

This is a pre-print.
Aside from formatting differences, it is identical to the version published
in Springer Lecture Notes for Computer Science Vol. 13886

Out-of-Distribution Detection for Adaptive Computer Vision

Simon Kristoffersson Lind¹ – simon.kristoffersson_lind@cs.lth.se,
Rudolph Triebel^{3,4} – rudolph.triebel@dlr.de,
Luigi Nardi^{1,2} – luigi.nardi@cs.lth.se, and
Volker Krueger¹ – volker.krueger@cs.lth.se

¹ Lund University LTH

² Stanford University

³ German Aerospace Center DLR

⁴ Technical University of Munich

Corresponding Author: Simon Kristoffersson Lind

Abstract. It is well known that computer vision can be unreliable when faced with previously unseen imaging conditions. This paper proposes a method to adapt camera parameters according to a normalizing flow-based out-of-distribution detector. A small-scale study is conducted which shows that adapting camera parameters according to this out-of-distribution detector leads to an average increase of 3 to 4 percentage points in mAP, mAR and F1 performance metrics of a YOLOv4 object detector. As a secondary result, this paper also shows that it is possible to train a normalizing flow model for out-of-distribution detection on the COCO dataset, which is larger and more diverse than most benchmarks for out-of-distribution detectors.

Keywords: Autonomous Systems · Out-of-Distribution Detection · Normalizing Flows · Object Detection.

1 Introduction

In the past decade computer vision has become a de facto component in autonomous systems. However, it is well known that vision can be unreliable when faced with new, previously unseen situations. Uncertainty is especially abundant in the field of robotics, where the autonomous agents are expected to perform tasks in real-world scenarios that often differ significantly from any training data.

There is an abundance of research to suggest that *Out-of-Distribution* (OOD) detection can help improve the reliability of vision algorithms [6]. Relevant literature reasons that OOD detection allows systems to take action when uncertainty

is encountered [6]. However, there is little research to show that OOD detection helps the reliability of autonomous systems in practice. Most literature on OOD detection tends to focus either on synthetic benchmarks, or controlled real-world scenarios where in- and out-of-distribution examples are very distinct. Neither of these scenarios are reflective of the diverse situations that may be encountered by autonomous systems in practice [6].

OOD detectors are commonly used by setting a threshold OOD score [9], and simply discarding any input that falls beyond this threshold. While this makes intuitive sense, on the basis that vision systems tend to be unreliable when faced with OOD data, it only utilises one end of the OOD scores. We hypothesize that a machine learning model will on average perform more reliably when an input has a lower OOD score. Therefore we propose to use an OOD detector as a quality metric, or un-normalized confidence; the lower the OOD score, the more certain we can be about our vision system’s output.

1.1 Our Contributions

- We expand the current knowledge on OOD detection by showing that it is possible to train normalizing flow models for OOD detection on large, diverse datasets while getting sensible results.
- We introduce a novel method for utilising OOD detection as a quality metric in autonomous systems and show that it can lead to reliability improvements for vision tasks.

2 Related Work

There has been some previous work investigating the uses of OOD detection in autonomous systems. For example Wellhausen *et al.* [17] perform anomaly detection on image data collected from a robot in real-world terrain. However, their main purpose is the evaluation of different OOD detection algorithms, and not an application of OOD detection to real-world operation.

Yuhas *et al.* [20] evaluate OOD detection as an emergency breaking system for an autonomous car, though not in real-world operation, but on a custom test track.

McAllister *et al.* [12] perform real-world crash avoidance experiments with an autonomous car. However, their method does not perform OOD detection directly. Instead they use a variational autoencoder to generate in-distribution samples from an out-of-distribution image, which they use as a measure of uncertainty.

Additionally, there is an abundance of work on OOD detection where the authors have taken care to ensure that in- and out-distributions are disjoint, for example [4, 7, 10, 18, 15, 13]. These works either use entirely different datasets as in- and out-distributions, or separate classes as in- and out-distributions. It is our belief that neither of these scenarios are reflective of real-world operation.

3 Background

3.1 Out-of-Distribution Detection

Conceptually, out-of-distribution detection is simple. First an in-distribution is defined, for example the training data for a machine learning-based vision system. Then, everything that deviates from said in-distribution is said to be out-of-distribution [19].

More formally, we assume that our vision system operates on data that is sampled I.I.D from some distribution $P_{train}(\mathbf{x})$. In practice, however, the system may encounter data that comes from a different distribution, in which case the system will produce unreliable results. Therefore, it is desirable to detect when data lies outside of the original distribution $P_{train}(\mathbf{x})$.

3.2 Normalizing Flows

This section aims to provide a basic introduction to normalizing flows. For more details we refer to Papamakarios *et al.* [14].

Normalizing flows have emerged in the last decade as a way to model complex probability distributions [14]. Let us assume that we want to evaluate or sample from a distribution $p_x(\mathbf{x})$, $\mathbf{x} \in \mathbb{R}^D$, and that p_x is intractable to evaluate or sample from. A normalizing flow can allow us to evaluate or sample from p_x by transforming $p_x(\mathbf{x})$ into a tractable distribution $p_u(\mathbf{u})$, $\mathbf{u} \in \mathbb{R}^D$.

In order to model $p_x(\mathbf{x})$, an invertible transformation T is constructed such that

$$\mathbf{x} = T(\mathbf{u}), \quad \mathbf{u} = T^{-1}(\mathbf{x})$$

where $\mathbf{u} \sim p_u(\mathbf{u})$. Here, we are free to define $p_u(\mathbf{u})$, for example $\mathbf{u} \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$.

By additionally requiring both T and T^{-1} to be differentiable, it is possible to recover $p_x(\mathbf{x})$ from $p_u(\mathbf{u})$ [16, 3, 14]:

$$p_x(\mathbf{x}) = p_u(\mathbf{u}) |\det J_T(\mathbf{u})|^{-1} = p_u(T^{-1}(\mathbf{x})) |\det J_{T^{-1}}(\mathbf{x})| \quad (1)$$

where J_T and $J_{T^{-1}}$ are the Jacobians of T and T^{-1} respectively.

This type of transformation is called a *diffeomorphism*. It has been shown that it is always possible to construct a diffeomorphism T under reasonable assumptions about $p_x(\mathbf{x})$ and $p_u(\mathbf{u})$ [14]. However, analytically constructing T is often infeasible in practice. Therefore, it is desirable to learn an approximation of T . Henceforth we will denote $T(\mathbf{x}; \Theta)$ as an approximation of T with learnable parameters Θ .

An important property is that the composition of two diffeomorphisms $T_2 \circ T_1$ is also a diffeomorphism [14]:

$$(T_2 \circ T_1)^{-1} = T_1^{-1} \circ T_2^{-1}$$

$$\det J_{T_2 \circ T_1}(\mathbf{u}) = \det J_{T_2}(T_1(\mathbf{u})) \cdot \det J_{T_1}(\mathbf{u})$$

This property opens up for many ways to construct powerful approximations $T(\mathbf{x}; \Theta)$. In this work, we focus on a particular model of so-called *affine coupling layers* [14].

Affine coupling layers first split the input \mathbf{x} into two parts [14]:

$$\mathbf{x} = [x_i \mid i \in \mathcal{I}], \quad \mathcal{I} = 1, 2, \dots, D$$

$$\mathbf{x}_1 = [x_i \mid i \in \mathcal{A}], \quad \mathbf{x}_2 = [x_i \mid i \in \mathcal{B}], \quad \mathcal{A} \cup \mathcal{B} = \mathcal{I}, \quad \mathcal{A} \cap \mathcal{B} = \emptyset.$$

While \mathcal{A} and \mathcal{B} can be chosen arbitrarily, the most common choice is to simply split \mathbf{x} into two halves:

$$\mathcal{A} = 1, 2, \dots, \frac{D}{2}, \quad \mathcal{B} = \frac{D}{2} + 1, \frac{D}{2} + 2, \dots, D.$$

Then, \mathbf{x}_1 is transformed as a function of \mathbf{x}_2 , while \mathbf{x}_2 is left as-is:

$$\mathbf{x}'_1 = [x'_i \mid i \in \mathcal{A}] = \boldsymbol{\alpha} \cdot \mathbf{x}_1 + \boldsymbol{\beta}, \\ \text{where } \boldsymbol{\alpha} = \exp(F_1(\mathbf{x}_2; \Theta_1)), \quad \boldsymbol{\beta} = F_2(\mathbf{x}_2, \Theta_2).$$

Here, \cdot denotes element-wise multiplication, and F_1, F_2 are arbitrary functions, for example neural networks. The output of the coupling layer is then simply the reconstruction of \mathbf{x} from \mathbf{x}'_1 and \mathbf{x}_2 :

$$T_{coupling}(\mathbf{x}; \Theta_1, \Theta_2) = \mathbf{x}' = [x'_i \mid i \in \mathcal{I}], \quad \text{where } x'_i = x_i \forall i \in \mathcal{B}$$

It is trivial to see that this construction results in an invertible and differential transformation, since \mathbf{x}_1 is subject to an affine transformation and \mathbf{x}_2 remains the same. Coupling layers also offer the benefit of a simple Jacobian [14]:

$$J_{T_{coupling}} = \mathbf{1} F_1(\mathbf{x}_2; \Theta_1)$$

where $\mathbf{1}$ denotes a row vector of ones – in other words, summing the elements of $F_1(\mathbf{x}_2; \Theta_1)$.

A common way to construct powerful learnable transformations is to simply compose several of these coupling layers, while changing which parts of \mathbf{x} are modified [14]. This approach is used in for example RealNVP [5], and it is what we use in our experiments.

3.3 Normalizing Flows for Image Out-of-Distribution Detection

Formulating OOD detection using normalizing flows is simple since we can directly compute the likelihood using (1), which represents the belief that an example \mathbf{x} is in-distribution [9]. Note that a high value for $p_x(\mathbf{x})$ equals a low likelihood of being OOD. It is otherwise common for OOD detection systems to use the opposite formulation, where a high value equals high likelihood of being OOD.

Training a normalizing flow for OOD detection is done by minimizing the KL-divergence between $p_x(\mathbf{x})$ and $p_u(\mathbf{u})$, which results in the following loss function [14]:

$$\mathcal{L} = -\frac{1}{N} \sum_{n=1}^N \log p_u(T^{-1}(\mathbf{x}_n)) + \log |\det J_{T^{-1}}(\mathbf{x}_n)| + C$$

where C is a constant.

Since we only need the inverse transformation T^{-1} for OOD detection, we simply let T^{-1} denote the forward pass of our affine coupling layers.

Despite seeming like perfect candidates for OOD detection, since equation (1) allows for the direct computation of the approximate probability distribution, normalizing flows often fail in the OOD detection setting [9]. Kirichenko *et al.* [9] suggest that common image flow models, such as RealNVP[5], simply learn pixel correspondences, and as such fail to grasp any semantic information that makes up the distribution. Instead of training a flow model on images directly, Kirichenko *et al.* suggest training a flow model on features extracted from another pretrained image network.

4 Our Method

Based on our hypothesis that a lower OOD score would lead to better reliability, we propose to let a vision system adapt, instead of simply discarding OOD data. In order to bring this into practice, we create a framework for adaption around an existing vision task, namely object detection with a YOLOv4 [2] network.

Most commercially available cameras expose a number of parameters that can alter the visual appearance of a captured image. Common examples are: saturation, contrast, and exposure. These camera parameters are a natural candidate for adapting to various imaging conditions.

Our proposed method is to simply adapt camera parameters in order to minimize the OOD score. Intuitively, our method can be deployed in two ways:

- either: if an image is marked as OOD, adapt camera parameters to minimize OOD score,
- or: continuously adapt camera parameters to minimize OOD score.

While most cameras have some built-in measures to adapt, for example auto-exposure, these are often not sufficient. Figure 1 shows two images of the same scene to illustrate this. In one picture, the camera’s default settings fail to produce an image suitable for object detection. With hand-tuned parameters however, a more suitable image is acquired. Note also the yellow bounding box indicates that YOLOv4 can detect the dog in the hand-tuned image, which is not possible with the camera’s default settings.

5 Experiments and Results

In this section we outline our two main experiments. Section 5.1 outlines the normalizing flow model used in our experiments. For our first experiment, in



Fig. 1. Left: camera’s default parameters, auto-exposure and auto-white-balance enabled. Right: hand-tuned camera parameters.

section 5.3, we train and validate a normalizing flow model as an OOD detector using the COCO dataset [11] as our in-distribution. Then, in section 5.4, we take the model from section 5.3 and apply it to a real-world experiment testing our proposed method from section 4.

5.1 Normalizing Flow Model

For our experiments we construct a normalizing flow model from affine coupling layers. We use a model very similar to the one used by Kirichenko *et al.* [9]. Our model consists of a series of 10 affine coupling layers, alternating between transforming the first and second half of the input vector. Each coupling layer consists of two shared linear layers with 512 hidden units followed by two parallel linear layers, one for α and one for β . This is illustrated in Figure 2.

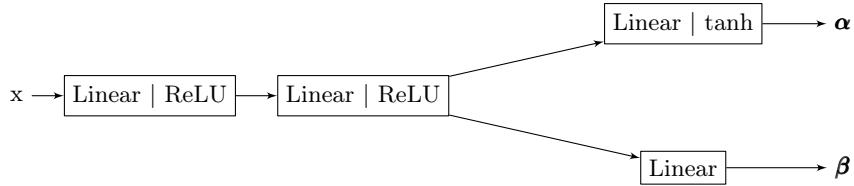


Fig. 2. Illustration of coupling layer layout.

We train our model on features extracted from a YOLOv4 [2] object detection network. Training is done for 200 epochs with a batch size of 128, using the Adam [8] optimizer with a learning rate of 10^{-4} .

5.2 Intel RealSense D435

Our experiments use an Intel RealSense D435[1], which is a RGB+Depth camera. In our experiments, however, we only utilise the RGB sensor. Like most commercially available cameras, the D435 exposes a number of parameters that affect the resulting image in various ways. For our experiments we manipulate a selection of these parameters, specifically: backlight compensation, brightness, contrast, exposure, gain, saturation, and sharpness.

5.3 Training Normalizing Flows on a Large Diverse Dataset

In this section, we train a normalizing flow model using the entire COCO training dataset as our in-distribution. We then test the model by comparing the log-likelihood outputs from different input image distributions. Specifically, we divide this section into two smaller experiments:

- One where we compare log-likelihoods of COCO images to randomly generated images.
- One where we capture many images of a static scene with randomized camera parameters, and compare their log-likelihood to COCO images.

For our first experiment we run the trained model on all images in the COCO training, validation and testing datasets, and record the resulting log-likelihood scores. In order to show that the model can distinguish images that are *definitely not* part of the in-distribution, we also record log-likelihood scores from randomly generated images. These random images were generated according to:

$$\begin{aligned}\sigma &\sim \text{Uniform}(1, 256) \\ v_{i,j} &\sim \mathcal{N}(0, \sigma) \\ pixel_{i,j} &= \min(255, |v_{i,j}|), \quad \forall i, j.\end{aligned}$$

Our primary motivation behind using this formula is to generate images with varying pixel ranges, while also generating some mostly-black and mostly-white images. By varying σ per image, we vary the pixel range in different images. When σ is small, we get mostly-black images, and when σ is large we get mostly-white images due to the min operation. Figure 3 shows histograms of randomly generated images, along with the COCO training, validation, and testing datasets.

For our second experiment we explore whether our normalizing flow can distinguish between different real images. We place a stationary D435 camera in our lab, and capture a large number of images. For each image, all camera parameters are randomized according to a uniform distribution. In order to be certain that all variance in log-likelihood scores are caused by changing camera parameters, we take precautions to minimize the amount of natural light entering

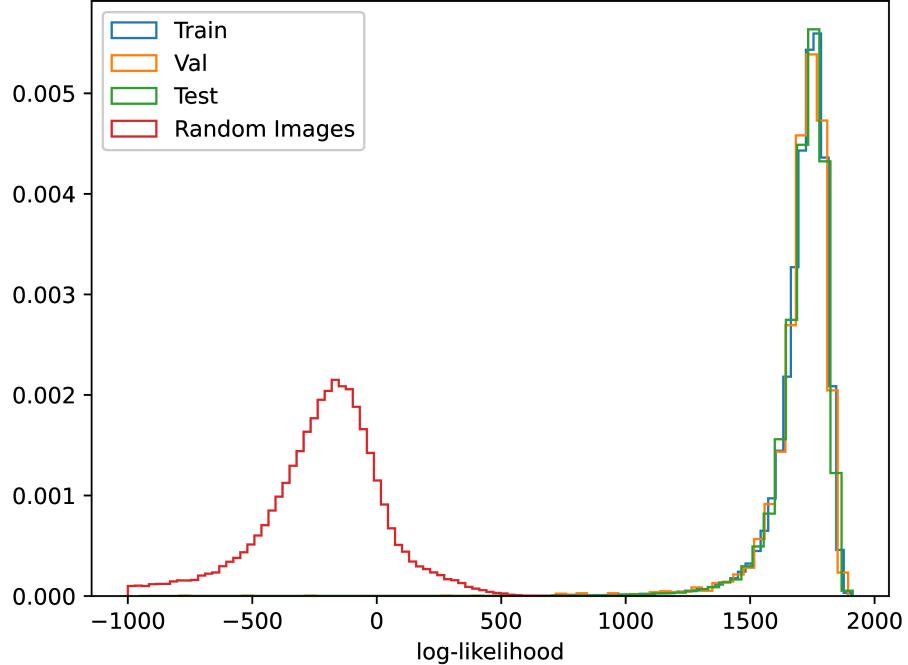


Fig. 3. Normalized histogram plot of log-likelihoods from COCO training, validation and test datasets, as well as random images. The histogram has been cut at -1000 log-likelihood for improved readability.

our lab. Then, we capture 10000 images with the ceiling lamp on, and 10000 with the ceiling lamp off. The number 10000 was chosen arbitrarily. Figure 4 shows the corresponding histograms of log-likelihood scores, along with the COCO training data for comparison.

5.4 Parameter Optimization

Here, we present results from a small-scale experiment conducted in one of our robot labs. In order to make the experiment as realistic as possible, the experiment was set up using one of our robots. Our robot has an industrial arm with an Intel Realsense D435 camera attached at the gripper.

For this experiment, we set up an optimization procedure using camera parameters as our input, and the log-likelihood from our normalizing flow model as output, as illustrated in Figure 5. More formally, let θ denote the camera parameters, and $C(\theta)$ denote the camera (a function that takes camera parameters and produces an image \mathbf{x}), then we solve:

$$\arg \max_{\theta} \log p_x(C(\theta))$$

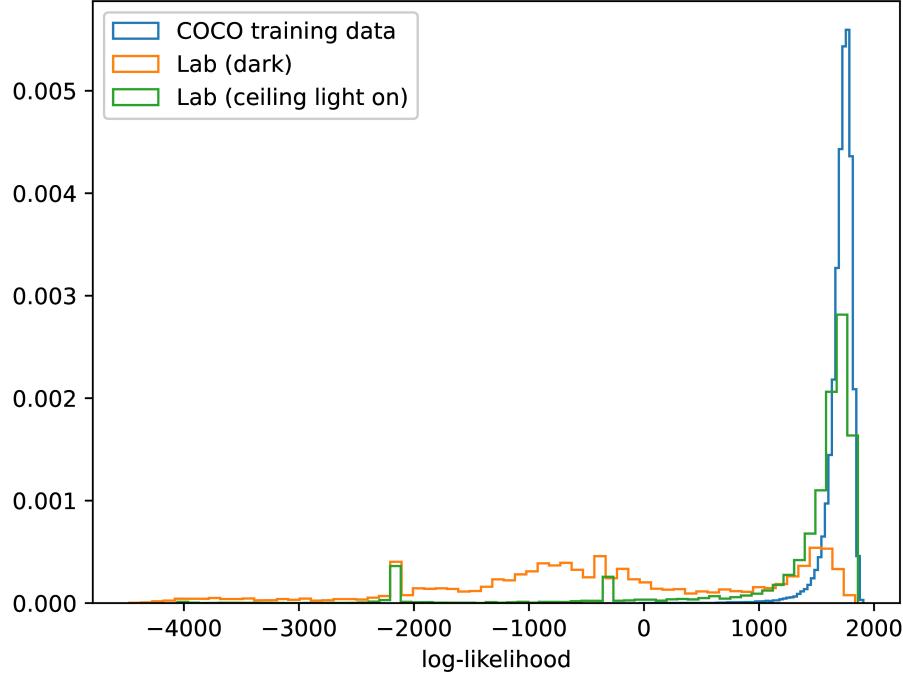
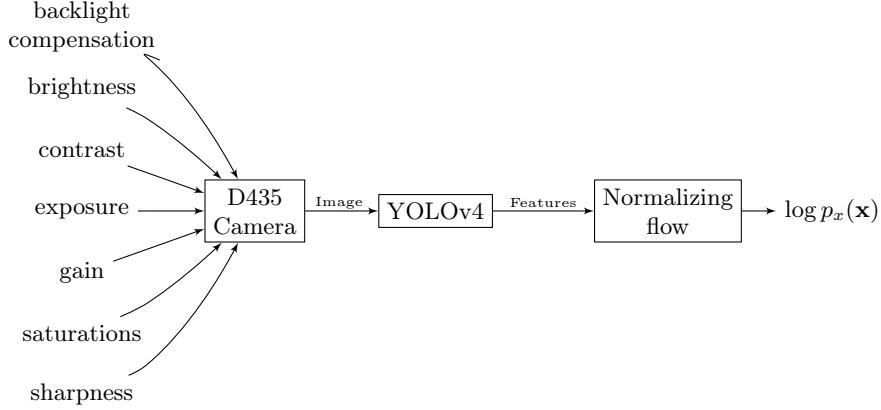


Fig. 4. Normalized histogram plot of log-likelihood values. Blue: COCO training images. Orange: 10000 images in our lab with randomized camera parameters, without any lights on. Green: 10000 images in our lab with randomized camera parameters, with ceiling lights on.

$$= \arg \max_{\theta} \log p_u(T^{-1}(C(\theta))) + \log |\det J_{T^{-1}}(C(\theta))|.$$

For the optimization procedure, we used a very simple evolutionary optimization with elitism selection, a population of 50, and a mutation rate of 20%. No crossover was used. Optimization was carried out for a total of 200 evaluations. Population size, mutation rate, and number of evaluations were chosen based on prior experience with evolutionary optimization. A small number of trial runs confirmed that these values work well.

In our lab, we set up 13 different scenarios with varying COCO objects, viewing angles, and lighting conditions. We took care to make the scenarios as challenging as possible by introducing difficult lighting conditions, reflective surfaces, and background clutter. With the robot stationary, optimization was performed as described above. During optimization, the images with the highest and lowest log-likelihood scores were saved along with their corresponding parameters, and compared to images captured with the camera's default parameter settings. Henceforth, these will be referred to as *best*, *worst* and *default* parameters, respectively.

**Fig. 5.** Parameter optimization setup.

After the optimization procedure, all images were annotated with bounding boxes for the COCO objects present. They were then fed through YOLOv4. Mean-Average-Precision (mAP), mean-Average-Recall (mAR), and F1 scores were computed and compared between the worst, default, and best camera parameters.

Table 1 shows average mAP, mAR, and F1 scores for best, default and worst camera parameters. In table 2 we show statistics for the improvement in mAP, mAR, and F1 scores of the best parameters compared to the default parameters. Similarly, Table 3 shows statistics for the improvement of the best compared to the worst parameters.

Table 1. Average mAP, mAR, and F1 score for best, default, and worst parameters.

	mAP	mAR	F1
Best	0.7977	0.3283	0.4605
Default	0.7591	0.2977	0.4219
Worst	0.1731	0.0431	0.0687

Table 2. mAP, mAR, and F1 improvement. Best compared to default parameters.

	Min	Max	Average
mAP	-0.2889	+0.5778	+0.0386
mAR	-0.1083	+0.25	+0.0306
F1	-0.1056	+0.349	+0.0386

Table 3. mAP, mAR, and F1 improvement. Best compared to worst parameters.

	Min	Max	Average
mAP	-0.104	+0.95	+0.6247
mAR	+0.1375	+0.4278	+0.2852
F1	+0.1794	+0.5703	+0.3918

Figure 6 shows box plots of the improvements in mAP, mAR, and F1 scores for the best parameters compared with the default and worst parameters respectively.

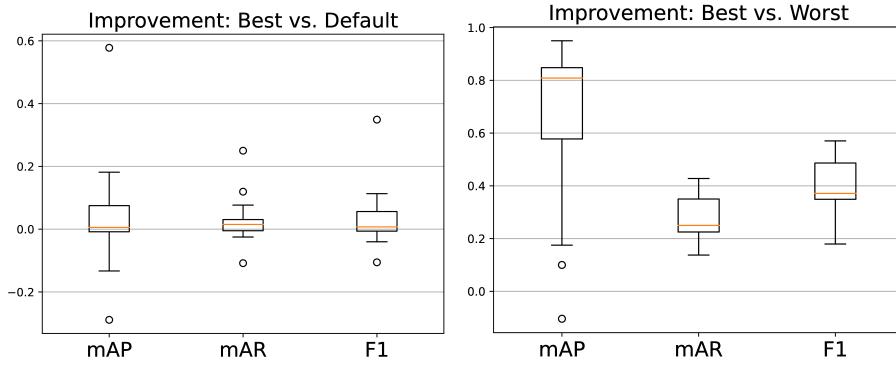


Fig. 6. Boxplot of mAP, mAR, and F1 improvement. Left: Best compared to default parameters. Right: Best compared to worst parameters.

In Figures 7, 8, and 9 we show example scenarios where optimization led to varying degrees of success. Each figure includes images along with the respective F1 scores from YOLOv4. Figure 7 shows examples where parameter optimization resulted in better object detection than the default parameters. Figure 8 shows examples where optimization resulted in no significant difference in object detection. Finally, Figure 9 shows the only scenario where optimization resulted in significantly worse object detection.

6 Discussion

6.1 Training on the COCO Dataset

When training on the full COCO dataset we see two important indicators that the model is learning properly. First, as seen in Figure 3, the output log-likelihoods span a wide range of values. It would be a red flag if all log-likelihoods were too similar, indicating that the learned probability distribution might be flat. Second, we observe that the validation and test datasets are distributed the same

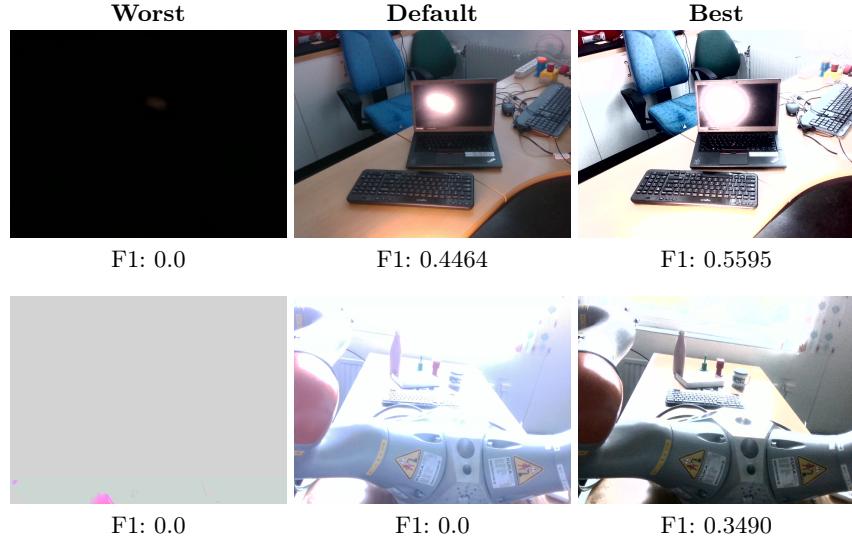


Fig. 7. Examples where parameter optimization resulted in an improvement in object detection. Left: lowest log-likelihood. Center: default parameters. Right: highest log-likelihood.

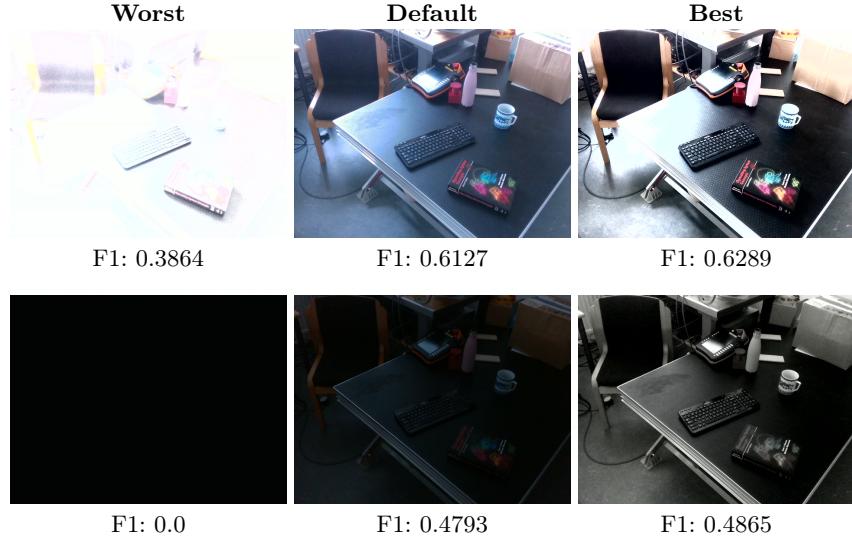


Fig. 8. Examples where parameter optimization resulted in no significant difference in object detection. Left: lowest log-likelihood. Center: default parameters. Right: highest log-likelihood.

way as the training set, while the random images are given distinctly lower log-likelihood scores.



Fig. 9. Example where parameter optimization resulted in worse performance in object detection. Left: lowest log-likelihood. Center: default parameters. Right: highest log-likelihood.

In figure 4, we see that the model is able to distinguish between real images from different distributions. As such, we have found no evidence to suggest that the size and diversity of the COCO dataset should pose any problems for OOD detection. This in turn suggests that normalizing flow models can be readily applied to diverse real-world scenarios.

Additionally, figure 4 shows that varying parameters in the D435 over the same scene can result in a wide range of log-likelihood values. This gives merit to the idea that camera parameters form a good basis for adapting to various imaging conditions.

6.2 Parameter Optimization Experiment

Table 1 tells us at a glance that our optimized camera parameters achieved the best average scores across the board. Looking at Table 2, we see an average improvement of 3 to 4 percentage points in terms of all metric scores. From the box plot in Figure 6, we see that most scenarios yield only a small difference in object detection compared to the default parameters. However, most importantly we see that the trend is a positive improvement. We also observe that the positive outliers are significantly larger than the negative outliers, again suggesting a positive trend.

Perhaps unsurprisingly, Table 3 shows a massive improvement (28 to 62 percentage points on average) between the best and worst parameters found. The box plot in Figure 6 further highlights this improvement.

In Figures 7, 8, and 9 we note that some of the parameter-optimized pictures look better to the human eye, while others look worse despite resulting in better object detection. This highlights the important fact that there is no equivalence between images that look good to the human eye, and images that are good for object detection.

Perhaps the most important result of this experiment is not the fact that parameter-optimization can lead to better object detection, but instead that it seems feasible to adapt and recover from a very poor visual scenario. Extrapolating from the fact that different camera parameters could result in images

where YOLO performs well, and where it fails almost completely on the same scenario, it seems feasible that these parameters have the necessary capacity to adapt to a wide range of difficult visual conditions. This is given further merit by the fact that the varying camera parameters resulted in widely varying log-likelihood values in figure 4. As such, when the normalizing flow model detects something with extremely low log-likelihood value, it is most likely possible to adapt camera parameters to get an image more suitable for object detection. While these experiments have only been concerned with object detection, we expect that this behaviour generalises to other vision tasks as well.

7 Conclusion

In this paper we have shown that it is possible to train a normalizing flow model for OOD detection on a large and diverse dataset such as COCO. We have also conducted a real-world experiment which shows that a normalizing flow OOD detector can be used as a quality metric for a vision task. Optimizing camera parameters with respect to the output from our normalizing flow OOD model yielded, on average, a 3 to 4 percent unit improvement in the reliability of YOLOv4 in terms of object detection performance.

8 Future Work

This paper lays the groundwork for a larger framework of quality metrics for vision task. In order to further expand this framework, a larger benchmark dataset is beneficial. It is also an interesting research direction to explore whether image improvements can be *learned* based on this OOD metric, rather than going through the somewhat slow optimization process used in our experiments.

Acknowledgements This research is funded by the Excellence Center at Linköping-Lund in Information Technology (ELLIIT), and the Wallenberg AI, Autonomous Systems and Software Program (WASP). Computations for this publication were enabled by the supercomputing resource Berzelius provided by the National Supercomputer Centre at Linköping University and the Knut and Alice Wallenberg foundation. Additionally, Luigi Nardi was supported in part by affiliate members and other supporters of the Stanford DAWN project – Ant Financial, Facebook, Google, Intel, Microsoft, NEC, SAP, Teradata, and VMware.

References

1. Intel realsense depth camera d435, <https://www.intelrealsense.com/depth-camera-d435/>
2. Bochkovskiy, A., Wang, C.Y., Liao, H.Y.M.: Yolov4: Optimal speed and accuracy of object detection (2020). <https://doi.org/10.48550/ARXIV.2004.10934>, <https://arxiv.org/abs/2004.10934>

3. Bogachev, V.I., Ruas, M.A.S.: Measure theory, vol. 1. Springer (2007)
4. Deecke, L., Vandermeulen, R., Ruff, L., Mandt, S., Kloft, M.: Image anomaly detection with generative adversarial networks. In: Berlingerio, M., Bonchi, F., Gärtner, T., Hurley, N., Ifrim, G. (eds.) Machine Learning and Knowledge Discovery in Databases. pp. 3–17. Springer International Publishing, Cham (2019)
5. Dinh, L., Sohl-Dickstein, J., Bengio, S.: Density estimation using real nvp. (2016), <https://arxiv.org/abs/1605.08803>
6. Gawlikowski, J., Tassi, C.R.N., Ali, M., Lee, J., Humt, M., Feng, J., Kruspe, A., Triebel, R., Jung, P., Roscher, R., Shahzad, M., Yang, W., Bamler, R., Zhu, X.X.: A survey of uncertainty in deep neural networks (2021). <https://doi.org/10.48550/ARXIV.2107.03342>, <https://arxiv.org/abs/2107.03342>
7. Hsu, Y.C., Shen, Y., Jin, H., Kira, Z.: Generalized odin: Detecting out-of-distribution image without learning from out-of-distribution data. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (June 2020)
8. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization (2014), <https://arxiv.org/abs/1412.6980>
9. Kirichenko, P., Izmailov, P., Wilson, A.G.: Why normalizing flows fail to detect out-of-distribution data. Advances in neural information processing systems **33**, 20578–20589 (2020)
10. Liang, S., Li, Y., Srikant, R.: Principled detection of out-of-distribution examples in neural networks. CoRR **abs/1706.02690** (2017), <http://arxiv.org/abs/1706.02690>
11. Lin, T.Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C.L., Dollár, P.: Microsoft coco: Common objects in context (2014). <https://doi.org/10.48550/ARXIV.1405.0312>, <https://arxiv.org/abs/1405.0312>
12. McAllister, R., Kahn, G., Clune, J., Levine, S.: Robustness to out-of-distribution inputs via task-aware generative uncertainty. In: 2019 International Conference on Robotics and Automation (ICRA). pp. 2083–2089 (2019). <https://doi.org/10.1109/ICRA.2019.8793552>
13. Mohseni, S., Pitale, M., Yadawa, J., Wang, Z.: Self-supervised learning for generalizable out-of-distribution detection. Proceedings of the AAAI Conference on Artificial Intelligence **34**(04), 5216–5223 (Apr 2020). <https://doi.org/10.1609/aaai.v34i04.5966>, <https://ojs.aaai.org/index.php/AAAI/article/view/5966>
14. Papamakarios, G., Nalisnick, E.T., Rezende, D.J., Mohamed, S., Lakshminarayanan, B.: Normalizing flows for probabilistic modeling and inference. J. Mach. Learn. Res. **22**(57), 1–64 (2021)
15. Ren, J., Liu, P.J., Fertig, E., Snoek, J., Poplin, R., Depristo, M., Dillon, J., Lakshminarayanan, B.: Likelihood ratios for out-of-distribution detection. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 32. Curran Associates, Inc. (2019), <https://proceedings.neurips.cc/paper/2019/file/1e79596878b2320cac26dd792a6c51c9-Paper.pdf>
16. Rudin, W.: Real and Complex Analysis. Tata McGraw-Hill Education (2006)
17. Wellhausen, L., Ranftl, R., Hutter, M.: Safe robot navigation via multi-modal anomaly detection. IEEE Robotics and Automation Letters **5**(2), 1326–1333 (2020). <https://doi.org/10.1109/LRA.2020.2967706>

18. Winkens, J., Bunel, R., Roy, A.G., Stanforth, R., Natarajan, V., Ledsam, J.R., MacWilliams, P., Kohli, P., Karthikesalingam, A., Kohl, S., Cemgil, A.T., Eslami, S.M.A., Ronneberger, O.: Contrastive training for improved out-of-distribution detection. CoRR **abs/2007.05566** (2020), <https://arxiv.org/abs/2007.05566>
19. Yang, J., Zhou, K., Li, Y., Liu, Z.: Generalized out-of-distribution detection: A survey (2021). <https://doi.org/10.48550/ARXIV.2110.11334>, <https://arxiv.org/abs/2110.11334>
20. Yuhas, M., Feng, Y., Ng, D.J.X., Rahiminasab, Z., Easwaran, A.: Embedded out-of-distribution detection on an autonomous robot platform. In: Proceedings of the Workshop on Design Automation for CPS and IoT. p. 13–18. Destion ’21, Association for Computing Machinery, New York, NY, USA (2021). <https://doi.org/10.1145/3445034.3460509>, <https://doi.org.ludwig.lub.lu.se/10.1145/3445034.3460509>