Unit-level surprise in neural networks

Cian Eastwood[*†] **Ian Mason**[*†] **Christopher K. I. Williams** [†‡]

[†] School of Informatics, University of Edinburgh
[‡] Alan Turing Institute, London

Abstract

To adapt to changes in real-world data distributions, neural networks must update their parameters. We argue that unit-level surprise *should* be useful for: (i) determining which few parameters should update to adapt quickly; and (ii) learning a modularization such that few modules need be adapted to transfer. We empirically validate (i) in simple settings and reflect on the challenges and opportunities of realizing both (i) and (ii) in more general settings.

1 Introduction

Neural networks have achieved remarkable successes on problems with *independent and identically-distributed* (IID) data [23]. However, real-world data is not IID—environmental conditions shift [8], new tasks are encountered [27], and agents alter their behaviour [6]. Ideally, we would like our networks to transfer or adapt *quickly* to such *out-of-distribution* (OOD) data [30].

Fast adaptation to OOD or *shifted* data is often achieved by updating only a few parameters—the final layer of an ImageNet-pretrained network [13, 42], the batch-normalization layers [24, 34, 38], or specific mechanisms or modules [31, 15]. For a given problem or (expected) *shift type*, a human usually decides: (i) which few parameters should be updated; and (ii) what architecture or *modularization* will allow the network to be adapted by updating only a few parameters (e.g. Rebuffi et al. [34] use residual adapters for domain adaptation). However, we often do not know what type of shift will occur and thus how to decide (i) and (ii).

In this work we argue that *unit-level surprise* can help identify *what* has changed, and this in turn can be used to: (i) infer which few parameters should be updated to adapt quickly by transferring past knowledge; and (ii) learn an appropriate modularization such that very few modules need be adapted to transfer. For (i), we propose the use of unit-level surprise—*"This looks the same as before, no need for me to update!"*— along with modulatory *update-in-progress* signals—*"This is being dealt with by someone else, hold tight!"*. These two additional pieces of information at the unit-level allow the network to update only the appropriate parameters to facilitate fast adaptation. This idea is depicted in Figure 1 and further motivated in Appendix A. For (ii), we propose to use the number of surprised units as a *proxy score* for the number of parameters that need to be adapted to a given shift, which can then be optimized to learn a modularization that changes sparsely and thus can be adapted quickly.

As a first step, we empirically validate the usefulness of unit-level surprise by focusing on use case (i) above. Through experiments on shifted EMNIST [7] datasets, we show that unit-level surprise can be used to create shift-dependent fine-tuning strategies that often align with our intuition about which few parameters should update. We then critique these results, identifying several challenges and opportunities for using unit-level surprise in more general settings where its full potential could be unlocked. Despite such challenges, the "beauty" of unit-level surprise—its alignment with our intuitions about how to handle OOD data, its seemingly strong signal for learning how to modularize knowledge, and its relations to neuroscience concepts like metaplasticity [1, 2, 41] and the free-energy principle [12]—convinces us that it *should be better* for realizing (i) and (ii) in more general settings.

[*]Equal contribution.

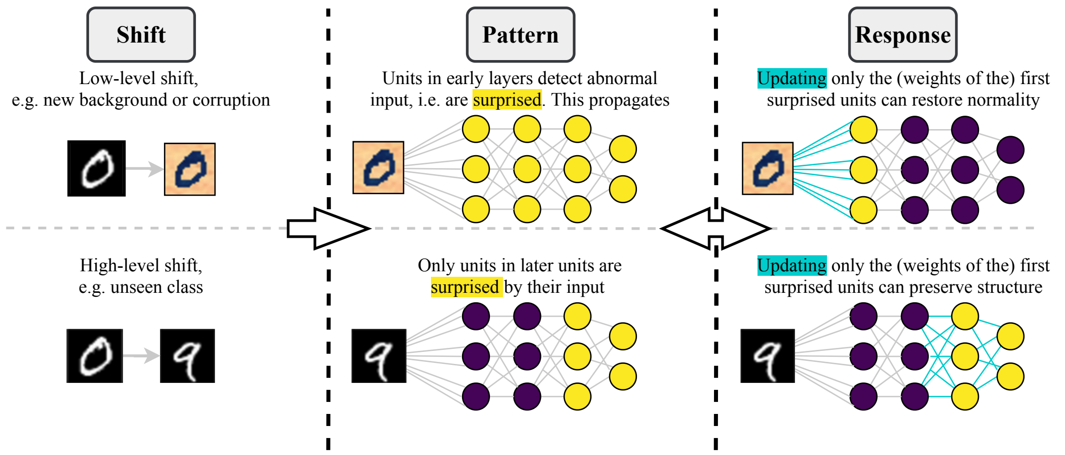

Shift
Pattern
Response
Low-level shift, e.g. new background or corruption
Units in early layers detect abnormal input, i.e. are surprised. This propagates
Updating only the (weights of the) first surprised units can restore normality
High-level shift, e.g. unseen class
Only units in later units are surprised by their input
Updating only the (weights of the) first surprised units can preserve structure

Figure 1: Unit-level surprise can help determine which few parameters should be updated. Purple units are unsurprised, yellow surprised. Blue indicates the weights to be updated. *Top row:* A low-level shift is noticed by units in the first layer (and all those that follow). By blocking those that follow, units in the first layer prevent unnecessary updates in later layers. *Bottom row:* A high-level shift is only noticed by units in later layers, so those in earlier layers don't need to update.

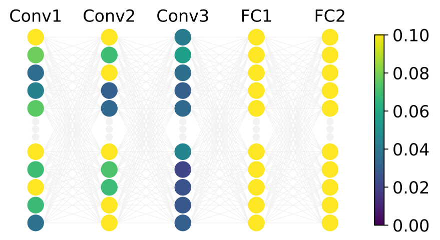
2 Initial validation

In this section we provide an initial empirical validation of the usefulness of unit-level surprise by showing how it can be used to determine which few parameters should be adapted to a given shift.

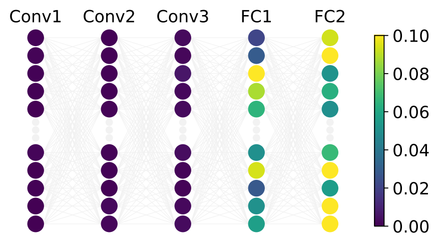
Experimental setup. We train a simple 5-layer network with 3 convolutional (conv) and 2 fully-connected (FC) layers on 37 of the 47 classes in EMNIST [7]. We then adapt this network to new data from 1 of 10 distribution shifts (see Appendix B.1) for which we expect it to be optimal to update different parts of the network. In particular, we use 7 low-level shifts where we expect only the early layers to need to adapt (e.g. new background) and 3 high-level shifts where we expect only the later layers to need to adapt (e.g. held-out/unseen classes). Full experimental details are in Appendix B.2, and code is available at `https://github.com/cianeastwood/unit-level-surprise`.

Calculating unit-level surprise. For each unit in a network we store the 1D distribution of its activations under the training data, $P(A)$, and compare this with the distribution of activations under a shifted data distribution, $Q(A)$. More specifically, we parameterize $P(A)$ and $Q(A)$ as softly-binned histograms [40] and infer the parameters of these distributions from the training and shifted data respectively. We then calculate the (Bayesian) surprise of a unit as $s(A) = D_{KL}(Q(A)||P(A))$ [19]. We discuss this quantity in Appendix B.3 and how to calculate it from bin counts in Appendix B.4.

Surprise patterns. Together, a network's surprise values yield shift-dependent patterns. These patterns, shown in Fig. 2, help us to understand the level of abstraction at which a shift occurs. For example, when the colour of the EMNIST characters is changed under the *crystals* shift (see Fig. 4 in Appendix B.1), we intuitively expect the low-level filters (e.g. black-and-white edge detectors) to be surprised. Fig. 2a shows this to indeed be the case with units in the first layer exhibiting high surprise. Naturally, the abnormal activations which surprised these units propagate through the rest of the network causing the succeeding units to also be surprised. In contrast, when the network is presented with unseen classes, we intuitively expect the low-level features to be unsurprised (similar distribution of black-and-white edges) but the later layers to be surprised (new combinations of low-level features). Fig. 2b shows this to indeed be the case with units in the later layers surprised while those in earlier layers are not.



(a) Low-level shift (crystals bckgr.)



(b) High-level shift (unseen classes)

Figure 2: Surprise patterns.

Creating an update rule. Once units have been equipped with the ability to calculate their surprise, how can we best

Table 1: 5-shot accuracy when training single layers, all layers (SGD), using FlexTune [35], or using a surprise-based update rule. *L:* low-level shifts (avg), *H:* high-level shifts (avg), *All:* all shifts (avg).

	Conv1	Conv2	Conv3	FC1	FC2	SGD	FlexTune	Upd. Rule
L	82.7 ± 0.4	70.2 ± 1.2	61.4 ± 0.2	56.3 ± 0.2	51.5 ± 0.3	71.3 ± 2.1	82.7 ± 0.4	$\mathbf{82.9 \pm 0.5}$
H	0.0 ± 0.0	0.3 ± 0.5	25.4 ± 3.8	80.6 ± 4.6	$\mathbf{94.5 \pm 0.9}$	74.4 ± 5.2	$\mathbf{94.5 \pm 0.9}$	79.2 ± 4.2
All	57.9 ± 0.3	49.3 ± 0.8	50.6 ± 1.2	63.6 ± 1.2	64.4 ± 0.3	72.2 ± 2.9	$\mathbf{86.3 \pm 0.5}$	81.8 ± 1.2

Table 2: Accuracy for a varying number of samples-per-class, averaged over different shifts.

	2	5	10	20	50	2000
SGD	60.0 ± 1.9	72.2 ± 2.9	78.9 ± 0.5	82.6 ± 0.2	86.1 ± 0.3	92.1 ± 0.1
FlexTune	$\mathbf{82.6 \pm 0.7}$	$\mathbf{86.3 \pm 0.5}$	$\mathbf{87.8 \pm 0.4}$	$\mathbf{89.2 \pm 0.2}$	$\mathbf{90.4 \pm 0.2}$	$\mathbf{92.5 \pm 0.0}$
Upd. Rule	66.3 ± 2.5	81.8 ± 1.2	86.2 ± 0.6	88.2 ± 0.4	89.7 ± 0.1	92.4 ± 0.0

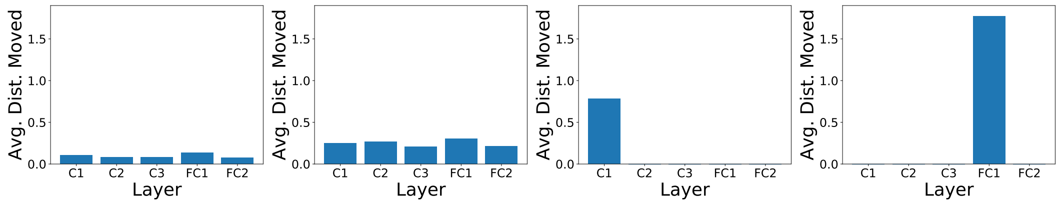
make use of this? We investigate whether or not surprise can give us some indication of the specific few units to update in order to adapt quickly to new data. In particular, we focus on few-shot learning as updating (the parameters of) fewer units should improve sample-efficiency. We consider three possible adaptation strategies or shift *responses*:

1. **SGD.** Ignore surprise patterns and use *stochastic gradient descent* (SGD) to update *all* parameters.
2. **FlexTune [35]**. Ignore surprise patterns and select the best individual layer to update using a test-domain validation set. This serves as an upper-bound on single-layer performance.
3. **Update rule based on unit-level surprise.** Use a rule that determines whether or not a unit should update based on surprise values.We design an update rule with the goal of leveraging unit-level surprise to preserve structure by preventing unnecessary updates. This rule can be summarised as *"update only if: (i) you are surprised; and (ii) your parents are not"*, with the key insight being that surprised parents (units in the preceding layer that are connected to it) indicate that the surprise-causing shift may be resolved in earlier layers. Here, (i) ensures that only affected units update—*"This looks the same as before, no need for me to update!"*—while (ii) acts as a modulatory *update-in-progress* signal from a unit's parents—*"This is being dealt with by someone else, hold tight!"*. We formally define this update rule in Appendix B.5.

Results. Table 1 gives the 5-shot results for training individual network layers and using each of our 3 responses. We see that low-level shifts are best dealt with by fine-tuning the first layer and high-level shifts the last. This aligns with our intuition about which units should update and also with the surprise patterns in Figure 2. Without *a priori* knowledge of the type of shift that will occur (choosing Conv1 for low-level shifts, FC2 for high-level) or exhaustively searching over all possibilities using an oracle-like test-domain validation set (FlexTune), we achieve the best average performance over low- and high-level shifts with our surprise-based update rule—where units only update if they are sufficiently surprised and their parents are not. These results show that it is not always optimal to fine-tune only the later layers of a network (as also found in [35]) and demonstrate that unit-level surprise can indeed help determine which few parameters should update to adapt quickly (i.e. with few samples) to new data. Results for individual shifts are given in Appendix E.

We also evaluate each of these adaptation strategies or responses as a function of the number of shifted samples that are available. As shown in Table 2, all strategies perform well using large amounts of data (2000 samples-per-class). However, as the number of samples-per-class drops, it becomes more important to update few parameters, and thus to choose the right strategy. With 5-50 samples-per-class, simple SGD (training all layers) is outperformed by both other strategies which seek to maintain structure by selecting few units to update. While FlexTune consistently outperforms the update rule (by a small margin), it requires training 5 different models on labelled test-domain data (one per layer) in order to select the best layer to update.

Which units does SGD update? To visualize which parameters are updated we calculate "how far" each unit's input parameters are moved and use this to compare SGD and our update rule. Specifically, we calculate the Euclidean distance between a parameter's value before and after adaptation, then average these distances layerwise to see which layers are being updated by each adaptation strategy. Figures 3a & 3b show that SGD moves all layers of the network similarly and makes no distinction between different types of shift. This confirms that a simple gradient signal is not sufficient to select



(a) Low, SGD (b) High, SGD (c) Low, Upd. Rule (d) High, Upd. Rule

Figure 3: Mean distance moved for units in each layer of a 5-layer CNN when using SGD and the update rule for low (crystals background shift) and high (H1, 5 unseen classes) level shifts.

which few parameters should update to a given shift, explaining the results in Table 1. In contrast, Figures 3c & 3d show that our update rule exhibits shift-specific behaviour which aligns with the results in Table 1—moving early layers for low-level shifts and later layers for high-level shifts.

3 Critique

While we achieved some promising results in carefully-designed settings, it remains unclear how unit-level surprise should be used in more realistic settings to determine which few parameters to update. Below we discuss the main challenges discovered during the initial idea validation above.

Networks are not sufficiently modular to show sparse surprise patterns. For the various shifts we considered, networks did not exhibit sparse or modular surprise patterns—low-level shifts surprised almost every unit in the network (Fig. 2a), while high-level shifts surprised all units in both FC layers (Fig. 2b). This then affects the updates of our update rule, with single, entire layers being updated rather than a modular selection of units (Fig. 3). We posit that such patterns of surprise occur because, using standard training techniques, simple neural networks do not form modules or mechanisms that change independently (and thus can be surprised independently). Moreover, we find ourselves asking if it is *ever* possible to learn such modules or mechanisms that change sparsely (i.e. align with the shifts in data distribution) without explicitly optimizing for it at training time over multiple shifts.

Parents *and* children may need to update. As discussed above, our update rule tends to update single layers, and this can lead to sub-optimal solutions (e.g. updating only FC1 for high-level shifts). We attribute this to the fact that surprised parent units tend to stay surprised, continually blocking their children from updating. One solution would be to update $P(A)$ with samples from $Q(A)$ as training progresses, for units that are being adapted. Intuitively, this would allow parent units to gradually become less surprised by the new data, eventually freeing their children to also update.

Surprise may not be sufficient. As shown in Table 1 and Figure 3, the update rule does not always select the "optimal" units to update (updating FC1 rather than FC2 for high-level shifts). This raises some questions—is surprise alone sufficient to determine which units should update? What other unit-level information may be required? Here we list some examples: (a) *gradient magnitude*—may help us to update FC2 rather than FC1 to adapt to high-level shifts; (b) *task importance* [21, 43]—may help in continual learning settings to best preserve performance on past tasks when adapting to new ones; (c) *parameter uncertainty*—may help speed-up learning by only updating the parameters with high uncertainty [3], akin to Bayesian optimization. Additionally, one can imagine shifts that are best resolved at a mid- or high-level but cause changes to the low-level feature distributions—e.g. if we occlude part of a digit, the low-level features (edges) should still be valid, we just need to update how they combine to form the class label. However, the changed low-level feature distributions (some edges are occluded) will surprise units in the early layers, causing them to update. Another similar challenge is that of changing feature frequency. In particular, if we take the view of units as feature detectors, where a specific unit is sensitive to a specific feature, then the activation distributions of units will change if the features they detect occur with a different frequency (i.e. appear more or less often). In this situation, a unit can show high surprise but we may not wish to update its parameters if we still wish to detect this feature. One potential solution to these problems is to explore alternative measures that distinguish between being more or less surprised than expected (see Appendix C).

Lack of interpretability makes validation difficult. Finally, it is difficult to come up with experimental setups where we know what the "right" units to update are. This makes the design and validation of update rules quite challenging. While we may hope for interpretable edges-parts-wholes feature hierarchies (see Figure 9 in Appendix E.1), this is often not the case. In fact, it can take as little as a change in random seed to change the semantic meaning of features, and with it, the optimal units to update. Work on unit-level interpretability [14, 29] may eventually help, but it is not currently at a level to decide which units should update for a given shift.

4 Future work

As discussed above, creating a general update rule using unit-level surprise is challenging as: (i) neural networks are not very modular by default; and (ii) surprise may not be sufficient to determine "who" should update. Below we discuss two potential avenues to overcome these two challenges.

Learning modular structure with unit-level surprise. Recent work in causal discovery assumes that the ground-truth data-generative process consists of independent mechanisms or modules, with many modules expected to behave similarly across different tasks and environments [5, 31, 32, 36, 37]. Thus, if an appropriate modular representation of the world has been learned, *very few modules should need to be adapted in order to transfer* [5, 15, 32, 37]. Bengio et al. [5] exploit this through a meta-learning objective that uses *adaptation speed* to optimize the way in which knowledge is represented or modularized. Here, adaptation speed is a proxy score for the number of modules and parameters that need be adapted and ultimately for how well the learned modularization fits the underlying causal dependencies. We believe surprise could provide an *alternative proxy score* (e.g. the number[2] of surprised units or groups of units) that is easier to meta-optimize (no inner-loop steps) and arguably a more direct measure of the number of modules and parameters that need be adapted.

Learning optimizers to better utilize unit-level surprise. As discussed in the previous section, unit-level surprise may not be sufficient for some shifts—further information may be required to determine "who" should update. This makes it difficult to handcraft a general update rule as one must specify how surprise should be used with this other information. In fact, even without additional information, there is still an enormous space to be explored surrounding how best to use unit-level surprise, e.g. setting the learning rate based on the surprise by using *soft* thresholding. One solution is to *learn* an optimizer [4, 33], or the parameters of an update rule [25], across multiple shifts. However, learned optimizers are notoriously difficult to train [28] and it can be difficult to beat the heavily-tuned few-shot benchmarks against which they are often compared [16].

5 Conclusion

In this work we provided a preliminary validation of the usefulness of unit-level surprise in neural networks. In particular, we showed that it can be useful for analysing *where* dataset shifts are noticed in a network and for devising *shift-dependent* fine-tuning strategies. We also discussed some of the "beauty" that makes us believe that unit-level surprise *should* be useful in more general settings—it often aligns with our intuitions about how to handle OOD data, it seems a strong signal for learning how to modularize knowledge such that only a few modules are surprised (and thus need be adapted), and it has several neuroscience relations such as metaplasticity and the free-energy principle. However, in formally specifying how to use unit-level surprise in simple settings, we encountered several challenges that made us wary of tackling the more complicated settings in which its full potential could be realized.

Acknowledgements

We thank Matthias Hennig, Tim Hospadales, Amos Storkey, and Sébastian Arnold for helpful discussions and comments. CE acknowledges support from The National University of Ireland via his Travelling Studentship in the Sciences. IM is supported by the Engineering and Physical Sciences Research Council (EPSRC).

[2]A continuous proxy can be used for differentiability.

References

[1] W. C. Abraham. Metaplasticity: tuning synapses and networks for plasticity. *Nature Reviews Neuroscience*, 9(5):387–387, 2008.

[2] W. C. Abraham and M. F. Bear. Metaplasticity: the plasticity of synaptic plasticity. *Trends in Neurosciences*, 19(4):126–130, 1996.

[3] L. Aitchison, J. Jegminat, J. A. Menendez, J.-P. Pfister, A. Pouget, and P. E. Latham. Synaptic plasticity as bayesian inference. *Nature Neuroscience*, 24(4):565–571, 2021.

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

[5] Y. Bengio, T. Deleu, N. Rahaman, N. R. Ke, S. Lachapelle, O. Bilaniuk, A. Goyal, and C. Pal. A meta-transfer objective for learning to disentangle causal mechanisms. In *International Conference on Learning Representations*, 2020.

[6] L. Buesing, T. Weber, Y. Zwols, N. Heess, S. Racaniere, A. Guez, and J.-B. Lespiau. Woulda, coulda, shoulda: Counterfactually-guided policy search. In *International Conference on Learning Representations*, 2019.

[7] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik. EMNIST: an extension of MNIST to handwritten letters. *arXiv preprint arXiv:1702.05373*, 2017.

[8] D. Dai and L. Van Gool. Dark model adaptation: Semantic image segmentation from daytime to nighttime. In *International Conference on Intelligent Transportation Systems*, pages 3819–3824. IEEE, 2018.

[9] C. Eastwood, I. Mason, C. K. I. Williams, and B. Schölkopf. Source-Free Adaptation to Measurement Shift via Bottom-Up Feature Restoration. *arXiv:2107.05446*, 2021.

[10] S. M. A. Eslami, D. J. Rezende, F. Besse, F. Viola, A. S. Morcos, M. Garnelo, A. Ruderman, A. A. Rusu, I. Danihelka, K. Gregor, D. P. Reichert, L. Buesing, T. Weber, O. Vinyals, D. Rosenbaum, N. Rabinowitz, H. King, C. Hillier, M. Botvinick, D. Wierstra, K. Kavukcuoglu, and D. Hassabis. Neural scene representation and rendering. *Science*, 360(6394):1204–1210, 2018.

[11] D. E. Feldman. Synaptic mechanisms for plasticity in neocortex. *Annual Review of Neuroscience*, 32:33–55, 2009.

[12] K. Friston. The free-energy principle: a unified brain theory? *Nature reviews neuroscience*, 11(2):127–138, 2010.

[13] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.

[14] G. Goh, N. C. †, C. V. †, S. Carter, M. Petrov, L. Schubert, A. Radford, and C. Olah. Multimodal neurons in artificial neural networks. *Distill*, 2021. https://distill.pub/2021/multimodal-neurons.

[15] A. Goyal, A. Lamb, J. Hoffmann, S. Sodhani, S. Levine, Y. Bengio, and B. Schölkopf. Recurrent independent mechanisms. In *International Conference on Learning Representations*, 2021.

[16] T. M. Hospedales, A. Antoniou, P. Micaelli, and A. J. Storkey. Meta-learning in neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[17] S. R. Hulme, O. D. Jones, and W. C. Abraham. Emerging roles of metaplasticity in behaviour and disease. *Trends in Neurosciences*, 36(6):353–362, 2013.

[18] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[19] L. Itti and P. Baldi. Bayesian surprise attracts human attention. *Vision research*, 49(10):1295–1306, 2009.

[20] P. S. Katz. *Beyond Neurotransmission: Neuromodulation and its Importance for Information Processing*. Oxford University Press New York:, 1999.

[21] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences of the United States of America*, 114(13):3521–3526, 2017.

[22] S. Kullback. *Information theory and statistics*. Wiley, 1959.

[23] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[24] Y. Li, N. Wang, J. Shi, J. Liu, and X. Hou. Revisiting batch normalization for practical domain adaptation. In *International Conference on Learning Representations Workshop*, 2017.

[25] Z. Li, F. Zhou, F. Chen, and H. Li. Meta-SGD: Learning to learn quickly for few-shot learning. *arXiv:1707.09835*, 2017.

[26] D. J. MacKay. Information-based objective functions for active data selection. *Neural computation*, 4(4):590–604, 1992.

[27] M. McCloskey and N. J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of Learning and Motivation*, volume 24, pages 109–165. Elsevier, 1989.

[28] L. Metz, N. Maheswaranathan, J. Nixon, D. Freeman, and J. Sohl-Dickstein. Understanding and correcting pathologies in the training of learned optimizers. In *International Conference on Machine Learning*, pages 4556–4565, 2019.

[29] C. Olah, A. Satyanarayan, I. Johnson, S. Carter, L. Schubert, K. Ye, and A. Mordvintsev. The building blocks of interpretability. *Distill*, 2018. https://distill.pub/2018/building-blocks.

[30] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[31] G. Parascandolo, N. Kilbertus, M. Rojas-Carulla, and B. Schölkopf. Learning independent causal mechanisms. In *International Conference on Machine Learning*, pages 4036–4044, 2018.

[32] J. Peters, D. Janzing, and B. Schölkopf. *Elements of Causal Inference: Foundations and Learning Algorithms*. MIT Press, 2017.

[33] S. Ravi and H. Larochelle. Optimization as a model for few-shot learning. In *International Conference on Learning Representations*, 2017.

[34] S.-A. Rebuffi, H. Bilen, and A. Vedaldi. Learning multiple visual domains with residual adapters. In *Advances in Neural Information Processing Systems*, pages 506–516, 2017.

[35] A. Royer and C. Lampert. A flexible selection scheme for minimum-effort transfer learning. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2191–2200, 2020.

[36] B. Schölkopf, D. Janzing, J. Peters, E. Sgouritsa, K. Zhang, and J. Mooij. On causal and anticausal learning. *arXiv:1206.6471*, 2012.

[37] B. Schölkopf, F. Locatello, S. Bauer, N. R. Ke, N. Kalchbrenner, A. Goyal, and Y. Bengio. Toward causal representation learning. *Proceedings of the IEEE*, 109(5):612–634, 2021.

[38] D. Wang, E. Shelhamer, S. Liu, B. Olshausen, and T. Darrell. TENT: Fully test-time adaptation by entropy minimization. In *International Conference on Learning Representations*, 2021.

[39] S.-Z. Wang and H. W. Tao. History matters: illuminating metaplasticity in the developing brain. *Neuron*, 64(2):155–157, 2009.

[40] Y. Yang, I. G. Morillo, and T. M. Hospedales. Deep neural decision trees. In *ICML Workshop on Human Interpretability in Machine Learning (WHI)*, 2018.

[41] A. X. Yee, Y.-T. Hsu, and L. Chen. A metaplasticity view of the interaction between homeostatic and Hebbian plasticity. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 372(1715), 2017.

[42] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, pages 818–833, 2014.

[43] F. Zenke, B. Poole, and S. Ganguli. Continual learning through synaptic intelligence. In *34th International Conference on Machine Learning*, pages 3987–3995, 2017.

A Example to further motivate surprise

Imagine you are given a network which has been trained to classify different Ford car models outdoors (in green pastures), and asked to classify pictures of these same models indoors (in a showroom). Intuitively, most of the learned function is still valid so only a small number of parameters need to be tweaked, requiring only a small number of indoor examples to do so. Perhaps tweaking parameters in the first layer would be sufficient, re-extracting the same features from e.g. the darker pixels of indoor images as were previously extracted from the outdoor images. Similarly, if you were asked to classify pictures of Fiat cars, you may expect that only a small number of parameters need to be adapted (likely in later layers this time), perhaps adjusting the (conceptual) wheel and mirror detectors for the smaller wheels and more rounded mirrors of Fiats. However, as we discuss in Section 2, minimizing a standard loss function (e.g. cross-entropy) with *stochastic gradient descent* (SGD) does not result in such intuitive updates—all of the network's parameters are updated and learnt structure is unnecessarily destroyed. These gradients tell us which parameters *can* reduce the error, but not which parameters *should* reduce the error in order to maximally-transfer knowledge and thus speed-up learning. Doing so requires additional information—such as unit-level surprise.

B Implementation details

B.1 Data

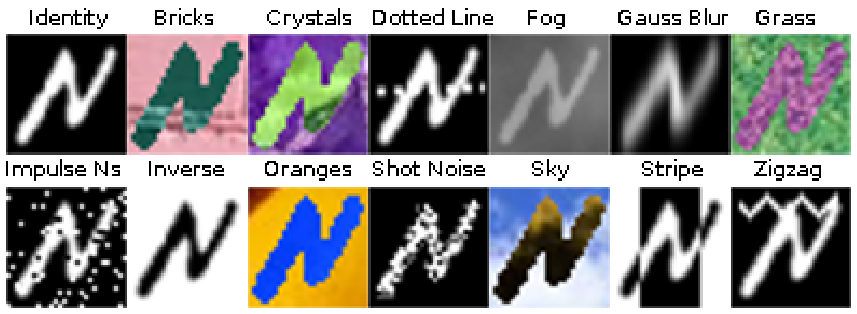


Figure 4: EMNIST-DA shifts. Figure adopted from [9].

Our networks are trained using the 47-class EMNIST dataset [7] ("identity" shift in Figure 4). We use 2000 samples per class from the training split with the remaining 400 forming a validation set, we report results on the separate test set also containing 400 examples per class.

We adapt to 10 new data distributions—7 low-level shifts from EMNIST-DA [9] and 3 high-level label shifts. From the 14 EMNIST-DA shifts depicted in Figure 4, we chose 7 shifts that adversely affect accuracy (without adaptation) and intuitively affect the early convolutional filters: crystals, fog, gaussian blur, grass, impulse noise, sky and stripe. To create the 3 label shifts, we train our networks on only the first 37 classes of EMNIST and then choose 5 of the 10 unseen classes three times from the range $[38, 47]$ to arrive at: H1:$[38, 39, 40, 41, 42]$, H2:$[43, 44, 45, 46, 47]$, and H3:$[38, 40, 42, 44, 46]$.

To evaluate sample efficiency we run experiments with varying amounts of data, we experiment with using 2, 5, 10, 20 and 50 samples per class as well as using all the data (2000 samples per class). The full results of these experiments are given in Appendix E.

B.2 Experimental setup

Table 3 provides the architectural details of the simple 5-layer convolutional neural network (CNN) that we use. During pre-training we use dropout between the layers, for adaptation we do not as it unnecessarily complicates the propagation of surprise and makes little difference to the final results.

During pre-training and adaptation we use a batch size of 256. We pre-train for 150 epochs with a learning rate of 0.01. Due to using small amount of data, during adaptation we train with early stopping for a maximum of 100 epochs using a patience of 10. We use a learning rate of 0.1 for all experiments except for when fine-tuning all layers simultaneously which requires a learning rate of 0.01 to prevent divergence. We optimize using stochastic gradient descent with momentum set to 0.9. For experiments using our update rule the thresholds α and β in Equation 3 are set to 0.01. Experiments are run over 3 seeds from which we report a mean and one standard deviation.

Table 3: Architecture of the CNN used. For conv. layers, the weights-shape is: *num. input channels* $\times$ *num. output channels* $\times$ *filter height* $\times$ *filter width*.

Layer	**Weights-Shape**	**Stride**	**Padding**	**Activation**	**Dropout Prob.**
Conv	$3 \times 64 \times 5 \times 5$	2	2	ReLU	0.1
Conv	$64 \times 128 \times 3 \times 3$	2	2	ReLU	0.3
Conv	$128 \times 256 \times 3 \times 3$	2	2	ReLU	0.5
FC	6400×128	N/A	N/A	ReLU	0.5
FC	128×47	N/A	N/A	Softmax	0

B.3 Measuring unit-level surprise

A single unit in a feed-forward neural network outputs an activation $a = g(\mathbf{w}^T\mathbf{h} + b)$, where $\mathbf{h}$ is the hidden unit activations of the previous layer, $\mathbf{w}$ the learned weight vector, b the learned bias and g some non-linearity. During training a unit can store a distribution $P(A)$ which captures the distribution that its activation can take. A unit is surprised by new data if the activation distribution changes, i.e. $P(A) \neq Q(A)$, where $Q(A)$ is the distribution of the unit's activations under the new data. We quantify surprise using the KL-divergence from $P(A)$ to $Q(A)$, i.e. $s(A) = D_{KL}(Q(A)||P(A))$ [19, 26].[3]

The surprisal (or information content) of an event $X = x$, with $X \sim P(X)$, is given by $\log(1/P(x))$. Intuitively, this quantity represents how "surprised" we are to see $X = x$, with unlikely events having high surprisal. Surprise itself is a somewhat overloaded term but can be used to describe the entropy, $H(X) = -\sum P(x) \log P(x)$, that is the expected surprisal. If we now receive a sample of a different random variable $Y = y$, $Y \sim Q(Y)$, but we have assumed as a prior that we are receiving samples from $P(X)$, the amount of additional surprisal we receive on account of our assumption is $\log(1/P(X = y)) - \log(1/Q(Y = y))$. The expected value of this quantity over Q is the KL-Divergence $D_{KL}(Q||P)$ [22], which is the expected surprisal of receiving samples from Q when we have assumed the distribution to be P. We can also interpret $D_{KL}(Q||P) = H(Q, P) - H(Q)$ as the expected extra message-length per datum that must be communicated if a code that is optimal for P is used to communicate Q, compared to a code that is optimal for Q. We have seen this quantity referred to as Bayesian surprise, information gain, asymmetric surprise or simply surprise [10, 19]. Throughout this work we refer to this quantity as surprise for simplicity.

B.4 Calculating surprise from bin counts

After pre-training we parameterize activation distributions with softly-binned histograms. To calculate $P(A)$ we run one further forward-pass of the network over the training data and bin the activations using the same procedure as in [9], with 10 bins, which outputs 10 normalized bin counts $\pi_1^p, \dots \pi_{10}^p$ for each unit. π_i^p represents the probability a falls into bin i and $\sum_{i=1}^{10} \pi_i^p = 1$. The blue curves in Fig. 5 depict examples of such distributions. These distributions can be considered as representing the "normal" activation values of a unit, i.e. the values it expects to take on.

We then receive some data from a new distribution, possibly the same as the pre-training data distribution, which is fed into the network. During adaptation we can parameterize $Q(A)$ in the same way as $P(A)$, using a batch of this new data (Fig. 5–orange curves) to calculate normalized bin counts $\pi_1^q, \dots \pi_{10}^q$ which change as the network learns. The surprise for a unit can then be calculated as

$$s(A) = D_{KL}\left(Q(A; \{\pi_i^q\}_{i=1}^{10}) \mid\mid P(A; \{\pi_i^p\}_{i=1}^{10})\right) = \sum_{i=1}^{10} \pi_i^q \log \frac{\pi_i^q}{\pi_i^p}. \tag{1}$$

[3]For convolutions, when creating $P(A)$ and $Q(A)$ we take each spatial location of a feature map to be one sample of the activation, a.

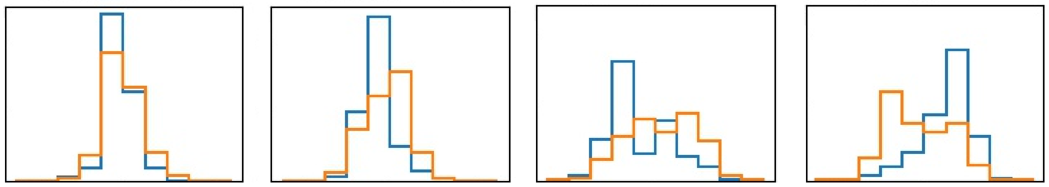


Figure 5: Examples of our histogram parameterizations of $P(A)$, in blue, and $Q(A)$, in orange. When new data is received, the activation distribution changes. From left to right, the surprise values s(A) are approximately 0.1, 0.2, 0.3, 0.4.

B.5 A surprise-based update rule

Let p_i denote the surprise of "parent" unit i in layer l, c_j the surprise of "child" unit j in layer $l+1$, and w_{ij} the weight that connects parent unit i with child unit j. This setup is depicted in Fig. 6 below.

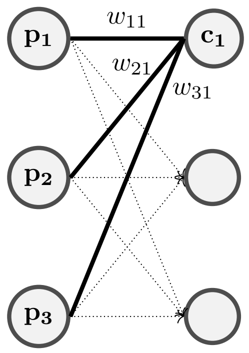


Figure 6: Update rule schematic. The highlighted weights update only if the aggregate parent surprise $\bar{p}_1$ is below some threshold β and the child surprise c_1 is above some threshold α.

For child unit j, we calculate its aggregate parent surprise $\bar{p}_j$ as a weighted average of its parent surprises. More specifically, we calculate

$$\bar{p}_j = \sum_i \frac{|w_{ij}|}{\sum_k |w_{kj}|} \cdot p_i \ , \tag{2}$$

where the normalized weight value $\frac{|w_{ij}|}{\sum_k |w_{kj}|}$ ensures comparable scaling across units in a layer. We then use this to create an update rule where the input weights of child j are updated if and only if child j is surprised (i.e. c_j is above some threshold α) but its parents are not (i.e. $\bar{p}$ is below some threshold β). In particular, we create the following update rule:

$$w_{ij} := w_{ij} - \mathbb{I}[c_j > \alpha]\mathbb{I}[\bar{p}_j < \beta] \cdot \eta \nabla_{w_{ij}} \mathcal{L}, \tag{3}$$

where $\mathbb{I}[c_j > \alpha]$ is an indicator function that is 1 when $c_j > \alpha$ and 0 otherwise, $\mathbb{I}[\bar{p}_j < \beta]$ is similarly defined, η the learning rate, and $\mathcal{L}$ the loss function (cross-entropy in our case). $\mathbb{I}[c_j > \alpha]$ ensures that only surprised units update, and can be compared with *metaplasticity*[4] in the brain. $\mathbb{I}[\bar{p}_j < \beta]$ prevents/blocks simultaneous changes in later units who may also surprised by their input, and can be compared with *neuromodulation*[5] in the brain.

[4]The modification of a neuron's future capacity for learning as a function of recent synaptic history [1, 2]. Believed to regulate the plasticity mechanisms themselves in order to generate adaptive behaviour [11, 17, 39, 41].

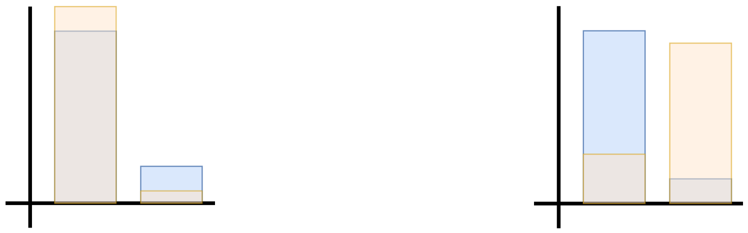
[5]Neuromodulators are neurotransmitters which, instead of conveying excitation or inhibition, change the properties of other neurons or synapses [20].

C Alternative surprise measures

Imagine a unit or feature detector with a bi-modal activation distribution where the modes roughly represent on (detected) and off (not detected). Perhaps such a unit being off *more often* in the new data (as when part of an image is occluded, discussed in Section 3) is not a good signal to update. To differentiate this situation from e.g. the unit being *on* more often in the new data, we can define a new measure which we call the *surprise increase* (SI):

$$SI = H(Q, P) - H(P), \tag{4}$$

where $H(P)$ is the entropy of P and $H(Q, P)$ is the cross-entropy of P relative to Q. Unlike the KL-divergence, SI *can* be negative. SI is negative (i.e. a surprise decrease) when an event that is already quite likely under P becomes even more likely under Q—as shown in Figure 7a. This could be an interesting alternative surprise measurement as it can distinguish between surprise increases and decreases.



(a) Surprise decrease

(b) Surprise increase

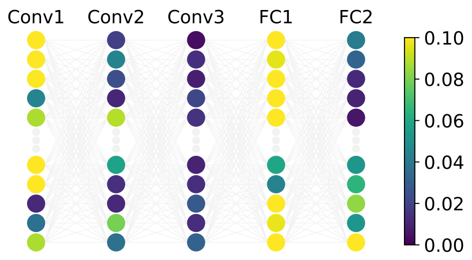
Figure 7: SI illustration. P is blue, Q is orange. For fixed P, SI depends only on $H(Q, P)$.

D Does batch normalization solve the problem?

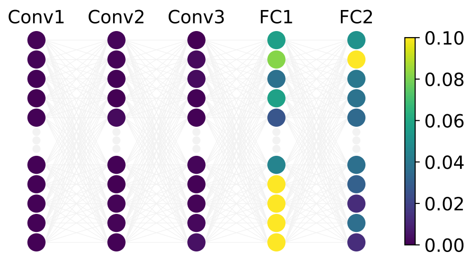
Batch normalization (BN, 18) standardizes activation distributions across batches, bringing $P(A)$ and $Q(A)$ closer together. This naturally raises the question as to whether or not BN solves the problem of differing unit-level distributions, thus removing the need for unit-level surprise. We investigate this below.

Do the same surprise patterns exist? Figure 8 shows the ideal situation for BN, where the BN statistics are (re)calculated using the *new* data distribution before we calculate surprise. Compared to Figure 2, the magnitude of the surprise is indeed lessened as BN standardizes $P(A)$ and $Q(A)$, but it does not make units unsurprised. Moreover, we still see the same patterns of surprise in the early layers for low-level shifts and in the later layers for high-level shifts.

Is an adaptative strategy still best? As shown in Table 4, it is indeed still best to employ an adaptive update strategy to train specific layers for specific shifts, e.g. use FlexTune [35] to select Conv1 & BN1 for low-level shifts and FC2 for high-level shifts. This further confirms that BN alone does *not* solve the problem of changes in activation distribution, as selective adaptation strategies are still superior to updating all layers with SGD. In Table 4 we also show that, if we only use the BN statistics without any training (AdaBN, 24), or only update the BN parameters and statistics on the new data (labelled "BN params"), performance is poor compared to the other strategies.



(a) Low-level shift (crystals bckgr.)



(b) High-level shift (unseen classes)

Figure 8: Network surprise patterns after updating the BN statistics on the new data.

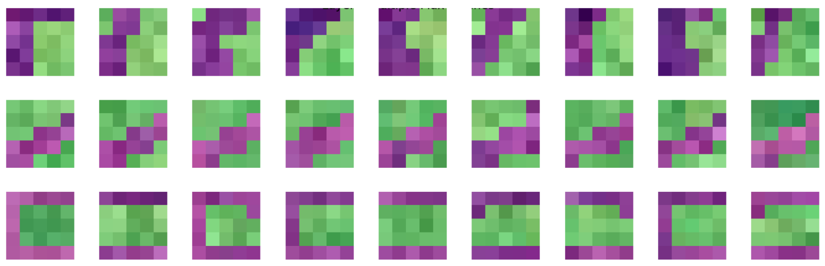
Table 4: 5-shot accuracy with BN. *L:* low-level shifts average, *H:* high-level shifts average. Zero-shot shows accuracy before adaptation. AdaBN [24] updates the BN statistics on the new data. "BN params" updates only the BN parameters and statistics. For all other methods/rows, only the layers listed are permitted to update.

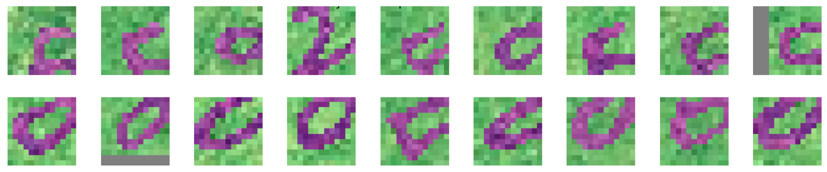
	L	H
Zero-Shot	22.1 ± 0.2	0.0 ± 0.0
AdaBN [24]	50.7 ± 0.4	0.0 ± 0.0
BN params	66.0 ± 1.3	0.0 ± 0.0
Conv1, BN1	$\mathbf{78.9 \pm 0.8}$	0.0 ± 0.0
Conv2, BN2	67.9 ± 1.5	0.0 ± 0.0
Conv3, BN3	60.4 ± 0.7	0.3 ± 0.3
FC1, BN4	56.4 ± 0.5	68.9 ± 1.7
FC2	52.7 ± 0.6	$\mathbf{95.4 \pm 0.5}$
FC1, BN4, FC2	55.8 ± 0.8	94.6 ± 0.9
All (SGD)	76.7 ± 0.8	90.7 ± 2.3

E Further Results

E.1 Max-activating patches

Figure 9 shows the maximum-activating image patches on the EMNIST-DA [9] grass shift for selected units in each layer. Note that we do not have a perfect edges-parts-wholes hierarchy—the receptive field of Conv2 seems too large as it almost sees the entire image (can be a "whole" rather than a part).


(a) Conv 1


(b) Conv 2

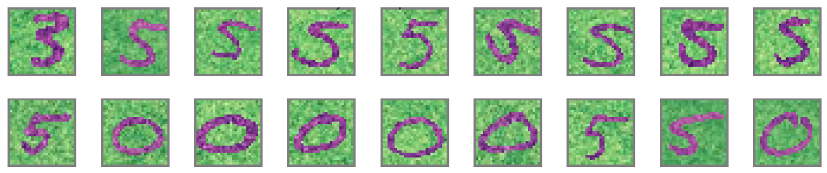

(c) Conv 3

Figure 9: Max-activating patches for different units on the EMNIST-DA grass shift.

E.2 Per-shift Results

What follows is tables of full results for each shift at each number of shots, these are provided for completeness with no further analysis.

Table 5: 2-shot accuracy across shifts: training different layers of a CNN

	Conv1	Conv2	Conv3	FC1	FC2	FC1 + FC2
H1	0.0 ± 0.0	0.0 ± 0.1	7.2 ± 4.9	65.3 ± 14.1	95.2 ± 1.4	77.8 ± 1.4
H2	0.0 ± 0.0	0.0 ± 0.0	5.0 ± 0.8	45.7 ± 8.8	80.5 ± 5.4	71.4 ± 4.2
H3	0.0 ± 0.0	0.0 ± 0.0	10.8 ± 7.3	49.4 ± 4.3	92.9 ± 2.3	81.4 ± 5.7
Crystals	69.9 ± 4.9	52.1 ± 0.9	51.0 ± 0.9	50.0 ± 0.6	47.8 ± 0.7	49.6 ± 1.1
Fog	86.5 ± 3.2	84.5 ± 0.4	84.0 ± 0.5	82.3 ± 0.8	78.8 ± 1.2	81.6 ± 0.4
Gauss. Blur	82.7 ± 0.3	80.4 ± 2.0	79.0 ± 0.3	74.9 ± 0.6	71.5 ± 1.0	74.1 ± 1.9
Grass	81.4 ± 0.8	21.6 ± 7.2	8.1 ± 0.7	7.3 ± 0.6	7.2 ± 0.5	7.7 ± 0.5
Imp. Noise	87.4 ± 0.9	86.0 ± 0.2	83.4 ± 1.4	83.2 ± 0.9	80.7 ± 0.6	81.9 ± 1.4
Sky	74.4 ± 3.9	54.4 ± 0.5	33.2 ± 2.8	18.4 ± 0.7	13.7 ± 2.7	18.4 ± 3.3
Stripe	74.3 ± 0.9	58.2 ± 4.8	58.2 ± 1.4	45.7 ± 2.1	36.7 ± 1.0	47.9 ± 2.6
Avg High	0.0 ± 0.0	0.0 ± 0.0	7.7 ± 4.0	53.5 ± 3.4	89.6 ± 1.5	76.9 ± 2.9
Avg Low	79.5 ± 1.3	62.4 ± 2.1	56.7 ± 0.5	51.7 ± 0.6	48.1 ± 0.5	51.6 ± 0.7
Avg All	55.7 ± 0.9	43.7 ± 1.5	42.0 ± 1.4	52.2 ± 0.8	60.5 ± 0.3	59.2 ± 0.9

Table 6: 2-shot accuracy across shifts: comparison of different responses. Zero-shot is the accuracy before any updates are performed.

	Zero-shot	SGD	FlexTune	Upd. Rule
H1	0.0 ± 0.0	56.7 ± 4.2	95.2 ± 1.4	33.8 ± 21.4
H2	0.0 ± 0.0	51.9 ± 5.7	80.5 ± 5.4	34.3 ± 11.2
H3	0.0 ± 0.0	57.6 ± 5.3	92.9 ± 2.3	41.4 ± 9.4
Crystals	46.3 ± 0.4	55.3 ± 1.3	69.9 ± 4.9	68.5 ± 5.4
Fog	78.4 ± 0.4	83.6 ± 1.2	87.1 ± 2.3	86.6 ± 3.2
Gauss. Blur	60.4 ± 2.1	81.2 ± 0.6	82.7 ± 0.3	81.8 ± 1.8
Grass	5.8 ± 0.2	20.1 ± 12.0	81.4 ± 0.8	81.4 ± 0.8
Imp. Noise	76.9 ± 0.9	87.9 ± 0.2	87.4 ± 0.9	87.7 ± 0.5
Sky	4.1 ± 0.5	35.2 ± 4.9	74.4 ± 3.9	74.4 ± 3.9
Stripe	16.3 ± 1.1	70.0 ± 1.4	74.3 ± 0.9	72.8 ± 2.1
Avg High	0.0 ± 0.0	55.4 ± 1.4	89.6 ± 1.5	36.5 ± 6.7
Avg Low	41.2 ± 0.3	61.9 ± 2.4	79.6 ± 1.2	79.0 ± 1.1
Avg All	28.8 ± 0.2	60.0 ± 1.9	82.6 ± 0.7	66.3 ± 2.5

Table 7: 5-shot accuracy across shifts: training different layers of a CNN

	Conv1	Conv2	Conv3	FC1	FC2	FC1 + FC2
H1	0.0 ± 0.0	0.1 ± 0.1	29.8 ± 8.2	82.8 ± 7.3	96.4 ± 1.0	91.9 ± 2.0
H2	0.0 ± 0.0	0.5 ± 0.9	26.1 ± 6.2	76.7 ± 6.6	91.3 ± 1.5	86.2 ± 3.8
H3	0.0 ± 0.0	0.3 ± 0.4	20.3 ± 6.1	82.3 ± 5.3	96.0 ± 0.6	93.0 ± 2.8
Crystals	78.1 ± 1.7	57.5 ± 0.9	52.3 ± 0.8	51.5 ± 0.5	49.2 ± 0.8	51.4 ± 0.6
Fog	89.6 ± 0.3	86.5 ± 0.7	84.8 ± 0.5	83.1 ± 0.6	81.3 ± 0.6	82.6 ± 0.6
Gauss. Blur	84.0 ± 0.4	82.5 ± 0.7	81.7 ± 0.6	78.3 ± 0.9	75.9 ± 0.5	78.1 ± 0.4
Grass	82.0 ± 2.2	41.7 ± 7.4	10.3 ± 1.3	8.4 ± 0.9	7.3 ± 0.5	9.4 ± 0.4
Imp. Noise	88.6 ± 0.3	86.8 ± 0.1	85.7 ± 0.3	84.3 ± 0.9	81.5 ± 0.4	83.9 ± 0.4
Sky	79.6 ± 0.7	65.9 ± 0.5	44.3 ± 1.9	27.9 ± 1.8	18.3 ± 2.4	28.8 ± 1.2
Stripe	77.1 ± 1.8	70.9 ± 2.6	70.6 ± 0.6	60.4 ± 1.6	47.0 ± 1.3	60.9 ± 2.5
Avg High	0.0 ± 0.0	0.3 ± 0.5	25.4 ± 3.8	80.6 ± 4.6	94.5 ± 0.9	90.4 ± 1.1
Avg Low	82.7 ± 0.4	70.2 ± 1.2	61.4 ± 0.2	56.3 ± 0.2	51.5 ± 0.3	56.4 ± 0.3
Avg All	57.9 ± 0.3	49.3 ± 0.8	50.6 ± 1.2	63.6 ± 1.2	64.4 ± 0.3	66.6 ± 0.1

Table 8: 5-shot accuracy across shifts: comparison of different responses. Zero-shot is the accuracy before any updates are performed.

	Zero-shot	SGD	FlexTune	Upd. Rule
H1	0.0 ± 0.0	68.0 ± 13.8	96.4 ± 1.0	80.3 ± 4.8
H2	0.0 ± 0.0	78.0 ± 2.0	91.3 ± 1.5	77.3 ± 7.2
H3	0.0 ± 0.0	77.1 ± 5.0	96.0 ± 0.6	79.9 ± 9.1
Crystals	46.3 ± 0.4	57.5 ± 3.8	78.1 ± 1.7	77.0 ± 3.4
Fog	78.4 ± 0.4	86.3 ± 0.2	89.6 ± 0.3	89.6 ± 0.3
Gauss. Blur	60.4 ± 2.1	84.3 ± 0.7	84.0 ± 0.4	83.9 ± 0.3
Grass	5.8 ± 0.2	50.9 ± 6.8	82.0 ± 2.2	82.3 ± 2.6
Imp. Noise	76.9 ± 0.9	88.3 ± 0.1	88.6 ± 0.3	87.6 ± 0.3
Sky	4.1 ± 0.5	56.6 ± 4.7	79.6 ± 0.7	80.8 ± 0.9
Stripe	16.3 ± 1.1	75.4 ± 1.0	77.1 ± 1.8	79.3 ± 0.8
Avg High	0.0 ± 0.0	74.4 ± 5.2	94.5 ± 0.9	79.2 ± 4.2
Avg Low	41.2 ± 0.3	71.3 ± 2.1	82.7 ± 0.4	82.9 ± 0.5
Avg All	28.8 ± 0.2	72.2 ± 2.9	86.3 ± 0.5	81.8 ± 1.2

Table 9: 10-shot accuracy across shifts: training different layers of a CNN

	Conv1	Conv2	Conv3	FC1	FC2	FC1 + FC2
H1	0.0 ± 0.0	0.5 ± 0.8	50.1 ± 5.1	91.3 ± 2.0	96.1 ± 0.8	94.5 ± 0.6
H2	0.0 ± 0.0	0.9 ± 0.8	45.7 ± 8.8	88.5 ± 1.7	92.8 ± 0.3	89.0 ± 2.2
H3	0.0 ± 0.0	0.3 ± 0.4	46.6 ± 16.0	89.7 ± 5.3	95.0 ± 1.1	92.8 ± 1.8
Crystals	80.5 ± 1.5	59.9 ± 3.1	54.4 ± 0.1	52.6 ± 0.7	50.6 ± 0.8	52.6 ± 0.8
Fog	90.1 ± 0.1	87.8 ± 0.5	85.6 ± 0.3	84.1 ± 0.2	82.1 ± 1.3	84.3 ± 0.4
Gauss. Blur	85.0 ± 0.4	83.9 ± 1.1	82.6 ± 0.4	81.4 ± 0.9	77.9 ± 2.3	80.7 ± 0.8
Grass	83.8 ± 0.9	57.1 ± 3.2	15.1 ± 0.8	10.0 ± 0.9	7.8 ± 0.1	11.2 ± 1.0
Imp. Noise	89.1 ± 0.0	87.2 ± 0.4	86.0 ± 0.3	84.7 ± 0.6	82.7 ± 0.1	84.5 ± 0.2
Sky	83.3 ± 1.5	71.4 ± 1.0	52.5 ± 0.5	36.7 ± 1.2	24.5 ± 2.4	38.0 ± 0.2
Stripe	82.4 ± 0.7	76.1 ± 2.8	76.1 ± 1.1	69.2 ± 0.2	56.8 ± 0.8	68.1 ± 1.8
Avg High	0.0 ± 0.0	0.6 ± 0.6	47.5 ± 6.4	89.8 ± 1.2	94.6 ± 0.6	92.1 ± 0.5
Avg Low	84.9 ± 0.3	74.8 ± 1.3	64.6 ± 0.2	59.8 ± 0.4	54.6 ± 0.5	59.9 ± 0.4
Avg All	59.4 ± 0.2	52.5 ± 0.8	59.5 ± 1.8	68.8 ± 0.5	66.6 ± 0.4	69.6 ± 0.4

Table 10: 10-shot accuracy across shifts: comparison of different responses. Zero-shot is the accuracy before any updates are performed.

	Zero-shot	SGD	FlexTune	Upd. Rule
H1	0.0 ± 0.0	84.8 ± 0.5	96.1 ± 0.8	90.4 ± 3.0
H2	0.0 ± 0.0	79.9 ± 1.7	92.8 ± 0.3	88.3 ± 2.0
H3	0.0 ± 0.0	87.1 ± 2.7	95.0 ± 1.1	89.7 ± 5.1
Crystals	46.3 ± 0.4	61.2 ± 1.6	80.5 ± 1.5	79.8 ± 0.9
Fog	78.4 ± 0.4	87.4 ± 0.2	90.1 ± 0.1	90.1 ± 0.1
Gauss. Blur	60.4 ± 2.1	85.7 ± 0.6	85.0 ± 0.4	85.3 ± 1.0
Grass	5.8 ± 0.2	69.2 ± 2.4	83.8 ± 0.9	84.3 ± 1.3
Imp. Noise	76.9 ± 0.9	88.2 ± 0.4	89.1 ± 0.0	88.0 ± 0.2
Sky	4.1 ± 0.5	66.8 ± 1.1	83.3 ± 1.5	83.4 ± 1.5
Stripe	16.3 ± 1.1	78.8 ± 1.1	82.4 ± 0.7	82.5 ± 0.9
Avg High	0.0 ± 0.0	84.0 ± 0.5	94.6 ± 0.6	89.5 ± 1.2
Avg Low	41.2 ± 0.3	76.8 ± 0.7	84.9 ± 0.3	84.8 ± 0.5
Avg All	28.8 ± 0.2	78.9 ± 0.5	87.8 ± 0.4	86.2 ± 0.6

Table 11: 20-shot accuracy across shifts: training different layers of a CNN

	Conv1	Conv2	Conv3	FC1	FC2	FC1 + FC2
H1	0.0 ± 0.0	2.1 ± 3.4	66.4 ± 8.9	95.7 ± 0.9	97.2 ± 0.7	95.1 ± 0.6
H2	0.0 ± 0.0	5.6 ± 0.9	64.9 ± 8.7	90.3 ± 1.5	94.8 ± 0.6	91.6 ± 1.1
H3	0.0 ± 0.0	9.5 ± 1.3	69.3 ± 6.2	93.5 ± 2.2	96.1 ± 1.0	95.3 ± 0.9
Crystals	83.7 ± 0.5	65.6 ± 2.3	56.7 ± 0.2	54.6 ± 0.3	51.5 ± 0.7	54.6 ± 0.4
Fog	90.4 ± 0.4	88.4 ± 0.1	86.1 ± 1.0	85.2 ± 0.2	84.0 ± 0.3	85.1 ± 0.2
Gauss. Blur	86.6 ± 0.4	85.0 ± 1.4	83.4 ± 0.6	82.2 ± 0.8	81.0 ± 0.9	82.3 ± 1.1
Grass	85.6 ± 1.0	67.6 ± 2.8	20.7 ± 1.8	13.4 ± 1.0	8.3 ± 0.3	14.1 ± 0.9
Imp. Noise	89.1 ± 0.6	87.7 ± 0.2	86.6 ± 0.2	85.2 ± 0.5	83.7 ± 0.5	84.9 ± 0.2
Sky	84.8 ± 1.0	76.3 ± 0.7	59.5 ± 0.4	45.1 ± 1.5	31.7 ± 0.7	47.0 ± 0.7
Stripe	83.9 ± 1.3	79.7 ± 2.0	78.9 ± 0.9	74.7 ± 0.2	62.2 ± 0.9	75.3 ± 0.3
Avg High	0.0 ± 0.0	5.7 ± 0.8	66.8 ± 2.9	93.2 ± 1.2	96.1 ± 0.3	94.0 ± 0.2
Avg Low	86.3 ± 0.3	78.6 ± 1.1	67.4 ± 0.2	62.9 ± 0.3	57.5 ± 0.4	63.3 ± 0.3
Avg All	60.4 ± 0.2	56.7 ± 0.9	67.3 ± 0.8	72.0 ± 0.5	69.0 ± 0.4	72.5 ± 0.3

Table 12: 20-shot accuracy across shifts: comparison of different responses. Zero-shot is the accuracy before any updates are performed.

	Zero-shot	SGD	FlexTune	Upd. Rule
H1	0.0 ± 0.0	89.3 ± 1.6	97.2 ± 0.7	95.5 ± 1.2
H2	0.0 ± 0.0	85.4 ± 3.1	94.8 ± 0.6	89.9 ± 1.4
H3	0.0 ± 0.0	90.3 ± 0.6	96.3 ± 0.9	93.5 ± 2.4
Crystals	46.3 ± 0.4	66.5 ± 1.8	83.7 ± 0.5	82.1 ± 0.9
Fog	78.4 ± 0.4	88.1 ± 0.3	90.4 ± 0.4	90.4 ± 0.4
Gauss. Blur	60.4 ± 2.1	86.0 ± 0.5	86.6 ± 0.4	86.9 ± 0.4
Grass	5.8 ± 0.2	77.0 ± 1.2	85.6 ± 1.0	85.5 ± 0.2
Imp. Noise	76.9 ± 0.9	88.5 ± 0.1	89.1 ± 0.6	88.3 ± 0.3
Sky	4.1 ± 0.5	73.3 ± 0.8	84.8 ± 1.0	84.7 ± 0.9
Stripe	16.3 ± 1.1	81.6 ± 0.4	83.9 ± 1.3	84.7 ± 1.1
Avg High	0.0 ± 0.0	88.3 ± 1.4	96.1 ± 0.2	93.0 ± 1.1
Avg Low	41.2 ± 0.3	80.1 ± 0.3	86.3 ± 0.3	86.1 ± 0.3
Avg All	28.8 ± 0.2	82.6 ± 0.2	89.2 ± 0.2	88.2 ± 0.4

Table 13: 50-shot accuracy across shifts: training different layers of a CNN

	Conv1	Conv2	Conv3	FC1	FC2	FC1 + FC2
H1	0.0 ± 0.0	25.8 ± 24.0	83.5 ± 5.3	97.0 ± 0.2	97.2 ± 0.5	95.9 ± 0.9
H2	0.0 ± 0.0	32.8 ± 4.5	84.4 ± 1.9	94.8 ± 0.7	95.2 ± 0.9	93.5 ± 1.2
H3	0.0 ± 0.0	28.1 ± 17.4	85.0 ± 4.7	95.5 ± 1.9	97.6 ± 0.5	96.1 ± 0.6
Crystals	85.2 ± 0.5	72.9 ± 1.2	59.3 ± 0.3	57.8 ± 0.3	53.1 ± 0.6	57.8 ± 0.2
Fog	90.6 ± 0.2	89.3 ± 0.1	87.7 ± 0.2	86.7 ± 0.2	85.1 ± 0.3	86.6 ± 0.2
Gauss. Blur	87.9 ± 0.5	87.3 ± 0.3	85.8 ± 0.4	84.6 ± 0.2	83.2 ± 0.6	84.6 ± 0.2
Grass	87.1 ± 0.6	75.2 ± 1.9	29.8 ± 1.6	19.3 ± 0.8	9.0 ± 0.2	19.7 ± 0.6
Imp. Noise	89.4 ± 0.3	88.3 ± 0.2	86.9 ± 0.3	85.9 ± 0.3	84.8 ± 0.2	85.9 ± 0.2
Sky	86.7 ± 0.5	80.9 ± 0.3	66.4 ± 0.6	54.3 ± 1.8	38.2 ± 0.8	56.7 ± 0.4
Stripe	87.4 ± 0.3	84.6 ± 0.7	83.2 ± 0.5	79.9 ± 0.9	67.6 ± 0.8	79.8 ± 0.8
Avg High	0.0 ± 0.0	28.9 ± 12.2	84.3 ± 3.7	95.8 ± 0.6	96.7 ± 0.6	95.2 ± 0.2
Avg Low	87.8 ± 0.1	82.6 ± 0.4	71.3 ± 0.3	66.9 ± 0.3	60.1 ± 0.3	67.3 ± 0.0
Avg All	61.4 ± 0.1	66.5 ± 3.8	75.2 ± 0.9	75.6 ± 0.4	71.1 ± 0.3	75.7 ± 0.1

Table 14: 50-shot accuracy across shifts: comparison of different responses. Zero-shot is the accuracy before any updates are performed.

	Zero-shot	SGD	FlexTune	Upd. Rule
H1	0.0 ± 0.0	93.3 ± 0.5	97.2 ± 0.5	96.8 ± 0.3
H2	0.0 ± 0.0	89.9 ± 0.9	95.2 ± 0.9	94.4 ± 0.6
H3	0.0 ± 0.0	93.8 ± 1.4	97.6 ± 0.5	95.5 ± 2.0
Crystals	46.3 ± 0.4	73.2 ± 0.9	85.2 ± 0.5	83.5 ± 0.5
Fog	78.4 ± 0.4	88.9 ± 0.3	90.6 ± 0.2	90.5 ± 0.3
Gauss. Blur	60.4 ± 2.1	87.3 ± 0.4	87.9 ± 0.5	87.4 ± 0.3
Grass	5.8 ± 0.2	81.8 ± 0.4	87.1 ± 0.6	86.2 ± 0.4
Imp. Noise	76.9 ± 0.9	88.9 ± 0.2	89.4 ± 0.3	88.5 ± 0.5
Sky	4.1 ± 0.5	79.1 ± 0.5	86.7 ± 0.5	86.9 ± 0.3
Stripe	16.3 ± 1.1	84.8 ± 1.3	87.4 ± 0.3	87.5 ± 0.4
Avg High	0.0 ± 0.0	92.3 ± 0.6	96.7 ± 0.6	95.6 ± 0.6
Avg Low	41.2 ± 0.3	83.4 ± 0.2	87.8 ± 0.1	87.2 ± 0.1
Avg All	28.8 ± 0.2	86.1 ± 0.3	90.4 ± 0.2	89.7 ± 0.1

Table 15: 2000-shot (all data) accuracy across shifts: training different layers of a CNN

	Conv1	Conv2	Conv3	FC1	FC2	FC1 + FC2
H1	0.0 ± 0.1	80.0 ± 9.9	97.6 ± 0.2	98.7 ± 0.1	98.6 ± 0.1	98.8 ± 0.1
H2	2.0 ± 3.2	78.2 ± 8.1	96.7 ± 0.2	98.0 ± 0.2	98.0 ± 0.2	98.3 ± 0.2
H3	0.0 ± 0.0	81.9 ± 2.8	97.7 ± 0.4	98.6 ± 0.3	98.7 ± 0.2	98.8 ± 0.0
Crystals	88.0 ± 0.3	85.5 ± 0.3	72.6 ± 0.2	69.4 ± 0.6	57.4 ± 0.4	68.2 ± 0.3
Fog	91.2 ± 0.3	91.2 ± 0.1	90.6 ± 0.1	90.2 ± 0.1	88.0 ± 0.3	89.6 ± 0.1
Gauss. Blur	90.1 ± 0.1	90.9 ± 0.1	90.3 ± 0.0	90.2 ± 0.1	86.7 ± 0.1	89.7 ± 0.2
Grass	89.1 ± 0.2	86.1 ± 0.5	56.6 ± 1.1	40.8 ± 0.9	17.0 ± 0.3	38.8 ± 1.0
Imp. Noise	89.8 ± 0.1	89.7 ± 0.2	88.2 ± 0.1	88.3 ± 0.2	86.6 ± 0.3	87.6 ± 0.2
Sky	89.0 ± 0.1	88.2 ± 0.3	81.7 ± 0.1	75.2 ± 0.3	48.8 ± 0.4	73.7 ± 0.4
Stripe	89.9 ± 0.1	90.8 ± 0.3	90.3 ± 0.1	89.0 ± 0.0	76.3 ± 0.5	88.3 ± 0.2
Avg High	0.7 ± 1.1	80.0 ± 5.1	97.4 ± 0.2	98.5 ± 0.1	98.4 ± 0.1	98.6 ± 0.0
Avg Low	89.6 ± 0.1	88.9 ± 0.0	81.5 ± 0.2	77.6 ± 0.1	65.8 ± 0.1	76.5 ± 0.1
Avg All	62.9 ± 0.3	86.3 ± 1.5	86.2 ± 0.1	83.8 ± 0.1	75.6 ± 0.1	83.2 ± 0.1

Table 16: 2000-shot (all data) accuracy across shifts: comparison of different responses. Zero-shot is the accuracy before any updates are performed.

	Zero-shot	SGD	FlexTune	Upd. Rule
H1	0.0 ± 0.0	97.9 ± 0.1	98.8 ± 0.1	98.7 ± 0.2
H2	0.0 ± 0.0	97.2 ± 0.4	98.3 ± 0.1	98.1 ± 0.2
H3	0.0 ± 0.0	98.2 ± 0.1	98.8 ± 0.0	98.7 ± 0.3
Crystals	46.3 ± 0.4	87.7 ± 0.3	88.0 ± 0.3	88.3 ± 0.2
Fog	78.4 ± 0.4	90.9 ± 0.0	91.3 ± 0.2	90.8 ± 0.2
Gauss. Blur	60.4 ± 2.1	90.5 ± 0.1	90.9 ± 0.1	90.6 ± 0.1
Grass	5.8 ± 0.2	89.2 ± 0.1	89.1 ± 0.2	89.5 ± 0.4
Imp. Noise	76.9 ± 0.9	89.3 ± 0.2	89.8 ± 0.1	89.5 ± 0.2
Sky	4.1 ± 0.5	89.1 ± 0.2	89.0 ± 0.1	89.4 ± 0.1
Stripe	16.3 ± 1.1	90.6 ± 0.3	90.8 ± 0.3	90.0 ± 0.2
Avg High	0.0 ± 0.0	97.8 ± 0.1	98.6 ± 0.0	98.5 ± 0.1
Avg Low	41.2 ± 0.3	89.6 ± 0.1	89.8 ± 0.0	89.7 ± 0.1
Avg All	28.8 ± 0.2	92.1 ± 0.1	92.5 ± 0.0	92.4 ± 0.0