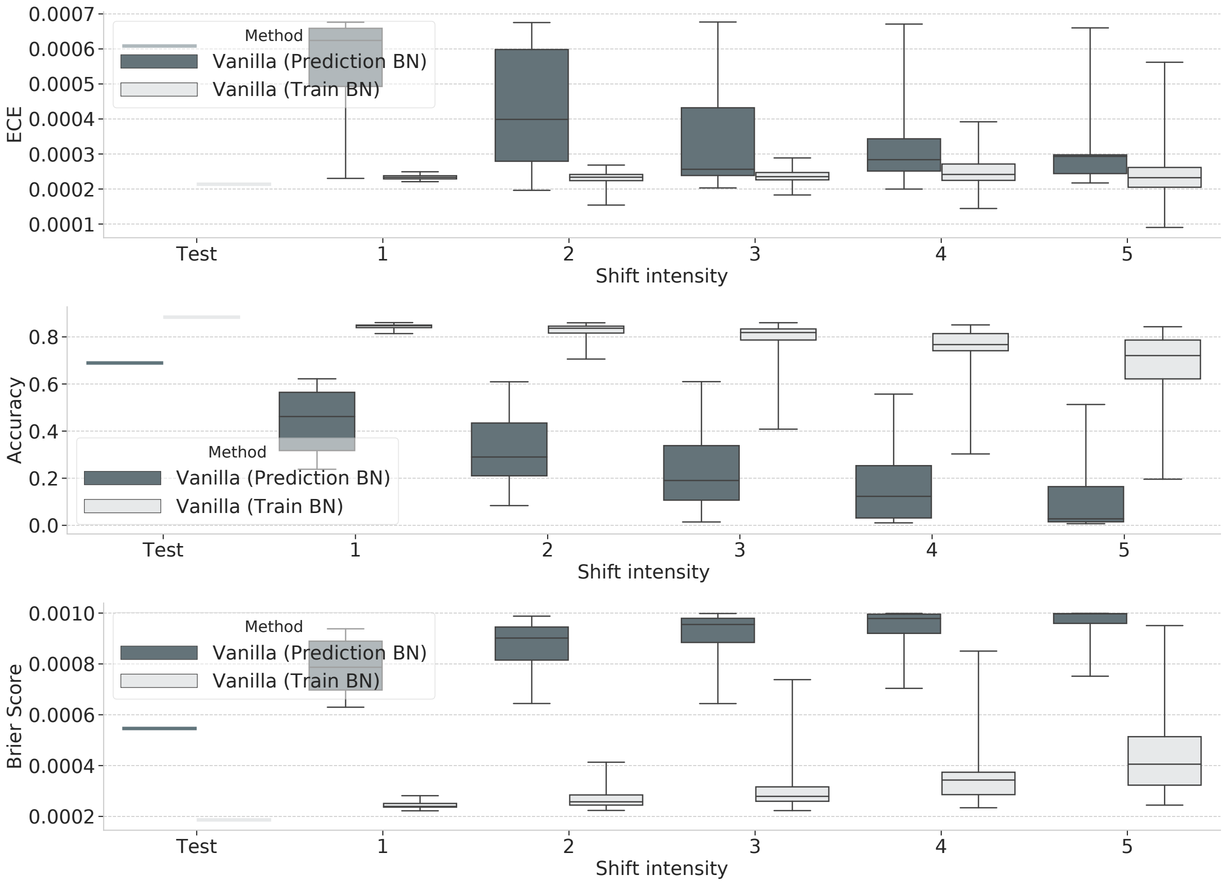

Figure S11. Calibration and accuracy under covariate shift with the ImageNet-C EfficientNet model trained with the Noisy Student technique.

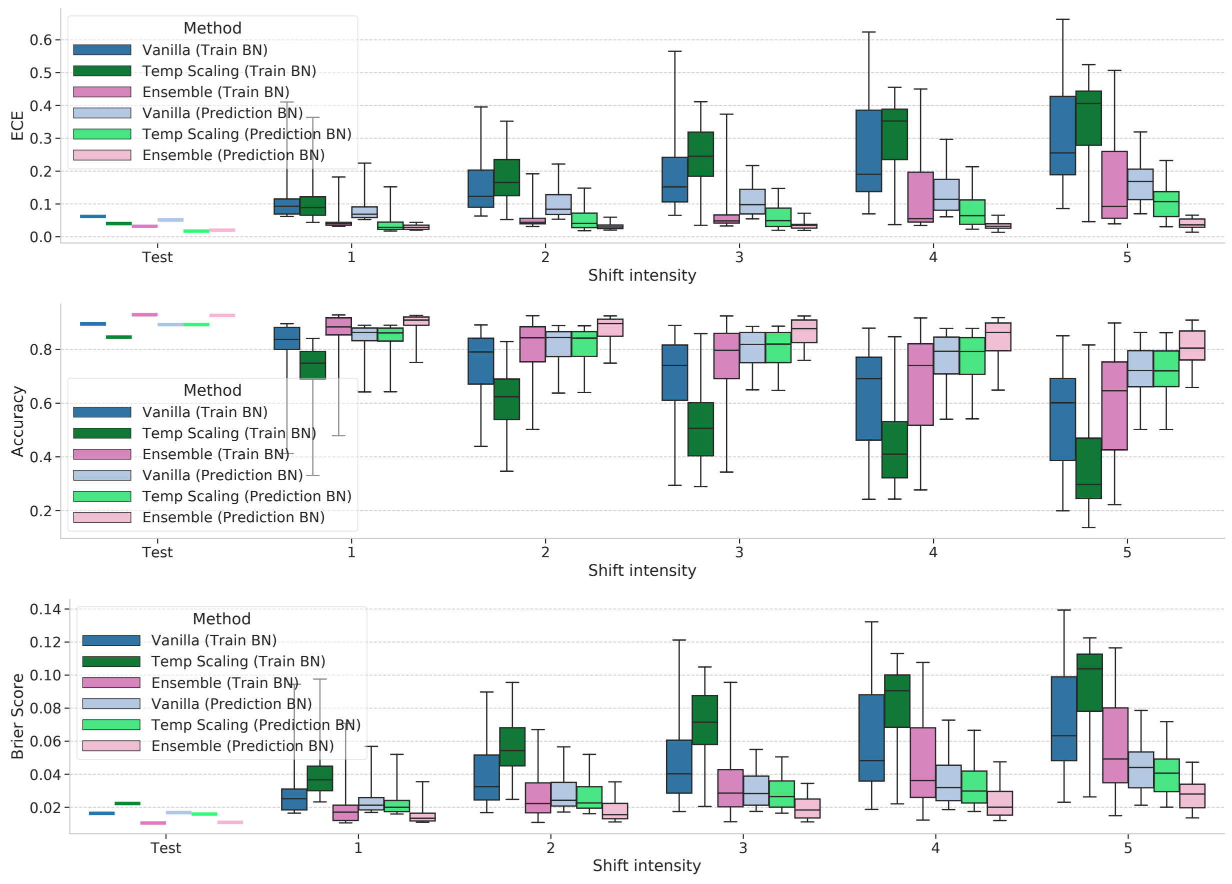

Figure S12. Calibration and accuracy under covariate shift on CIFAR-10-C for vanilla, ensemble, and temperature scaling methods, each with a test batch size of 500 and $\epsilon = 10^{-3}$.

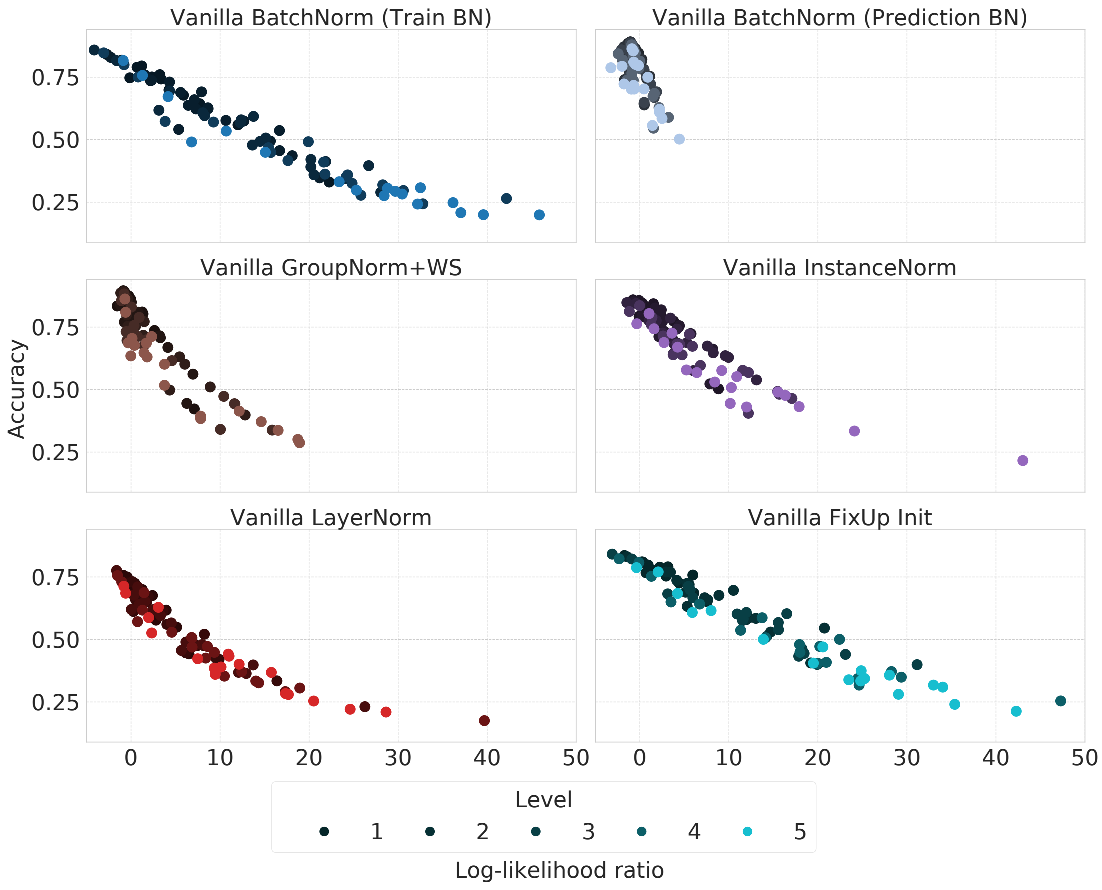

Figure S13. As first described in Figure 2, we also see a linear trend of degrading performance as the train and test activation distributions become further apart. Once again, higher shift intensities are plotted with higher color saturation, illustrating a clear relationship between shift intensity and distance between activation supports.

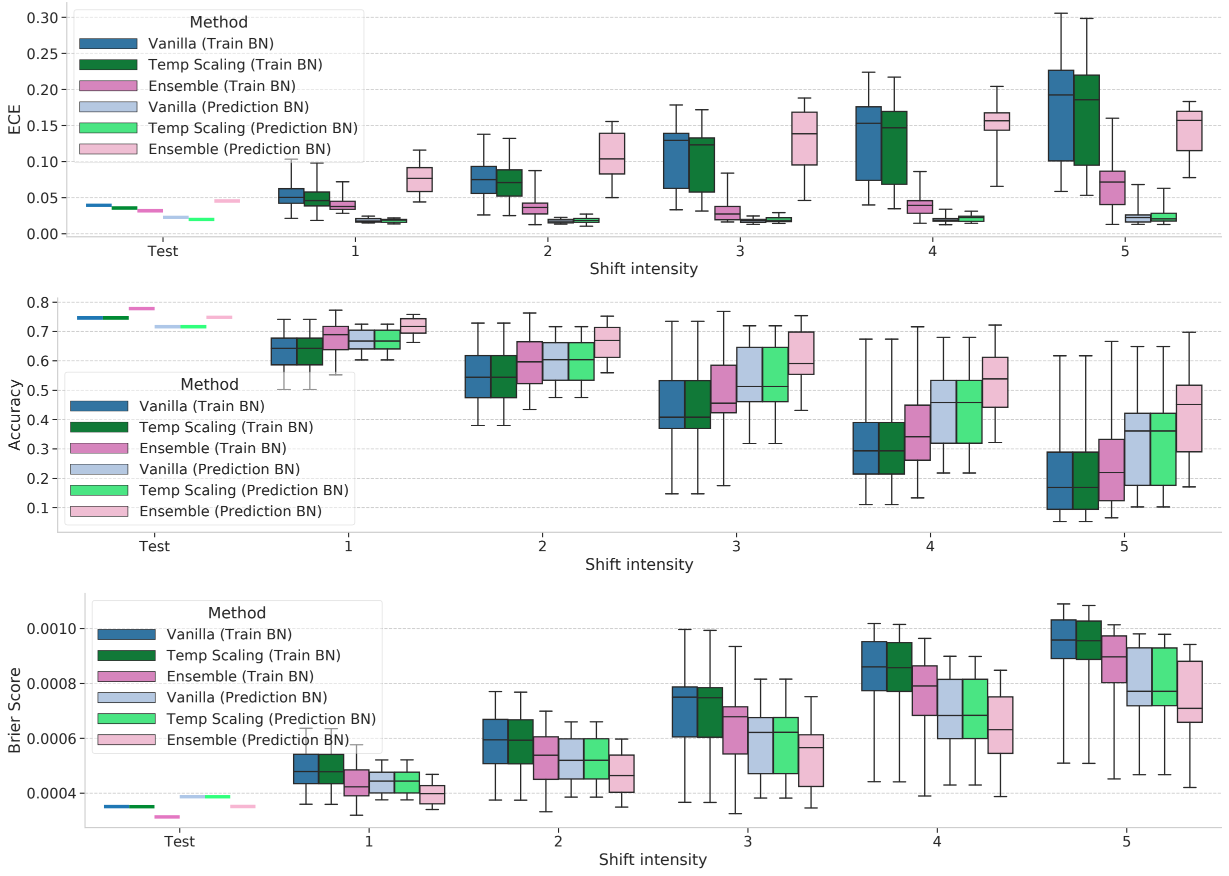

Figure S14. Calibration and accuracy under covariate shift on ImageNet-C for vanilla, ensemble, and temperature scaling methods, each with a test batch size of 100 and $\epsilon = 10^{-5}$. Note that for ensembles using prediction-time BN actually degrades performance on ECE, but achieves the best accuracy and Brier Score. This is likely due to an issue with the number of bins used to compute ECE, which was 30 for all ImageNet experiments, because we still see improvements on Brier Score which, unlike ECE, is a proper scoring rule.

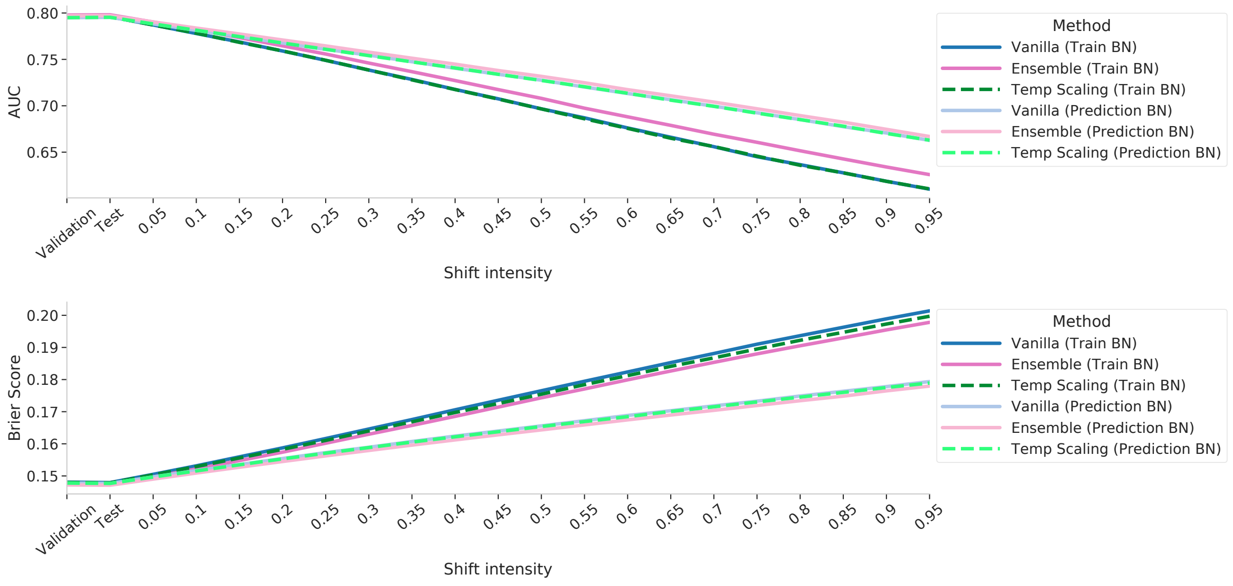

Figure S15. Calibration and AUC under covariate shift on the Criteo dataset.

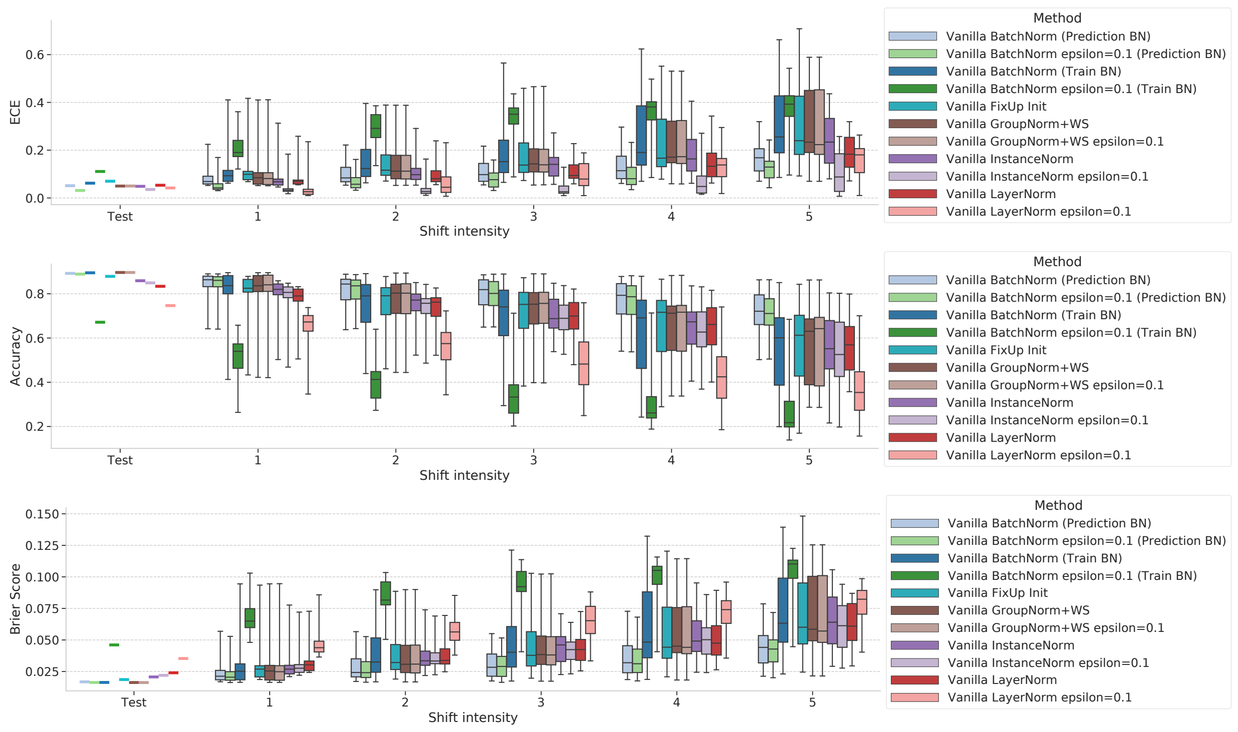

Figure S16. Calibration and accuracy under covariate shift on CIFAR-10-C for the vanilla model using various normalization techniques, each with a test batch size of 500. In addition to $\epsilon \in \{10^{-3}, 10^{-1}\}$ varieties of each normalization method, we also include FixUp initialization (Zhang et al., 2019) as a no normalization baseline. FixUp performs well on the in-distribution set but quickly degrades in a similar trend as the train BN method.

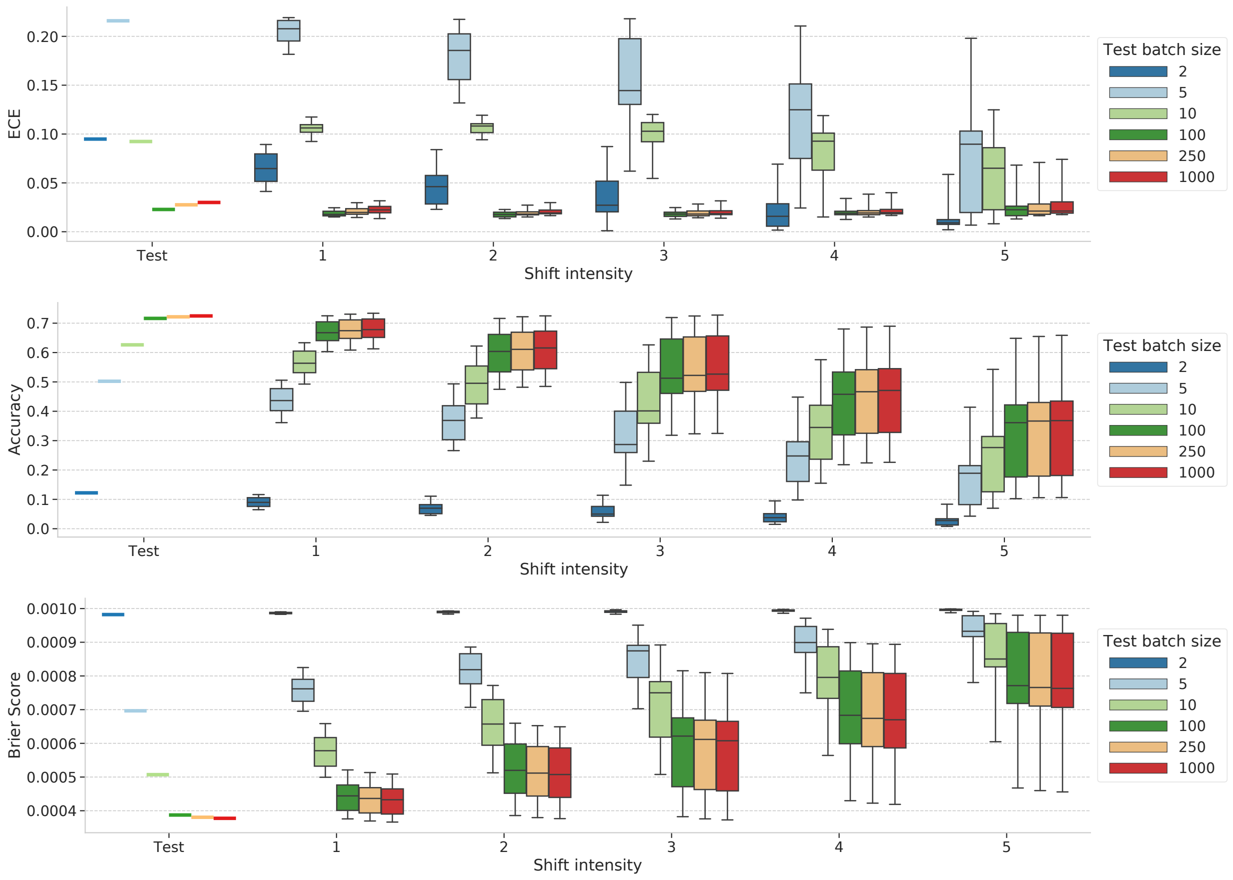

Figure S17. ImageNet-C calibration and accuracy for the vanilla model for different test batch sizes. We see an almost identical trend as in CIFAR-10-C, with performance plateauing after batch size 100 or 250. We do see a small increase in ECE after batch size 100, but accuracy and Brier Score continue to slightly improve, so this could be an artifact of how ECE bins confidences.

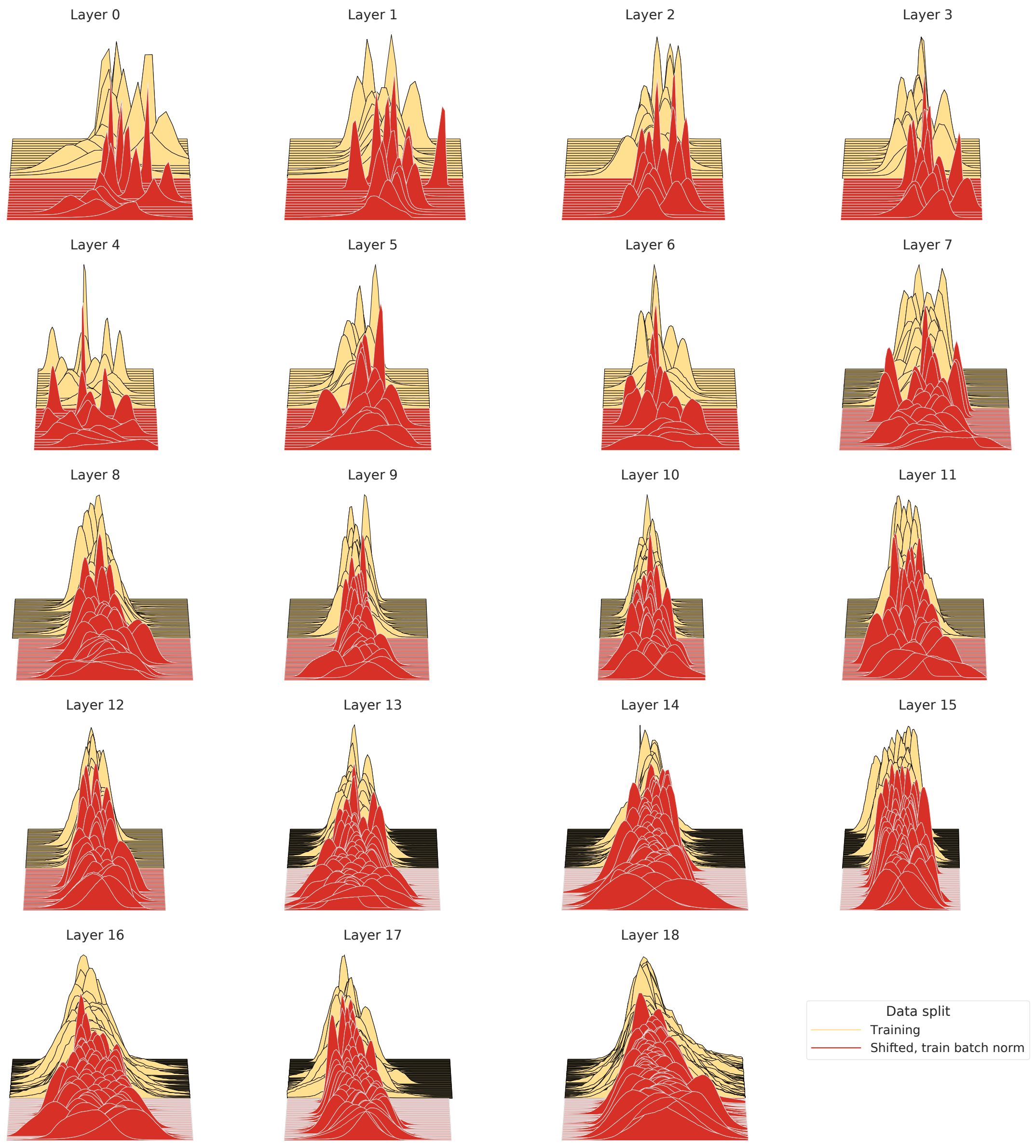

Figure S18. Empirical distributions of the outputs of each normalization layer in Resnet-20, for train BN.

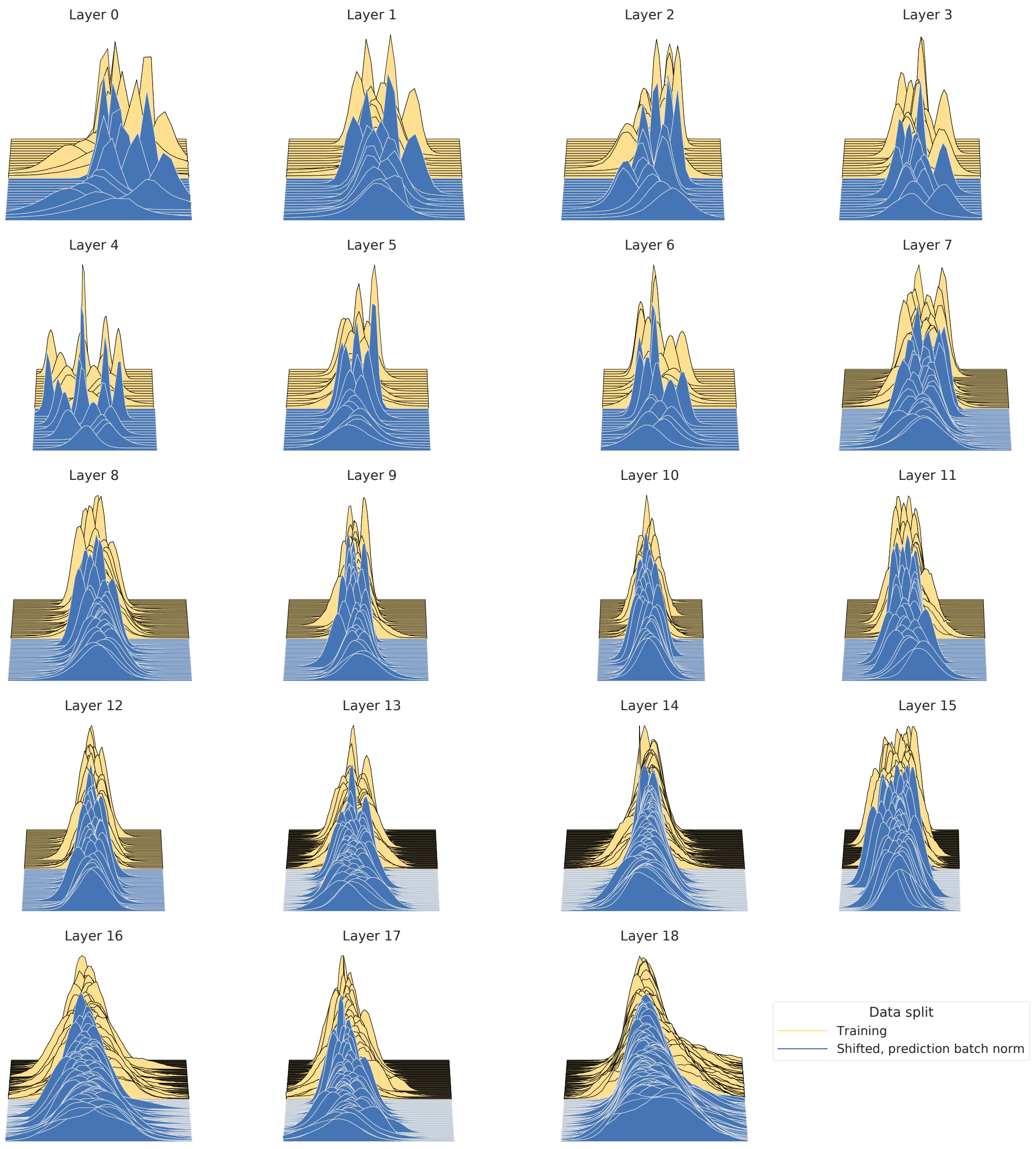

Figure S19. Empirical distributions of the outputs of each normalization layer in Resnet-20, for prediction-time BN.