Sketching Curvature for Efficient Out-of-Distribution Detection for Deep Neural Networks

Apoorva Sharma[1] **Navid Azizan**[1,2] **Marco Pavone**[1]

[1]Stanford University
[2]Massachusetts Institute of Technology

Abstract

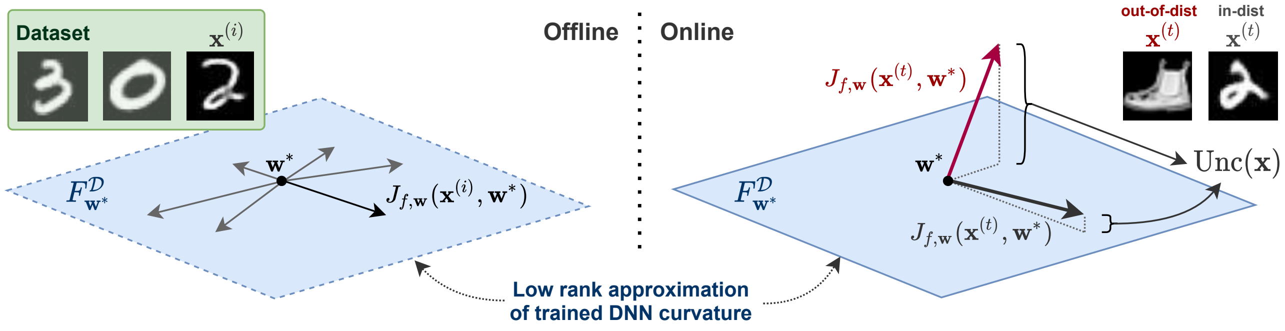
In order to safely deploy Deep Neural Networks (DNNs) within the perception pipelines of real-time decision making systems, there is a need for safeguards that can detect out-of-training-distribution (OoD) inputs both efficiently and accurately. Building on recent work leveraging the local curvature of DNNs to reason about epistemic uncertainty, we propose *Sketching Curvature for OoD Detection (SCOD)*, an architecture-agnostic framework for equipping any trained DNN with a task-relevant epistemic uncertainty estimate. Offline, given a trained model and its training data, SCOD employs tools from matrix sketching to tractably compute a low-rank approximation of the Fisher information matrix, which characterizes which directions in the weight space are most influential on the predictions over the training data. Online, we estimate uncertainty by measuring how much perturbations orthogonal to these directions can alter predictions at a new test input. We apply SCOD to pre-trained networks of varying architectures on several tasks, ranging from regression to classification. We demonstrate that SCOD achieves comparable or better OoD detection performance with lower computational burden relative to existing baselines.

1 INTRODUCTION

Deep Neural Networks (DNNs) have enabled breakthroughs in extracting actionable information from high-dimensional input streams, such as videos or images. However, a key limitation of these black-box models is that their performance can be erratic when queried with inputs that are significantly different from those seen during training. To alleviate this, there has been a growing field of literature aimed at equipping pre-trained DNN models with a measure of their uncertainty. These approaches aim to characterize what a given DNN model has learned from the dataset it was trained on, so as to detect at test time whether a new input is inconsistent.

One appealing direction to this end has leveraged the curvature of a pre-trained DNN about its optimized weights [Madras et al., 2019, Ritter et al., 2018]. The second-order analysis of the local sensitivity of the network to its weights can offer a post-hoc approximation of the intractable Bayesian posterior on the network weights. Known as the Laplace approximation, this is an attractive approach in its promise of adding a principled measure of epistemic uncertainty to any pre-trained network. However, computing the required curvature matrix is quadratic in the number of weights of the network, and is still intractable for today's DNN models with millions of weights. Thus, the literature has focused on methods to approximate this curvature to yield scalable approaches, yet these approaches have typically relied on imposing sparsity structures on the curvature matrix, e.g., by ignoring cross terms between layers of the network. Furthermore, these approaches tend to focus on estimating the *posterior predictive* distribution by marginalizing over the approximate posterior over the weights, which often requires the computationally intensive process of sampling weights and estimating the posterior predictive through Monte-Carlo integration.

Contributions. In this work, we build from the framework of the Laplace approximation to propose a novel, architecture-agnostic method for equipping any trained DNN with estimates of task-relevant input atypicality. Rather than incorporate epistemic uncertainty into the probabilistic prediction of the network, we propose augmenting the network output with a scalar uncertainty measure that quantifies the degree of atypicality for a given input. Our specific contributions are as follows: (1) We leverage information geometry to translate curvature-informed weight uncertainty in DNNs to a task-relevant measure of input



Dataset
$\mathbf{x}^{(i)}$
Offline
Online
$F^{\mathcal{D}}_{\mathbf{w}^*}$
$\mathbf{w}^*$
$J_{f,\mathbf{w}}(\mathbf{x}^{(i)}, \mathbf{w}^*)$
$J_{f,\mathbf{w}}(\mathbf{x}^{(t)}, \mathbf{w}^*)$
out-of-dist
in-dist
$\mathbf{x}^{(t)}$
$\mathrm{Unc}(\mathbf{x})$
Low rank approximation of trained DNN curvature

Figure 1: Offline, SCOD uses gradients $\boldsymbol{J}_{f,\boldsymbol{w}}$ of the DNN on training data to build a low-rank estimate of the network curvature, as defined by the Fisher information matrix, employing tools from matrix sketching to make this estimation tractable and scalable. Online, SCOD produces a metric of atypicality Unc using gradient information at a test datapoint $\boldsymbol{x}^{(t)}$, and comparing this to the low-rank approximation of the Fisher. Derived from an approximation of the Bayesian posterior over the weights, this metric can be viewed, for scalar regression DNNs, as measuring how orthogonal a new test point gradient is from the gradients seen on training inputs.

atypicality; (2) we show how this measure can be computed efficiently by leveraging backpropagation and a characterization of the top eigenspace of the Fisher matrix; (3) we develop a technique based on matrix sketching to tractably approximate this top eigenspace, even for large models and large datasets; (4) we empirically demonstrate that this atypicality measure matches or exceeds the OoD detection performance relative to baselines on a diverse set of problems and architectures from regression to classification.

2 PROBLEM STATEMENT

We take a probabilistic interpretation, and define a DNN model as a mapping from inputs $\boldsymbol{x} \in \mathscr{X}$ to probability distributions over targets $\boldsymbol{y} \in \mathscr{Y}$. Typically, this is broken down into the composition of a neural network architecture f mapping inputs $\boldsymbol{x}$ and weights $\boldsymbol{w} \in \mathbb{R}^N$ to parameters $\boldsymbol{\theta} \in \mathbb{R}^d$, and and a distributional family $\mathscr{P}$ mapping these parameters to a distribution over the targets:

$$\boldsymbol{\theta} = f(\boldsymbol{x}, \boldsymbol{w}), \qquad p(\boldsymbol{y}) = \mathscr{P}(\boldsymbol{\theta}).$$

We use $p_{\boldsymbol{w}}(\boldsymbol{y} \mid \boldsymbol{x}) := \mathscr{P} \circ f$ as a convenient shorthand for the conditional distribution that the DNN model defines. This model is trained on a dataset of examples $\mathscr{D} = \{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^M$, where we assume the inputs are drawn i.i.d. from some training distribution $p_{\text{train}}(x, y)$, to minimize the Kullback-Leibler (KL) divergence from the empirical distribution in the dataset to $p_{\boldsymbol{w}}(\boldsymbol{y} \mid \boldsymbol{x})$ [1].

Our goal is to take a trained DNN, and equip it with an OoD monitor which can detect if a given input is far from the data seen at train time, and thus likely to result in incorrect and overconfident predictions. Specifically, we assume we are given the functional forms f and $\mathscr{P}$, the trained weights $\boldsymbol{w}^*$, as well as a set of training data $\mathscr{D}$, and wish to construct an uncertainty measure $\mathrm{Unc}(\cdot) : \mathscr{X} \to \mathbb{R}$. Intuitively, we wish for this measure to be low when queried on inputs drawn from $p_{\text{train}}(x)$, but high for inputs far from this data manifold. This can be thought of as providing a measure of *epistemic uncertainty* due to an input not being covered by the training set, or a measure of *novelty* with respect to the training data, useful for detecting *anomalies* or *out-of-distribution* inputs.

We wish to design a monitor that can provide a real-time uncertainty signal alongside the DNN's predictions. Thus, we desire a function $\mathrm{Unc}(\cdot)$ that is both computationally efficient (to run at test time) as well as informative, providing a meaningful anomaly signal that can effectively separate held-out validation inputs that are known to be in-distribution, from inputs which we know to be out-of-distribution and outside the DNN's domain of competency.

3 BACKGROUND: CURVATURE AND LAPLACE APPROXIMATION

Key to our approach is the curvature of DNN models: a second-order characterization of how perturbations to the weights of a DNN influence its probabilistic output. In this section, we review tools to characterize DNN curvature, and discuss the connections between curvature and epistemic uncertainty estimates through the Laplace approximation.

3.1 FISHER INFORMATION

A crucial aspect of our approach relies on understanding how the parameters of the model influence its output dis-

[1] This general formulation covers almost all applications of DNNs; for example, choosing $\mathscr{P}$ to be a categorical distribution parameterized by logits $\boldsymbol{\theta}$ recovers the standard softmax training objective for classification, while choosing $\mathscr{P}$ to be unit variance Gaussian with mean $\boldsymbol{\theta}$ recovers the typical mean-squared error objective for regression.

tribution. To do so, we leverage tools from information geometry. We can view the family of distributions defined by the model $\mathscr{P}(\boldsymbol{\theta})$ as defining a statistical manifold with coordinates $\boldsymbol{\theta}$. The Riemannian metric for this manifold is the Fisher information matrix

$$F_{\boldsymbol{\theta}}(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{y}}\left[(\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{y}))(\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{y}))^{\mathsf{T}}\right] \quad (1)$$

$$= \mathbb{E}_{\boldsymbol{y}}\left[-\frac{\partial^2}{\partial \boldsymbol{\theta}^2} \log p_{\boldsymbol{\theta}}(\boldsymbol{y})\right], \quad \text{(see note)}^2 \quad (2)$$

where $p_{\boldsymbol{\theta}}(\cdot) = \mathscr{P}(\boldsymbol{\theta})$, the pdf of the probability distribution on $\mathscr{Y}$ defined by parameters $\boldsymbol{\theta}$. The Fisher information matrix (henceforth referred to as the Fisher) represents the second-order approximation of the local KL divergence, describing how the output distribution of a model changes with small perturbations to the distribution parameters $\boldsymbol{\theta}$:

$$D_{\mathsf{KL}}(\mathscr{P}(\boldsymbol{\theta})||\mathscr{P}(\boldsymbol{\theta}+d\boldsymbol{\theta})) \approx d\boldsymbol{\theta}^{\mathsf{T}} F_{\boldsymbol{\theta}}(\boldsymbol{\theta}) d\boldsymbol{\theta} + O(d\boldsymbol{\theta}^3). \quad (3)$$

The subscript on $F_{\boldsymbol{\theta}}$ serves to make explicit the Fisher's dependence on the model's parameterization.

For many common parametric distributions, the Fisher can be computed analytically. For example, for the family of Gaussian distributions with fixed covariance Σ, parameterized by the mean vector $\boldsymbol{\theta} = \boldsymbol{\mu}$, the Fisher information is simply the constant $F_{\boldsymbol{\theta}}(\boldsymbol{\theta}) = \Sigma^{-1}$. For a categorical distribution parameterized such that $\boldsymbol{\theta}_i$ represents the probability assigned to class i, $F(\boldsymbol{\theta}) = (\mathsf{diag}(\boldsymbol{\theta}))^{-1}$. In cases where this analytic computation is not possible or difficult, one can compute a Monte-Carlo approximation of the Fisher by sampling $\boldsymbol{y} \sim p_{\boldsymbol{\theta}}(\cdot)$ to estimate the expectation in (1).

3.2 FISHER FOR DEEP NEURAL NETWORKS

For DNNs, we can also consider the Fisher in terms of the network weights $\boldsymbol{w}$ using a change of variables. Since $\boldsymbol{\theta} = f(x, \boldsymbol{w})$, we have

$$F_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{J}_{f,\boldsymbol{w}}^{\mathsf{T}} F_{\boldsymbol{\theta}}(f(\boldsymbol{x}, \boldsymbol{w})) \boldsymbol{J}_{f,\boldsymbol{w}}, \quad (4)$$

where $\boldsymbol{J}_{f,\boldsymbol{w}}$ is the Jacobian matrix with $[\boldsymbol{J}_{f,\boldsymbol{w}}]_{ij} = \frac{\partial f_i}{\partial w_j}$, evaluated at $(\boldsymbol{x}, \boldsymbol{w})$. Note that $F_{\boldsymbol{w}}$ is a function of both $\boldsymbol{w}$ and the input $\boldsymbol{x}$. We will henceforth use the shorthand $F_{\boldsymbol{w}^*}^{(t)} := F_{\boldsymbol{w}}(\boldsymbol{x}^{(t)}, \boldsymbol{w}^*)$ to denote this weight-space Fisher evaluated for a particular input $\boldsymbol{x}^{(t)}$ and the trained weights $\boldsymbol{w}^*$.

From (3), we see that the Fisher defines a second-order approximation of how perturbations in the weight space influence the DNN's probabilistic predictions:

$$\delta_{\mathsf{KL}}(\boldsymbol{x}^{(t)}) := D_{\mathsf{KL}}\left(p_{\boldsymbol{w}^*}(\cdot \mid \boldsymbol{x}^{(t)}) || p_{\boldsymbol{w}^*+d\boldsymbol{w}}(\cdot \mid \boldsymbol{x}^{(t)})\right)$$
$$\approx d\boldsymbol{w}^{\mathsf{T}} F_{\boldsymbol{w}^*}^{(t)} d\boldsymbol{w}.$$

[2]This equality holds under mild regularity conditions on $\mathscr{P}(\boldsymbol{\theta})$

We can also use the Fisher to consider the impact of weight perturbations on the predictions over the entire dataset as

$$\delta_{\mathsf{KL}}(\mathscr{D}) = \frac{1}{M}\sum_{i=1}^{M} \delta_{\mathsf{KL}}(\boldsymbol{x}^{(i)}) = \frac{1}{M}\sum_{i=1}^{M} d\boldsymbol{w}^{\mathsf{T}} F_{\boldsymbol{w}^*}^{(i)} d\boldsymbol{w}$$
$$= d\boldsymbol{w}^{\mathsf{T}} \underbrace{\left(\frac{1}{M}\sum_{i=1}^{M} F_{\boldsymbol{w}^*}^{(i)}\right)}_{:=F_{\boldsymbol{w}^*}^{\mathscr{D}}} d\boldsymbol{w}.$$

3.3 CONNECTION TO THE HESSIAN

As is evident from (1), there are strong connections between the dataset Fisher $F_{\boldsymbol{w}^*}^{\mathscr{D}}$ and the Hessian with respect to $\boldsymbol{w}$ of the log likelihood of the training data. If we define $L(\boldsymbol{w}) = \sum_{i=1}^{M} \log p_{\boldsymbol{w}^*}(\boldsymbol{y}^{(i)} \mid \boldsymbol{x}^{(i)})$, the Hessian of L evaluated at $\boldsymbol{w}^*$ can be well approximated by the dataset Fisher[3] [Ritter et al., 2018, Martens and Grosse, 2015]:

$$H_L = \left.\frac{\partial^2 L}{\partial \boldsymbol{w}^2}\right|_{\boldsymbol{w}=\boldsymbol{w}^*} \approx M F_{\boldsymbol{w}^*}^{\mathscr{D}}.$$

Unlike the Hessian, the Fisher is always guaranteed to be positive semidefinite. For this reason, this approximation is common in the field of second-order optimization, where preconditioning gradient steps with the inverse Fisher tends to be more efficient and numerically stable than using the Hessian.

3.4 THE LAPLACE APPROXIMATION OF EPISTEMIC UNCERTAINTY

The Hessian of the log-likelihood has strong connections to Bayesian ideas of *epistemic uncertainty*, the uncertainty due to lack of data. From a Bayesian perspective, one can choose a prior over the weights of a DNN, $p(\boldsymbol{w})$, and then reason about the posterior distribution on these weights given the dataset, $p(\boldsymbol{w} \mid \mathscr{D})$. Often, due to the overparameterized nature of DNNs, many values of $\boldsymbol{w}$ are likely under this dataset, corresponding to different ways the DNN can fit the training data [Azizan et al., 2019]. By characterizing the posterior, and then marginalizing over it to produce a posterior predictive distribution $p(\boldsymbol{y} \mid \boldsymbol{x}) = \int p(\boldsymbol{w} \mid \mathscr{D}) p_{\boldsymbol{w}}(\boldsymbol{y} \mid \boldsymbol{x}) d\boldsymbol{w}$, one can hope to detect atypical data by incorporating uncertainty due to lack of data into the network's probabilistic predictions.

[3]By application of the chain rule, $H_L = \hat{F}_{\boldsymbol{w}^*}^{\mathscr{D}} + C(\boldsymbol{w}^*)$, where $\hat{F}_{\boldsymbol{w}^*}^{\mathscr{D}}$ is the empirical Fisher where the expectation in (1) is replaced by an empirical expectation over the dataset, and $C(\boldsymbol{w}^*)$ is a term involving first derivatives of the log likelihood and second derivatives of the network f. For trained networks, we expect the Fisher and the empirical Fisher to be closely aligned, and $C(\boldsymbol{w}^*) \approx 0$ since the first derivatives of the log likelihood are near zero at the end of training. Furthermore, for piece-wise linear networks (e.g., with ReLU activations), the second deriviatves of f are 0, and so, $C(\boldsymbol{w}^*) = 0$.

While computing this posterior is intractable for DNNs, due to their nonlinearity and high-dimensional weight space, many approximations exist, leveraging Monte-Carlo sampling, or by assuming a distributional form and carrying out variational inference. One approximation is the Laplace approximation [MacKay, 1992], which involves a second-order approximation of the log posterior $\log p(\boldsymbol{w} \mid \mathscr{D})$ about a point estimate $\boldsymbol{w}^*$. This quadratic form yields a Gaussian posterior over the weights. If the prior on the weights is $p(\boldsymbol{w}) = \mathscr{N}(\cdot; 0, \varepsilon^2 I_N)$, the Laplace posterior is given by $\Sigma^* = \frac{1}{2}\left(H_L + \frac{1}{2\varepsilon^2} I_N\right)^{-1}$. The Laplace approximation is attractive as it uses local second-order information (the Hessian H_L) to produce an estimate of the Bayesian posterior for any pretrained model. However, even computing this approximation to the exact Bayesian posterior can be challenging for large models, where estimating and inverting an $N \times N$ matrix to compute Σ^* is demanding.

4 PROPOSED METHOD: SKETCHING CURVATURE FOR OOD DETECTION

In this section, we describe the main steps in our OoD detection method. We build upon ideas from the Laplace approximation, but using the dataset Fisher to approximate the Hessian. Thus, our approximate posterior is given by

$$\Sigma^* = \frac{1}{2}\left(M F^{\mathscr{D}}_{\boldsymbol{w}^*} + \frac{1}{2\varepsilon^2} I\right)^{-1}. \tag{5}$$

As we are only focused on the downstream-task OoD detection, we avoid the computationally expensive step of marginalizing over this uncertainty to form a posterior predictive distribution, and instead, directly quantify how this posterior uncertainty on the weights impacts the networks probabilistic predictions. A natural metric for that purpose is the expected change in the output distribution (measured by the KL divergence) when the weights are perturbed according to the Laplace posterior distribution, i.e.,

$$\begin{aligned} \mathrm{Unc}(\boldsymbol{x}^{(t)}) &= \mathop{\mathbb{E}}_{d\boldsymbol{w} \sim \mathscr{N}(0,\Sigma^*)} \left[\delta_{\mathsf{KL}}(\boldsymbol{x}^{(t)})\right] \\ &\approx \mathop{\mathbb{E}}_{d\boldsymbol{w} \sim \mathscr{N}(0,\Sigma^*)} \left[d\boldsymbol{w}^\top F^{(t)}_{\boldsymbol{w}^*} d\boldsymbol{w}\right] \\ &= \mathrm{Tr}\left(F^{(t)}_{\boldsymbol{w}^*} \Sigma^*\right). \end{aligned} \tag{6}$$

Crucially, by using the Fisher to estimate the change in the output distribution due to weight perturbation, we obtain a quadratic form whose expectation we can compute analytically. Note that this metric is the Frobenius inner product between the test input's Fisher and a regularized inverse of the dataset Fisher. Thus, we can view this metric as measuring novelty in the terms of DNN's curvature, i.e., measuring the abnormality of a test input $\boldsymbol{x}$ by comparing the curvature at this input against the curvature on training inputs.

4.1 EFFICIENTLY COMPUTING THE UNCERTAINTY METRIC

While (6) is a clean expression for an uncertainty metric, its naïve computation is intractable for typical DNNs, as both Σ^* and $F^{(t)}_{\mathscr{D}}$ are $N \times N$ matrices. To make this computation amenable for real-time operation, we exploit the fact that both matrices are low-rank. From their definitions, it follows that $\mathsf{rank}(F^{(t)}_{\boldsymbol{w}^*}) \le d$, and thus $\mathsf{rank}(F^{\mathscr{D}}_{\boldsymbol{w}^*}) \le Md$. The number of weights in a neural network N is always greater than the output dimension d, and for large models and typical dataset sizes, often also greater than Md. Thus, we choose instead to express both Fisher matrices in factored forms

$$F^{(t)}_{\boldsymbol{w}^*} = L^{(t)}_{\boldsymbol{w}^*} L^{(t)\top}_{\boldsymbol{w}^*}, \qquad F^{\mathscr{D}}_{\boldsymbol{w}^*} = U \mathsf{diag}(\boldsymbol{\lambda}) U^T, \tag{7}$$

where $L^{(t)}_{\boldsymbol{w}^*} \in \mathbb{R}^{N \times d}, U \in \mathbb{R}^{N \times Md}$, and $\boldsymbol{\lambda} \in \mathbb{R}^{Md}$. Note that if we write $F_{\boldsymbol{\theta}}(f(\boldsymbol{x}^{(t)}, \boldsymbol{w}^*)) = L^{(t)}_{\boldsymbol{\theta}^*} L^{(t)\top}_{\boldsymbol{\theta}^*}$, then from (4), we can see that $L^{(t)}_{\boldsymbol{w}^*} = \boldsymbol{J}^\top_{f,\boldsymbol{w}} L^{(t)}_{\boldsymbol{\theta}^*}$. For many common choices of parametric distributions, $L^{(t)}_{\boldsymbol{\theta}^*}$ can be computed analytically. Furthermore, leveraging the linearity of the derivative, we can compute each row of $L^{(t)}_{\boldsymbol{w}^*}$ efficiently via backpropagation (see Appendix A for details and examples for common distributions).

Given these factored forms, an application of the Woodbury matrix identity allows us to simplify the computation to

$$\mathrm{Unc}(\boldsymbol{x}^{(t)}) = \\ \varepsilon^2 \left\| L^{(t)}_{\boldsymbol{w}^*} \right\|^2_{\mathrm{F}} - \varepsilon^2 \left\| \mathsf{diag}\left(\sqrt{\frac{\boldsymbol{\lambda}}{\boldsymbol{\lambda} + 1/(2M\varepsilon^2)}}\right) U^\top L^{(t)}_{\boldsymbol{w}^*} \right\|^2_{\mathrm{F}}. \tag{8}$$

A derivation is provided in Appendix B.

In this new form, the computation is split into computing the factor $L^{(t)}_{\boldsymbol{w}^*}$, carrying out the matrix product with $U^\top$, and then computing Frobenius norms. The main bottleneck in this procedure, both in terms of memory and computation, is the matrix multiplication with the $N \times Md$ matrix U. However, several empirical analyses of neural network curvature have found that the Hessian and dataset Fisher $F^{\mathscr{D}}_{\boldsymbol{w}^*}$ exhibit rapid spectral decay [Sagun et al., 2017, Madras et al., 2019]). Indeed, we see that if $\lambda_i << 1/(2M\varepsilon^2)$, the corresponding element in the diagonal matrix tends to 0. Thus, we can approximate this computation by only considering the top k eigenvalues and eigenvectors of the dataset Fisher, drastically reducing memory and computation requirements at test time. Notably, making this low-rank approximation gives us a strict *over-estimate* of the exact quantity, which is well-suited for safety-critical settings, where being under-confident is more desirable than being over-confident. An error bound is provided in Appendix B.

4.2 TRACTABLY APPROXIMATING THE DATASET FISHER VIA MATRIX SKETCHING

In order to compute $\mathrm{Unc}(\boldsymbol{x}^{(t)})$ online, we require the top k eigenvalues and eigenvectors of $F^{\mathscr{D}}_{\boldsymbol{w}^*}$. While this computation can happen offline, it is still intractable to carry out exactly for common DNN models and large datasets, since simply representing $F^{\mathscr{D}}_{\boldsymbol{w}^*}$ exactly requires storing an $Md \times N$ factor, which can easily grow beyond the capacity of common GPU memories for large perception networks and datasets with tens of thousands of parameters. To alleviate this issue, prior work has considered imposing sparsity patterns on the Fisher, e.g., diagonal (which ignores correlations between weights), or layer-wise block-diagonal [Ritter et al., 2018] (which ignores correlations between layers). Instead, we note that only the top eigenvectors of the datastet Fisher are important to the computation of our uncertainty metric, and thus we turn to tools from matrix sketching to tractably estimate a low-rank approximation of the Fisher *without* imposing any sparsity structure on the matrix.

The key idea in matrix sketching, to avoid working with a large matrix directly, is to apply a randomized linear map S to the matrix of interest [Tropp et al., 2017]. By appropriately randomizing this map (the *sketching operator*), we obtain high-probability guarantees that the image of the original matrix produced by the map (the *sketch*) encodes sufficient information about the original matrix. Given a bound on the desired approximation error, the size required for the sketch depends on the desired rank of the approximation k, and not the original size of the large matrix. Thus, this can enable our technique to be applied to arbitrarily large datasets.

The linearity of the sketching operator allows us to form this sketch iteratively, using a single pass over the dataset, without storing the full dataset Fisher in memory. Specifically, from the definition of $F^{\mathscr{D}}_{\boldsymbol{w}^*}$, we can compute its sketch as the sum of smaller sketches:

$$\mathsf{S}\left(F^{\mathscr{D}}_{\boldsymbol{w}^*}\right) = \frac{1}{M}\sum_{i=1}^{M} \mathsf{S}\left(F^{(i)}_{\boldsymbol{w}^*}\right) = \frac{1}{M}\sum_{i=1}^{M} \mathsf{S}\left(L^{(i)}_{\boldsymbol{w}^*} L^{(i)\top}_{\boldsymbol{w}^*}\right). \quad (9)$$

Following Tropp et al. [2017], we choose S to independently left- and right-multiply the Fisher by random sketching matrices. Specifically, the sketch of each component is computed as $Y^{(i)}, W^{(i)} \leftarrow \mathsf{S}(L^{(i)}_{\boldsymbol{w}^*} L^{(i)\top}_{\boldsymbol{w}^*})$, where

$$Y^{(i)} = \left(\left(\Omega L^{(i)}_{\boldsymbol{w}^*}\right) L^{(i)\top}_{\boldsymbol{w}^*}\right)^{\top}, \quad W = \left(\Psi L^{(i)}_{\boldsymbol{w}^*}\right) L^{(i)\top}_{\boldsymbol{w}^*}, \quad (10)$$

where $\Omega \in \mathbb{R}^{r \times N}$ and $\Psi \in \mathbb{R}^{s \times N}$ are random sketching matrices, with $T = r + s$ defining the total size of the sketch. Note that following the operation order indicated by the parentheses avoids instantiating any $N \times N$ matrix. The memory complexity of this operation is $O(T(N+d))$. Tropp et al. [2017] suggest splitting the budget to $r = (T-1)/3, s = T - r$, and suggest choosing $T = 6k + 4$ as a minimal value of T for a given k to minimize a high-probability bound on the approximation error of the sketch. We refer the reader to [Tropp et al., 2017] for a discussion of these theoretical results.

Algorithm 1 SCOD Offline

```
Require: Dataset $\mathscr{D} = \{\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}\}_{i=1}^{M}$, DNN architecture $f, \mathscr{P}$, trained weights $\boldsymbol{w}^*$
 1: function SKETCHCURVATURE($f, \mathscr{P}, \boldsymbol{w}^*, \mathscr{D}$)
 2:     Sample $\Omega, \Psi$ as in (11).   ▷ construct sketching map
 3:     $Y, W \leftarrow 0, 0$   ▷ initialize sketch
 4:     for $(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$ in $\mathscr{D}$ do
 5:         $\boldsymbol{\theta}^{(i)} \leftarrow f(\boldsymbol{x}^{(i)}, \boldsymbol{w}^*)$   ▷ forward pass
 6:         Compute $L^{(i)}_{\boldsymbol{w}^*}$ from $\boldsymbol{\theta}^{(i)}$   ▷ $d$ backward passes
 7:         $Y \leftarrow Y + \frac{1}{M}\left(\left(\Omega L^{(i)}_{\boldsymbol{w}^*}\right) L^{(i)\top}_{\boldsymbol{w}^*}\right)^{\top}$   ▷ update sketch
 8:         $W \leftarrow W + \frac{1}{M}\left(\Psi L^{(i)}_{\boldsymbol{w}^*}\right) L^{(i)\top}_{\boldsymbol{w}^*}$
 9:     end for
10:     $U_{\text{top}}, \boldsymbol{\lambda}_{\text{top}} \leftarrow$ FIXEDRANKSYM($\Omega, \Psi, Y, W$)
11:     return $U_{\text{top}}, \boldsymbol{\lambda}_{\text{top}}$
12: end function
```

These sketching matrices can be as simple as matrices with i.i.d. standard Gaussian entries. However, to further reduce the memory and computation overhead of sketching, in this work, we use Subsampled Randomized Fourier Transform (SRFT) sketching matrices [Woolfe et al., 2008]

$$\Omega = P_1 F_N \mathsf{diag}(\boldsymbol{d}_1), \qquad \Psi = P_2 F_N \mathsf{diag}(\boldsymbol{d}_2), \qquad (11)$$

where $\boldsymbol{d}_1, \boldsymbol{d}_2 \in \mathbb{R}^N$ are vectors with entries sampled from independent Rademacher random variables[4], F_N is the linear operator which applies the discrete cosine transform on each N-length column, and P_1, P_2 are matrices which each select a random subset of r and s rows respectively. The SRFT sketching matrices offer similar approximation performance when compared to Gaussian matrices, but adding only a $(T + 2N)$ parameter overhead, as opposed to TN of the Gaussian case [Tropp et al., 2017].

Armed with these tools, we can now follow the procedure detailed in Algorithm 1 to produce a low-rank approximation of the dataset Fisher. Our approach uses one pass over the dataset $\mathscr{D}$, incrementally building the sum in (9) by applying the SRFT sketching matrices to the $L^{(i)}_{\boldsymbol{w}^*}$, the factor for the Fisher for a single input, which can be computed with d backward passes for each input. Having constructed the sketch, we can use this much lower-dimensional, $T \times N$ representation of the dataset Fisher to extract a low-rank, diagonalized representation $F^{\mathscr{D}}_{\boldsymbol{w}^*} = U_{\text{top}} \mathsf{diag}(\boldsymbol{\lambda}_{\text{top}}) U_{\text{top}}^{\top}$ by following the `FixedRankSymmetric` algorithm detailed in Tropp et al. [2017].

[4]A Rademacher random variable has a value of $+1$ or -1 with equal probability.

5 RELATED WORK

There is a large body of work on characterizing epistemic uncertainty in DNNs. Specific to softmax-based classification models, there has been some effort in using the predictive uncertainty directly as a measure of epistemic uncertainty for OoD detection [Hendrycks and Gimpel, 2018], which can be improved through temperature scaling and input preprocessing [Liang et al., 2018]. Bayesian approaches often employ Monte-Carlo [Neal, 2012, Gal and Ghahramani, 2015] or variational [Graves, 2011, Blundell et al., 2015, Liu and Wang, 2016] methods, but these are not generally applicable to pre-trained networks. In contrast, the Laplace approximation to the Bayesian posterior [MacKay, 1992] is appealing as it can be applied to any pre-trained DNN, yet requires estimating the network curvature.

The most closely-related approaches to ours are the post-training uncertainty methods of Madras et al. [2019] and Ritter et al. [2018]. The main differences between such approaches are (1) how to approximate the curvature (Hessian/Fisher) and (2) how to propagate uncertainty at test time. The true Bayesian posterior is based on the log-likelihood, and a second-order approximation of the log-likelihood would involve the Hessian (the Laplace approximation). However, for DNNs, we often do not optimize to convergence, and thus, the Hessian is not guaranteed to be PSD. Madras et al. [2019] consider only the principal components, but the computation still does not scale well to large datasets. Moreover, stochastic versions of Lanczos algorithm [Lanczos, 1950] or power iteration are difficult to tune. An alternative is to consider the Fisher (or the Gauss–Newton [Botev et al., 2017]) approximation of the full Hessian. For certain cases, such as for the exponential family distributions and piecewise linear networks, the Fisher is the same as the Hessian[5]. An advantage of the Fisher is that it is easier to compute, as it only requires the first-order terms. Further, it is guaranteed to be PSD, and thus is often used in second-order optimization.

As mentioned earlier, Fisher is still intractable to compute, store, and invert for large models, and so, various approximation schemes for that have been proposed. Ritter et al. [2018] use a Kronecker-factored representation of the Fisher, which is easy to store and invert, but it requires certain approximations of the expectations of a Kronecker product, and ignores the cross terms between layers to form a block diagonal structure. We do not enforce this block diagonal structure, and instead use matrix sketching to tractably estimate only the top eigenvectors and eigenvalues of the full Fisher. The KFAC approximation (and its derivatives) are suitable for second-order optimization, when curvature is used to scale the gradient step in the training loop (making the block diagonal approximation enables quick computation of an invertible Fisher). However, in our case, computing the dataset Fisher (and its eigen-decomposition) happens only once, offline, and thus, we can afford to take a slower, more memory-intensive approach.

While proposed for a different purpose, there are similar ideas that have been used for the problem of continual learning, i.e., learning different tasks in a sequential fashion. The challenge there is to represent the information of the previous training data in a compact form and to update the weights in such a way that preserves the previous information as much as possible when training for a new task. Elastic weight consolidation (EWC) of Kirkpatrick et al. [2017] uses the Fisher information matrix to weight each parameter based on its "importance" for the previous task, and uses a regularizer that penalizes changing important parameters more. Farajtabar et al. [2020] proposed orthogonal gradient descent (OGD), which represents the information about the previous data in the form of gradients of the predictions, and then updates the weights in a direction orthogonal to those gradients.

6 EXPERIMENTAL RESULTS

We are interested in evaluating how *efficiently* SCOD produces uncertainty estimates, and how *useful* these estimates are. We define the utility of the uncertainty estimate in terms of how well it serves to classify atypical inputs from typical ones. Following the literature on OoD detection, we quantify this utility by computing the area under the ROC and precision-recall curves, to produce the AUROC and AUPR metrics, respectively.

Using these metrics, we explore: (1) how choices like the sketch budget T and rank of approximation k impact performance; (2) how performance of SCOD can be improved on large DNNs; and (3) how SCOD compares to baselines on a suite of problem settings, from regression to classification. The details of all the problem settings and the experimental evaluation are included in Appendix C.1.

6.1 CHOOSING THE SKETCH BUDGET AND RANK OF APPROXIMATION

A key aspect of SCOD is using matrix sketching to approximate the dataset Fisher as a low-rank matrix. This presents the practitioner with two key hyperparameters: the memory budget T to allocate for the sketching, and the rank k of the approximation used in online computation. While memory budget T is generally set by hardware constraints, it impacts the quality of low-rank approximation attainable through sketching. Indeed, the sketching procedure produces a rank $2(T-1)/3$ approximation, and theoretical results suggest

[5]For piecewise linear networks, e.g., those with ReLU activations, the Hessian of $\boldsymbol{\theta}$ with respect to the weights is 0 wherever the network is differentiable, and the nondifferentiable points of such networks form a measure zero set [Singla et al., 2019]

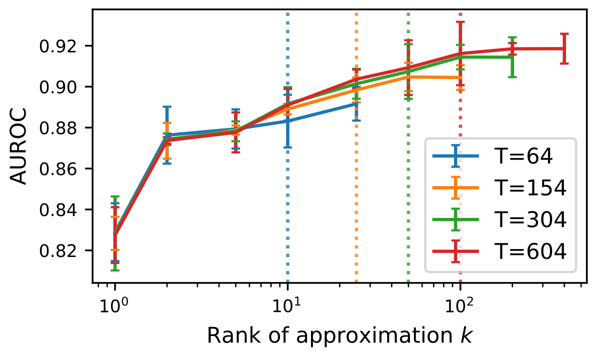


Figure 2: The impact of the sketched approximation on OoD detection (measured by AUROC) for Rotated MNIST. We see that, in general, increasing the rank of the approximation yields a higher AUROC, but with diminishing returns (note the log scale). AUROC is not significantly impacted by the sketch budget T if $k \leq (T-4)/6$, the threshold visualized by the dashed lines.

keeping only the top $(T-4)/6$ eigenvalues and eigenvectors from this approximation [Tropp et al., 2017].

We explore this trade-off empirically on the Rotated MNIST domain. We choose a range of values for the sketching budget T, and sketch the dataset Fisher. For each sketch size, we choose k from a range of values from 1 all the way to the theoretical maximum $2(T-1)/3$, and test the AUROC performance of Unc. Figure 2 shows the results of these experiments. These results show two key trends. First, we see that increasing the rank k improves performance, but with diminishing returns. Second, for a fixed rank k, we see that the performance is insensitive to the sketch budget, especially if $k < (T-4)/6$. For larger k, we start seeing benefits from increasing the sketch budget, consistent with the theory; performance starts to plateau as k increases beyond $(T-4)/6$. Beyond the rank, Unc is also impacted by the scale of the prior ε^2. In our experiments, we found performance to be insensitive to this hyperparameter and thus use $\varepsilon = 1$ in all the experiments. See Appendix C.3 for more details.

6.2 FURTHER IMPROVING EFFICIENCY FOR LARGE DNNS

These results suggest that the best classification performance is achieved by setting T as high as memory allows, and subsequently choosing $k \geq (T-4)/6$. While sketching enables us to avoid the quadratic dependency on N, the memory footprint of the offline stage of SCOD is still linear, i.e., $O(NT)$. As a result, GPU memory constraints can restrict T, and thus k, substantially for large models. To alleviate this issue, we study how performance is impacted if we simply restrict our analysis to the last few layers of the network, thus lowering the effective value of N. For convolutional networks, the first layers tend to learn generic filter patterns which apply across many domains, while later layers learn task-specific representations [Zeiler and Fergus, 2014]. Thus, it is possible that the curvature on the later-layer parameters is more informative from an OoD detection standpoint.

To test the impact of this, we compare to **SCOD (LL)**, an ablation which applies this analysis only to the last layers of the network. For these experiments, we chose to limit analysis to the last L layers, where we chose L for each network architecture such that a minimum of 2 layers were considered, and at least 1 Conv layer was considered. For the larger models in TaxiNet and CIFAR10, we restricted our analysis to the last 15% of the layers. In these latter two domains, we also increased the sketch budget T and associated rank k up to the capacity of our GPU. Full details of the setup are included in Appendix C.1. Results are included in Figure 3. For an in-depth look into the impact of focusing on the last layer, see Appendix C.4.

6.3 PERFORMANCE RELATIVE TO BASELINES

We compare SCOD against several baselines. First, we compare with the two closely-related approaches, namely, **Local Ensembles** [Madras et al., 2019] and **KFAC Laplace** approximation [Ritter et al., 2018], which use curvature estimation to augment a trained model with uncertainty estimates. Next, while *not* a method applicable to a pre-trained model, we compare against **Deep Ensembles** [Lakshminarayanan et al., 2017] as a benchmark, as it has shown strong OoD detection performance across regression and classification examples. For this baseline, we retrain $K = 5$ models of the same architecture from different initializations. Both KFAC Laplace and Deep Ensembles output mixture distributions, which we turn into a scalar uncertainty measure by computing the total variance (summed over output dimension) for regression problems, or the predictive entropy for classification, as in [Lakshminarayanan et al., 2017]. Finally, we compare to a **Naive** baseline which produces an uncertainty measure directly from the output of the pre-trained model. For regression models which output only a mean estimate, extracting an uncertainty estimate is not possible, and thus this baseline outputs a constant signal $\text{Unc}(x) = 1$. For classification models, we use the entropy of the output distribution as a measure of uncertainty, as in [Lakshminarayanan et al., 2017].

We compare the performance of these baselines across three regression domains (Wine, Rotated MNIST, and TaxiNet), as well as three classification domains (Binary MNIST, MNIST, and CIFAR10). For each domain, we create a dataset known to be semantically different from the training data. Where possible, we consider OoD data that is realistic — for example, data from white wines as OoD for a model trained on data form red wines; or in TaxiNet, images from

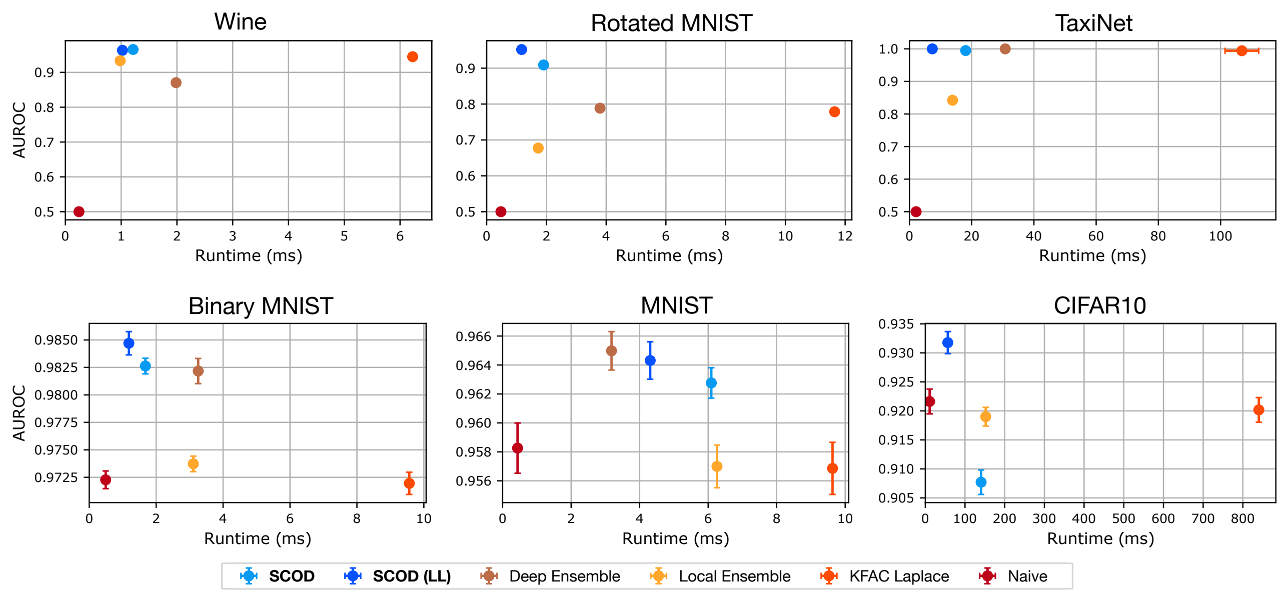


Figure 3: Comparing OoD detection performance and runtime against baselines. We see that SCOD is consistently on a Pareto frontier in terms of the runtime/efficacy tradeoff among post-hoc uncertainty methods (top-left is better). Indeed, SCOD, applied onto a pre-trained model, often matches or exceeds the performance of Deep Ensembles, which is not a post-training approach, and requires retraining multiple models from scratch. Note that the y-axes are independently scaled.

a wing-mounted camera in different times of day and different weather conditions for a model trained only on data from the morning with clear weather conditions. We also consider using the model's own accuracy to label points as in- or out-of-distribution in Appendix C.5. For all domains except CIFAR10, we train all networks, and then apply the post-training algorithms. For the CIFAR10 domain, we test on a pre-trained DenseNet121 model [Huang et al., 2019, Phan, 2021] to highlight the fact that SCOD can be applied to augment any pre-trained model, independent of training methodology, with uncertainty estimates.

The results are summarized in Figure 3. Overall, we see three key trends. First, SCOD consistently provides the most informative uncertainty measures out of the methods applicable on a pre-trained model. In fact, its AUROC often matches or exceeds that of Deep Ensembles. While KFAC Laplace also matches Ensemble performance in many settings, our approach tends to dominate it on a runtime/AUROC Pareto frontier. Computing uncertainty metrics using KFAC Laplace requires the computationally-intensive repeated evaluations of the DNN with different sampled weights. Furthermore, this sampling process is very sensitive to the regularization hyperparameters. In contrast, our uncertainty metric does not require any sampling, and thus adds very little overhead. Local Ensembles similarly add little overhead, but have a worse AUROC on many domains, especially for large models and datasets, e.g., in the TaxiNet domain, where $M = 50,000$. It is likely that sketching the Fisher yields a better low-rank curvature estimate than using stochastic mini-batching to estimate the top eigenspace of the Hessian.

On regression problems, the output of the network provides no estimate of uncertainty, and, unsurprisingly, the Naive baseline performs poorly. In contrast, on classification problems, the Naive strategy of interpreting the predictive uncertainty as epistemic uncertainty works quite well [Hendrycks and Gimpel, 2018]. Nevertheless, we see that, SCOD is generally able to exceed the Naive performance on these classification domains, and match the performance of Deep Ensembles. Importantly, unlike Deep Ensembles or even KFAC Laplace, we do not directly use the base DNN's output uncertainty as part of our score, and only consider how weight perturbations may change the output. Given that the output uncertainty is a strong baseline for softmax classification problems, connecting these ideas is an interesting avenue for future work.

In general, we see that restricting analysis to the last layers often leads to better AUROC performance as well as faster runtime. This is particularly evident on CIFAR10, where without restricting analysis to the last layers, SCOD performed worse than the Naive baseline (although still achieving AUROC > 0.9). We suspect this is due to the first layers representing generic features that are suited to most natural images, and thus, the curvature of the later layers is more useful for OoD detection. This is supported by a case study on the CIFAR10 model, with results in Appendix C.4.

7 CONCLUSION

We presented *Sketching Curvature for OoD Detection (SCOD)*, a scalable, architecture-agnostic framework for equipping any pre-trained DNN with a task-relevant epistemic uncertainty estimate. Through extensive experiments, we demonstrated that the proposed method achieves comparable or better OoD detection performance with a lower computational burden relative to existing approaches.

There are several important avenues for future work, in addition to what was mentioned earlier. Our framework can be potentially extended to continual learning settings, since the sketch can be built incrementally without the need to loop over the previous data. Furthermore, within the context of autonomous systems, another important direction is to tailor OoD detection and uncertainty estimation for the downstream control task.

Acknowledgements

A.S. and M.P. were supported in part by DARPA under the Assured Autonomy program and by NASA under the University Leadership Initiative program. N.A. was supported by MIT. The authors wish to thank Robin Brown and Edward Schmerling for helpful input and discussions during the development of these ideas.

References

Navid Azizan, Sahin Lale, and Babak Hassibi. Stochastic mirror descent on overparameterized nonlinear models: Convergence, implicit regularization, and generalization. *arXiv preprint arXiv:1906.03830*, 2019.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015.

Aleksandar Botev, Hippolyt Ritter, and David Barber. Practical gauss-newton optimisation for deep learning. In *International Conference on Machine Learning*, pages 557–565. PMLR, 2017.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

Mehrdad Farajtabar, Navid Azizan, Alex Mott, and Ang Li. Orthogonal gradient descent for continual learning. In *International Conference on Artificial Intelligence and Statistics*, pages 3762–3773. PMLR, 2020.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *arXiv preprint arXiv:1506.02142*, 2015.

Noah Golmant, Zhewei Yao, Amir Gholami, Michael Mahoney, and Joseph Gonzalez. pytorch-hessian-eigenthings: efficient pytorch hessian eigendecomposition, October 2018. URL `https://github.com/noahgolmant/pytorch-hessian-eigenthings`.

Alex Graves. Practical variational inference for neural networks. In *Advances in neural information processing systems*, pages 2348–2356, 2011.

Dan Hendrycks and Kevin Gimpel. A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks. *arXiv:1610.02136 [cs]*, October 2018. URL `http://arxiv.org/abs/1610.02136`. arXiv: 1610.02136.

Gao Huang, Zhuang Liu, Geoff Pleiss, Laurens Van Der Maaten, and Kilian Weinberger. Convolutional networks with dense connectivity. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

Alex Krizhevsky et al. Learning multiple layers of features from tiny images. 2009.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in neural information processing systems*, pages 6402–6413, 2017.

Cornelius Lanczos. *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*. United States Governm. Press Office Los Angeles, CA, 1950.

Shiyu Liang, Yixuan Li, and R. Srikant. Enhancing The Reliability of Out-of-distribution Image Detection in Neural Networks. *arXiv:1706.02690 [cs, stat]*, February 2018. URL `http://arxiv.org/abs/1706.02690`. arXiv: 1706.02690.

Qiang Liu and Dilin Wang. Stein variational gradient descent: A general purpose bayesian inference algorithm. In *Advances in neural information processing systems*, pages 2378–2386, 2016.

David JC MacKay. A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3): 448–472, 1992.

David Madras, James Atwood, and Alex D'Amour. Detecting Extrapolation with Local Ensembles.

arXiv:1910.09573 [cs, stat], October 2019. URL http://arxiv.org/abs/1910.09573. arXiv: 1910.09573.

James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417. PMLR, 2015.

Radford M Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

Huy Phan. huyvnphan/pytorch_cifar10, January 2021. URL https://doi.org/10.5281/zenodo.4431043.

Hippolyt Ritter, Aleksandar Botev, and D. Barber. A scalable laplace approximation for neural networks. In *ICLR*, 2018.

Levent Sagun, Utku Evci, V Ugur Guney, Yann Dauphin, and Leon Bottou. Empirical analysis of the hessian of over-parametrized neural networks. *arXiv preprint arXiv:1706.04454*, 2017.

Sahil Singla, Eric Wallace, Shi Feng, and Soheil Feizi. Understanding impacts of high-order loss approximations and features in deep learning interpretation. In *International Conference on Machine Learning*, pages 5848–5856. PMLR, 2019.

Joel A. Tropp, Alp Yurtsever, Madeleine Udell, and Volkan Cevher. Practical sketching algorithms for low-rank matrix approximation. *SIAM J. Matrix Anal. & Appl.*, 38 (4):1454–1485, January 2017. ISSN 0895-4798, 1095-7162. doi: 10.1137/17M1111590. URL http://arxiv.org/abs/1609.00048. arXiv: 1609.00048.

Franco Woolfe, Edo Liberty, Vladimir Rokhlin, and Mark Tygert. A fast randomized algorithm for the approximation of matrices. *Applied and Computational Harmonic Analysis*, 25(3):335–366, 2008.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

A COMPUTING PER-INPUT FISHER VIA BACKPROP

We work with the factorization $F^{(t)}_{\boldsymbol{w}^*} = L^{(t)}_{\boldsymbol{w}^*} L^{(t)\top}_{\boldsymbol{w}^*}$ Recall that $L^{(t)}_{\boldsymbol{w}^*} = L^{(t)}_{\boldsymbol{w}^*} = \boldsymbol{J}^{\top}_{f,\boldsymbol{w}} L^{(t)}_{\boldsymbol{\theta}^*}$. Leveraging the linearity of the derivative, we can avoid carrying out this matrix multiplication on $\boldsymbol{J}^{(t)}_{f,\boldsymbol{w}^*} \in \mathbb{R}^{d\times N}$, and instead perform the matrix multiplication prior to computing the Jacobian via backpropagation. Defining the function $\boldsymbol{g}(A, \boldsymbol{x}, \boldsymbol{w}) = Af(\boldsymbol{x}, \boldsymbol{w})$, which applies a linear transformation A to the output of the DNN, we have

$$L^{(t)\top}_{\boldsymbol{w}^*} = \begin{bmatrix} \frac{\partial}{\partial \boldsymbol{w}} g_1(L^{(t)\top}_{\boldsymbol{\theta}^*}, \boldsymbol{x}^{(t)}, \boldsymbol{w}^*) \\ \vdots \\ \frac{\partial}{\partial \boldsymbol{w}} g_d(L^{(t)\top}_{\boldsymbol{\theta}^*}, \boldsymbol{x}^{(t)}, \boldsymbol{w}^*) \end{bmatrix}, \tag{12}$$

where each row can be computed by backpropagation from the corresponding output dimension of $\tilde{\boldsymbol{\theta}}$, but ignoring the gradients that flow through the dependence of $L^{(t)\top}_{\boldsymbol{\theta}^*}$ on $\boldsymbol{w}$.

Alternatively, for models with large output dimensions, exactly computing the Fisher as outlined above can be expensive. For such settings, it is possible to exploit the definition of the Fisher as an expectation over $\boldsymbol{y} \sim P(\boldsymbol{\theta})$, and turn to numerical integration techniques such as Monte-Carlo estimation. In our experiments, we use the exact Fisher in both the offline and online phases.

Below, we provide analytic forms of $L^{(t)}_{\boldsymbol{\theta}^*}$ for common parameteric distributions:

- **Fixed Diagonal Variance Gaussian.** If $\mathscr{P}(\boldsymbol{\theta}) = \mathscr{N}(\boldsymbol{\theta}, \mathsf{diag}(\boldsymbol{\sigma}))$, then
$$L^{(t)}_{\boldsymbol{\theta}^*} = \mathsf{diag}(\boldsymbol{\sigma})^{-1/2}.$$
- **Bernoulli with Logit Parameter.** If the output distribution is chosen to be a Bernoulli parameterized by the logit $\boldsymbol{\theta} \in \mathbb{R}$, such that the probability of a positive outcome is $p = 1/(1+\exp(-\boldsymbol{\theta}^*))$, then, $F^{(t)}_{\boldsymbol{\theta}^*} = p(1-p)$. So,
$$L^{(t)}_{\boldsymbol{\theta}^*} = \sqrt{p(1-p)}.$$
- **Categorical with Logits.** If the output distribution is a categorical one parameterized by the logits $\boldsymbol{\theta} \in \mathbb{R}^d$, such that $p(y=k) = \frac{\exp(\boldsymbol{\theta}_k)}{\sum_{j=1}^{d}\exp(\boldsymbol{\theta}_j)}$, then,
$$L^{(t)}_{\boldsymbol{\theta}^*} = \mathsf{diag}(\boldsymbol{p})^{1/2}(I_d - \mathbf{1}_d \boldsymbol{p}^{\top}),$$
where $\boldsymbol{p}$ is the vector of class probabilities according to $\boldsymbol{\theta}^*$.

B DERIVATION OF THE SIMPLIFIED UNCERTAINTY METRIC

If we substitute the eigenvalue decomposition of the dataset Fisher $F^{\mathscr{D}}_{\boldsymbol{w}^*} = U\mathsf{diag}(\boldsymbol{\lambda})U^T$ into the expression for the posterior covariance (5), and apply the Woodbury identity, we obtain

$$\Sigma^* = \varepsilon^2 \left(I - U\mathsf{diag}\left(\frac{\boldsymbol{\lambda}}{\boldsymbol{\lambda} + 1/(2M\varepsilon^2)}\right) U^{\top} \right), \tag{13}$$

where the operations in the diagonal are applied element-wise.

Now, if we plug this expression into our expression for Unc, we obtain

$$\begin{aligned} &\mathrm{Unc}(\boldsymbol{x}^{(t)}) \\ &= \mathrm{Tr}\left(\varepsilon^2 \left(I - U\mathsf{diag}\left(\frac{\boldsymbol{\lambda}}{\boldsymbol{\lambda}+1/(2M\varepsilon^2)}\right)U^{\top}\right) F^{(t)}_{\boldsymbol{w}^*}\right) && (14) \\ &= \varepsilon^2 \,\mathrm{Tr}\left(F^{(t)}_{\boldsymbol{w}^*}\right) - \varepsilon^2 \,\mathrm{Tr}\left(U\mathsf{diag}\left(\frac{\boldsymbol{\lambda}}{\boldsymbol{\lambda}+1/(2M\varepsilon^2)}\right)U^{\top}F^{(t)}_{\boldsymbol{w}^*}\right) && (15) \\ &= \varepsilon^2 \left\|L^{(t)}_{\boldsymbol{w}^*}\right\|^2_{\mathrm{F}} - \varepsilon^2 \,\mathrm{Tr}\left(L^{(t)\top}_{\boldsymbol{w}^*} U\mathsf{diag}\left(\frac{\boldsymbol{\lambda}}{\boldsymbol{\lambda}+1/(2M\varepsilon^2)}\right)U^{\top}L^{(t)}_{\boldsymbol{w}^*}\right) && (16) \\ &= \varepsilon^2 \left\|L^{(t)}_{\boldsymbol{w}^*}\right\|^2_{\mathrm{F}} - \varepsilon^2 \left\|\mathsf{diag}\left(\sqrt{\frac{\boldsymbol{\lambda}}{\boldsymbol{\lambda}+1/(2M\varepsilon^2)}}\right)U^{\top}L^{(t)}_{\boldsymbol{w}^*}\right\|^2_{\mathrm{F}}, && (17) \end{aligned}$$

where we make use of the cyclic property of the trace and the fact that $\|A\|_F = \sqrt{\mathrm{Tr}(AA^{\top})}$.

B.1 LOW-RANK APPROXIMATION AND ERROR BOUNDS

We notice that the elements of the diagonal $\frac{\lambda_j}{\lambda_j + 1/2M\varepsilon^2}$ tend to 1 for $\lambda_i >> 1/2M\varepsilon^2$, and 0 for $\lambda_i << 1/2M\varepsilon^2$. Therefore, only the top eigenvectors of the dataset Fisher are relevant to this posterior. We see that assuming a fixed spectral decay rate of $F^{\mathscr{D}}_{\boldsymbol{w}^*}$, more eigenvectors are relevant if we choose ε^2 to be large (wider prior weights), or M to be large (more data points collected). The number of eigenvectors we keep influences memory and compute requirements. So, alternatively, we can choose a fixed rank of approximation k, and then choose ε^2 as appropriate.

Thus, we see that the dataset Fisher characterizes how the weights of the DNN are determined by the dataset. In fact, we see that this posterior distribution on the weights has a wide variance in all directions expect in the directions of the top eigenvectors of $F^{\mathscr{D}}_{\boldsymbol{w}^*}$, for which $\lambda_j / \left(\lambda_j + (2M\varepsilon^2)^{-1}\right)$ is non-negligible.

We can characterize the error made by keeping only the top k eigenvalues and eigenvectors. Let $\boldsymbol{\lambda}^{\top} = [\boldsymbol{\lambda}^{\top}_{\mathrm{top}}, \boldsymbol{\lambda}^{\top}_{\mathrm{bot}}], U = [U_{\mathrm{top}} U_{\mathrm{bottom}}]$, where top selects the top k eigenvalues. If we

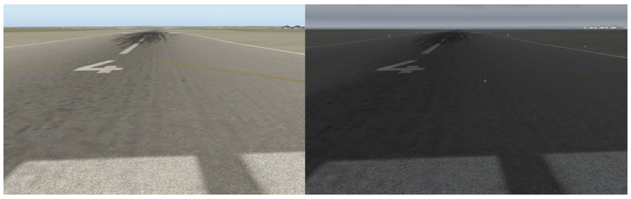


Figure 4: Example input images from the TaxiNet domain, in clear morning (left), and cloudy evening (right) conditions.

define $\tilde{\text{Unc}}(\boldsymbol{x})$ as using the low-rank approximation, we have

$$\tilde{\text{Unc}}(\boldsymbol{x}^{(t)}) = \varepsilon^2 \left\| L_{\boldsymbol{w}^*}^{(t)} \right\|_{\mathrm{F}}^2 - \varepsilon^2 \left\| \mathsf{diag}\left(\sqrt{\frac{\boldsymbol{\lambda}_{\text{top}}}{\boldsymbol{\lambda}_{\text{top}} + 1/(2M\varepsilon^2)}} \right) U_{\text{top}}^{\mathsf{T}} L_{\boldsymbol{w}^*}^{(t)} \right\|_{\mathrm{F}}^2 .$$

The error in the approximation can then be characterized as

$$\tilde{\text{Unc}}(\boldsymbol{x}^{(t)}) - \text{Unc}(\boldsymbol{x}^{(t)})$$

$$= \varepsilon^2 \left\| \mathsf{diag}\left(\sqrt{\frac{\boldsymbol{\lambda}_{\text{bottom}}}{\boldsymbol{\lambda}_{\text{bottom}} + 1/(2M\varepsilon^2)}} \right) U_{\text{bottom}}^{\mathsf{T}} L_{\boldsymbol{w}^*}^{(t)} \right\|_{\mathrm{F}}^2 \tag{18}$$

$$= \varepsilon^2 \sum_{j=k+1}^{\mathsf{rank}\left(F_{\boldsymbol{w}^*}^{\mathscr{D}}\right)} \frac{\lambda_j}{\lambda_j + 1/(2M\varepsilon^2)} \left\| L_{\boldsymbol{w}^*}^{(t)\mathsf{T}} \boldsymbol{u}_j \right\|_2^2 \tag{19}$$

$$\leq \varepsilon^2 \left\| L_{\boldsymbol{w}^*}^{(t)\mathsf{T}} \right\|_2^2 \sum_{j=k+1}^{\mathsf{rank}\left(F_{\boldsymbol{w}^*}^{\mathscr{D}}\right)} \frac{\lambda_j}{\lambda_j + 1/(2M\varepsilon^2)} \tag{20}$$

$$\leq \varepsilon^2 \left\| L_{\boldsymbol{w}^*}^{(t)} \right\|_F^2 \sum_{j=k+1}^{\mathsf{rank}\left(F_{\boldsymbol{w}^*}^{\mathscr{D}}\right)} \frac{\lambda_j}{\lambda_j + 1/(2M\varepsilon^2)} \tag{21}$$

$$\leq \varepsilon^2 \left\| L_{\boldsymbol{w}^*}^{(t)} \right\|_F^2 \left(\mathsf{rank}\left(F_{\boldsymbol{w}^*}^{\mathscr{D}}\right) - k \right) \frac{\lambda_k}{\lambda_k + 1/(2M\varepsilon^2)} . \tag{22}$$

C FURTHER EXPERIMENTAL DISCUSSION

C.1 EXPERIMENTAL DOMAINS

We evaluate SCOD on several problem settings ranging from classification to regression. Here, we provide implementation details for all the domains. For all results, we report 95% confidence bounds on AUROC and AUPR computed by bootstrapping. We measure performance on a system with an AMD Ryzen 9 3950X 16-core CPU and an NVIDIA GeForce RTX 2070 GPU.

Within regression, we consider the following datasets.

- **Wine** is a dataset from the UCI Machine Learning Repository, in which inputs are various chemical properties of a wine, and labels are a scalar quality score of the wine. We train on the dataset of red wine quality and use the white wine dataset as OoD. The network architecture is a 3-layer fully connected network with ReLU activations, and each hidden layer having 100 units, yielding $N = 11401$. Here $T = 604, k = 100$. for SCOD. For Local Ensembles, we use $k = 20$ using all $M = 1000$ datapoints to compute exact Hessian-vector products.
- **Rotated MNIST**, where the input is an MNIST digit 2, rotated by a certain angle, and the regression target is the rotated angle. We consider two types of OoD data: the digit 2 rotated by angles outside the range seen at train time, as well as inputs that are other digits from MNIST. The model starts with three conv layers with 3×3 filters with a stride of 1. The number of channels in the conv layers is 16,32, and 32, with MaxPooling with a kernel size of 2 between each conv layer. The result is flattened and then processed by a linear layer with hidden dimension of 10 before the linear output layer, yielding a total of $N = 16949$. We use ReLU activations. Here we use $T = 304, k = 50$ for SCOD. For Local Ensembles, we use $k = 50$ eigenvectors and all $M = 5000$ datapoints to compute exact Hessian-vector products.
- **TaxiNet** which is a network architecture designed to process $3 \times 260 \times 300$ RGB input images from a wing-mounted camera and produce estimates of the aircraft's distance in meters from the centerline of the runway as well as its heading in radians relative to the runway, both of which can be used for downstream control during taxiing. It was developed by Boeing as part of the DARPA Assured Autonomy program[6]. The model is based on a ResNet18 backbone, pre-trained on ImageNet, with the last layer replaced with a linear layer to the 2 output dimensions. This yields a total of $N = 11177538$ weights. We fine-tune the network on data collected in the X-Plane 11[7] flight simulator with clear weather and at 9am. Here, we tested against realistic OoD data, by changing weather conditions to cloudy and changing the time-of-day to the afternoon and evening, which change the degree to which shadows impact the scene. Figure 4 visualizes inputs under different conditions. Here $M = 30,000$. SCOD processes all 30,000 datapoints in its sketch in under 30 mins. Here, $T = 46, k = 7$, for SCOD, and for SCOD LL, $T = 124, k = 20$. For Local Ensembles, we use $k = 14$ eigenvectors of the Hessian.

For classification, we consider:

- **BinaryMNIST**. We consider a binary classification problem created by keeping only the digits 0 and 1

[6] `https://www.darpa.mil/program/assured-autonomy`

[7] `https://www.x-plane.com/`

from MNIST. As OoD data, we consider other MNIST digits, as well as FashionMNIST Xiao et al. [2017], a dataset of images of clothing that is compatible with an MNIST architecture. The network architecture we use has a convolutional backbone identical to that in the Rotated MNIST, with the flattened output of the conv layers processed directly by a linear output layer to a scalar output, for a total of $N = 14337$ parameters. This output $\boldsymbol{\theta}$ is interpreted to be pre-sigmoid activation for logistic regression; i.e. the output parametric distribution is a Bernoulli with the probability of success given by sigmoid$(\boldsymbol{\theta})$. Here, we use $T = 304, k = 50$ for SCOD. For Local Ensembles, we use $k = 50$ eigenvectors of the Hessian, computed with exact Hessian-vector products using all $M = 5000$ datapoints.

- **MNIST**. We also consider a categorical classification example formulated on MNIST, this time interpreting the output of the network as logits mapped via a softmax to class probabilities. We train on 5-way classification on the digits 0-4. We use digits 5-9 as OoD data, along with FashionMNIST. The network architecture is identical to that of BinaryMNIST, except here the output $\boldsymbol{\theta} \in \mathbb{R}^5$ represents the logits such that the probability of each class is given by softmax$(\boldsymbol{\theta})$. This yields $N = 15493$. Here, we use $T = 604, k = 100$ for SCOD. For Local Ensembles, we use $k = 100$ eigenvectors.
- **CIFAR10**. To test on larger, more realistic inputs, we consider the CIFAR-10 dataset [Krizhevsky et al., 2009]. Here, unlike the previous experiments, we use a pre-trained DenseNet121 model from Phan [2021], and do not train the model from scratch, to highlight that SCOD can be applied to any pre-trained model. While this model is trained on all 10 classes of CIFAR-10, we use only the first 5, and thus keep only the first 5 outputs as $\boldsymbol{\theta} \in \mathbb{R}^5$, and use them as the pre-softmax logits. This yields a total of $N = 6956426$. We process $M = 5000$ images sampled from the first 5 classes of the train split of CIFAR10, and use the val split as in-distribution examples to test on. For the experiments in the body of the paper, we use as the out-of-distribution dataset TinyImageNet[8], a scaled down version of the ImageNet [Deng et al., 2009] dataset, keeping only 200 classes, and resizing inputs to 32×32 RGB images. In the tests in Appendix C.4, we also consider OoD data from the 5 held-out classes from CIFAR-10, as well as from the Street View House Numbers dataset Netzer et al. [2011] which has a compatible size. Here, we use $T = 76, k = 12$ for SCOD, and $T = 184, k = 30$ for SCOD (LL). For Local Ensembles, we use $k = 20$ eigenvectors.

[8]`http://cs231n.stanford.edu/tiny-imagenet-200.zip`

C.2 BASELINES

We outline details of the implementation of each baseline here:

- **Local Ensembles.** We reimplement this baseline in PyTorch, using the `pytorch-hessian-eigenthings` library [Golmant et al., 2018] to compute the top eigenspace of the Hessian. We implement Local Ensembles using a stochastic minibatch estimator for the Hessian-vector product for all domains except for Wine and the MNIST-based domains, where the models are small enough that using the full dataset to compute the Hessian vector product remains computationally feasible. At test time, we use the prediction gradient to compute the extrapolation score, while for multivariate regression and classification problems, we use the loss gradient, sampling possible outputs, computing the gradient on each, projecting it, and then aggregating by taking the minimum over the resulting scores.
- **KFAC Laplace.** We use the KFAC Laplace implementation found at `https://github.com/DLR-RM/curvature`. We use 30 samples from the posterior prediction to estimate the Fisher, on a batch size of 32. We found that in many cases, choosing default values for the norm and scale hyperparameters would lead to singular matrices making. For each experiment, we performed a coarse sweep over these hyperparameters, and chose the best performing set on a validation set to use for the results. Indeed, especially on the classification examples, we found that we required large values of the norm parameter to obtain accurate predictions, which regularizes the posterior towards a delta distribution centered around $\boldsymbol{w}^*$.
- **Deep Ensemble.** We train $K = 5$ models of identical architecture from random initializations using SGD on the same dataset. In all domains where Deep Ensembles are used, the first member of the ensemble is the same model as in the post-training and Naive approaches.

C.3 SENSITIVITY TO SCALE OF PRIOR

We test the impact of the prior scale ε^2 on the performance of SCOD by performing a sweep on the Rotated MNIST domain, with $T = 604, k = 100$. The results, shown in Figure 5, suggest that this has little impact on the performance, unless it is set to be very small corresponding to very tight prior on the weights. This is unsurprising, as, apart from linearly changing the scale of our metric, the term only enters in the diagonal matrix, and has a significant impact if $1/(2M\varepsilon^2)$ is of comparable magnitude to the eigenvalues of the Fisher. This is rarely the case for the first k eigenvalues of the Fisher, especially when M is large (here, $M = 5000$).

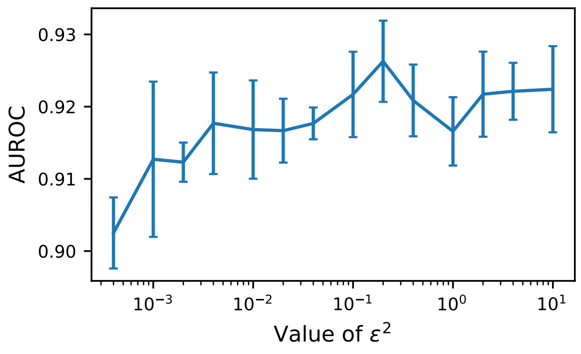


Figure 5: AUROC on RotatedMNIST as a function of prior scale ϵ^2.

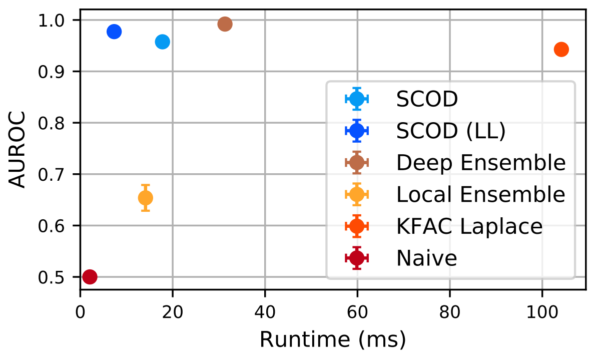


Figure 7: Error-based OoD detection on TaxiNet, where inputs where the DNN made an error greater than 0.5 in Mahalanobis distance are deemed to be out-of-distribution, i.e., out of the DNN's domain of competency.

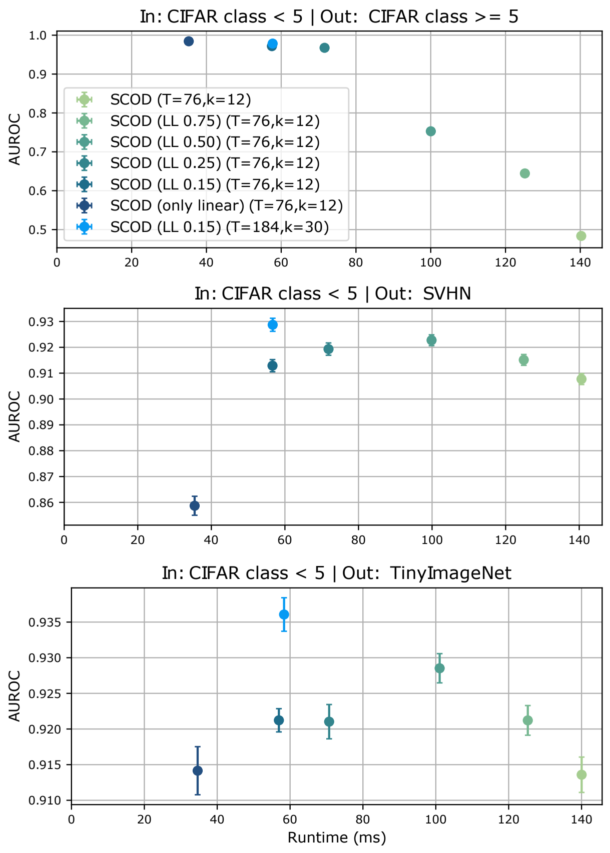


Figure 6: Impact of restricting analysis on the last layers of a network. We see that depending on the OoD dataset, there is a different optimal choice for which layers to consider.

C.4 LIMITING SCOD TO THE LAST LAYERS

To explore the impact of only considering the last layers of a network, we consider restricting SCOD to different subsets of the layers of the DenseNet121 model used in the CIFAR experiments. We hold the sketching parameters constant, and only vary how much of the network we consider. We denote this by the fraction after LL. For example, LL 0.5 means we restrict SCOD to the last 50% of the network. With this notation, LL 1.0 represents considering the full network, which is simply SCOD. At the other extreme, we consider restricting our analysis to just the last linear layer of the model, which we denote "SCOD (only linear)." We consider performance with several different OoD datasets. Figure 6 shows the results. The first OoD dataset is simply the other classes of CIFAR10 which we did not use to create the sketch. We see that here, using just the last linear layer performs best. Indeed, as the network was trained on all 10 classes, it is not surprising that only the last layer analysis provides any meaningful separation between the classes. Including the earlier layers "dilutes" this signal, and, for a fixed sketch budget, considering more layers leads to worse performance. On SVHN and TinyImageNet, we see that the optimal choice of the layers to consider is somewhere in between the two extremes. Here, it is possible that as we increase the fraction of the network that we analyze, we first gain the benefits of the information stored in the last layers, and then start to suffer the consequences of approximation error in the sketching, which depends on the size of the original matrix. We also visualize on this plot the performance of the version of SCOD (LL) used to produce the results in the body of the paper. As is evident, we did not use the results of this sweep to optimally choose the number of layers to consider for the experiments in the body.

C.5 ERROR-BASED OOD DETECTION

In Figure 3, we measure how informative an uncertainty measure is with respect to how well it can classify examples that come from an altogether different dataset as anomalous. However, ideally, we would like this measure of uncertainty to also correspond to the network's own accuracy. To test this, we use the TaxiNet domain as a test case, where we construct a dataset which includes images from all day, morning and afternoon, on a clear day. The training dataset consists only of images from the morning, so this all-day dataset shares some support with the training dataset. Moreover, the differences in these inputs are quite subtle, as the images remain brightly lit, though some shadows appeare in the afternoon. We choose an error threshold, and consider inputs for which the network has an error (measured in Mahalanobis distance) less than this threshold to be "in-distribution" and those where the network has a high error to be "out-of-distribution."

Figure 7 shows the results of this experiment. We see that even in this setup, SCOD produces very high AUROC scores, similar to Deep Ensembles, and outperforms all post-training baselines. In this experiment, we find that applying SCOD to a single pre-trained DNN almost perfectly characterizes the DNNs domain of competency.

Domain	**Method**	**Runtime** (ms)	**AUROC**	**AUPR**
Wine (regression)	**SCOD**	1.214±0.009	0.966±0.002	0.962±0.003
	SCOD (LL)	1.023±0.007	0.963±0.002	0.961±0.002
	Deep Ensemble	1.985±0.008	0.870±0.004	0.896±0.003
	Local Ensemble	0.983±0.007	0.933±0.002	0.931±0.002
	KFAC Laplace	6.226±0.043	0.945±0.002	0.949±0.002
	Naive	0.244±0.002	0.500±0.000	0.750±0.000
Rotated MNIST (regression)	**SCOD**	1.906±0.011	0.909±0.004	0.920±0.003
	SCOD (LL)	1.170±0.026	0.952±0.002	0.940±0.004
	Deep Ensemble	3.794±0.012	0.788±0.004	0.789±0.004
	Local Ensemble	1.726±0.010	0.677±0.005	0.674±0.005
	KFAC Laplace	11.650±0.016	0.779±0.004	0.775±0.005
	Naive	0.477±0.004	0.500±0.000	0.750±0.000
TaxiNet (regression)	**SCOD**	18.074±0.018	0.995±0.000	0.994±0.001
	SCOD (LL)	7.344±0.022	1.000±0.000	1.000±0.000
	Deep Ensemble	30.786±0.026	1.000±0.000	1.000±0.000
	Local Ensemble	13.857±0.024	0.842±0.003	0.820±0.004
	KFAC Laplace	106.839±5.409	0.994±0.000	0.994±0.000
	Naive	2.134±0.007	0.500±0.000	0.750±0.000
Binary MNIST (classification / logistic)	**SCOD**	1.679±0.007	0.983±0.001	0.982±0.001
	SCOD (LL)	1.185±0.008	0.985±0.001	0.985±0.001
	Deep Ensemble	3.258±0.011	0.982±0.001	0.981±0.001
	Local Ensemble	3.111±0.007	0.974±0.001	0.971±0.001
	KFAC Laplace	9.566±0.046	0.972±0.001	0.970±0.001
	Naive	0.492±0.002	0.972±0.001	0.970±0.001
MNIST (classification / softmax	**SCOD**	6.093±0.005	0.963±0.001	0.961±0.001
	SCOD (LL)	4.309±0.008	0.964±0.001	0.966±0.002
	Deep Ensemble	3.179±0.013	0.965±0.001	0.969±0.001
	Local Ensemble	6.258±0.014	0.957±0.001	0.956±0.002
	KFAC Laplace	9.631±0.012	0.957±0.002	0.961±0.002
	Naive	0.436±0.003	0.958±0.002	0.962±0.002
CIFAR10 (classification / softmax)	**SCOD**	140.624±0.330	0.908±0.002	0.864±0.004
	SCOD (LL)	56.677±0.177	0.932±0.002	0.905±0.003
	Local Ensembles	152.154±0.179	0.919±0.002	0.881±0.003
	KFAC Laplace	840.195±1.364	0.920±0.002	0.915±0.002
	Naive	10.929±0.017	0.922±0.002	0.917±0.003

Table 1: Full Numerical Results. For each domain, we apply each method to output an uncertainty score on both in and out-of-distribution inputs. We evaluate the performance in terms of runtime per example, and the area under the ROC curve (AUROC) and the area under the precision-recall curve (AUPR). For the latter two metrics, 95% confidence bounds are produced by bootstrapping.