Sketching Curvature for Efficient Out-of-Distribution Detection for Deep Neural Networks (Supplementary material)

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

[1]Stanford University, Stanford, California, USA,
[2]Massachusetts Institute of Technology, Cambridge, Massachusetts, USA,

A COMPUTING PER-INPUT FISHER VIA BACKPROP

We work with the factorization $F^{(t)}_{\boldsymbol{w}^*} = L^{(t)}_{\boldsymbol{w}^*} L^{(t)\mathsf{T}}_{\boldsymbol{w}^*}$ Recall that $L^{(t)}_{\boldsymbol{w}^*} = L^{(t)}_{\boldsymbol{w}^*} = \boldsymbol{J}^{\mathsf{T}}_{f,\boldsymbol{w}} L^{(t)}_{\boldsymbol{\theta}^*}$. 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

$$L^{(t)\mathsf{T}}_{\boldsymbol{w}^*} = \begin{bmatrix} \frac{\partial}{\partial \boldsymbol{w}} g_1(L^{(t)\mathsf{T}}_{\boldsymbol{\theta}^*}, \boldsymbol{x}^{(t)}, \boldsymbol{w}^*) \\ \vdots \\ \frac{\partial}{\partial \boldsymbol{w}} g_d(L^{(t)\mathsf{T}}_{\boldsymbol{\theta}^*}, \boldsymbol{x}^{(t)}, \boldsymbol{w}^*) \end{bmatrix}, \tag{1}$$

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 $L^{(t)\mathsf{T}}_{\boldsymbol{\theta}^*}$ 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 parametric distributions:

- **Fixed Diagonal Variance Gaussian.** If $\mathcal{P}(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}, \mathsf{diag}(\boldsymbol{\sigma}))$, 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,

$$L^{(t)}_{\boldsymbol{\theta}^*} = \mathsf{diag}(\boldsymbol{p})^{1/2}(I_d - \mathbf{1}_d \boldsymbol{p}^{\mathsf{T}}),$$

  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 $F^{\mathcal{D}}_{\boldsymbol{w}^*} = U\mathsf{diag}(\boldsymbol{\lambda})U^T$ into the expression for the posterior covariance (5), and apply the Woodbury identity, we obtain

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

where the operations in the diagonal are applied elementwise.

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

$$\begin{aligned}
\mathrm{Unc}(\boldsymbol{x}^{(t)}) &= \mathrm{Tr}\left(\epsilon^2 \left(I - U\mathsf{diag}\left(\frac{\boldsymbol{\lambda}}{\boldsymbol{\lambda} + 1/(M\epsilon^2)} \right) U^\mathsf{T} \right) F^{(t)}_{\boldsymbol{w}^*} \right) && (3)\\
&= \epsilon^2 \,\mathrm{Tr}\left(F^{(t)}_{\boldsymbol{w}^*} \right) - \epsilon^2 \,\mathrm{Tr}\left(U\mathsf{diag}\left(\frac{\boldsymbol{\lambda}}{\boldsymbol{\lambda} + 1/(M\epsilon^2)} \right) U^\mathsf{T} F^{(t)}_{\boldsymbol{w}^*} \right) && (4)\\
&= \epsilon^2 \left\| L^{(t)}_{\boldsymbol{w}^*} \right\|^2_{\mathrm{F}} - \epsilon^2 \,\mathrm{Tr}\left(L^{(t)\mathsf{T}}_{\boldsymbol{w}^*} U\mathsf{diag}\left(\frac{\boldsymbol{\lambda}}{\boldsymbol{\lambda} + 1/(M\epsilon^2)} \right) U^\mathsf{T} L^{(t)}_{\boldsymbol{w}^*} \right) && (5)\\
&= \epsilon^2 \left\| L^{(t)}_{\boldsymbol{w}^*} \right\|^2_{\mathrm{F}} - \epsilon^2 \left\| \mathsf{diag}\left(\sqrt{\frac{\boldsymbol{\lambda}}{\boldsymbol{\lambda} + 1/(M\epsilon^2)}} \right) U^\mathsf{T} L^{(t)}_{\boldsymbol{w}^*} \right\|^2_{\mathrm{F}}, && (6)
\end{aligned}$$

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

B.1 LOW-RANK APPROXIMATION AND ERROR BOUNDS

We notice that the elements of the diagonal $\frac{\lambda_j}{\lambda_j + 1/M\epsilon^2}$ tend to 1 for $\lambda_i >> 1/M\epsilon^2$, and 0 for $\lambda_i << 1/M\epsilon^2$. Therefore, only the top eigenvectors of the dataset Fisher are relevant to this posterior. We see that assuming a fixed spectral decay rate of $F^{\mathcal{D}}_{\boldsymbol{w}^*}$, 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 $F^{\mathcal{D}}_{\boldsymbol{w}^*}$, for which $\lambda_j / \left(\lambda_j + (M\epsilon^2)^{-1}\right)$ is non-negligible.

We can characterize the error made by keeping only the top k eigenvalues and eigenvectors. Let $\boldsymbol{\lambda}^\mathsf{T} = [\boldsymbol{\lambda}^\mathsf{T}_{\mathrm{top}}, \boldsymbol{\lambda}^\mathsf{T}_{\mathrm{bot}}], U = [U_{\mathrm{top}} U_{\mathrm{bottom}}]$, where top selects the top k eigenvalues. If we define $\widetilde{\mathrm{Unc}}(\boldsymbol{x})$ as using the low-rank approximation, we have

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

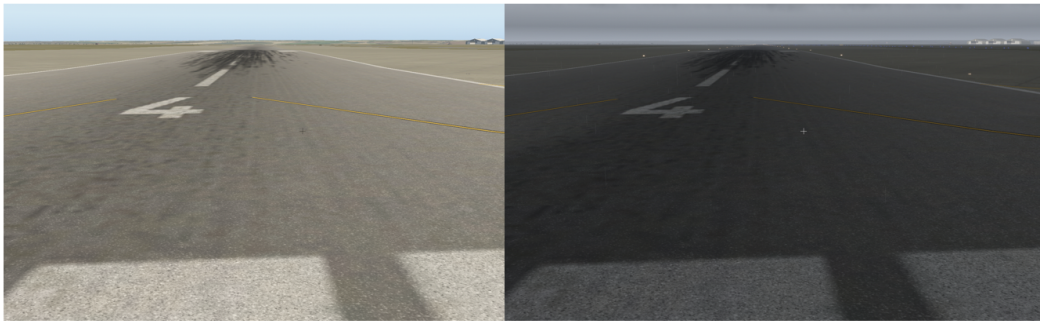

Figure 1: Example input images from the TaxiNet domain, in clear morning (left), and cloudy evening (right) conditions.

The error in the approximation can then be characterized as

$$\widetilde{\mathrm{Unc}}(\boldsymbol{x}^{(t)}) - \mathrm{Unc}(\boldsymbol{x}^{(t)}) = \epsilon^2 \left\| \mathsf{diag}\left(\sqrt{\frac{\boldsymbol{\lambda}_{\mathrm{bottom}}}{\boldsymbol{\lambda}_{\mathrm{bottom}} + 1/(2M\epsilon^2)}} \right) U_{\mathrm{bottom}}^{\mathsf{T}} L_{\boldsymbol{w}^*}^{(t)} \right\|_{\mathrm{F}}^2 \tag{7}$$

$$= \epsilon^2 \sum_{j=k+1}^{\mathsf{rank}\left(F_{\boldsymbol{w}^*}^{\mathcal{D}}\right)} \frac{\lambda_j}{\lambda_j + 1/(M\epsilon^2)} \left\| L_{\boldsymbol{w}^*}^{(t)\mathsf{T}} \boldsymbol{u}_j \right\|_2^2 \tag{8}$$

$$\leq \epsilon^2 \left\| L_{\boldsymbol{w}^*}^{(t)\mathsf{T}} \right\|_2^2 \sum_{j=k+1}^{\mathsf{rank}\left(F_{\boldsymbol{w}^*}^{\mathcal{D}}\right)} \frac{\lambda_j}{\lambda_j + 1/(M\epsilon^2)} \tag{9}$$

$$\leq \epsilon^2 \left\| L_{\boldsymbol{w}^*}^{(t)} \right\|_F^2 \sum_{j=k+1}^{\mathsf{rank}\left(F_{\boldsymbol{w}^*}^{\mathcal{D}}\right)} \frac{\lambda_j}{\lambda_j + 1/(M\epsilon^2)} \tag{10}$$

$$\leq \epsilon^2 \left\| L_{\boldsymbol{w}^*}^{(t)} \right\|_F^2 \left(\mathsf{rank}\left(F_{\boldsymbol{w}^*}^{\mathcal{D}}\right) - k\right) \frac{\lambda_k}{\lambda_k + 1/(M\epsilon^2)} \tag{11}$$

$$\leq \epsilon^2 \left\| L_{\boldsymbol{w}^*}^{(t)} \right\|_F^2 (\min(N, Md) - k) \ln\left(1 + M\epsilon^2 \lambda_k\right) \tag{12}$$

where the last step uses the identity that $\ln(1+x) \geq \frac{x}{x+1} \forall x > -1$.

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[1]. 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[2] 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 1 visualizes inputs under different conditions. Here $M = 30,000$. SCOD processes all 30,000 datapoints in its sketch in under 30 minutes. 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 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 $\mathrm{sigmoid}(\boldsymbol{\theta})$. 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 $\mathrm{softmax}(\boldsymbol{\theta})$. 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, 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 a random selection of data from three datasets:
  1. TinyImageNet[3], a scaled down version of the ImageNet [Deng et al., 2009] dataset, keeping only 200 classes, and resizing inputs to 32×32 RGB images.
  2. LSUN [Yu et al., 2015], scaled and cropped to match the size of CIFAR 10 images.
  3. Street View House Numbers (SVHN), pictures of digits from house numbers cropped to the same size [Netzer et al., 2011].

  The hyperparameters for this experiment are $T = 76, k = 12$ for SCOD, and $T = 184, k = 30$ for SCOD (LL). For Local Ensembles, we use $k = 20$ eigenvectors. In the tests in Appendix C.4, we also consider OoD data from the 5 held-out classes from CIFAR-10, and investigate each OoD dataset independently.

C.2 BASELINES

We outline details of the implementation of each baseline here:

[1] `https://www.darpa.mil/program/assured-autonomy`
[2] `https://www.x-plane.com/`
[3] `http://cs231n.stanford.edu/tiny-imagenet-200.zip`

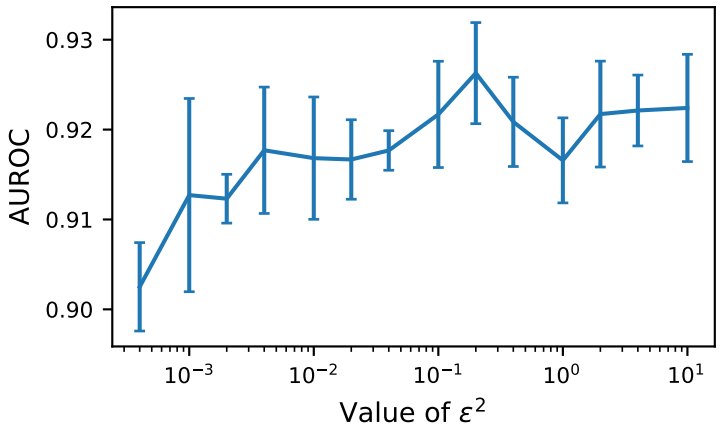

Figure 2: AUROC on RotatedMNIST as a function of prior scale ϵ^2.

- **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 2, 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 $1/(M\epsilon^2)$ 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, $M = 5000$).

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 3 shows the results. The first

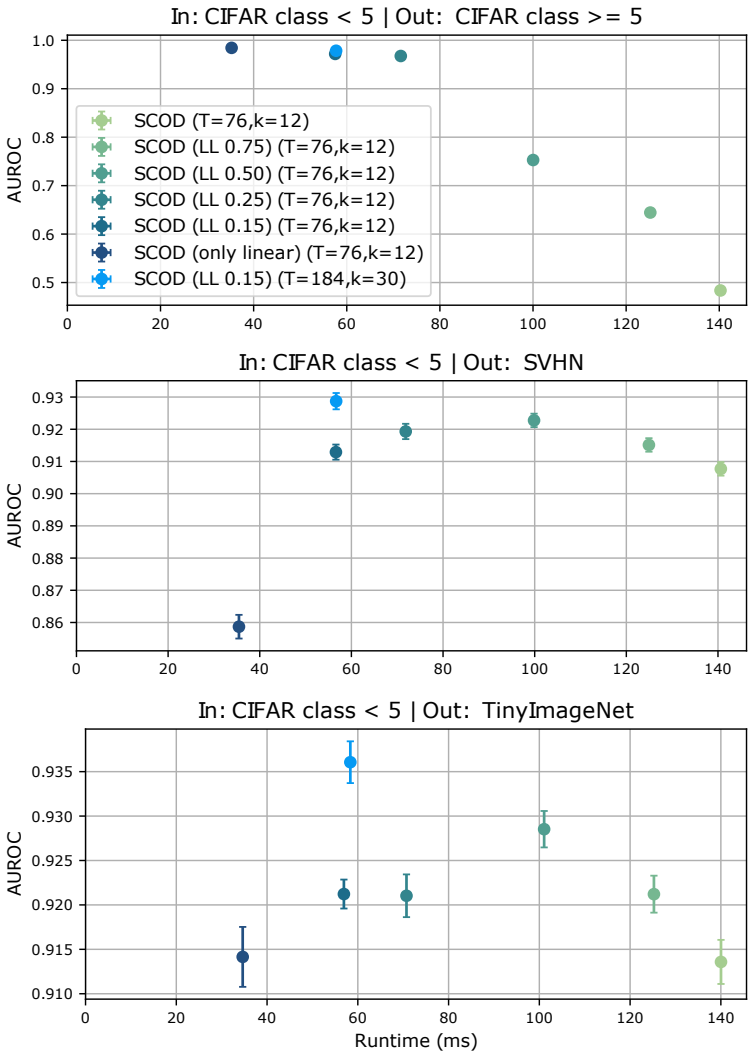

Figure 3: Impact of restricting analysis on the last layers of a network. We see that depending on the OoD dataset, there is a different optimal choice for which layers to consider.

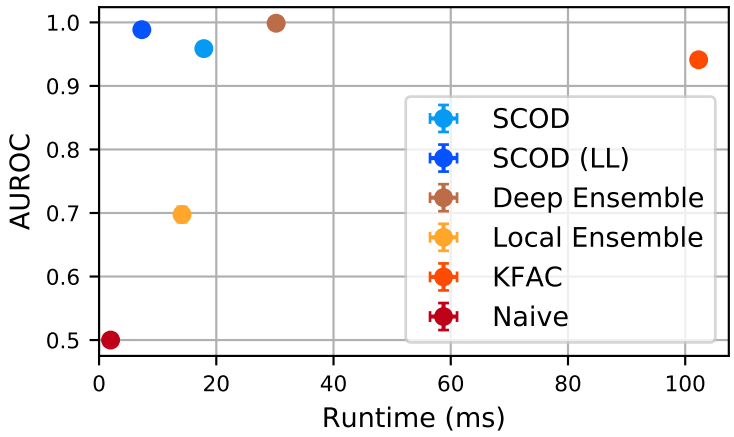

Figure 4: Error-based OoD detection on TaxiNet, where inputs where the DNN made an error greater than 0.5 in Mahalanobis distance are deemed to be out-of-distribution, i.e., out of the DNN's domain of competency.

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 appear 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 4 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. On this domain, we find that applying SCOD to a single pre-trained DNN almost perfectly characterizes the DNNs domain of competency.

References

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.

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 Krizhevsky. Learning multiple layers of features from tiny images. *University of Toronto*, 2009.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *Advances in Neural Information Processing Systems*, 2011.

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

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Fisher Yu, Ari Seff, Yinda Zhang, Shuran Song, Thomas Funkhouser, and Jianxiong Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015.

Domain	Method	**Runtime** (ms)	AUROC	AUPR
Wine (regression)	**SCOD**	1.263±0.009	0.968±0.002	0.967±0.002
	SCOD (LL)	1.033±0.006	0.970±0.001	0.969±0.001
	Deep Ensemble	2.045±0.005	0.881±0.004	0.907±0.003
	Local Ensemble	0.997±0.007	0.958±0.002	0.955±0.003
	KFAC Laplace	6.054±0.019	0.957±0.002	0.955±0.002
	Naive	0.244±0.002	0.500±0.000	0.750±0.000
Rotated MNIST (regression)	**SCOD**	1.971±0.011	0.892±0.003	0.896±0.003
	SCOD (LL)	1.202±0.007	0.942±0.002	0.939±0.002
	Deep Ensemble	3.764±0.008	0.721±0.005	0.753±0.005
	Local Ensemble	1.741±0.011	0.601±0.005	0.528±0.004
	KFAC Laplace	10.934±0.019	0.704±0.005	0.667±0.005
	Naive	0.475±0.004	0.500±0.000	0.750±0.000
TaxiNet (regression)	**SCOD**	17.857±0.020	0.994±0.000	0.993±0.000
	SCOD (LL)	7.300±0.009	1.000±0.000	1.000±0.000
	Deep Ensemble	29.960±0.018	1.000±0.000	1.000±0.000
	Local Ensemble	13.735±0.025	0.912±0.003	0.925±0.002
	KFAC Laplace	103.629±2.466	0.993±0.000	0.993±0.000
	Naive	2.040±0.006	0.500±0.000	0.750±0.000
Binary MNIST (classification / logistic)	**SCOD**	1.752±0.011	0.984±0.001	0.983±0.001
	SCOD (LL)	1.266±0.008	0.984±0.001	0.983±0.001
	Deep Ensemble	3.171±0.009	0.981±0.001	0.981±0.001
	Local Ensemble	3.109±0.009	0.979±0.001	0.977±0.001
	KFAC Laplace	9.346±0.069	0.976±0.001	0.975±0.002
	Naive	0.463±0.003	0.976±0.001	0.974±0.002
MNIST (classification / softmax	**SCOD**	6.041±0.007	0.963±0.002	0.964±0.002
	SCOD (LL)	4.341±0.009	0.962±0.002	0.963±0.002
	Deep Ensemble	3.103±0.010	0.966±0.001	0.970±0.001
	Local Ensemble	6.303±0.009	0.949±0.002	0.946±0.003
	KFAC Laplace	9.592±0.016	0.957±0.002	0.960±0.002
	Naive	0.458±0.003	0.957±0.002	0.961±0.002
CIFAR10 (classification / softmax)	**SCOD**	141.874±0.249	0.927±0.002	0.872±0.005
	SCOD (LL)	57.993±0.100	0.950±0.002	0.930±0.003
	Local Ensembles	148.347±0.253	0.911±0.003	0.841±0.005
	KFAC Laplace	836.307±0.779	0.929±0.003	0.912±0.005
	Naive	10.235±0.016	0.928±0.003	0.912±0.004

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.