TRADI: Tracking deep neural network weight distributions

Gianni Franchi[1,2], Andrei Bursuc[3], Emanuel Aldea[2], Séverine Dubuisson[4], and Isabelle Bloch[5]

[1] ENSTA Paris, Institut polytechnique de Paris
[2] SATIE, Université Paris-Sud, Université Paris-Saclay
[3] valeo.ai
[4] CNRS, LIS, Aix Marseille University
[5] LTCI, Télécom Paris, Institut polytechnique de Paris ⋆

Abstract. During training, the weights of a Deep Neural Network (DNN) are optimized from a random initialization towards a nearly optimum value minimizing a loss function. Only this final state of the weights is typically kept for testing, while the wealth of information on the geometry of the weight space, accumulated over the descent towards the minimum is discarded. In this work we propose to make use of this knowledge and leverage it for computing the distributions of the weights of the DNN. This can be further used for estimating the epistemic uncertainty of the DNN by aggregating predictions from an ensemble of networks sampled from these distributions. To this end we introduce a method for tracking the trajectory of the weights during optimization, that does neither require any change in the architecture, nor in the training procedure. We evaluate our method, TRADI, on standard classification and regression benchmarks, and on out-of-distribution detection for classification and semantic segmentation. We achieve competitive results, while preserving computational efficiency in comparison to ensemble approaches.

Keywords: Deep neural networks, weight distribution, uncertainty, ensembles, out-of-distribution detection

1 Introduction

In recent years, Deep Neural Networks (DNNs) have gained prominence in various computer vision tasks and practical applications. This progress has been in part accelerated by multiple innovations in key parts of DNN pipelines, *e.g.*, architecture design [18, 30, 47, 49], optimization [27], initialization [12, 17], regularization [22, 48], *etc.*, along with a pool of effective heuristics identified by practitioners. Modern DNNs achieve now strong accuracy across tasks and domains, leading to their potential utilization as key blocks in real-world applications.

⋆ This work was supported by ANR Project MOHICANS (ANR-15-CE39-0005). We would like to thank Saclay-IA cluster and CNRS Jean-Zay supercomputer.

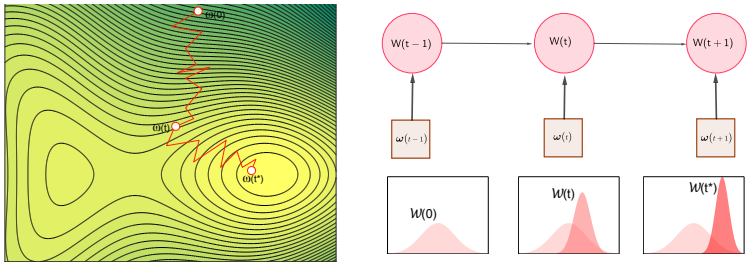

Fig. 1: Our algorithm uses Kalman filtering for tracking the distribution $\mathcal{W}$ of all DNN weights across training steps from a generic prior $\mathcal{W}(0)$ to the final estimate $\mathcal{W}(t^*)$. We also estimate the covariance matrix of all the trainable network parameters. Popular alternative approaches rely typically either on ensembles of models trained independently [31] with a significant computational cost, approximate ensembles [10] or on averaging weights collected from different local minima [23].

However, DNNs have also been shown to be making mostly over-confident predictions [15], a side-effect of the heuristics used in modern DNNs. This means that for ambiguous instances bordering two classes (*e.g.*, human wearing a cat costume), or on unrelated instances (*e.g.*, plastic bag not "seen" during training and classified with high probability as rock), DNNs are likely to fail silently, which is a critical drawback for decision making systems. This has motivated several works to address the predictive uncertainty of DNNs [6, 10, 31], usually taking inspiration from Bayesian approaches. Knowledge about the distribution of the network weights during training opens the way for studying the evolution of the underlying covariance matrix, and the uncertainty of the model parameters, referred to as the epistemic uncertainty [26]. In this work we propose a method for estimating the distribution of the weights by tracking their trajectory during training. This enables us to sample an ensemble of networks and estimate more reliably the epistemic uncertainty and detect out-of-distribution samples.

The common practice in training DNNs is to first initialize its weights using an appropriate random initialization strategy and then slowly adjust the weights through optimization according to the correctness of the network predictions on many mini-batches of training data. Once the stopping criterion is met, the final state of the weights is kept for evaluation. We argue that the trajectory of weights towards the (local) optimum reveals abundant information about the structure of the weight space that we could exploit, instead of discarding it and looking only at the final point values of the weights. Popular DNN weight initialization techniques [12, 17] consist of an effective layer-wise scaling of random weight values sampled from a Normal distribution. Assuming that weights follow a Gaussian distribution at time $t = 0$, owing to the central limit theorem weights will also converge towards a Gaussian distribution. The final state is reached here through a noisy process, where the stochasticity is induced by the weight initialization, the order and configuration of the mini-batches, *etc.* We find it thus reasonable to see optimization as a random walk leading to a (local) minimum, in

which case "tracking" the distribution makes sense (Fig. 1). To this end, Kalman filtering (KF) [14] is an appropriate strategy for tractability reasons, as well as for the guaranteed optimality as long as the underlying assumptions are valid (linear dynamic system with Gaussian assumption in the predict and update steps).[6] To the best of our knowledge, our work is the first attempt to use such a technique to track the DNN weight distributions, and subsequently to estimate its epistemic uncertainty.

Contributions. The keypoints of our contribution are: **(a)** this is the first work which filters in a tractable manner the trajectory of the entire set of trainable parameters of a DNN during the training process; **(b)** we propose a tractable approximation for estimating the covariance matrix of the network parameters; **(c)** we achieve competitive or state of the art results on most regression datasets, and on out-of-distribution experiments our method is better calibrated on three segmentation datasets (CamVid [7] , StreetHazards [20], and BDD Anomaly [20]); **(d)** our approach strikes an appealing trade-off in terms of performance and computational time (training + prediction).

2 TRAcking of the weight DIstribution (TRADI)

In this section, we detail our approach to first estimate the distribution of the weights of a DNN at each training step, and then generate an ensemble of networks by sampling from the computed distributions at training conclusion.

2.1 Notations and hypotheses

- X and Y are two random variables, with $X \sim \mathcal{P}_X$ and $Y \sim \mathcal{P}_Y$. Without loss of generality we consider the observed samples $\{\mathbf{x}_i\}_{i=1}^n$ as vectors and the corresponding labels $\{y_i\}_{i=1}^n$ as scalars (class index for classification, real value for regression). From this set of observations, we derive a training set of n_l elements and a testing set of n_τ elements: $n = n_l + n_\tau$.
- Training/Testing sets are denoted respectively by $\mathcal{D}_l = (\mathbf{x}_i, y_i)_{i=1}^{n_l}$, $\mathcal{D}_\tau = (\mathbf{x}_i, y_i)_{i=1}^{n_\tau}$. Data in $\mathcal{D}_l$ and $\mathcal{D}_\tau$ are assumed to be i.i.d. distributed according to their respective unknown joint distribution $\mathcal{P}_l$ and $\mathcal{P}_\tau$.
- The DNN is defined by a vector containing the K trainable weights $\boldsymbol{\omega} = \{\omega_k\}_{k=1}^K$. During training, $\boldsymbol{\omega}$ is iteratively updated for each mini-batch and we denote by $\boldsymbol{\omega}(t)$ the state of the DNN at iteration t of the optimization algorithm, realization of the random variable $W(t)$. Let g denote the architecture of the DNN associated with these weights and $g_{\boldsymbol{\omega}(t)}(x_i)$ its output at t. The initial set of weights $\boldsymbol{\omega}(0) = \{\omega_k(0)\}_{k=1}^K$ follows $\mathcal{N}(0, \sigma_k^2)$, where the values σ_k^2 are fixed as in [17].
- $\mathcal{L}(\boldsymbol{\omega}(t), y_i)$ is the loss function used to measure the dissimilarity between the output $g_{\boldsymbol{\omega}(t)}(\mathbf{x}_i)$ of the DNN and the expected output y_i. Different loss functions can be considered depending on the type of task.

[6] Recent theoretical works [40] show connections between optimization and KF, enforcing the validity of our approach.

– Weights on different layers are assumed to be independent of each another at all times. This assumption is not necessary from a theoretical point of view, yet we need it to limit the complexity of the computation. Many works in the related literature rely on such assumptions [13], and some take the assumptions even further, *e.g.* [5], one of the most popular modern BNNs, supposes that all weights are independent (even from the same layer). Each weight $\omega_k(t)$, $k = 1, \ldots, K$, follows a non-stationary Normal distribution (i.e. $W_k(t) \sim \mathcal{N}(\mu_k(t), \sigma_k^2(t))$) whose two parameters are tracked.

2.2 TRAcking of the DIstribution (TRADI) of weights of a DNN

Tracking the mean and variance of the weights DNN optimization typically starts from a set of randomly initialized weights $\boldsymbol{\omega}(0)$. Then, at each training step t, several SGD updates are performed from randomly chosen mini-batches towards minimizing the loss. This makes the trajectory of the weights vary or oscillate, but not necessarily in the good direction each time [33]. Since gradients are averaged over mini-batches, we can consider that weight trajectories are averaged over each mini-batch. After a certain number of epochs, the DNN converges, *i.e.* it reaches a local optimum with a specific configuration of weights that will then be used for testing. However, this general approach for training does not consider the evolution of the distribution of the weights, which may be estimated from the training trajectory and from the dynamics of the weights over time. In our work, we argue that the history of the weight evolution up to their final state is an effective tool for estimating the epistemic uncertainty.

More specifically, our goal is to estimate, for all weights $\omega_k(t)$ of the DNN and at each training step t, $\mu_k(t)$ and $\sigma_k^2(t)$, the parameters of their normal distribution. Furthermore, for small networks we can also estimate the covariance $\text{cov}(W_k(t), W_{k'}(t))$ for any pair of weights $(\omega_k(t), \omega'_k(t))$ at t in the DNN (see supplementary material for details). To this end, we leverage mini-batch SGD in order to optimize the loss between two weight realizations. The loss derivative with respect to a given weight $\omega_k(t)$ over a mini-batch $B(t)$ is given by:

$$\nabla \mathcal{L}_{\omega_k(t)} = \frac{1}{|B(t)|} \sum_{(\mathbf{x}_i, y_i) \in B(t)} \frac{\partial \mathcal{L}(\boldsymbol{\omega}(t-1), y_i)}{\partial \omega_k(t-1)} \tag{1}$$

Weights $\omega_k(t)$ are then updated as follows:

$$\omega_k(t) = \omega_k(t-1) - \eta \nabla \mathcal{L}_{\omega_k(t)} \tag{2}$$

with η the learning rate.

The weights of DNNs are randomly initialized at $t = 0$ by sampling $W_k(0) \sim \mathcal{N}(\mu_k(0), \sigma_k^2(0))$, where the parameters of the distribution are set empirically on a per-layer basis [17]. By computing the expectation of $\omega_k(t)$ in Eq. (2), and using its linearity property, we get:

$$\mu_k(t) = \mu_k(t-1) - \mathbb{E}\left[\eta \nabla \mathcal{L}_{\omega_k(t)}\right] \tag{3}$$

We can see that $\mu_k(t)$ depends on $\mu_k(t-1)$ and on another function at time $(t-1)$: this shows that the means of the weights follow a Markov process.

As in [2, 53] we assume that during back-propagation and forward pass weights to be independent. We then get:

$$\sigma_k^2(t) = \sigma_k^2(t-1) + \eta^2 \mathbb{E}\left[(\nabla \mathcal{L}_{\omega_k(t)})^2\right] - \eta^2 \mathbb{E}^2\left[\nabla \mathcal{L}_{\omega_k(t)}\right] \tag{4}$$

This leads to the following state and measurement equations for $\mu_k(t)$:

$$\begin{cases} \mu_k(t) = \mu_k(t-1) - \eta \nabla \mathcal{L}_{\omega_k(t)} + \varepsilon_\mu \\ \omega_k(t) = \mu_k(t) + \tilde{\varepsilon}_\mu \end{cases} \tag{5}$$

with ε_μ being the state noise, and $\tilde{\varepsilon}_\mu$ being the observation noise, as realizations of $\mathcal{N}(0, \sigma_\mu^2)$ and $\mathcal{N}(0, \tilde{\sigma}_\mu^2)$ respectively. The state and measurement equations for the variance σ_k are given by:

$$\begin{cases} \sigma_k^2(t) = \sigma_k^2(t-1) + \left(\eta \nabla \mathcal{L}_{\omega_k(t)}\right)^2 + \varepsilon_\sigma \\ z_k(t) = \sigma_k^2(t) - \mu_k(t)^2 + \tilde{\varepsilon}_\sigma \\ \text{with } z_k(t) = \omega_k(t)^2 \end{cases} \tag{6}$$

with ε_σ being the state noise, and $\tilde{\varepsilon}_\sigma$ being the observation noise, as realizations of $\mathcal{N}(0, \sigma_\sigma^2)$ and $\mathcal{N}(0, \tilde{\sigma}_\sigma^2)$, respectively. We ignore the square empirical mean of the gradient on the equation as in practice its value is below the state noise.

Approximating the covariance Using the measurement and state transition in Eq. (5-6), we can apply a Kalman filter to track the state of each trainable parameter. As the computational cost for tracking the covariance matrix is significant, we propose to track instead only the variance of the distribution. For that, we approximate the covariance by employing a model inspired from Gaussian Processes [52]. We consider the Gaussian model due to its simplicity and good results. Let $\mathbf{\Sigma}(t)$ denote the covariance of $W(t)$, and let $\mathbf{v}(t) = \big(\sigma_0(t), \sigma_1(t), \sigma_2(t), \ldots, \sigma_K(t)\big)$ be a vector of size K composed of the standard deviations of all weights at time t. The covariance matrix is approximated by $\hat{\mathbf{\Sigma}}(t) = (\mathbf{v}(t)\mathbf{v}(t)^T) \odot \boldsymbol{\mathcal{K}}(t)$, where $\odot$ is the Hadamard product, and $\boldsymbol{\mathcal{K}}(t)$ is the kernel corresponding to the $K \times K$ Gram matrix of the weights of the DNN, with the coefficient (k, k') given by $\boldsymbol{\mathcal{K}}(\omega_k(t), \omega_{k'}(t)) = \exp\left(-\frac{\|\omega_k(t) - \omega_{k'}(t)\|^2}{2\sigma_{\text{rbf}}^2}\right)$. The computational cost for storing and processing the kernel $\boldsymbol{\mathcal{K}}(t)$ is however prohibitive in practice as its complexity is quadratic in terms of the number of weights (*e.g.*, $K \approx 10^9$ in recent DNNs).

Rahimi and Recht [45] alleviate this problem by approximating non-linear kernels, *e.g.* Gaussian RBF, in an unbiased way using random feature representations. Then, for any translation-invariant positive definite kernel $\boldsymbol{\mathcal{K}(t)}$, for all $(\omega_k(t), \omega_{k'}(t))$, $\boldsymbol{\mathcal{K}}(\omega_k(t), \omega_{k'}(t))$ depends only on $\omega_k(t) - \omega_{k'}(t)$. We can then approximate the matrix by:

$$\boldsymbol{\mathcal{K}}(\omega_k(t), \omega_{k'}(t)) \equiv \mathbb{E}\left[\cos(\Theta \omega_k(t) + \Phi) \cos(\Theta \omega_{k'}(t) + \Phi)\right]$$

where $\Theta \sim \mathcal{N}(0, \sigma_{\text{rbf}}^2)$ (this distribution is the Fourier transform of the kernel distribution) and $\Phi \sim \mathcal{U}_{[0,2\pi]}$. In detail, we approximate the high-dimensional feature space by projecting over the following N-dimensional feature vector:

$$\mathbf{z}(\omega_k(t)) \equiv \sqrt{\frac{2}{N}} \left[\cos(\theta_1 \omega_k(t) + \phi_1), \ldots, \cos(\theta_N \omega_k(t) + \phi_N))\right]^\top \tag{7}$$

where the $\theta_1, \ldots, \theta_N$ are i.i.d. from $\mathcal{N}(0, \sigma_{\text{rbf}}^2)$ and $\phi_1, \ldots, \phi_N$ are i.i.d. from $\mathcal{U}_{[0,2\pi]}$. In this new feature space we can approximate kernel $\boldsymbol{\mathcal{K}}(t)$ by $\hat{\boldsymbol{\mathcal{K}}}(t)$ defined by:

$$\hat{\boldsymbol{\mathcal{K}}}(\omega_k(t), \omega_{k'}(t)) = \mathbf{z}(\omega_k(t))^\top \mathbf{z}(\omega_{k'}(t)) \tag{8}$$

Furthermore, it was proved in [45] that the probability of having an error of approximation greater than $\epsilon \in \mathbb{R}^+$ depends on $\exp(-N\epsilon^2)/\epsilon^2$. To avoid the Hadamard product of matrices of size $K \times K$, we evaluate $\boldsymbol{r}(\omega_k(t)) = \sigma_k(t)\boldsymbol{z}(\omega_k(t))$, and the value at index (k, k') of the approximate covariance matrix $\hat{\boldsymbol{\Sigma}}(t)$ is given by:

$$\hat{\boldsymbol{\Sigma}}(t)(k, k') = \boldsymbol{r}(\omega_k(t))^\top \boldsymbol{r}(\omega_k(t)). \tag{9}$$

2.3 Training the DNNs

In our approach, for classification we use the cross-entropy loss to get the log-likelihood similarly to [31]. For regression tasks, we train over two losses sequentially and modify $g_{\boldsymbol{\omega}(t)}(\boldsymbol{x}_i)$ to have two output heads: the classical regression output $\mu_{\text{pred}}(\boldsymbol{x}_i)$ and the predicted variance of the output σ_{pred}^2. This modification is inspired by [31]. The first loss is the MSE $\mathcal{L}_1(\boldsymbol{\omega}(t), \boldsymbol{y}_i) = \|g_{\boldsymbol{\omega}(t)}(\boldsymbol{x}_i) - \boldsymbol{y}_i\|_2^2$ as used in the traditional regression tasks. The second loss is the negative log-likelihood (NLL) [31] which reads:

$$\mathcal{L}_2(\boldsymbol{\omega}(t), y_i) = \frac{1}{2\sigma_{\text{pred}}(\mathbf{x}_i)^2} \|\mu_{\text{pred}}(\mathbf{x}_i) - y_i\|^2 + \frac{1}{2} \log \sigma_{\text{pred}}(\mathbf{x}_i)^2 \tag{10}$$

We first train with loss $\mathcal{L}_1(\boldsymbol{\omega}(t), y_i)$ until reaching a satisfying $\boldsymbol{\omega}(t)$. In the second stage we add the variance prediction head and start fine-tuning from $\boldsymbol{\omega}(t)$ with loss $\mathcal{L}_2(\boldsymbol{\omega}(t), y_i)$. In our experiments we observed that this sequential training is more stable as it allows the network to first learn features for the target task and then to predict its own variance, rather than doing both in the same time (which is particularly unstable in the first steps).

2.4 TRADI training algorithm overview

We detail the TRADI steps during training in Appendix, Section 1.3. For tracking purposes we must store $\mu_k(t)$ and $\sigma_k(t)$ for all the weights of the network. Hence, the method computationally lighter than Deep Ensembles, which has a training complexity scaling with the number of networks composing the ensemble. In addition, TRADI can be applied to any DNN without any modification of the architecture, in contrast to MC Dropout that requires adding dropout layers to the underlying DNN. For clarity we define $\mathcal{L}(\boldsymbol{\omega}(t), B(t)) = \frac{1}{|B(t)|} \sum_{(x_i, y_i) \in B(t)} \mathcal{L}(\boldsymbol{\omega}(t), y_i)$.

Here $\mathbf{P}_\mu$, $\mathbf{P}_\sigma$ are the noise covariance matrices of the mean and variance respectively and $\mathbf{Q}_\mu$, $\mathbf{Q}_\sigma$ are the optimal gain matrices of the mean and variance respectively. These matrices are used during Kalman filtering [24].

2.5 TRADI uncertainty during testing

After having trained a DNN, we can evaluate its uncertainty by sampling new realizations of the weights from to the tracked distribution. We call $\tilde{\boldsymbol{\omega}}(t) = \{\tilde{\omega}_k(t)\}_{k=1}^K$ the vector of size K containing these realizations. Note that this vector is different from $\boldsymbol{\omega}(t)$ since it is sampled from the distribution computed with TRADI, that does not correspond exactly to the DNN weight distribution. In addition, we note $\boldsymbol{\mu}(t)$ the vector of size K containing the mean of all weights at time t.

Then, two cases can occur. In the first case, we have access to the covariance matrix of the weights (by tracking or by an alternative approach) that we denote $\boldsymbol{\Sigma}(t)$, and we simply sample new realizations of $W(t)$ using the following formula:

$$\tilde{\boldsymbol{\omega}}(t) = \boldsymbol{\mu}(t) + \boldsymbol{\Sigma}^{1/2}(t) \times \boldsymbol{m}_1 \tag{11}$$

in which $\boldsymbol{m}_1$ is drawn from the multivariate Gaussian $\mathcal{N}(\mathbf{0}_K, \boldsymbol{I}_K)$, where $\mathbf{0}_K, \boldsymbol{I}_K$ are respectively the K-size zero vector and the $K \times K$ size identity matrix.

When we deal with a DNN (the considered case in this paper), we are constrained for tractability reasons to approximate the covariance matrix following the random projection trick proposed in the previous section, and we generate new realizations of $W(t)$ as follows:

$$\tilde{\boldsymbol{\omega}}(t) = \boldsymbol{\mu}(t) + \boldsymbol{R}(\boldsymbol{\omega}(t)) \times \boldsymbol{m}_2 \tag{12}$$

where $\boldsymbol{R}(\boldsymbol{\omega}(t))$ is a matrix of size $K \times N$ whose rows $k \in [1, K]$ contain the $\boldsymbol{r}(\omega_k(t))^\top$ defined in Section 2.2. $\boldsymbol{R}(\boldsymbol{\omega}(t))$ depends on $(\theta_1, \ldots, \theta_N)$ and on $(\phi_1, \ldots, \phi_N)$ defined in Eq.(7). $\boldsymbol{m}_2$ is drawn from the multivariate Gaussian $\mathcal{N}(\mathbf{0}_N, \boldsymbol{I}_N)$, where $\mathbf{0}_N, \boldsymbol{I}_N$ are respectively the zero vector of size N and the identity matrix of size $N \times N$. Note that since $N \ll K$, computations are significantly accelerated.

Then similarly to works in [26, 37], given input data $(\mathbf{x}^*, y^*) \in \mathcal{D}_\tau$ from the testing set, we estimate the marginal likelihood as Monte Carlo integration. First, a sequence $\{\tilde{\boldsymbol{\omega}}^j(t)\}_{j=1}^{N_{\text{model}}}$ of N_{model} realizations of $W(t)$ is drawn (typically, $N_{\text{model}} = 20$). Then, the marginal likelihood of y^* over $W(t)$ is approximated by:

$$\mathcal{P}(y^*|x^*) = \frac{1}{N_{\text{model}}} \sum_{j=1}^{N_{\text{model}}} \mathcal{P}(y^*|\tilde{\boldsymbol{\omega}}^j(t), \mathbf{x}^*) \tag{13}$$

For regression, we use the strategy from [31] to compute the log-likelihood of the regression and consider that the outputs of the DNN applied on $\mathbf{x}^*$ are the parameters $\{\mu_{\text{pred}}^j(\mathbf{x}^*), (\sigma_{\text{pred}}^j(\mathbf{x}^*))^2\}_{j=1}^{N_{\text{model}}}$ of a Gaussian distribution

(see Section 2.3). Hence, the final output is the result of a mixture of N_{model} Gaussian distributions $\mathcal{N}(\mu^j_{\text{pred}}(\mathbf{x}^*), (\sigma^j_{\text{pred}}(\mathbf{x}^*))^2)$. During testing, if the DNN has BatchNorm layers, we first update BatchNorm statistics of each of the sampled $\tilde{\boldsymbol{\omega}}^j(t)$ models, where $j \in [1, N_{\text{model}}]$ [23].

3 Related work

Uncertainty estimation is an important aspect for any machine learning model and it has been thoroughly studied across years in statistical learning areas. In the context of DNNs a renewed interest has surged in dealing with uncertainty, In the following we briefly review methods related to our approach.

Bayesian methods. Bayesian approaches deal with uncertainty by identifying a distribution of the parameters of the model. The posterior distribution is computed from a prior distribution assumed over the parameters and the likelihood of the model for the current data. The posterior distribution is iteratively updated across training samples. The predictive distribution is then computed through Bayesian model averaging by sampling models from the posterior distribution. This simple formalism is at the core of many machine learning models, including neural networks. Early approaches from Neal [39] leveraged Markov chain Monte Carlo variants for inference on Bayesian Neural Networks. However for modern DNNs with millions of parameters, such methods are intractable for computing the posterior distribution, leaving the lead to gradient based methods.

Modern Bayesian Neural Networks (BNNs). Progress in variational inference [28] has enabled a recent revival of BNNs. Blundell *et al.* [6] learn distributions over neurons via a Gaussian mixture prior. While such models are easy to reason along, they are limited to rather medium-sized networks. Gal and Ghahramani [10] suggest that Dropout [48] can be used to mimic a BNN by sampling different subsets of neurons at each forward pass during test time and use them as ensembles. MC Dropout is currently the most popular instance of BNNs due to its speed and simplicity, with multiple recent extensions [11, 32, 50]. However, the benefits of Dropout are more limited for convolutional layers, where specific architectural design choices must be made [25, 38]. A potential drawback of MC Dropout concerns the fact that its uncertainty is not reducing with more training steps [41, 42]. TRADI is compatible with both fully-connected and convolutional layers, while uncertainty estimates are expected to improve with training as it relies on the Kalman filter formalism.

Ensemble Methods. Ensemble methods are arguably the top performers for measuring epistemic uncertainty, and are largely applied to various areas, *e.g.* active learning [3]. Lakshminarayan *et al.* [31] propose training an ensemble of DNNs with different initialization seeds. The major drawback of this method is its computational cost since one has to train multiple DNNs, a cost which is particularly high for computer vision architectures, *e.g.*, semantic segmentation, object detection. Alternatives to ensembles use a network with multiple prediction heads [35], collect weight checkpoints from local minima and average them [23] or fit a distribution over them and sample networks [37]. Although

the latter approaches are faster to train than ensembles, their limitation is that the observations from these local minima are relatively sparse for such a high dimensional space and are less likely to capture the true distributions of the space around these weights. With TRADI we are mitigating these points as we collect weight statistics at each step of the SGD optimization. Furthermore, our algorithm has a lighter computational cost than [31] during training.

Kalman filtering (KF). The KF [24] is a recursive estimator that constructs an inference of unknown variables given measurements over time. With the advent of DNNs, researchers have tried integrating ideas from KF in DNN training: for SLAM using RNNs [8, 16], optimization [51], DNN fusion [36]. In our approach, we employ KF for keeping track of the statistics of the network during training such that at "convergence" we have a better coverage of the distribution around each parameter of a multi-million parameter DNN. The KF provides a clean and relatively easy to deploy formalism to this effect.

Weight initialization and optimization. Most DNN initialization techniques [12, 17] start from weights sampled from a Normal distribution, and further scale them according to the number of units and the activation function. BatchNorm [22] stabilizes training by enforcing a Normal distribution of intermediate activations at each layer. WeightNorm [46] has a similar effect over the weights, making sure they are sticking to the initial distributions. From a Bayesian perspective the L_2 regularization, known as weight decay, is equivalent to putting a Gaussian prior over the weights [4]. We also consider a Gaussian prior over the weights, similar to previous works [6, 23] for its numerous properties, ease of use and natural compatibility with KF. Note that we use it only in the filtering in order to reduce any major drift in the estimation of distributions of the weights across training, while mitigating potential instabilities in SGD steps.

4 Experiments

We evaluate TRADI on a range of tasks and datasets. For regression , in line with prior works [10, 31], we consider a toy dataset and the regression benchmark [21]. For classification we evaluate on MNIST [34] and CIFAR-10 [29]. Finally, we address the Out-of-Distribution task for classification, on MNIST/notMNIST [31], and for semantic segmentation, on CamVid-OOD, StreetHazards [19], and BDD-Anomaly [19]. Unless otherwise specified, we use mini-batches of size 128 and Adam optimizer with fixed learning rate of 0.1 in all our experiments.

4.1 Toy experiments

Experimental setup. As evaluation metric we use mainly the NLL uncertainty. In addition for classification we consider the accuracy, while for regression we use the root mean squared error (RMSE). For the out- of-distribution experiments we use the AUC, AUPR, FPR-95%-TPR as in [20], and the Expected Calibration Error (ECE) as in [15]. For our implementations we use PyTorch [44]. Unless otherwise specified, we use mini-batches of size 128 and Adam optimizer with

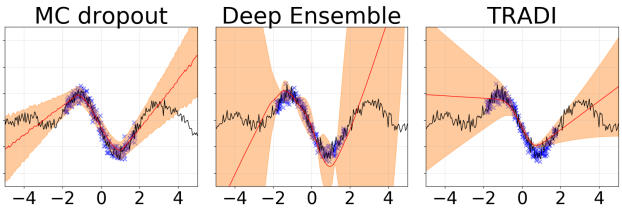

Fig. 2: Results on a synthetic regression task comparing MC dropout, Deep Ensembles, and TRADI. x-axis: spatial coordinate of the Gaussian process. Black lines: ground truth curve. Blue points: training points. Orange areas: estimated variance.

fixed learning rate of 0.1 in all our experiments. We provide other implementation details on per-experiment basis.

First we perform a qualitative evaluation on a one-dimensional synthetic dataset generated with a Gaussian Process of zero mean vector and as covariance function an RBF kernel $\boldsymbol{\mathcal{K}}$ with $\sigma^2 = 1$, denoted $GP(\mathbf{0}, \boldsymbol{\mathcal{K}})$. We add to this process a zero mean Gaussian noise of variance 0.3. We train a neural network composed of one hidden layer and 200 neurons. In Fig. 2 we plot the regression estimation provided by TRADI, MC Dropout [10] and Deep Ensembles [31]. Although $GP(\mathbf{0}, \boldsymbol{\mathcal{K}})$ is one of the simplest stochastic processes, results show clearly that the compared approaches do not handle robustly the variance estimation, while TRADI neither overestimates nor underestimates the uncertainty.

4.2 Regression experiments

For the regression task, we consider the experimental protocol and the data sets from [21], and also used in related works [10, 31]. Here, we consider a neural network with one hidden layer, composed of 50 hidden units trained for 40 epochs. For each dataset, we do 20-fold cross-validation. For all datasets, we set the dropout rate to 0.1 except for *Yacht Hydrodynamics* and *Boston Housing* for which it is set to 0.001 and 0.005, respectively. We compare against MC Dropout [10] and Deep Ensembles [31] and report results in Table 1. TRADI outperforms both methods, in terms of both RMSE and NLL. Aside from the proposed approach to tracking the weight distribution, we assume that an additional reason for which our technique outperforms the alternative methods resides in the sequential training (MSE and NLL) proposed in Section 2.3.

4.3 Classification experiments

For the classification task, we conduct experiments on two datasets. The first one is the MNIST dataset [34], which is composed of a training set containing 60k images and a testing set of 10k images, all of size 28×28. Here, we use a neural network with 3 hidden layers, each one containing 200 neurons, followed by ReLU non-linearities and BatchNorm, and fixed the learning rate $\eta = 10^{-2}$. We share our results in Table 2. For the MNIST dataset, we generate $N_{\text{model}} = 20$

Table 1: Comparative results on regression benchmarks

Datasets	RMSE			NLL		
	MC Dropout	Deep Ensembles	TRADI	MC Dropout	Deep Ensembles	TRADI
Boston Housing	2.97 ± 0.85	3.28 ± 1.00	$\mathbf{2.84 \pm 0.77}$	2.46 ± 0.25	2.41 ± 0.25	$\mathbf{2.36 \pm 0.17}$
Concrete Strength	5.23 ± 0.53	6.03 ± 0.58	$\mathbf{5.20 \pm 0.45}$	3.04 ± 0.09	3.06 ± 0.18	$\mathbf{3.03 \pm 0.08}$
Energy Efficiency	1.66 ± 0.16	2.09 ± 0.29	$\mathbf{1.20 \pm 0.27}$	1.99 ± 0.09	$\mathbf{1.38 \pm 0.22}$	1.40 ± 0.16
Kin8nm	0.10 ± 0.00	0.09 ± 0.00	$\mathbf{0.09 \pm 0.00}$	-0.95 ± 0.03	$\mathbf{-1.2 \pm 0.02}$	-0.98 ± 0.06
Naval Propulsion	0.01 ± 0.00	0.00 ± 0.00	$\mathbf{0.00 \pm 0.00}$	-3.80 ± 0.05	$\mathbf{-5.63 \pm 0.05}$	-2.83 ± 0.24
Power Plant	4.02 ± 0.18	4.11 ± 0.17	$\mathbf{4.02 \pm 0.14}$	2.80 ± 0.05	$\mathbf{2.79 \pm 0.04}$	2.82 ± 0.04
Protein Structure	4.36 ± 0.04	4.71 ± 0.06	$\mathbf{4.35 \pm 0.03}$	2.89 ± 0.01	2.83 ± 0.02	$\mathbf{2.80 \pm 0.02}$
Wine Quality Red	0.62 ± 0.04	0.64 ± 0.04	$\mathbf{0.62 \pm 0.03}$	0.93 ± 0.06	0.94 ± 0.12	$\mathbf{0.93 \pm 0.05}$
Yacht Hydrodynamics	1.11 ± 0.38	1.58 ± 0.48	$\mathbf{1.05 \pm 0.25}$	1.55 ± 0.12	1.18 ± 0.21	$\mathbf{1.18 \pm 0.39}$

models, in order to ensure a fair comparison with Deep Ensembles. The evaluation underlines that in terms of performance TRADI is positioned between Deep Ensembles and MC Dropout. However, in contrast to Deep Ensembles our algorithm is significantly lighter because only a single model needs to be trained, while Deep Ensembles approximates the weight distribution by a very costly step of independent training procedures (in this case 20).

We conduct the second experiment on CIFAR-10 [29], with WideResnet 28×10 [55] as DNN. The chosen optimization algorithm is SGD, $\eta = 0.1$ and the dropout rate was fixed to 0.3. Due to the long time necessary for Deep Ensembles to train the DNNs we set $N_{\text{model}} = 15$. Comparative results on this dataset, presented in Table 2, allow us to make similar conclusions with experiments on the MNIST dataset.

Table 2: Comparative results on image classification

Method	**MNIST**		**CIFAR-10**	
	NLL	ACCU	NLL	ACCU
Deep Ensembles	**0.035**	**98.88**	0.173	95.67
MC Dropout	0.065	98.19	0.205	95.27
SWAG	0.041	98.78	**0.110**	**96.41**
TRADI (ours)	0.044	98.63	0.205	95.29

4.4 Uncertainty evaluation for out-of-distribution (OOD) test samples.

In these experiments, we evaluate uncertainty on OOD classes. We consider four datasets, and the objective of these experiments is to evaluate to what extent the trained DNNs are overconfident on instances belonging to classes which are not present in the training set. We report results in Table 3.

Baselines. We compare against Deep Ensembles and MC Dropout, and propose two additional baselines. The first is the Maximum Classifier Prediction (MCP) which uses the maximum softmax value as prediction confidence and has shown competitive performance [19, 20]. Second, we propose a baseline to emphasize the ability of TRADI to capture the distribution of the weights. We take a *trained* network and randomly perturb its weights with noise sampled

Table 3: Distinguishing in- and out-of-distribution data for semantic segmentation (CamVid, StreetHazards, BDD Anomaly) and image classification (MNIST/notMNIST)

Dataset	OOD technique	AUC	AUPR	FPR-95%-TPR	ECE	Train time
MNIST/notMNIST 3 hidden layers	Baseline (MCP)	94.0	96.0	24.6	**0.305**	2m
	Gauss. perturbation ensemble	94.8	96.4	19.2	0.500	2m
	MC Dropout	91.8	94.9	35.6	0.494	2m
	Deep Ensemble	**97.2**	**98.0**	**9.2**	0.462	31m
	SWAG	90.9	94.4	31.9	0.529	
	TRADI (ours)	96.7	97.6	11.0	0.407	2m
CamVid-OOD ENET	Baseline (MCP)	75.4	10.0	65.1	0.146	30m
	Gauss. perturbation ensemble	76.2	10.9	62.6	0.133	30m
	MC Dropout	75.4	10.7	63.2	0.168	30m
	Deep Ensemble	**79.7**	**13.0**	**55.3**	0.112	5h
	SWAG	75.6	12.1	65.8	0.133	
	TRADI (ours)	79.3	**12.8**	57.7	**0.110**	41m
StreetHazards PSPNet	Baseline (MCP)	88.7	6.9	26.9	0.055	13h14m
	Gauss. perturbation ensemble	57.08	2.4	71.0	0.185	13h14m
	MC Dropout	69.9	6.0	32.0	0.092	13h14m
	Deep Ensemble	**90.0**	7.2	25.4	0.051	132h19m
	TRADI (ours)	89.2	**7.2**	**25.3**	**0.049**	15h36m
BDD Anomaly PSPNet	Baseline (MCP)	86.0	5.4	27.7	0.159	18h08
	Gauss. perturbation ensemble	86.0	4.8	27.7	0.158	18h08m
	MC Dropout	85.2	5.0	29.3	0.181	18h08m
	Deep Ensemble	87.0	6.0	**25.0**	0.170	189h40m
	TRADI (ours)	86.1	5.6	26.9	**0.157**	21h48m

from a Normal distribution. In this way we generate an ensemble of networks, each with different noise perturbations – we practically sample networks from the vicinity of the local minimum. We refer to it as *Gaussian perturbation ensemble*.

First we consider MNIST trained DNNs and use them on a test set composed of 10k MNIST images and 19k images from NotMNIST [1], a dataset of instances of ten classes of letters. Standard DNNs will assign letter instances of NotMNIST to a class number with high confidence as shown in [1]. For these OOD instances, our approach is able to decrease the confidence as illustrated in Fig. 3a, in which we represent the *accuracy vs confidence* curves as in [31].

The *accuracy vs confidence* curve is constructed by considering, for different confidence thresholds, all the test data for which the classifier reports a confidence above the threshold, and then by evaluating the accuracy on this data. The confidence of a DNN is defined as the maximum prediction score. We also evaluate the OOD uncertainty using AUC, AUPR and FPR-95%-TPR metrics, introduced in [20] and the ECE metrics[7] introduced in [15]. These criteria characterize the quality of the prediction that a testing sample is OOD with respect to the training dataset. We also measured the computational training times of all algorithms implemented in PyTorch on a PC equipped with Intel Core i9-9820X and one GeForce RTX 2080 Ti and report them in Table 3. We note that TRADI DNN

[7] Please note that the ECE is calculated over the joint dataset composed of the In distribution and the Out of distribution test data.

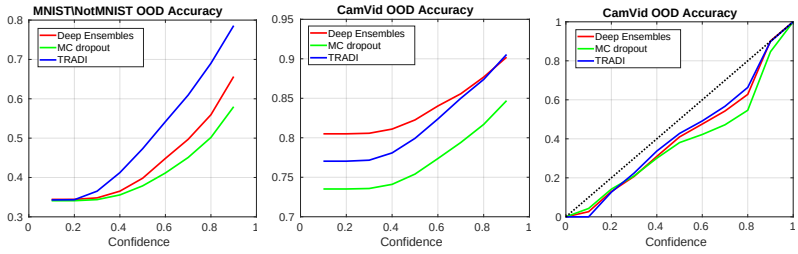

Fig. 3: (a) and (b) Accuracy vs confidence plot on the MNIST \notMNIST and CamVid experiments, respectively. (c) Calibration plot for the CamVid experiment.

with 20 models provides incorrect predictions on such OOD samples with lower confidence than Deep Ensembles and MC Dropout.

In the second experiment, we train a Enet DNN [43] for semantic segmentation on CamVid dataset [7]. During training, we delete three classes (pedestrian, bicycle, and car), by marking the corresponding pixels as unlabeled. Subsequently, we test with data containing the classes represented during training, as well as the deleted ones. The goal of this experiment is to evaluate the DNN behavior on the deleted classes which represent thus OOD classes. We refer to this setup as CamVid-OOD. In this experiment we use $N_{\text{model}} = 10$ models trained for 90 epochs with SGD and using a learning rate $\eta = 5 \times 10^{-4}$. In Fig. 3b and 3c we illustrate the *accuracy vs confidence* curves and the *calibration* curves [15] for the CamVid experiment. The calibration curve as explained in [15] consists in dividing the test set into bins of equal size according to the confidence, and in computing the accuracy over each bin. Both the calibration and the *accuracy vs confidence* curves highlight whether the DNN predictions are good for different levels of confidence. However, the calibration provides a better understanding of what happens for different scores.

Finally, we conducted experiments on the recent OOD benchmarks for semantic segmentation StreetHazards [19] and BDD Anomaly [19]. The former consists of 5,125/1,031/1,500 (train/test-in-distribution/test-OOD) synthetic images [9] with annotations for 12 classes for training and a 13^{th} OOD class found only in the test-OOD set. The latter is a subset of BDD [54] and is composed of 6,688/951/361 images, with the classes *motorcycle* and *train* as anomalous objects. We follow the experimental setup from [19], *i.e.*, PSPNet [56] with ResNet50 [18] backbone. On StreetHazards, TRADI outperforms Deep Ensembles and on BDD Anomaly Deep Ensembles has best results close to the one of TRADI.

Results show that TRADI outperforms the alternative methods in terms of calibration, and that it may provide more reliable confidence scores. Regarding *accuracy vs confidence*, the most significant results for a high level of confidence, typically above 0.7, show how overconfident the network tends to behave; in this range, our results are similar to those of Deep Ensembles. Lastly, in all experiments TRADI obtains performances close to the best AUPR and AUC,

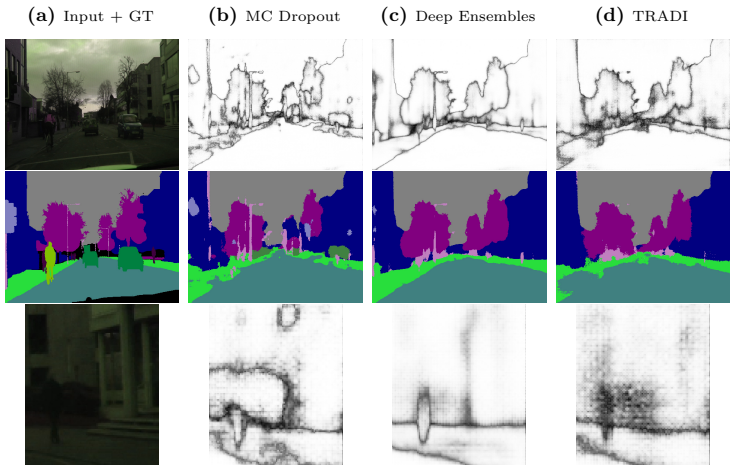

Fig. 4: Qualitative results on CamVid-OOD. Columns: (a) input image and ground truth; (b)-(d) predictions and confidence scores by MC Dropout, Deep Ensembles, and TRADI. Rows: (1) input and confidence maps; (2) class predictions; (3) zoomed-in area on input and confidence maps

while having a computational time /training time significantly smaller than Deep Ensembles.

Qualitative discussion. In Fig. 4 we give as example a scene featuring the three OOD instances of interest (*bike*, *car*, *pedestrian*). Overall, MC Dropout outputs a noisy uncertainty map, but fails to highlight the OOD samples. By contrast, Deep Ensembles is overconfident, with higher uncertainty values mostly around the borders of the objects. TRADI uncertainty is higher on borders and also on pixels belonging to the actual OOD instances, as shown in the zoomed-in crop of the pedestrian in Fig. 4 (row 3).

5 Conclusion

In this work we propose a novel technique for computing the epistemic uncertainty of a DNN. TRADI is conceptually simple and easy to plug to the optimization of any DNN architecture. We show the effectiveness of TRADI over extensive studies and compare against the popular MC Dropout and the state of the art Deep Ensembles. Our method exhibits an excellent performance on evaluation metrics for uncertainty quantification, and in contrast to Deep Ensembles, for which the training time depends on the number of models, our algorithm does not add any significant cost over conventional training times.

Future works involve extending this strategy to new tasks, *e.g.*, object detection, or new settings, *e.g.*, active learning. Another line of future research concerns transfer learning. So far TRADI is starting from randomly initialized weights sampled from a given Normal distribution. In transfer learning, we start from a

pre-trained network where weights are expected to follow a different distribution. If we have access to the distribution of the DNN weights we can improve the effectiveness of transfer learning with TRADI.

Bibliography

[1] Notmnist dataset. http://yaroslavvb.blogspot.com/2011/09/notmnist-dataset.html

[2] Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M.W., Pfau, D., Schaul, T., Shillingford, B., De Freitas, N.: Learning to learn by gradient descent by gradient descent. In: Advances in neural information processing systems. pp. 3981–3989 (2016)

[3] Beluch, W.H., Genewein, T., Nürnberger, A., Köhler, J.M.: The power of ensembles for active learning in image classification. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 9368–9377 (2018)

[4] Bishop, C.M.: Pattern recognition and machine learning. springer (2006)

[5] Blundell, C., Cornebise, J., Kavukcuoglu, K., Wierstra, D.: Weight uncertainty in neural network. In: Bach, F., Blei, D. (eds.) Proceedings of the 32nd International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 37, pp. 1613–1622. PMLR, Lille, France (07–09 Jul 2015), http://proceedings.mlr.press/v37/blundell15.html

[6] Blundell, C., Cornebise, J., Kavukcuoglu, K., Wierstra, D.: Weight uncertainty in neural networks. arXiv preprint arXiv:1505.05424 (2015)

[7] Brostow, G.J., Shotton, J., Fauqueur, J., Cipolla, R.: Segmentation and recognition using structure from motion point clouds. In: European conference on computer vision. pp. 44–57. Springer (2008)

[8] Chen, C., Lu, C.X., Markham, A., Trigoni, N.: Ionet: Learning to cure the curse of drift in inertial odometry. In: The Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18) (2018)

[9] Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., Koltun, V.: CARLA: An open urban driving simulator. In: Proceedings of the 1st Annual Conference on Robot Learning. pp. 1–16 (2017)

[10] Gal, Y., Ghahramani, Z.: Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In: international conference on machine learning. pp. 1050–1059 (2016)

[11] Gal, Y., Hron, J., Kendall, A.: Concrete dropout. In: NIPS (2017)

[12] Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the thirteenth international conference on artificial intelligence and statistics. pp. 249–256 (2010)

[13] Graves, A.: Practical variational inference for neural networks. In: Advances in neural information processing systems. pp. 2348–2356 (2011)

[14] Grewal, M.S.: Kalman filtering. Springer (2011)

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

[16] Haarnoja, T., Ajay, A., Levine, S., Abbeel, P.: Backprop kf: Learning discriminative deterministic state estimators. In: Advances in Neural Information Processing Systems. pp. 4376–4384 (2016)
[17] He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE international conference on computer vision. pp. 1026–1034 (2015)
[18] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
[19] Hendrycks, D., Basart, S., Mazeika, M., Mostajabi, M., Steinhardt, J., Song, D.: A benchmark for anomaly segmentation. arXiv preprint arXiv:1911.11132 (2019)
[20] Hendrycks, D., Gimpel, K.: A baseline for detecting misclassified and out-of-distribution examples in neural networks. arXiv preprint arXiv:1610.02136 (2016)
[21] Hernández-Lobato, J.M., Adams, R.: Probabilistic backpropagation for scalable learning of bayesian neural networks. In: International Conference on Machine Learning. pp. 1861–1869 (2015)
[22] Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167 (2015)
[23] Izmailov, P., Podoprikhin, D., Garipov, T., Vetrov, D., Wilson, A.G.: Averaging weights leads to wider optima and better generalization. arXiv preprint arXiv:1803.05407 (2018)
[24] Kalman, R.E.: A new approach to linear filtering and prediction problems. Journal of basic Engineering **82**(1), 35–45 (1960)
[25] Kendall, A., Badrinarayanan, V., , Cipolla, R.: Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding. arXiv preprint arXiv:1511.02680 (2015)
[26] Kendall, A., Gal, Y.: What uncertainties do we need in bayesian deep learning for computer vision? In: Advances in neural information processing systems. pp. 5574–5584 (2017)
[27] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
[28] Kingma, D.P., Welling, M.: Auto-encoding variational bayes. In: 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings (2014)
[29] Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images. Tech. rep., Citeseer (2009)
[30] Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp. 1097–1105 (2012)
[31] Lakshminarayanan, B., Pritzel, A., Blundell, C.: Simple and scalable predictive uncertainty estimation using deep ensembles. In: Advances in Neural Information Processing Systems. pp. 6402–6413 (2017)

[32] Lambert, J., Sener, O., Savarese, S.: Deep learning under privileged information using heteroscedastic dropout. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition pp. 8886–8895 (2018)
[33] Lan, J., Liu, R., Zhou, H., Yosinski, J.: Lca: Loss change allocation for neural network training. In: Advances in Neural Information Processing Systems. pp. 3614–3624 (2019)
[34] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998)
[35] Lee, S., Purushwalkam, S., Cogswell, M., Crandall, D., Batra, D.: Why m heads are better than one: Training a diverse ensemble of deep networks. arXiv preprint arXiv:1511.06314 (2015)
[36] Liu, C., Gu, J., Kim, K., Narasimhan, S.G., Kautz, J.: Neural rgb (r) d sensing: Depth and uncertainty from a video camera. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 10986–10995 (2019)
[37] Maddox, W., Garipov, T., Izmailov, P., Vetrov, D., Wilson, A.G.: A simple baseline for bayesian uncertainty in deep learning. arXiv preprint arXiv:1902.02476 (2019)
[38] Mukhoti, J., Gal, Y.: Evaluating bayesian deep learning methods for semantic segmentation. CoRR **abs/1811.12709** (2018), http://arxiv.org/abs/1811.12709
[39] Neal, R.M.: Bayesian Learning for Neural Networks. Springer-Verlag, Berlin, Heidelberg (1996)
[40] Ollivier, Y.: The extended kalman filter is a natural gradient descent in trajectory space. arXiv preprint arXiv:1901.00696 (2019)
[41] Osband, I.: Risk versus uncertainty in deep learning : Bayes , bootstrap and the dangers of dropout (2016)
[42] Osband, I., Aslanides, J., Cassirer, A.: Randomized prior functions for deep reinforcement learning. In: NeurIPS (2018)
[43] Paszke, A., Chaurasia, A., Kim, S., Culurciello, E.: Enet: A deep neural network architecture for real-time semantic segmentation. arXiv preprint arXiv:1606.02147 (2016)
[44] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: Pytorch: An imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems. pp. 8024–8035 (2019)
[45] Rahimi, A., Recht, B.: Random features for large-scale kernel machines. In: Advances in neural information processing systems. pp. 1177–1184 (2007)
[46] Salimans, T., Kingma, D.P.: Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In: Advances in Neural Information Processing Systems. pp. 901–909 (2016)
[47] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
[48] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A simple way to prevent neural networks

from overfitting. J. Mach. Learn. Res. **15**(1), 1929–1958 (Jan 2014), http://dl.acm.org/citation.cfm?id=2627435.2670313

[49] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A., et al.: Going deeper with convolutions. arxiv 2014. arXiv preprint arXiv:1409.4842 **1409** (2014)

[50] Teye, M., Azizpour, H., Smith, K.: Bayesian uncertainty estimation for batch normalized deep networks. In: ICML (2018)

[51] Wang, G., Peng, J., Luo, P., Wang, X., Lin, L.: Batch kalman normalization: Towards training deep neural networks with micro-batches. arXiv preprint arXiv:1802.03133 (2018)

[52] Williams, C.K., Rasmussen, C.E.: Gaussian processes for machine learning, vol. 2. MIT press Cambridge, MA (2006)

[53] Yang, G.: Scaling limits of wide neural networks with weight sharing: Gaussian process behavior, gradient independence, and neural tangent kernel derivation. arXiv preprint arXiv:1902.04760 (2019)

[54] Yu, F., Xian, W., Chen, Y., Liu, F., Liao, M., Madhavan, V., Darrell, T.: Bdd100k: A diverse driving video database with scalable annotation tooling. arXiv preprint arXiv:1805.04687 (2018)

[55] Zagoruyko, S., Komodakis, N.: Wide residual networks. arXiv preprint arXiv:1605.07146 (2016)

[56] Zhao, H., Shi, J., Qi, X., Wang, X., Jia, J.: Pyramid scene parsing network. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2881–2890 (2017)