SafetyCage:
A misclassification detector for feed-forward neural networks

Pål Vegard Johnsen*[1] and Filippo Remonato[1]

[1]Sintef Digital
pal.johnsen@sintef.no, filippo.remonato@sintef.no

Abstract

Deep learning classifiers have reached state-of-the-art performance in many fields, particularly so image classification. Wrong class assignment by the classifiers can often be inconsequential when distinguishing pictures of cats and dogs, but in more critical operations like autonomous driving vehicles or process control in industry, wrong classifications can lead to disastrous events. While reducing the error rate of the classifier is of primary importance, it is impossible to completely remove it. Having a system that is able to flag wrong or suspicious classifications is therefore a necessary component for safety and robustness in operations. In this work, we present a general statistical inference framework for detection of misclassifications. We test our approach on two well-known benchmark datasets: MNIST and CIFAR-10. We show that, given the underlying classifier is well trained, SafetyCage is effective at flagging wrong classifications. We also include a detailed discussion of the drawbacks, and what can be done to improve the approach.

1 Introduction

Image classification with deep learning (DL) classifiers has in recent years seen extensive use. While convolutional Neural Networks (CNN) are widely used for this purpose on a variety of situations [1–4], deep learning models exclusively based on multilayer perceptrons have recently shown equal success [5, 6]. While these approaches showcase great results, it is impossible to create a perfect system: Some wrong classifications are bound to happen. While this oftentimes does not pose a problem, wrong results from the classifier can lead to serious consequences in difficult or safety-critical operations, e.g. autonomous driving vehicles. Work has been put into establishing frameworks to increase the robustness of classification models, especially so for out-of-distribution (OOD) detection [7, 8]. A common approach in OOD detection consists in constructing a second DL-based model that acts as a supervisor for the classifier. In this direction Henriksson et.al. [9] propose a structured way to assess the performance of the supervisor for OOD detection in DL classifiers. Bayesian neural networks, which would be an obvious choice for basing wrong classification detection on uncertainty estimation, have instead been shown to be unsuitable [10].

In our setting, instead, we are interested in *in distribution* detection of wrong classification. That is, while OOD detection concerns input samples for which the classifier has not been trained, e.g. a classifier trained on cats and dogs receiving an eagle as input, we are interested in detecting samples that do belong to the same family as the training samples (cats and dogs) but are misclassified, possibly due to unknown, unexpected, or rare properties that position them close to the boundary of the distribution. Some work has been done also in this direction, but mostly by simply evaluating the network's subjective "uncertainty" by thresholding the value of a softmax function at the output layer [11].

We take a route not based on additional DL supervisors or limited to the last layer, by treating the neurons' activation values in a neural network as realisation of a *multivariate* distribution. Based on this we infer whether the classification is likely to be correct. In simpler terms, for a given predicted class, we want to describe the subset of all samples that the model correctly classifies. We posit that, for well-performing models, wrongly classified samples will tend to appear away from the centres of the distribution of correctly classified samples. Our method is simple to implement, readily applicable to all layers of a feed-forward neural network—and not just the final one—and grounded in well-established statistical theory. Since the method must be applicable to already trained classifiers without demanding retraining or other modifications to the architecture, we trained a simple neural network to classify images, and froze it after training was concluded. We would like to stress this part: Our method does not take any advantage of the specific architecture or training procedure of the underlying classifier. The only requirement is the possibility to access the network's internal activation values during forward passes. As such, minute details on how the classifier was built and trained are irrelevant to the scope of this work; still, for the sake of reproducibility, the

*Corresponding Author.
This work has been supported by the NFR project 304843 - EXAIGON

Proceedings of the 5th Northern Lights Deep Learning Conference (NLDL), PMLR 233, 2024.
© 2024 Pål Vegard Johnsen & Filippo Remonato. This is an open access article distributed under the terms and conditions of the Creative Commons Attribution license (http://creativecommons.org/licenses/by/4.0/).

main information on the architecture of the classifier is given in the Results section. Everything is built on top of the classifier, making sure to never exploit model-specific information such as particular neuron connections or other structural elements. In this view our method acts as a cage around a classifier, capturing and flagging wrong classifications, hence the name *SafetyCage*, and is applicable to any existing feed-forward network.

2 Multivariate SafetyCage

Let us set the notation for the rest of the paper: Consider an input sample $x \in \mathbb{R}^p$ with y its true class (label) and $\hat{y}$ the predicted class by the classifier, in our case a simple MLP with L layers. Each layer l is composed by N_l neurons, which during a forward pass activate with an activation value $a_n = g(z_n)$, where g is the activation function and z_n is the *pre-activation* value for a given neuron n. We denote with Ω the set of all input samples to the classifier, and with $\Omega_y \subseteq \Omega$ the subset of correctly classified samples with true class y, i.e. $\Omega_y = \{x \in \Omega \,|\, \hat{y} = y\}$. Then, our method of uncertainty estimation under classification is based on using the training set $\Omega^{(tr)}$ of the classifier to construct the probability distribution of the pre-activation values $z^{(tr)}$ corresponding to Ω_y for each class y. In other words, for each class y we construct the distributions of z from the samples in $\Omega_y^{(tr)}$: $P(z \mid x \in \Omega_y^{(tr)})$. The process of constructing and storing these distributions is what we call *training* for the SafetyCage. Then, given a test sample x^* not seen during the SafetyCage's training, its predicted class $\hat{y}^*$, and its corresponding pre-activation values z^* from the full network, we infer whether z^* originated from the probability distribution corresponding to $\Omega_{\hat{y}^*}^{(tr)}$ by means of a hypothesis test:

$$\begin{aligned} H_0 &: Z^* \sim P(z \mid \Omega_{\hat{y}^*}) \\ &\text{against} \\ H_1 &: Z^* \not\sim P(z \mid \Omega_{\hat{y}^*}). \end{aligned} \tag{1}$$

Note that in (1) we dropped the superscript $^{(tr)}$ to lighten the notation. We now make a few remarks:

Remark 1: The reason we prefer to use the pre-activation values is twofold: First, we want the SafetyCage to operate regardless of how the classifier was trained. For models with ReLu, activation values often present large inflations at 0, which make two-sided p-value computations prolematic; ReLu deletes information that could have been useful, which is instead preserved in the pre-activations. Second, we observed that pre-activation values appear more normally distributed than the actual activations, see Figure 1.

Remark 2: While we train the SafetyCage on the correctly classified samples in the *train* set of the classifier, we could use also (or exclusively) the ones from the *test* set. Indeed, the only strict requirement is that the SafetyCage is trained on correctly classified samples. The reason to use the test set would be avoiding the overfitting the model did on the train data, leading to exaggerated confidence, and the influence of borderline wrong classifcation that would pollute the distributions in the SafetyCage. Note that, since SafetyCage and the classifier model absolve to two completely different tasks, there is no information leak in using the classifier's test set for training SafetyCage. Due to both time and space constraints, we will include this investigation in another work.

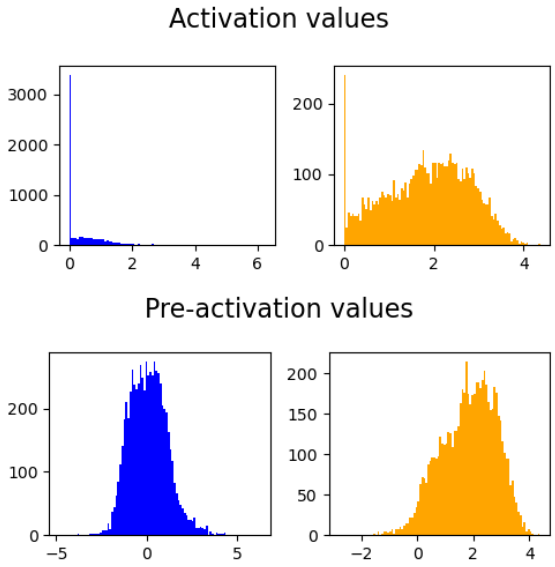

Figure 1. Observed probability distribution of ReLu-activation values (top) and pre-activation values (bottom) for two particular nodes in the penultimate (last hidden) layer of a trained MLP model used in this paper.

Remark 3: For a given predicted class, we test membership against *not* membership. This might become problematic when several classes are present, as the space of "not membership" becomes large. In those cases, it might be beneficial to set up the hypothesis test between *another* class and the predicted class, for each other class. Then, one can accept the predicted class when all tests are failed, or reject the predicted class if at least one test accepts the null hypothesis for membership in one of the other classes. For the same reasons as Remark 2, we will include this investigation in another work.

Remark 4: One can similarly employ the observed probability distribution of wrongly predicted samples for each class, and infer whether the sample to be tested is likely to belong to the distribution of correctly or wrongly predicted samples for the corresponding predicted class via a likelihood ratio test. This approach can be relevant with less-performing classifier models, in which the distribution of cor-

rectly and wrongly predicted samples overlap.

2.1 Mahalanobis distance

Our multivariate approach to detecting wrong classifications is based on the Mahalanobis distance, inspired by the OOD-detection approach explained in [8]. With motivation from Figure 1, assume the pre-activation random vector Z of size q has a multivariate Gaussian distribution, $Z \sim \mathcal{N}_q(\mu, \Sigma)$, with mean μ and covariance matrix Σ. Then it can be shown [12] that the *Mahalanobis distance* $U = (Z-\mu)^T\Sigma^{-1}(Z-\mu)$ has a Chi-squared distribution with q degrees of freedom, $U \sim \chi^2_q$.

Given a sample x to be tested, one can then calculate the appropriate p-value as $p = 1 - F_{\chi^2_q}(u)$, with $F_{\chi^2_q}$ being the cumulative of the Chi-squared and u the realisation of the Mahalanobis distance for the pre-activation values z of x.

The above holds when μ and Σ are known. If they are unknown one needs to estimate them, usually with the sample mean $\bar{\mu}$ and the unbiased sample covariance $\bar{\Sigma}_u$. In those cases, the corresponding distribution of $\bar{U}$ will still converge asymptotically to a χ^2_q as the amounts of samples in the estimators $n \longrightarrow \infty$ thanks to the continuous mapping theorem [13], but it may be inaccurate if Ω_y is small for a particular class y, e.g. in cases with unbalanced classes.

The assumption regarding the Gaussianity of pre-activation values is strong. Nonetheless, when the pre-activation random vector exhibits a multivariate probability distribution closely approximating that of a Gaussian, the utilization of the Mahalanobis distance remains a justifiable choice.

In this context, one can argue that an alternative way to infer the null hypothesis would be computing the p-value of the *empirical cumulative distribution function* (ECDF) of the Mahalanobis distance. Note that this puts no assumptions on the underlying probability distribution of the pre-activation values. Denoting $\hat{F}$ the corresponding ECDF, the p-value is then computed as $1 - \hat{F}(u)$.

Separation between layers

In a neural network the input x goes through several transformations equal to the number of hidden layers plus one. It can therefore be natural to separate the pre-activation values Z per layer $Z = [Z_1, ..., Z_h, Z_o]$, with h the total number of hidden layers and Z_o denoting the output layer. One can then also separate the null hypothesis given in (1) in the same way, thus obtaining one p-value for each layer, which can later be combined into one single p-value for the whole sample. There are several p-value combination tests, among them Fisher's combination test [14], as well as the more recent Cauchy combination test [15].

It may therefore be beneficial to only regard a subset of all layers when choosing which p-values to combine for the final decision. In this work we investigate the concatenation of all layers (giving one p-value directly), separation between all layers (giving $h+1$ p-values, then combined to one), only including the hidden layers (giving h p-values, then combined to one), and only including the penultimate layer (giving one p-value).

3 Results

We tested our approach on two famous benchmark datasets: MNIST [16] and CIFAR-10 [17]. In each case, the underlying classifier is a feed-forward, fully-connected neural network composed of two hidden layers with 256 and 128 neurons, respectively, and one output layer with 10 neurons. The hidden layers have ReLu activation functions, while the output layer has sigmoid activation. The network was trained with the standard Adam optimizer. After the classifier was trained, the SafetyCage has been trained as described in Section 2, and tested on 10000 unseen samples not used for training. We evaluate the SafetyCage with the performance metrics precision (P), recall (R), specificity (Spe), negative predictive value (NPV), and the Matthews correlation coefficient (MCC), which we regard as the main performance metric due to its summarising properties with respect to the other metrics [18].

3.1 Results on a good classifier

We first applied the SafetyCage on the MNIST dataset. On this dataset, the underlying classifier does a good job at distinguishing different handwritten digits with an accuracy of 0.98. See also Figure 2 where we show scikit-learn's t-SNE [19, 20] decomposition of the pre-activation values in the penultimate layer of the network for the digits 1 and 8. There we see how the classes are well separated, and also how wrong classifications have a tendency to appear closer to the edge of the feature space, which makes sense when the classifier is well trained. Those are the points we are interested in detecting with the SafetyCage.

Table 1 presents the performance metrics for the SafetyCage applied to the MNIST dataset where p-value rejections, and consequent flags as wrong classifications, were performed at a significance level of $\alpha = 0.05$. The column M indicates on what basis the p-values are computed: Either via the empirical cumulative distribution function (ECDF), or via the χ^2-asymptotic under the assumption of Gaussianity. The column L indicates to which layer the SafetyCage was applied: 'p' for only penultimate layer, 'h' for all hidden layers, or 'a' or all layers (hidden and output). In the cases where multiple layers have

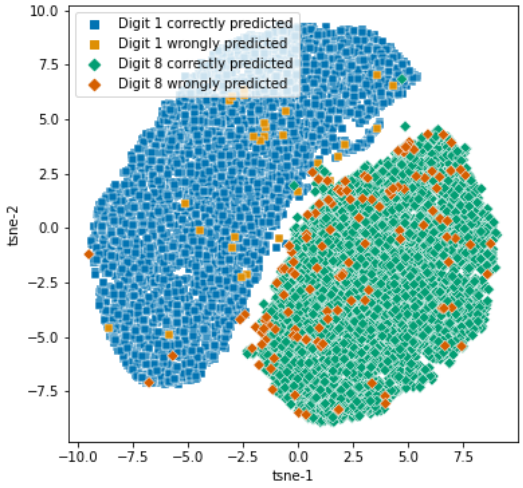

Figure 2. A t-SNE visualisation of the feature space of the penultimate layer of the MLP network for the digits 1 and 8 in the MNIST dataset and whether the classifier assigned the correct or incorrect label. The classes are clearly separated. Wrong classifications tend to appear near the border of the feature space of the correct classifications.

been considered (L=a or h), p-values coming from each layer have been combined using the Fisher or Cauchy combination test, as indicated in the column CT in the table. We tested all combinations of L and p-value aggregation procedures for both approaches, ECDF and χ^2-asymptotic. Additionally, we varied the significance level α to maximise the MCC. Results for the optimal α are presented in Table 2, with the corresponding confusion matrix shown in Figure 3.

M	L	CT	P	R	Spe	NPV	MCC
ECDF	p	-	0.19	0.57	0.95	0.99	0.31
ECDF	h	Cau.	0.17	0.57	0.94	0.99	0.29
ECDF	a	Cau.	0.13	0.71	0.91	0.99	0.28
ECDF	a	Fis.	0.11	0.83	0.87	1.00	0.27
ECDF	h	Fis.	0.13	0.69	0.91	0.99	0.27
ECDF	c	-	0.14	0.44	0.95	0.99	0.22
χ^2	p	-	0.073	0.91	0.77	1.00	0.22
χ^2	a	Fis.	0.062	0.93	0.72	1.00	0.20
χ^2	a	Cau.	0.062	0.90	0.73	1.00	0.19
χ^2	h	Cau.	0.06	0.90	0.72	1.00	0.19
χ^2	h	Fis.	0.06	0.90	0.72	1.00	0.19
χ^2	c	-	0.06	0.81	0.76	0.99	0.18

Table 1. The evaluation of the SafetyCage when applied on the MNIST data, for different combinations, ordered with respect to MCC when $\alpha = 0.05$.

As we can see from the tables, the SafetyCage has high recall but very low precision. On the one side, precision is not the most informative metric in this case, as it does not take into account the true negatives, that is, the samples that are correctly classified and correctly accepted by the SafetyCage.

M	L	CT	Best α	P	R	Spe	NPV	MCC
χ^2	p	-	2.32e-11	0.18	0.60	0.95	0.99	0.31
ECDF	p	-	5.01e-2	0.19	0.58	0.95	0.99	0.31
ECDF	a	Fis.	1.36e-2	0.15	0.73	0.92	0.99	0.31
ECDF	a	Cau.	2.62e-2	0.17	0.63	0.94	0.99	0.30
ECDF	h	Cau.	5.45e-2	0.17	0.61	0.94	0.99	0.30
ECDF	h	Fis.	2.69e-2	0.17	0.60	0.94	0.99	0.29
χ^2	a	Fis.	4.05e-14	0.14	0.68	0.91	0.99	0.28
χ^2	a	Cau.	2.65e-12	0.13	0.71	0.90	0.99	0.28
χ^2	h	Cau.	3.41e-9	0.11	0.75	0.88	0.99	0.26
χ^2	h	Fis.	3.23e-8	0.10	0.80	0.86	1.00	0.25
ECDF	c	-	4.50e-2	0.15	0.44	0.95	0.99	0.24
χ^2	c	-	3.17e-9	0.11	0.62	0.90	0.99	0.23

Table 2. The evaluation of the SafetyCage when applied on the MNIST data, ordered with respect to the MCC, with the α that maximizes MCC for each combination.

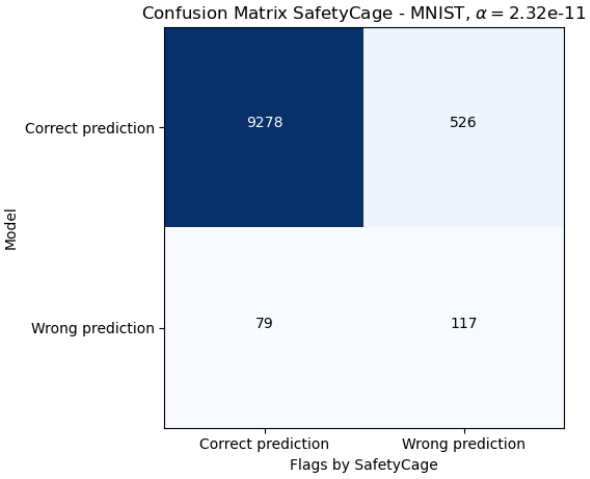

Figure 3. Confusion matrix for the first line in Table 2 with significance level $\alpha = 2.32 \cdot 10^{-11}$.

In critical operations having a high recall is more important than high precision, which is the scenario the SafetyCage is built to tackle. On the other side, even though it is less important, precision cannot be completely neglected, especially if the method needs to be used by real human operators, since too many false alarms would induce little trust. In order to tackle this, we adjusted the significance level α to maximise the MCC scores, rebalancing precision and recall. By looking at the confusion matrix for the configuration with the highest MCC, shown in Figure 3, we can see how of the total 196 examples wrongly classified by the neural network model, 117 (60%) have been correctly detected and flagged by the SafetyCage. At the same time, the SafetyCage wrongly flagged 526 examples that were indeed correctly classified. While this is the source of the poor precision score, it represents only 5.4 % of the total negative samples.

We show in Figure 4 a histogram of the Mahalanobis distance calculated from the penultimate

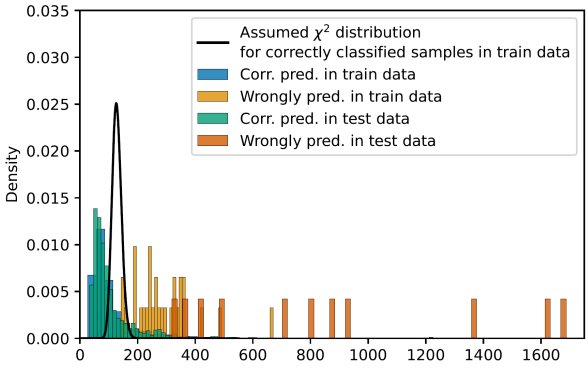

Figure 4. The Mahalanobis distance from the penultimate layer of the model for the digit 1 in MNIST. Also included is the estimated χ^2 distribution of the Mahalanobis distance for the correct predictions of the digit in the training data.

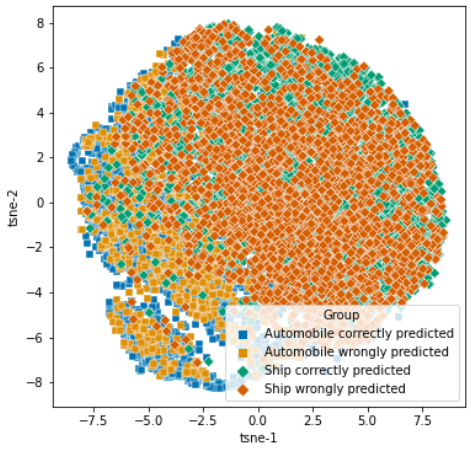

Figure 5. A t-SNE visualisation of the feature space of the penultimate layer of the network for the classes "automobile" and "ship" in the CIFAR-10 dataset and whether the classifier assigned the correct or incorrect label. The distributions of the classes and assigned labels are overlapping.

layer of the classifier. We see how the distance is much larger for wrongly classified samples, which confirms our idea of using it as a metric to evaluate the samples. At the same time, we see that the distribution of the Mahalanobis distance is far from converging to a χ^2. This might indicate that the pre-activation values are not (sufficiently) normally distributed.

3.2 Results on a poor classifier

We additionally trained a model on a more difficult dataset, the CIFAR-10. When constructing the model for MNIST, we tuned the architecture and training to obtain a reasonably decent (but not perfect) classifier. This has not been done for CIFAR-10, where the exact same architecture and training procedure as for MNIST has been used. This resulted in an accuracy of 0.49. The purpose was to test the SafetyCage on a poor underlying classifier.

Figure 5 shows a visualisation of the first two components in the t-SNE decomposition of the pre-activation values in the penultimate layer of the network for the CIFAR-10 classes "automobile" and "ship". In opposition to the MNIST case, we clearly see the poor performance of the classifier: The two classes are not separated, and the wrongly classified samples are no longer appearing close to the edge, but are completely overlapping the distribution of the correctly classified ones.

Poor performace of the classifier clearly impacts also the SafetyCage. In Figure 6 we see how the empirical distribution of the Mahalanobis distance is very far from the assumed χ^2 distribution. As a result, the SafetyCage performance is comparable to random guessing on this example.

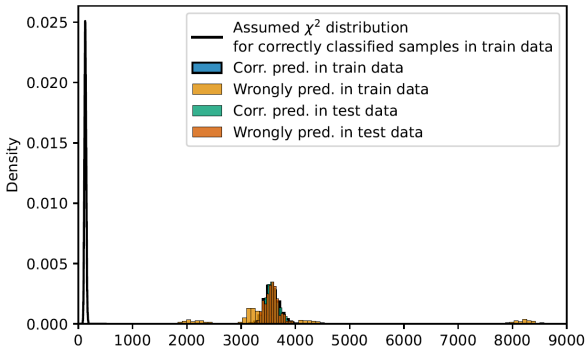

Figure 6. The Mahalanobis distance from the penultimate layer of the model for the "automobile" class in CIFAR-10. Also included the estimated χ^2-distribution of the Mahalanobis distance for the correct predictions in the training data.

4 Discussion and conclusions

We have seen that for a well-performing neural network model, the multivariate SafetyCage introduced in this paper performs well at distinguishing correctly and wrongly classified samples, detecting and flagging 60% of the wrong classifications against a drawback of 5.4% unnecessary flags. This gives acceptably large recall, specificity, and negative predictive value, but low precision. We argue that this can be due to two reasons: The distribution of the pre-activation values being too far from a Normal distribution, leading to the Mahalanobis distance being an inappropriate measure in our setting, or that the distributions of correctly and wrongly predicted samples of the same class are overlapping.

The second case is particularly evident when the model is less-performing such as for the example on the CIFAR-10 data set, where the distribution of

correctly and wrongly classified samples overlap, see Figure 6.

From our results, see Table 1 and Table 2, the two approaches, using the χ^2-asymptotics or the ECDF, yield comparable results, with the ECDF having perhaps a marginal advantage. The small difference between those two approaches might indicate that the Mahalanobis distance does not capture the statistical distance between the distributions of correctly and incorrectly classified samples.

Possible steps for improving the SafetyCage could be: Reformulating the hypothesis test to test the predicted label against every other class, or construct it as a likelihood-ratio test comparing the probability distributions of both correctly and wrongly predicted samples. We expect that this could strengthen the test, leading to fewer false positives. Using a different distance measure than the Mahalanobis distance could also improve performance, particularly when the distribution of the (pre)-activation values is far from Gaussian; see also comment above regarding the χ^2 vs ECDF approach. We also note in passing that one could additionally use the F-beta score, in the cases where it can be interesting to weigh between Type I and Type II errors.

The procedure introduced in this paper can be successfully used in an online learning fashion [21] for two reasons: Firstly, the significance level α that maximises the performance, with respect to some measure, can be learnt based on new incoming data. Secondly, new incoming data can be used to re-fit the parameters of the parametric distributions assumed in the layers of the neural network, in this case μ and Σ in the Multivariate Gaussian distribution. However, doing this in practice would require new incoming data to be manually labeled, which may be expensive. A cheaper alternative would be to hand-pick particular samples of interest to be labeled. This can be chosen based on the p-values returned from the SafetyCage.

Even though the SafetyCage has been presented on a simple MLP neural network, there is in principle nothing stopping it from being generalized to any feed-forward layers, such as convolutional or pooling, or more complex architectures sporting e.g. skip connections.

Lastly, a radically different approach would be to develop a univariate SafetyCage, that is, treating the (pre-)activations of the neurons as coming from univariate random variables. Our initial work in this direction is interesting, and we plan to publish it separately, with a comparison to the multivariate SafetyCage that has here been presented in detail.

References

[1] D. Marmanis, M. Datcu, T. Esch, and U. Stilla. "Deep Learning Earth Observation Classification Using ImageNet Pretrained Networks". In: *IEEE Geoscience and Remote Sensing Letters* 13.1 (2016), pp. 105–109. DOI: 10.1109/LGRS.2015.2499239.

[2] D. Duarte, F. Nex, N. Kerle, and G. Vosselman. "Multi-Resolution Feature Fusion for Image Classification of Building Damages with Convolutional Neural Networks". In: *Remote Sensing* 10.10 (2018). DOI: 10.3390/rs10101636.

[3] D. Griffiths and J. Boehm. "Rapid Object Detection Systems, Utilising Deep Learning and Unmanned Aerial Systems (UAS) for Civil Engineering Applications". In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLII-2 (2018), pp. 391–398. DOI: 10.5194/isprs-archives-XLII-2-391-2018.

[4] J. J. Bird, D. R. Faria, L. J. Manso, P. P. S. Ayrosa, and A. Ekárt. "A study on CNN image classification of EEG signals represented in 2D and 3D". In: *Journal of Neural Engineering* 18.2 (2021). DOI: 10.1088/1741-2552/abda0c.

[5] I. O. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, A. Steiner, D. Keysers, J. Uszkoreit, M. Lucic, and A. Dosovitskiy. "MLP-Mixer: An all-MLP Architecture for Vision". In: (2021). arXiv:2105.01601. arXiv: 2105.01601.

[6] C. A. Barajas, J. C. Polf, and M. K. Gobbert. "Deep residual fully connected neural network classification of Compton camera based prompt gamma imaging for proton radiotherapy". In: *Frontiers in Physics* 11 (2023). DOI: 10.3389/fphy.2023.903929.

[7] T. DeVries and G. W. Taylor. *Learning Confidence for Out-of-Distribution Detection in Neural Networks*. arXiv:1802.04865. 2018. arXiv: 1802.04865.

[8] K. Lee, K. Lee, H. Lee, and J. Shin. "A Simple Unified Framework for Detecting Out-of-Distribution Samples and Adversarial Attacks". In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. arXiv.1807.03888. Curran Associates Inc., 2018, pp. 7167–7177.

[9] J. Henriksson, C. Berger, M. Borg, L. Tornberg, C. Englund, S. R. Sathyamoorthy, and S. Ursing. "Towards Structured Evaluation of Deep Neural Network Supervisors". In: (2019). arXiv:1903.01263. arXiv: 1903.01263.

[10] F. D'Angelo and C. Henning. "On out-of-distribution detection with Bayesian neural networks". In: (2022). arXiv:2110.06020. arXiv: 2110.06020.

[11] D. Hendrycks and K. Gimpel. "A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks". In: (2016). arXiv:1610.02136. arXiv: 1610.02136.

[12] W. K. Härdle and L. Simar. "Theory of the Multinormal". In: *Applied Multivariate Statistical Analysis*. Ed. by W. K. Härdle and L. Simar. Berlin, Heidelberg: Springer, 2015, p. 184. DOI: 10.1007/978-3-662-45171-7_5.

[13] H. B. Mann and A. Wald. "On Stochastic Limit and Order Relationships". In: *The Annals of Mathematical Statistics* 14.3 (1943), pp. 217–226. DOI: 10.1214/aoms/1177731415.

[14] R. A. Fisher. *Statistical Methods for Research Workers*. Springer Series in Statistics. New York, NY: Springer, 1992, pp. 66–70. DOI: 10.1007/978-1-4612-4380-9_6.

[15] Y. Liu and J. Xie. "Cauchy combination test: a powerful test with analytic p-value calculation under arbitrary dependency structures". In: *Journal of the American Statistical Association* 115.529 (2020), pp. 393–402. DOI: 10.1080/01621459.2018.1554485.

[16] Y. LeCun and C. Cortes. "MNIST handwritten digit database". In: (2010). URL: http://yann.lecun.com/exdb/mnist/.

[17] A. Krizhevsky. "Learning Multiple Layers of Features from Tiny Images". In: (2009). URL: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

[18] D. Chicco and G. Jurman. "The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation". In: *BMC Genomics* 21.1 (Jan. 2020), p. 6. DOI: 10.1186/s12864-019-6413-7.

[19] L. van der Maaten and G. Hinton. "Visualizing Data using t-SNE". In: *Journal of Machine Learning Research* 9.86 (2008), pp. 2579–2605. URL: http://jmlr.org/papers/v9/vandermaaten08a.html.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830. DOI: 10.3390/info11040193.

[21] S. C. H. Hoi, D. Sahoo, J. Lu, and P. Zhao. "Online learning: A comprehensive survey". In: *Neurocomputing* 459 (2021), pp. 249–289. DOI: 10.1016/j.neucom.2021.04.112.