Contextual Squeeze-and-Excitation

Massimiliano Patacchiola
University of Cambridge
mp2008@cam.ac.uk

John Bronskill
University of Cambridge
jfb54@cam.ac.uk

Aliaksandra Shysheya
University of Cambridge
as2975@cam.ac.uk

Katja Hofmann
Microsoft Research
kahofman@microsoft.com

Sebastian Nowozin*
nowozin@gmail.com

Richard E. Turner
University of Cambridge
ret26@cam.ac.uk

Abstract

Several applications require effective knowledge transfer across tasks in the low-data regime. For instance in personalization a pretrained system is adapted by learning on small amounts of labeled data belonging to a specific user (context). This setting requires high accuracy under low computational complexity, meaning low memory footprint in terms of parameters storage and adaptation cost. Meta-learning methods based on Feature-wise Linear Modulation generators (FiLM) satisfy these constraints as they can adapt a backbone without expensive fine-tuning. However, there has been limited research on viable alternatives to FiLM generators. In this paper we focus on this area of research and propose a new adaptive block called Contextual Squeeze-and-Excitation (CaSE). CaSE is more efficient than FiLM generators for a variety of reasons: it does not require a separate set encoder, has fewer learnable parameters, and only uses a scale vector (no shift) to modulate activations. We empirically show that CaSE is able to outperform FiLM generators in terms of parameter efficiency (a 75% reduction in the number of adaptation parameters) and classification accuracy (a 1.5% average improvement on the 26 datasets of the VTAB+MD benchmark).

1 Introduction

In several applications it is crucial to ensure fast and efficient knowledge transfer across tasks. In personalization for instance, a pretrained system must be adapted by exploiting small amounts of data belonging to a specific user (context), often relying on scarce computational resources (e.g. on portable devices). These settings call for few-shot learning methods that can retain high accuracy under low computational complexity, meaning low memory footprint in terms of parameters storage and adaptation cost.

Meta-learning methods (Schmidhuber, 1987; Hospedales et al., 2020) are good candidates to satisfy those constraints since they are significantly less expensive in terms of adaptation cost than fine-tuning routines (Bronskill et al., 2021). The key element to superior performances, is often the use of adapters that can modify dynamically a pretrained model conditioned on the context data. The most common adapters used in few-shot learning are the Feature-wise Linear Modulation layers (FiLM, Perez et al. 2018). Dynamic adapters based on FiLM, use a convolutional set-encoder to generate an embedding of the context set that is forwarded to local MLPs to produce scale and shift vectors that modulate a pretrained neural network backbone. Variations of this adapter have been used in several methods, such as TADAM (Oreshkin et al., 2018), CNAPs (Requeima et al., 2019), SimpleCNAPs (Bateni et al., 2020), CAVIA (Zintgraf et al., 2019), and LITE (Bronskill et al., 2021).

*Work done while the author was at Microsoft Research – Cambridge (UK)

6th Workshop on Meta-Learning at NeurIPS 2022, New Orleans.

We will use the generic term *FiLM generator* to refer to these adapters and the term *FiLM* to refer to the scale and shift vectors used to modulate the activations. FiLM generators are effective but they have been taken from granted by the community and there has been no exploration of possible alternatives. These considerations raise a question: is it possible to build dynamic adapters that are more effective and efficient than FiLM generators?

Contributions In this work we propose a novel extension of the popular Squeeze-and-Excitation block of Hu et al. (2018) to the meta-learning setting that we call **C**ontextu**a**l **S**queeze-and-**E**xcitation (**CaSE**). CaSE is better than FiLM generators for a variety of reasons: it does not require a separate set encoder, has fewer learnable parameters, and only uses a scale vector (no shift) to modulate the backbone activations. We empirically show that CaSE is able to outperform FiLM generators in terms of parameter efficiency (a 75% reduction in the number of adaptation parameters) and classification accuracy (a 1.5% improvement on MetaDataset and VTAB). The code is released with an open-source license and available on the authors' repository [1].

2 Contextual Squeeze-and-Excitation (CaSE)

Problem formulation In this paragraph we introduce the few-shot learning notation, as this will be used to describe the functioning of a CaSE adaptive block. Let us define a collection of meta-training tasks as $\mathcal{D} = \{\tau_1, \ldots, \tau_D\}$ where $\tau_i = (\mathcal{C}_i, \mathcal{T}_i)$ represents a generic task composed of a context set $\mathcal{C}_i = \{(\mathbf{x}, y)_1, \ldots, (\mathbf{x}, y)_M\}$ and a target set $\mathcal{T}_i = \{(\mathbf{x}, y)_1, \ldots, (\mathbf{x}, y)_D\}$ of input-output pairs. Following common practice we use the term *shot* to identify the number of samples per class (e.g. 5-shot is 5 samples per class) and the term *way* to identify the number of classes (e.g. 10-way is 10 classes per task). Given an evaluation task $\tau_* = \{\mathcal{C}_*, \mathbf{x}_*\}$ the goal is to predict the true label y_* of the unlabeled target point $\mathbf{x}_*$ conditioned on the context set $\mathcal{C}_*$. In meta-learning methods training and evaluation are performed episodically (Vinyals et al., 2016), with training tasks sampled from a meta-train dataset and evaluation tasks sampled from an unseen meta-test dataset. Parameters are divided in two groups: $\boldsymbol{\phi}$ task-common (shared across all tasks), and $\boldsymbol{\psi}_\tau$ task-specific (estimated on the task at hand). The parameters $\boldsymbol{\phi}$ and $\boldsymbol{\psi}_\tau$ can be estimated via gradient updates (e.g. MAML, Finn et al. 2017), learned metrics (e.g. ProtoNets, Snell et al. 2017), or Bayesian methods (Gordon et al., 2018; Patacchiola et al., 2020; Sendera et al., 2021).

Standard Squeeze-Excite (SE) We briefly introduce standard SE (Hu et al., 2018), as we are going to build on top of this work. SE is an adaptive layer used in the supervised learning setting to perform instance based channel-wise feature adaptation, which is trained following a supervised protocol together with the parameters of the neural network backbone. Given a convolutional neural network, consider a subset of L layers and associate to each one of them a Multi-Layer Perceptron (MLP), here represented as a function $g_{\boldsymbol{\phi}}(\cdot)$. The number of hidden units in the MLP is defined by the number of inputs divided by a reduction factor. Given a mini-batch of B input images, each convolution produces an output of size $B \times C \times H \times W$ where C is the number of channels, H the height, and W the width of the resulting tensor. For simplicity we split this tensor into sub-tensors that are grouped into a set $\{\mathbf{H}_1, \ldots, \mathbf{H}_B\}$ with $\mathbf{H}_i \in \mathbb{R}^{C \times H \times W}$. To avoid clutter, we suppress the layer indexing when possible. SE perform a spatial pooling that produces a tensor of shape $B \times C \times 1 \times 1$; this can be interpreted as a set of vectors $\{\mathbf{h}_1, \ldots, \mathbf{h}_B\}$ with $\mathbf{h}_i \in \mathbb{R}^C$. For each layer l, the set is passed to the associated MLP that will generate an individual scale vector $\boldsymbol{\gamma}_i \in \mathbb{R}^C$, where

$$\boldsymbol{\gamma}_1^{(l)} = g_{\boldsymbol{\phi}}^{(l)}\left(\mathbf{h}_1^{(l)}\right) \cdots \boldsymbol{\gamma}_B^{(l)} = g_{\boldsymbol{\phi}}^{(l)}\left(\mathbf{h}_B^{(l)}\right). \tag{1}$$

An elementwise product is then performed between the scale vector and the original tensor

$$\hat{\mathbf{H}}_1^{(l)} = \mathbf{H}_1^{(l)} * \boldsymbol{\gamma}_1^{(l)} \cdots \hat{\mathbf{H}}_B^{(l)} = \mathbf{H}_B^{(l)} * \boldsymbol{\gamma}_B^{(l)}, \tag{2}$$

with the aim of modulating the activation along the channel dimension. This operation can be interpreted as a soft attention mechanism, with the MLP conditionally deciding which channel must be attended to. A graphical representation of SE is provided in Figure 1 (left).

Contextual Squeeze-Excite (CaSE) Standard SE is an instance-based mechanism that is suited for i.i.d. data in the supervised setting. In a meta-learning setting we can exploit the distinction in tasks to define a new version of SE for task-based channel-wise feature adaptation. For a task $\tau = (\mathcal{C}, \mathcal{T})$,

[1] `https://github.com/mpatacchiola/contextual-squeeze-and-excitation`

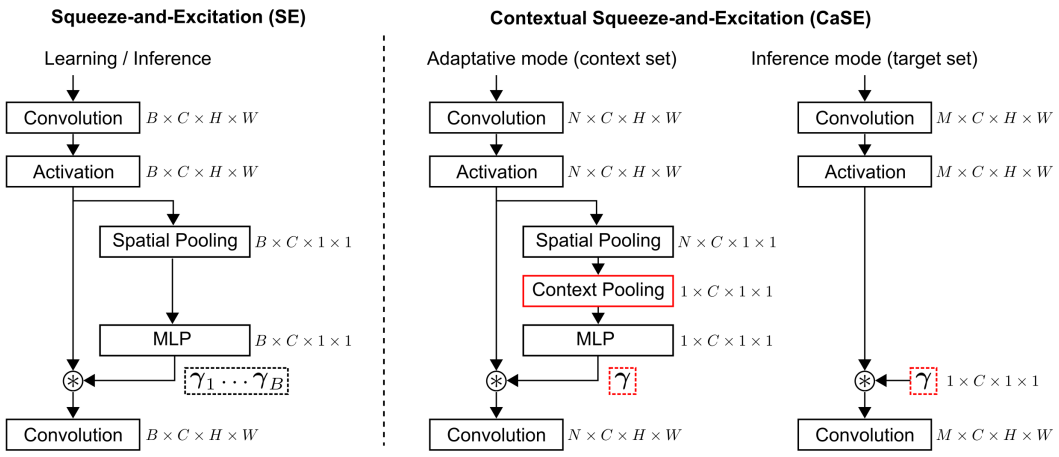

Figure 1: Comparison between the standard Squeeze-Excite (left) and the proposed Contextual Squeeze-Excite (right). Red frames highlight the two key differences between SE and CaSE: context pooling and scale transfer from context to target. B = mini-batch size, C = channels, H = height, W = width, N = context-set size, M = target-set size, $*$ elementwise multiplication.

consider the N images from the context set $\mathcal{C}$, and the tensors produced by each convolution in the layers of interest $\{\mathbf{H}_1, \ldots, \mathbf{H}_N\}$ with $\mathbf{H}_i \in \mathbb{R}^{C \times H \times W}$. As in standard SE, we first apply a *spatial* pooling to each tensor $\mathbf{H}_i$ which produces N vectors $\{\mathbf{h}_1, \ldots, \mathbf{h}_N\}$ of shape $\mathbf{h}_i \in \mathbb{R}^C$. Then a *context* pooling is performed; this corresponds to an empirical mean over $\{\mathbf{h}_1, \ldots, \mathbf{h}_N\}$ (see Appendix A for more details about context pooling). The pooled representation is passed to the associated MLP to produce a single scale-vector for that layer

$$\boldsymbol{\gamma}^{(l)} = g_{\boldsymbol{\phi}}^{(l)}\left(\bar{\mathbf{h}}^{(l)}\right) \quad \text{with} \quad \bar{\mathbf{h}}^{(l)} = \frac{1}{N}\left(\mathbf{h}_1^{(l)} + \cdots + \mathbf{h}_N^{(l)}\right), \tag{3}$$

which is then multiplied elementwise by the original tensors

$$\hat{\mathbf{H}}_1^{(l)} = \mathbf{H}_1^{(l)} * \boldsymbol{\gamma}^{(l)} \;\cdots\; \hat{\mathbf{H}}_N^{(l)} = \mathbf{H}_N^{(l)} * \boldsymbol{\gamma}^{(l)}. \tag{4}$$

The scale vector is estimated in adaptive mode and transferred to the target points $\mathcal{T}$ in inference mode (no forward pass on the MLPs), as shown in the rightmost part of Figure 1. In synthesis, the three major differences between SE and CaSE are: (i) CaSE uses a contextual pooling with the aim of generating an adaptive vector per-task instead of per-instance as in SE; (ii) CaSE distinguishes between an adaptive mode and an inference mode that transfers the scale from context to target, while SE does not make such a distinction; and (iii) CaSE parameters are estimated via episodic meta-training while SE parameters via standard supervised-training. In Section 3 we show that those differences are fundamental to achieve superior performance in the few-shot setting. A representation of a CaSE block is reported in Figure 1 (right), additional technical details are provided in Appendix A.

3 Experiments

In this sub-section we report empirical results related to CaSE blocks in three directions: **1)** we compared standard SE (Hu et al., 2018) and CaSE on MDv2 (Triantafillou et al., 2019) and VTAB (Zhai et al., 2019), confirming that a) adaptation helps over not adapting, b) contextual adaptation (CaSE) outperforms instance based adaptation (SE); **2)** we compare CaSE against a SOTA FiLM generator (Bronskill et al., 2021), showing that CaSE is significantly more efficient using 75% fewer parameters while boosting the classification accuracy on average by $+1.5\%$ on VTAB and MD2; and **3)** we provide an insight on the effectiveness of CaSE blocks with a series of qualitative analysis.

Comparing SE vs. CaSE We compare standard SE and the proposed CaSE on VTAB and MD-v2. For a fair comparison we keep constant all factors of variation (backbone, training schedule, hyperparameters, etc.) and use the same reduction of 32 (0.8M adaptive parameters). In order to compare the results with the other experiments in this section, we use a Mahalanobis-distance head as in Bronskill et al. (2021). We summarize the results in Figure 2 (left) and add a tabular breakdown

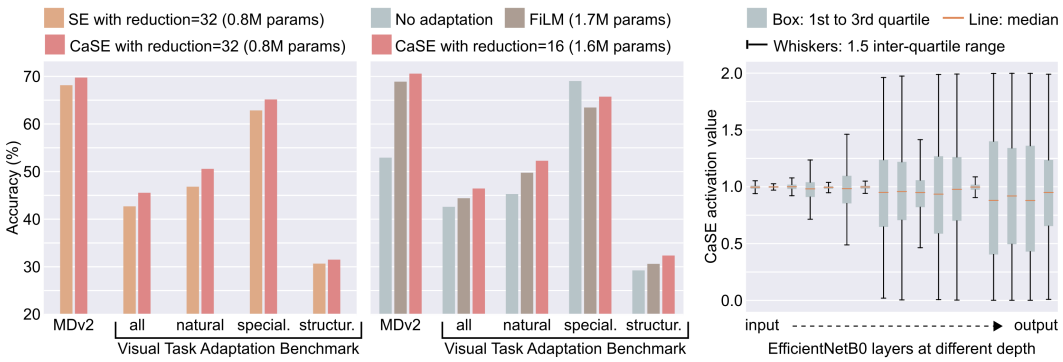

Figure 2: **Left**: CaSE vs Squeeze-and-Excitation (SE) (both methods use EfficientNetB0, 84×84 inputs, Mahalanobis-distance head). CaSE outperforms SE in all conditions. **Center**: CaSE vs. FiLM generators (Bronskill et al., 2021) and a baseline with no body adaptation (all methods use EfficientNetB0, 84×84 inputs, Mahalanobis-distance head). CaSE outperforms FiLM generators in all conditions. **Right**: boxplot of CaSE activations at different depth of an EfficientNetB0 for 800 tasks sampled from the MDv2 test set. The modulation of CaSE is minimal at early stages for general-purpose filters and increases at deeper stages.

in Appendix B. CaSE outperforms SE in all conditions, confirming that a contextual adaptation mechanism is fundamental to transfer knowledge effectively across tasks.

Comparing adaptation mechanisms We perform a comparison on VTAB+MD of CaSE against FiLM generators (Bronskill et al., 2021), and a baseline that uses a pretrained model but no adaptation of the body. Methods are compared in identical conditions, using a Mahalanobis-distance head, an EfficientNetB0 backbone, and same training schedule on 84×84 images. We show a summary of the results in Figure 2 (center) and provide a tabular breakdown in the appendix. CaSE is able to outperform FiLM generators in all conditions. In Appendix B we report the results for CaSE with reduction 64 (0.4M parameters) showing that it is able to outperform FiLM generators (1.7M parameters) using a fraction of the parameters. The comparison with the baseline with no adaptation, shows that in all but one condition (VTAB specialized) adaptation is beneficial. This is likely due to the strong domain shift introduced by some of the specialized datasets.

Role of CaSE blocks To examine the role of CaSE blocks we analyze the aggregated activations at different stages of the body for 800 tasks sampled from the MDv2 test set using an EfficientNetB0. In Figure 2 (right) we report the aggregated distribution as boxplots, and in Appendix D we provide a per-dataset breakdown. Overall the median is close to 1.0 (identity) which is the expected behavior as on average we aim at exploiting the underlying pretrained model. The variance is small at early stages, indicating that CaSE has learned to take advantage of general-purpose filters that are useful across all tasks. In deeper layers the variance increases, showing a task-specific modulation effect. In Appendix D we also include a plot with per-channel activations for all datasets at different depths, showing that the modulation is similar across datasets at early stages and it diverges later on.

4 Conclusions

We have introduced a new adaptive block called CaSE, which is based on the popular Squeeze-and-Excitation (SE) block proposed by Hu et al. (2018). CaSE is effective at modulating a pretrained model in the few-shot setting, outperforming other adaptation mechanisms such as FiLM generators in terms of classification accuracy on the 26 datasets of VTAB+MD. Additionally we have showed that CaSE uses fewer learnable parameters and is easier to implement as it does not need a separate set encoder module as in FiLM generators. **Limitations**: while the experimental results are promising, further investigation is needed to validate the effectiveness of CaSE in other settings (e.g. larger images) and in conjunction with other methods (e.g. ProtoNets, MatchingNets, etc).

Acknowledgments and Disclosure of Funding

Funding in direct support of this work: Massimiliano Patacchiola, John Bronskill, Aliaksandra Shysheya, and Richard E. Turner are supported by an EPSRC Prosperity Partnership EP/T005386/1 between the EPSRC, Microsoft Research and the University of Cambridge. The authors would like to thank: anonymous reviewers for useful comments and suggestions; Aristeidis Panos, Daniela Massiceti, and Shoaib Ahmed Siddiqui for providing suggestions and feedback on the preliminary version of the manuscript.

References

Bateni, P., Goyal, R., Masrani, V., Wood, F., and Sigal, L. (2020). Improved few-shot visual classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Bronskill, J., Massiceti, D., Patacchiola, M., Hofmann, K., Nowozin, S., and Turner, R. (2021). Memory efficient meta-learning with large images. *Advances in Neural Information Processing Systems*.

Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*.

Garnelo, M., Schwarz, J., Rosenbaum, D., Viola, F., Rezende, D. J., Eslami, S., and Teh, Y. W. (2018). Neural processes. *arXiv preprint arXiv:1807.01622*.

Gordon, J., Bronskill, J., Bauer, M., Nowozin, S., and Turner, R. E. (2018). Meta-learning probabilistic inference for prediction. *arXiv preprint arXiv:1805.09921*.

Hospedales, T., Antoniou, A., Micaelli, P., and Storkey, A. (2020). Meta-learning in neural networks: A survey. *arXiv preprint arXiv:2004.05439*.

Hu, J., Shen, L., and Sun, G. (2018). Squeeze-and-excitation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Oreshkin, B., Rodríguez López, P., and Lacoste, A. (2018). Tadam: Task dependent adaptive metric for improved few-shot learning. *Advances in neural information processing systems*, 31.

Patacchiola, M., Turner, J., Crowley, E. J., O'Boyle, M., and Storkey, A. J. (2020). Bayesian meta-learning for the few-shot setting via deep kernels. *Advances in Neural Information Processing Systems*.

Perez, E., Strub, F., De Vries, H., Dumoulin, V., and Courville, A. (2018). Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Requeima, J., Gordon, J., Bronskill, J., Nowozin, S., and Turner, R. E. (2019). Fast and flexible multi-task classification using conditional neural adaptive processes. *Advances in Neural Information Processing Systems*.

Schmidhuber, J. (1987). *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München.

Sendera, M., Tabor, J., Nowak, A., Bedychaj, A., Patacchiola, M., Trzcinski, T., Spurek, P., and Zieba, M. (2021). Non-gaussian gaussian processes for few-shot regression. *Advances in Neural Information Processing Systems*.

Snell, J., Swersky, K., and Zemel, R. (2017). Prototypical networks for few-shot learning. *Advances in Neural Information Processing Systems*.

Triantafillou, E., Zhu, T., Dumoulin, V., Lamblin, P., Evci, U., Xu, K., Goroshin, R., Gelada, C., Swersky, K., Manzagol, P.-A., et al. (2019). Meta-dataset: A dataset of datasets for learning to learn from few examples. *arXiv preprint arXiv:1903.03096*.

Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D., et al. (2016). Matching networks for one shot learning. *Advances in Neural Information Processing Systems*.

Zhai, X., Puigcerver, J., Kolesnikov, A., Ruyssen, P., Riquelme, C., Lucic, M., Djolonga, J., Pinto, A. S., Neumann, M., Dosovitskiy, A., et al. (2019). A large-scale study of representation learning with the visual task adaptation benchmark. *arXiv preprint arXiv:1910.04867*.

Zintgraf, L., Shiarli, K., Kurin, V., Hofmann, K., and Whiteson, S. (2019). Fast context adaptation via meta-learning. In *International Conference on Machine Learning*.

A CaSE: additional details

A.1 CaSE implementation

Standardization Empirically we have observed that standardizing the pooled representations before passing them to the MLP improves the training stability in CaSE (but not in SE). Standardization is performed by taking the pooled representation at layer l as showed in Equation (3), that is $\bar{\mathbf{h}}^{(l)} \in \mathbb{R}^C$, subtracting the mean and dividing by the standard deviation.

Activation function for the output layer Standard SE blocks usually rely on a sigmoid function in the last layer of the MLPs. This works well when the adaptive block is trained in parallel with the underlying neural network. However, in our case we use a pretrained model and learning can be speeded up considerably by enforcing the identity function as output of the MLPs. We achieve this by multiplying the output of the sigmoid by a constant scalar $c = 2$ which extends the range to $[0, 2]$, and then set to zero the weights and bias of the layer. This has the effect of enforcing the identity function at the beginning of the training. We have also used a linear activation function instead of a sigmoid, with good results. When using a linear output the identity can be enforced by setting the weights of the last layer to zero, and the bias to one.

CaSE location For the choice of CaSE location in the feature extractor, we followed the same principles used in Bronskill et al. (2021) for FiLM generators. In EfficientNetB0 we place CaSE at the beginning of each hyperblock and the last layer (excluding the first layer). Differently from FiLM (placed after the BatchNorm) we place CaSE after the non-linearity (as done in standard SE) and before the Squeeze-and-Excitation block (included by default in EfficientNet):

Conv2d→BatchNorm2d→SiLU→CaSE→SqueezeExcitation→Conv2d→BatchNorm2d

This results in a total of 18 CaSE blocks for EfficientNetB0. Increasing the number of blocks did not provide a significant benefit. In ResNet18 we place two CaSE blocks per each basic block as:

Conv2d→BatchNorm2d→ReLU→CaSE→Conv2d→BatchNorm2d→ReLU→CaSE

Similarly we place two CaSE blocks inside a bottleneck block in ResNet50. See the code for more details.

Based on the qualitative analysis reported in Appendix D we hypothesize that adaptive blocks are not needed in the initial layers of the network, since at those stages their activity is minimal. Identifying which layer needs adapters and which layer does not, can reduce even more the parameter count of adaptive blocks. Additional work is needed to fully understand this factor.

CaSE reduction The number of parameters allocated to the CaSE blocks is regulated by a divider r that is used to compute the number of hidden units in the MLPs. Given the input size C (corresponding to the number of channels in that layer) the number of hidden units is given by C/r. We also use a clipping factor r_{min} that prevents the number of units to fall under a given threshold. This prevents the allocation of a low number of units for layers with a small number of channels.

A.2 Context pooling

In this section we provide additional details about the context pooling operation performed in a CaSE adaptive block (described in Section 2).

Similarities with other methods Context pooling is a way to summarize a task with a permutation-invariant aggregation of the embeddings. A similar mechanism has been exploited in various meta-learning methods. For instance, in ProtoNets (Snell et al., 2017) a prototype for a single class is computed by taking the average over all the context embeddings associated to the inputs for that class. The embeddings are generated in the last layer of the feature extractor. In Simple-CNAPs (Bateni et al., 2020) a prototype is estimated as in ProtoNets but it is used to define a Gaussian distribution instead of a mean vector. Neural latent variable models, such as those derived from the Neural Processes family (Garnelo et al., 2018) also rely on similar permutation-invariant aggregations to define distributions over functions.

Global vs. local context-pooling Comparing CaSE with the FiLM generators of Bronskill et al. (2021) it is possible to distinguish between two types of context pooling: global and local. The FiLM

generators of Bronskill et al. (2021) rely on a *global* pooling strategy, meaning that the aggregation is performed once-for-all by using a dedicated convolutional set encoder. More specifically, the encoder takes as input all the context images and produces embeddings for each one of them, followed by an average-pooling of those embeddings. The aggregated embedding is then passed to MLPs in each layer that generates a scale and shift parameter. Crucially, each MLP receives the same embedding.

CaSE exploits a *local* context-pooling at the layer level. The convolutional set encoder is discarded, and the feature maps produces by the backbone itself at each stage are used as context embeddings. Therefore, the MLPs responsible for generating the scale parameters receive a unique embedding. As showed in the experimental section, local pooling improves performances and uses less parameters, as no convolutional encoder is needed.

B Tabular results

Table 1: Comparing CaSE adaptive blocks (with reduction 64, 32, 16) on VTAB+MD against the FiLM generators used in Bronskill et al. (2021), standard Squeeze-and-Excitation (SE) (Hu et al., 2018), and a baseline with no body adaptation. CaSE blocks are more efficient in terms of adaptive and amortization parameters while providing higher classification accuracy. All models have been trained and tested on 84×84 images, using a Mahalanobis distance head. Best results in bold.

Adaptation type	None	FiLM	SE32	CaSE64	CaSE32	CaSE16
Adaptive Params (M)	n/a	0.02	**0.01**	**0.01**	**0.01**	**0.01**
Amortiz. Params (M)	n/a	1.7	0.8	**0.4**	0.8	1.6
MetaDataset (all)	53.4	68.4	67.8	69.8	69.6	**70.4**
VTAB (all)	43.5	44.7	43.6	46.2	45.3	**46.4**
VTAB (natural)	45.4	49.5	47.5	52.1	50.2	**52.6**
VTAB (specialized)	**69.4**	63.8	63.6	66.3	64.9	65.5
VTAB (structured)	29.1	31.7	30.6	31.8	31.8	**32.1**

C Pytorch code for CaSE

Implementation of a CaSE adaptive block in Pytorch. The script is also available as `case.py` at `https://github.com/mpatacchiola/contextual-squeeze-and-excitation`.

```
import torch
from torch import nn

class CaSE(nn.Module):
  def __init__(self, cin, reduction=32, min_units=32,
               standardize=True, out_mul=2.0,
               device=None, dtype=None):
    """
    Initialize a CaSE adaptive block.

    Parameters:
    cin (int): number of input channels.
    reduction (int): divider for computing number of hidden units.
    min_units (int): clip hidden units to this value (if lower).
    standardize (bool): standardize the input for the MLP.
    out_mul (float): multiply the MLP output by this value.
    """
    factory_kwargs = {'device': device, 'dtype': dtype}
    super(CaSE, self).__init__()
    self.cin = cin
    self.standardize = standardize
    self.out_mul = out_mul
    hidden = max(min_units, cin // reduction)
    self.gamma_generator = nn.Sequential(
      nn.Linear(cin, hidden, bias=True, **factory_kwargs),
      nn.SiLU(),
      nn.Linear(hidden, hidden, bias=True, **factory_kwargs),
      nn.SiLU(),
      nn.Linear(hidden, cin, bias=True, **factory_kwargs),
      nn.Sigmoid() )
    self.reset_parameters()

  def reset_parameters(self):
    nn.init.zeros_(self.gamma_generator[4].weight)
    nn.init.zeros_(self.gamma_generator[4].bias)
    self.gamma = torch.tensor([1.0])

  def forward(self, x):
    if(self.training): # adaptive mode
      self.gamma = torch.mean(x, dim=[2,3]) # spatial pooling
      self.gamma = torch.mean(self.gamma, dim=[0]) # context pooling
      if(self.standardize):
        self.gamma = (self.gamma - torch.mean(self.gamma)) / \
            torch.sqrt(torch.var(self.gamma, unbiased=False) + 1e-5)
      self.gamma = self.gamma.unsqueeze(0)
      self.gamma = self.gamma_generator(self.gamma) * self.out_mul
      self.gamma = self.gamma.reshape([1,-1,1,1])
      return self.gamma * x
    else: # inference mode
      self.gamma = self.gamma.to(x.device)
      return self.gamma * x

  def extra_repr(self):
      return 'cin={}'.format(self.cin)
```

D Role of CaSE blocks

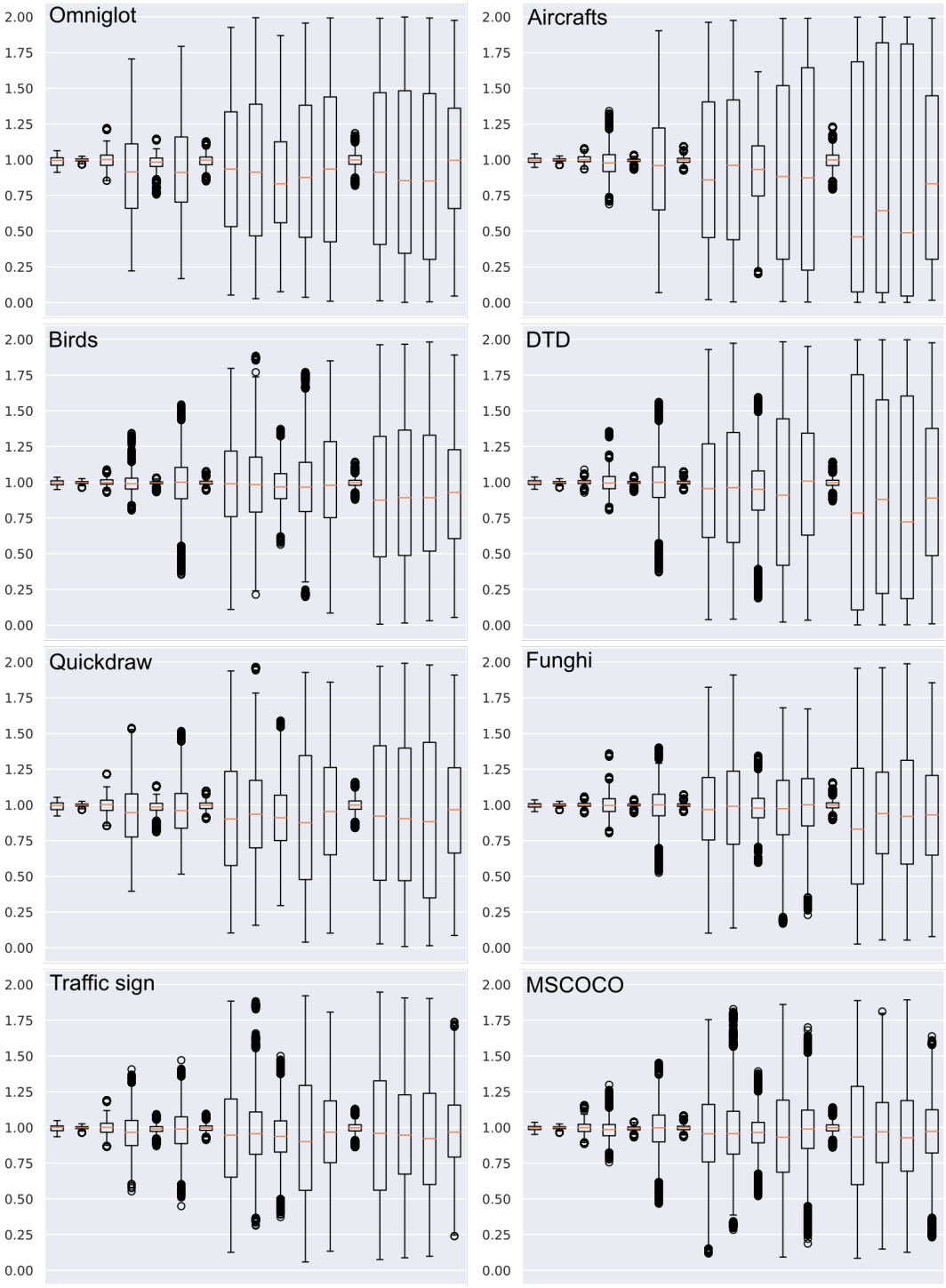

Figure 3: Boxplots for all the MDv2 test datasets (100 tasks per dataset) reporting the CaSE activation (vertical axis) at different stages of an EfficientNetB0 (horizontal axis, with early stages on the left). The box encloses first to third quartile, with the median represented by the orange line. The whiskers extend from the box by 1.5 the inter-quartile range. Outlier (point past the end of the whiskers) are represented with black circles.

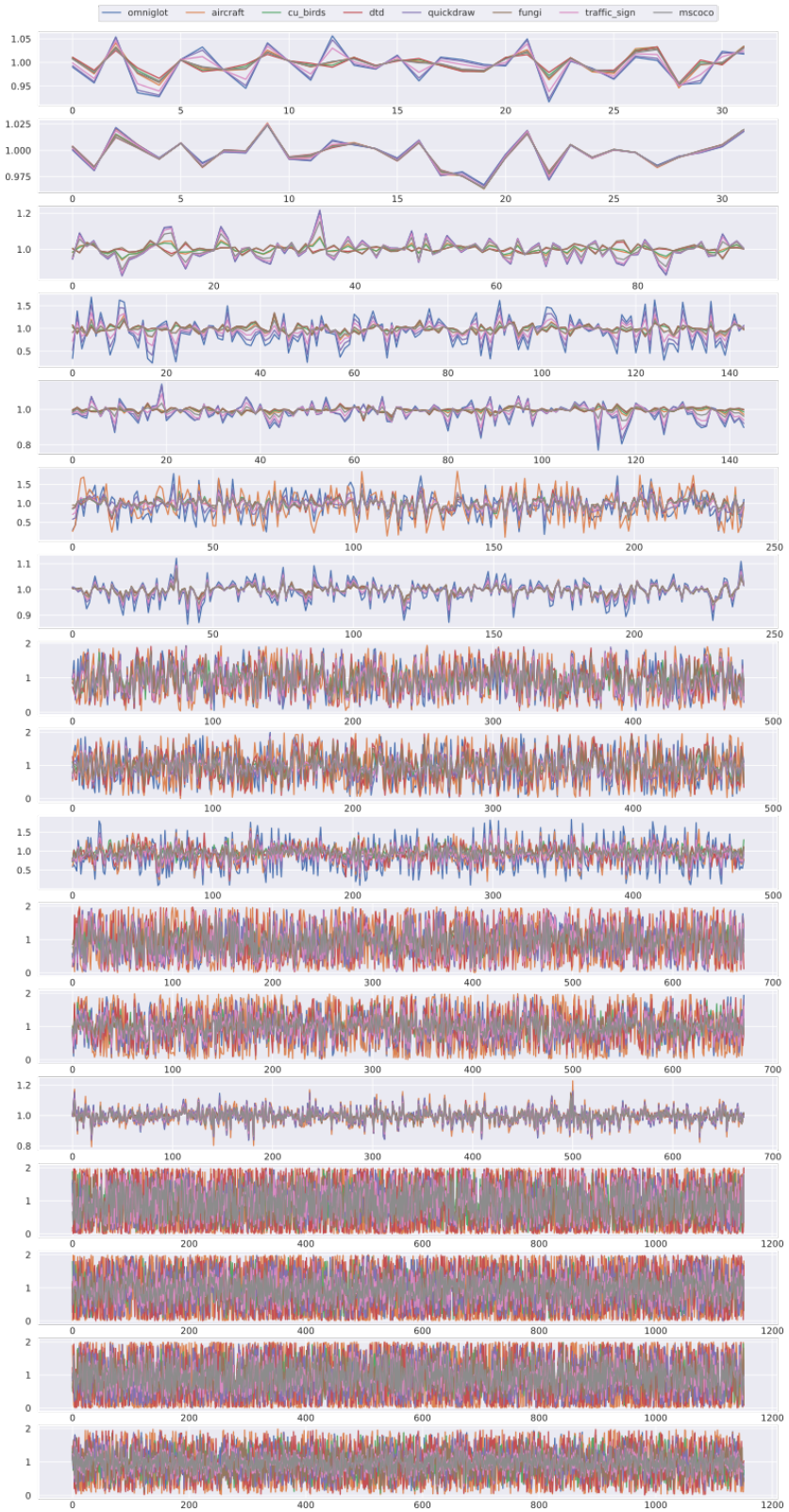

Figure 4: CaSE activation values (vertical axis) for all channels (horizontal axis) at different stages (top plots are early stages) in EfficientNetB0 for the MDv2 test dataset (one task per dataset). Values are similar and closer to one in the first stages but diverge in the latest. The magnitude tends to increase with depth.