Contextual Squeeze-and-Excitation for Efficient Few-Shot Image Classification

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

Recent years have seen a growth in user-centric applications that require effective knowledge transfer across tasks in the low-data regime. An example is personalization, where a pretrained system is adapted by learning on small amounts of labeled data belonging to a specific user. This setting requires high accuracy under low computational complexity, therefore the Pareto frontier of accuracy vs. adaptation cost plays a crucial role. In this paper we push this Pareto frontier in the few-shot image classification setting with a key contribution: a new adaptive block called Contextual Squeeze-and-Excitation (CaSE) that adjusts a pretrained neural network on a new task to significantly improve performance with a single forward pass of the user data (context). We use meta-trained CaSE blocks to conditionally adapt the body of a network and a fine-tuning routine to adapt a linear head, defining a method called UpperCaSE. UpperCaSE achieves a new state-of-the-art accuracy relative to meta-learners on the 26 datasets of VTAB+MD and on a challenging real-world personalization benchmark (ORBIT), narrowing the gap with leading fine-tuning methods with the benefit of orders of magnitude lower adaptation cost.

1 Introduction

In recent years, the growth of industrial applications based on recommendation systems (Bennett et al., 2007), speech recognition (Xiong et al., 2018), and personalization (Massiceti et al., 2021) has sparked an interest in machine learning techniques that are able to adapt a model on small amounts of data belonging to a specific user. A key factor in many of these applications is the Pareto frontier of accuracy vs. computational complexity (cost to adapt). For example, in a real-time classification task on a phone, a pretrained model must be personalized by exploiting small amounts of data on the user's device (context). In these applications the goal is twofold: maximize the classification accuracy on unseen data (target) while avoiding any latency and excessive use of computational resources.

Methods developed to face these challenges in the few-shot classification setting can be grouped in two categories: meta-learning and fine-tuning. Meta-learning is based on the idea of learning-how-to-learn by improving the algorithm itself (Schmidhuber, 1987; Hospedales et al., 2020). Meta-learners are trained across multiple tasks to ingest a labeled context set, adapt the model, and predict the class membership of an unlabeled target point. Fine-tuning methods adjust the parameters of a pretrained neural network on the task at hand by iterative gradient-updates (Chen et al., 2019; Triantafillou et al., 2019; Tian et al., 2020; Kolesnikov et al., 2020; Dumoulin et al., 2021).

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

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

We can gain an insight on the differences between those two paradigms by comparing them in terms of accuracy and adaptation cost. Figure 1 illustrates this comparison by showing on the vertical axis the average classification accuracy on the 18 datasets of the Visual Task Adaptation Benchmark (VTAB, Dumoulin et al. 2021), and on the horizontal axis the adaptation cost measured as the number of multiply–accumulate operations (MACs) required to adapt on a single task (see Appendix C.1 for details). Overall, fine-tuners achieve a higher classification accuracy than meta-learners but are more expensive to adapt. The comparison between two state-of-the-art methods for both categories, Big Transfer (BiT, Kolesnikov et al. 2020) and LITE (Bronskill et al., 2021), shows a substantial performance gap of 14% in favor of the fine-tuner but at a much higher adaptation cost, with BiT requiring 526×10^{12} MACs and LITE only 0.2×10^{12} MACs.

It is crucial to find solutions that retain the best of both worlds: the accuracy of fine-tuners and low adaptation cost of meta-learners. The main bottleneck that hampers the adaptation of fine-tuners is the need for multiple gradient adjustments over the entire set of network parameters. Restricting those adjustments to the last linear layer (head) significantly speeds up fine-tuning, but it harms performance (e.g. see experiments in Section 5.1). Finding a way to rapidly adapt the feature extractor (body) is therefore the main obstacle to bypass. In this paper we propose a hybrid solution to this issue, exploiting meta-learned adapters for rapidly adjusting the body and a fine-tuning routine for optimizing the head.

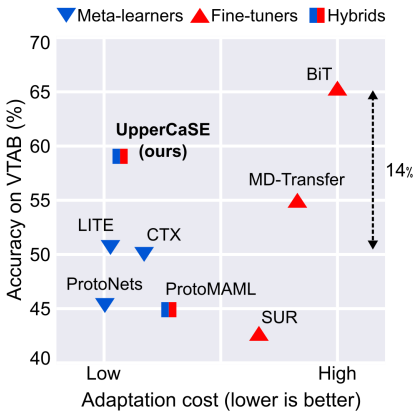

Figure 1: Accuracy and adaptation cost on VTAB for meta-learners (blue), fine-tuners (red), and hybrids (blue-red). Black dotted-line is the previous Pareto front across categories. UpperCaSE narrows the gap with the leading fine-tuning method and represents the best trade-off in terms of accuracy/adaptation-cost.

At the core of our approach is a novel extension of the popular Squeeze-and-Excitation block proposed by Hu et al. (2018) to the meta-learning setting that we call **C**ontextu**a**l **S**queeze-and-**E**xcitation (**CaSE**). We exploit CaSE as building block of a hybrid training protocol called **UpperCaSE** which is based on the idea of adjusting the body of the network in a single forward pass over the context, and reserving the use of expensive fine-tuning routines for the linear head, similarly to methods like MetaOptNet (Lee et al., 2019), R2D2 (Bertinetto et al., 2018), and ANIL (Raghu et al., 2019). Figure 1 shows how UpperCaSE substantially improves the performance in the low-cost regime, outperforming meta-learners, fine-tuners such as MD-Transfer (Triantafillou et al., 2019), and reducing the gap with the current state of the art (BiT). When adaptation cost is critical, UpperCaSE is the best method currently available since it can provide substantial computation savings and compelling classification performance.

Our *contributions* can be summarized as follows:

1. We introduce a new adapter called **C**ontextu**a**l **S**queeze-and-**E**xcitation (**CaSE**), based on the popular Squeeze-and-Excitation model proposed by Hu et al. (2018), that outperforms other adaptation mechanisms (e.g. the FiLM generators used in Bronskill et al. 2021) 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 [1].
2. We use CaSE adaptive blocks in conjuction with a fine-tuning routine for the linear head in a model called UpperCaSE, reporting an improved classification accuracy compared to the SOTA meta-learner (Bronskill et al., 2021) on the 8 datasets of MDv2 ($+2.5\%$ on average) and the 18 datasets of VTAB ($+6.8\%$ on average), narrowing the gap with BiT (Kolesnikov et al., 2020) with the benefit of orders of magnitude lower adaptation cost.
3. We showcase the potential of UpperCaSE in a real-world personalization task on the ORBIT dataset (Massiceti et al., 2021), where it compares favorably with the leading methods in the challenging cross-domain setting (training on MDv2, testing on ORBIT).

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

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 fine-tuning methods, we are given a neural network $f_{\boldsymbol{\theta}}(\cdot)$, with parameters $\boldsymbol{\theta}$ estimated via standard supervised-learning on a large labeled dataset (e.g. ImageNet). Given a test task τ_* adaptation consists of minimizing the loss $\mathcal{L}(\cdot)$ via gradient updates to find the task-specific parameters $\boldsymbol{\theta}_{\tau_*} \leftarrow G(\epsilon, \mathcal{L}, \tau_*, f_{\boldsymbol{\theta}})$, where ϵ is a learning rate, and $G(\cdot)$ is a functional representing an iterative routine that returns the adapted parameters $\boldsymbol{\theta}_{\tau_*}$ (used for prediction). This procedure is particularly effective because it can exploit efficient mini-batching, parallelization, and large pretrained models.

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. The distinction in tasks is exploited to define a hierarchy. The parameters are divided in two groups: $\boldsymbol{\phi}$ task-common parameters shared across all tasks (top of the hierarchy), and $\boldsymbol{\psi}_\tau$ task-specific parameters estimated on the task at hand as part of an adaptive mechanism (bottom of the hierarchy). The way $\boldsymbol{\phi}$ and $\boldsymbol{\psi}_\tau$ come into play is method dependent; they 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 2 (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 $\tau = (\mathcal{C}, \mathcal{T})$, 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

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

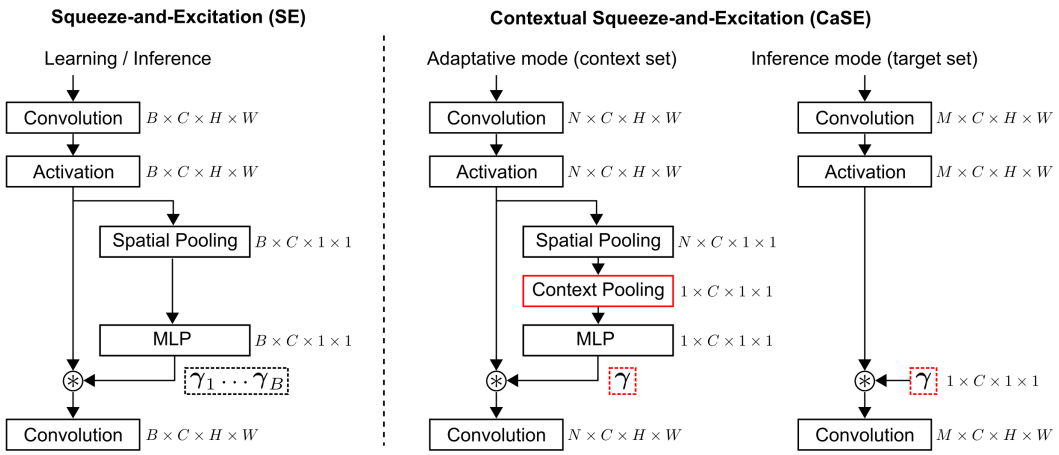

Figure 2: Comparison between the standard Squeeze-Excite (left) and the proposed Contextual Squeeze-Excite (right). Red frames highlight the two key differences between SE and CaSE: context pooling and scale transfer from context to target. B = mini-batch size, C = channels, H = height, W = width, N = context-set size, M = target-set size, $*$ elementwise multiplication.

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 2. 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 5.1 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 2 (right), additional technical details are provided in Appendix A.

Comparison with other adapters Popular adaptation mechanisms for few-shot learning are based on Feature-wise Linear Modulation layers (FiLM, Perez et al. 2018). Those mechanisms perform adaptation using a separate convolutional set-encoder to produce an embedding of the context set. The embedding is forwarded to local MLPs to produce the scale and shift vectors of the FiLM layers that modulate a pretrained model. 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). 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. There are two key differences between FiLM and CaSE: (i) CaSE exploits context pooling to aggregate the activations of the backbone instead of a separate set-encoder as in FiLM generators (see Appendix A for details) which is more efficient in terms of parameter count and implementation overhead; and (ii) FiLM uses scale and shift to modulate the activations, CaSE only the scale, therefore 50% less parameters are stored in memory and transferred during inference. In Section 5.1 we compare CaSE and the FiLM generators used in a recent SOTA method (LITE, Bronskill et al. 2021), showing that CaSE is superior in terms of accuracy while using a fraction of the amortization parameters.

3 UpperCaSE: system description and optimization protocol

We exploit CaSE blocks as part of UpperCaSE, a hybrid training protocol based on Coordinate-Descent (CD). We call this protocol *hybrid* because it combines a meta-training procedure to optimize the CaSE parameters (body) with a fine-tuning routine to estimate the task-specific parameters (head).

Preliminaries We are given a feature extractor (body) pretrained with supervised learning on a large dataset (e.g. ImageNet), defined as $b_{\boldsymbol{\theta}}(\cdot)$ where $\boldsymbol{\theta}$ are the pretrained parameters. CaSE blocks,

parameterized by $\boldsymbol{\phi}$, are added to the model at specific locations to give $b_{\boldsymbol{\theta},\boldsymbol{\phi}}(\cdot)$ (see Appendix A for details about this step). We are interested in learning the CaSE parameters $\boldsymbol{\phi}$ keeping constant the pretrained parameters $\boldsymbol{\theta}$ (omitted from here to keep the notation uncluttered). At training time, we are given a series of tasks $\tau = \{\mathcal{C}, \mathcal{T}\} \sim \mathcal{D}$, where $\mathcal{D}$ is the training set. The number of classes (way) is calculated from the context set and used to define a linear classification head $h_{\boldsymbol{\psi}_\tau}(\cdot)$ parameterized by $\boldsymbol{\psi}_\tau$. The complete model is obtained by nesting the two functions as $h_{\boldsymbol{\psi}_\tau}(b_{\boldsymbol{\phi}}(\cdot))$. We indicate a forward pass through the body over the context inputs with the shorthand $b_{\boldsymbol{\phi}}(\mathcal{C}^x) \rightarrow \{\mathbf{z}_1, \ldots, \mathbf{z}_N\}$, where $\mathbf{z}_n$ is the context embedding for the input $\mathbf{x}_n$. All the context embeddings and the associated labels are stored in $\mathcal{M} = \{(\mathbf{z}_n, y_n)\}_{n=1}^N$.

Optimization challenges We have two sets of learnable parameters, $\boldsymbol{\phi}$ the CaSE parameters, and $\boldsymbol{\psi}_\tau$ the parameters of the linear head for the task τ. While $\boldsymbol{\phi}$ is shared across all tasks (task-common), $\boldsymbol{\psi}_\tau$ must be inferred on the task at hand (task-specific). In both cases, the objective is the minimization of a classification loss $\mathcal{L}$. There are some challenges in optimizing the CaSE parameters in the body, as shown by the decomposition of the full gradient

$$\frac{d\mathcal{L}}{d\boldsymbol{\phi}} = \sum_\tau \left(\frac{\partial \mathcal{L}_\tau}{\partial \boldsymbol{\psi}_\tau} \frac{d\boldsymbol{\psi}_\tau}{d\boldsymbol{\phi}} + \frac{\partial \mathcal{L}_\tau}{\partial \boldsymbol{\phi}} \right). \tag{5}$$

The first term $\partial \mathcal{L}_\tau / \partial \boldsymbol{\psi}_\tau$ (sensitivity of the loss w.r.t. the head) and the direct gradient $\partial \mathcal{L} / \partial \boldsymbol{\phi}$ (sensitivity of the loss w.r.t. the adaptation parameters with a fixed head) can be obtained with auto-differentiation as usual. The second term $d\boldsymbol{\psi}_\tau / d\boldsymbol{\phi}$ (sensitivity of the head w.r.t. the adaptation parameters) is problematic because $\boldsymbol{\psi}_\tau$ is obtained iteratively after a sequence of gradient updates. Backpropagating the gradients to $\boldsymbol{\phi}$ includes a backpropagation through all the gradient steps performed to obtain the task-specific $\boldsymbol{\psi}_\tau$. Previous work has showed that this produces instability, vanishing gradients, and high memory consumption (Antoniou et al., 2018; Rajeswaran et al., 2019).

Meta-training via Coordinate-Descent A potential solution to these issues is the use of implicit gradients (Chen et al., 2020; Rajeswaran et al., 2019; Chen et al., 2022). The main problem with implicit gradients is the computation and inversion of the Hessian matrix as part of Cauchy's implicit function theorem, which is infeasible when the number of parameters in the linear head is large. Another possible solution is the use of an alternating-optimization scheme, similar to the one proposed in a number of recent methods such as MetaOptNet (Lee et al., 2019), R2D2 (Bertinetto et al., 2018), and ANIL (Raghu et al., 2019). These methods share the idea of inner-loop-head/outer-loop-body meta-training, and they find the parameters of the linear head with closed form solutions or by stochastic optimization. Starting from similar assumptions we propose a simple yet effective alternating-optimization scheme, which we formalize using Coordinate-Descent (CD) (Wright, 2015). The idea behind CD is to consider the minimization of a complex multi-variate function as a set of simpler objectives that can be solved one at a time. In our case, we can consider the joined landscape w.r.t. $\boldsymbol{\phi}$ and $\boldsymbol{\psi}_\tau$ as composed of two separate sets of coordinates (block CD, Wright 2015). By minimizing $\boldsymbol{\psi}_\tau$ first, we reach a local minimum where $\partial \mathcal{L}_\tau / \partial \boldsymbol{\psi}_\tau \approx 0$. Therefore CD induces a direct optimization objective w.r.t. $\boldsymbol{\phi}$, with Equation (5) reducing to $\partial \mathcal{L}_\tau / \partial \boldsymbol{\phi}$ (no red term). The time complexity of this method is only affected by the number of classes but is constant w.r.t. the number of training points due to the use of mini-batching, which scales well with large tasks (e.g. those in MetaDataset and VTAB). See Appendix B for more details.

In practice, at each training iteration we sample a task $\tau = (\mathcal{C}, \mathcal{T}) \sim \mathcal{D}$, perform a forward pass on the body (with CaSE in adaptive mode) to get

$$b_{\boldsymbol{\phi}}(\mathcal{C}^x) \rightarrow \{\mathbf{z}_1, \ldots, \mathbf{z}_N\}. \tag{6}$$

The context embeddings are temporarily stored in a buffer with their associated labels $\mathcal{M} = \{(\mathbf{z}_n, y_n)\}_{n=1}^N$ to avoid expensive calls to $b_{\boldsymbol{\phi}}(\cdot)$. We then set the head parameters to zero, and solve the first minimization problem (inner-loop), obtaining the task-specific parameters $\boldsymbol{\psi}_\tau$ via

$$\boldsymbol{\psi}_\tau \leftarrow G\left(\alpha, \mathcal{M}, \mathcal{L}, h_{\boldsymbol{\psi}_\tau}\right) \tag{7}$$

where α is a learning rate, and $G(\cdot)$ is a functional representing an iterative gradient-descent routine for parameter estimation (e.g. maximum likelihood estimation or maximum a posteriori estimation). Note that the iterative routine in Equation (7) only relies on the head $h_{\boldsymbol{\psi}_\tau}(\cdot)$ and not on the body $b_{\boldsymbol{\phi}}(\cdot)$, which is the primary source of memory savings and the crucial difference with common fine-tuning methods. Moreover, the inner-loop is agnostic to the choice of optimizer, it can handle many gradient steps without complications, exploit parallelization and efficient mini-batching.

We then turn our attention to the second coordinate: the task-common parameters of the CaSE blocks in the body. For a single task, the update consists of a single optimization step w.r.t. $\boldsymbol{\phi}$ (outer-loop) given support/target points and the task-specific parameters $\boldsymbol{\psi}_\tau$ identified previously. The final form of the equation depends on the optimizer, for a generic SGD the update is given by

$$\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} - \beta \nabla_{\boldsymbol{\phi}} \mathcal{L}\left(\mathcal{C}^y \cup Q^y, h_{\boldsymbol{\psi}_\tau}, b_{\boldsymbol{\phi}}\right), \tag{8}$$

where β is a learning rate. CaSE blocks must be in adaptive mode to allow the backpropagation of the gradients to the MLPs. The process repeats, alternating the minimization along the two sets of coordinates. The pseudo-code for train and test is provided in Appendix B.

Inference on unseen tasks After the training phase, we are given an unseen task $\tau_* = (\mathcal{C}_*, \mathbf{x}_*)$ where $\mathbf{x}_*$ is a single target input and y_* the associate true label to estimate. Inference consists of three steps: (i) forward pass on the body for all the context inputs with CaSE set to adaptive mode as in Equation (6) and embeddings/labels stored in $\mathcal{M}$, (ii) estimation of the task-specific parameters $\boldsymbol{\psi}_*$ via iterative updates as in Equation (7), and (iii) inference of the target-point membership via a forward pass over body and head $\hat{y}_* = h_{\boldsymbol{\psi}_*}\left(b_{\boldsymbol{\phi}}(\mathbf{x}_*)\right)$ with CaSE in inference mode.

4 Related work

Meta-learning There has been a large volume of publications related to meta-learning. Here we focus on those methods that are the most related to our work, and refer the reader to a recent survey for additional details (Hospedales et al., 2020). LITE (Bronskill et al., 2021) is a protocol for training meta-learners on large images, that achieved SOTA accuracy on VTAB+MD. LITE is particularly relevant in this work, as its best performing method is based on Simple CNAPs (Bateni et al., 2020) that exploits FiLM for fast body adaptation. We compare against LITE in Section 5.2 showing that UpperCaSE is superior in terms of classification accuracy and parameter efficiency.

Fine-tuning Chen et al. (2019) were the first to expose the potential of simple fine-tuning baselines for transfer learning. MD-Transfer has been proposed in Triantafillou et al. (2019) as an effective fine-tuning baseline for the MetaDataset benchmark. More recently Kolesnikov et al. (2020) have presented Big Transfer (BiT), showing that large models pretrained on ILSVRC-2012 ImageNet and the full ImageNet-21k are very effective at transfer learning. MD-Transfer and BiT differ in terms of classification head, learning schedule, normalization layers, and batching. Fine-tuning only the last linear layer can be effective (Bauer et al., 2017; Tian et al., 2020). We compare against this baseline in Section 5.1, showing that adapting the body via CaSE significantly boosts the performance. Overall, fine-tuners have consistently outperformed meta-learners in terms of classification accuracy, only under particular conditions (e.g. strong class-imbalance) the trend is reversed (Ochal et al., 2021a,b).

Hybrids Hybrid methods are trained episodically like meta-learners but rely on fine-tuning routines for adaptation. Model Agnostic Meta-Learning (MAML, Finn et al. 2017) finds a set of parameters that is a good starting point for adaptation towards new tasks in a few gradient steps. MAML has been the inspiration for a series of other models such as MAML++ (Antoniou et al., 2018), ProtoMAML (Triantafillou et al., 2019), and Reptile (Nichol et al., 2018).

Dynamic networks CaSE blocks belong to the wider family of dynamic networks, models that can adapt their structure or parameters to different inputs (Han et al., 2021). Adaptive components have been used in a variety of applications, such as neural compression (Veit and Belongie, 2018; Wu et al., 2018), generation of artistic styles (Dumoulin et al., 2016; Huang and Belongie, 2017), or routing (Guo et al., 2019). Residual adapters (Rebuffi et al., 2017, 2018) have been used in transfer learning (non few-shot) but they rely on fine-tuning routines which are significantly slow during adaptation. More recently, Li et al. (2022) have used serial and residual adapters in the few-shot setting, with the task-specific weights being adapted from scratch on the context set. This approach has similar limitations, since it requires backpropagation to the task-specific weights in the body of the network which is costly. In Sun et al. (2019) the authors introduce a Meta-Transfer Learning (MTL) method for the few-shot setting. In MTL a series of scale and shift parameters are meta-learned across tasks and then dynamically adapted during the test phase via fine-tuning. This method suffers of similar limitations, as the fine-tuning stage is expensive during adaptation. Moreover, MTL relies on scale and shift vectors to perform adaptation whereas CaSE only relies on a scale vector, meaning that it needs to store and transfer 50% less parameters at test time.

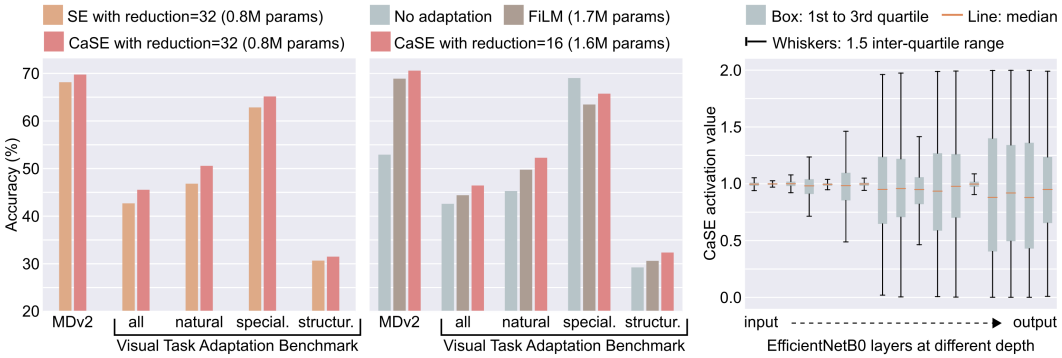

Figure 3: **Left**: CaSE vs Squeeze-and-Excitation (SE) (both methods use EfficientNetB0, 84×84 inputs, Mahalanobis-distance head). CaSE outperforms SE in all conditions. **Center**: CaSE vs. FiLM generators (Bronskill et al., 2021) and a baseline with no body adaptation (all methods use EfficientNetB0, 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 (224×224 inputs, UpperCaSE). The modulation of CaSE is minimal at early stages for general-purpose filters and increases at deeper stages.

5 Experiments

In this section we report on experiments on VTAB+MD (Dumoulin et al., 2021) and ORBIT (Massiceti et al., 2021). VTAB+MD has become the standard evaluation protocol for few-shot approaches, and it includes a large number of datasets (8 test dataset for MD, 18 for VTAB). For a description of ORBIT, see Section 5.2. In all experiments we used the following pretrained (on ImageNet) backbones: EfficientNetB0 from the official Torchvision repository; ResNet50x1-S released with BiT (Kolesnikov et al., 2020). We used three workstations (CPU 6 cores, 110GB of RAM, and a Tesla V100 GPU), the meta-training protocol of Bronskill et al. (2021) ($10K$ training tasks, updates every 16 tasks), the Adam optimizer with a linearly-decayed learning rate in $[10^{-3}, 10^{-5}]$ for both the CaSE and linear-head. The head is updated 500 times using a random mini-batch of size 128. MD test results are averaged over 1200 tasks per-dataset (confidence intervals in appendix). We did not use data augmentation. Code to reproduce the experiments is available at `https://github.com/mpatacchiola/contextual-squeeze-and-excitation`.

5.1 Analysis of CaSE blocks

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 and VTAB, 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), reporting results with a linear head in the appendix. We summarize the results in Figure 3 (left) and add a tabular breakdown in Appendix C.2. 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. We show a summary of the results in

Figure 3 (center) and provide a full breakdown in the appendix. CaSE is able to outperform FiLM generators in all conditions. In Appendix C.3 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 trained with UpperCaSE on 224×224 images. In Figure 3 (right) we report the aggregated distribution as boxplots, and in Appendix C.5 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 C.5 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. An ablation study of different factors (e.g. reduction, number of hidden layers, activation functions) is reported in Appendix C.4.

5.2 Performance evaluation of UpperCaSE

In this sub-section we analyze the performance of UpperCaSE in two settings: **1)** comparison on the VTAB+MD benchmark against SOTA fine-tuners and meta-learners, where we show that UpperCaSE is able to outperform all the meta-learners, narrowing the gap with Big Transfer (BiT) on VTAB; **2)** we show an application of UpperCaSE in a real-world personalization task on the challenging ORBIT dataset (Massiceti et al., 2021) for the cross-domain condition MDv2→ORBIT, where we achieve the best average-score in most metrics, although these improvements are within the error bars.

Table 1: **UpperCaSE outperforms fine-tuners on MDv2 and narrows the gap on VTAB with the leading method (BiT) with a much lower adaptation cost**. Average accuracy on the 26 datasets of VTAB+MD. RN=ResNet, EN=EfficientNet. Img: image size. Param.: total parameters (no adapters) in millions. Cost: MACs to adapt on a task (10-shot, 100-way), in Teras. Best results in bold.

					Cost ↓	**MDv2 ↑**	**VTAB ↑**			
Method	**Protocol**	**Net**	**Img**	**Param.**	**MACs**	**all**	**all**	**natur.**	**spec.**	**struc.**
MD-Transfer	fine-tuning	RN18	126	11.2	118.6	63.4	55.6	52.4	72.9	49.3
SUR	fine-tuning	RN50	224	164.6	28.8	71.3	43.7	50.9	66.2	27.2
Big Transfer	fine-tuning	RN50	224	23.5	526.3	73.3	**65.4**	**69.4**	**81.0**	**54.5**
UpperCaSE	hybrid	RN50	224	23.5	0.8	74.9	56.6	66.3	80.1	37.6
UpperCaSE	hybrid	ENB0	224	4.0	**0.2**	**76.1**	58.4	69.1	80.3	39.4

Table 2: **UpperCaSE outperforms all meta-learning/hybrid methods and uses the lowest number of parameters per adaptive blocks**. Average accuracy on the 26 datasets of VTAB+MD. RN=ResNet, EN=EfficientNet. Img: image size. Param.: total parameters (excluding adapters). Adapt.: total adaptive parameters in millions. Best results in bold.

					Adapt. ↓	**MDv2 ↑**	**VTAB ↑**			
Method	**Protocol**	**Net**	**Img**	**Param.**	**count**	**all**	**all**	**natur.**	**spec.**	**struc.**
ProtoMAML	hybrid	RN18	126	11.2	n/a	64.2	45.0	45.7	70.7	31.5
CTX	meta-learning	RN34	224	21.3	n/a	71.6	50.5	61.1	67.3	34.0
ProtoNet	meta-learning	ENB0	224	4.0	n/a	72.7	46.1	60.9	64.2	25.9
LITE	meta-learning	ENB0	224	4.0	1.7	73.8	51.4	65.2	71.9	30.8
UpperCaSE	hybrid	RN50	224	23.5	0.8	74.9	56.6	66.3	80.1	37.6
UpperCaSE	hybrid	ENB0	224	4.0	**0.4**	**76.1**	**58.4**	**69.1**	**80.3**	**39.4**

Comparison on VTAB+MD We compare UpperCaSE against fine-tuners, meta-learners, and hybrids on the 18 datasets of VTAB and the 8 datasets of MetaDataset-v2 (MDv2) and report the results

Table 3: **ORBIT: UpperCaSE obtains the best average-score in most metrics, being within error bars with leading methods.** Average accuracy and 95% confidence interval for frames, videos, and frames to recognition (FTR). Cost: average MACs over all tasks (Teras). Results and setup from Massiceti et al. (2021): meta-train on MetaDataset and test on ORBIT, image-size 84×84, ResNet18 backbone, 85 test tasks (17 test users, 5 tasks per user). Best results (within error bars) in bold.

	Cost	**Clean Video Evaluation (CLE-VE)**			**Clutter Video Evaluation (CLU-VE)**		
Method	**MACs↓**	**frame acc.↑**	**FTR↓**	**video acc.↑**	**frame acc.↑**	**FTR↓**	**video acc.↑**
ProtoNet	**3.2**	**59.0±2.2**	**11.5±1.8**	**69.2±3.0**	**47.0±1.8**	**20.4±1.7**	**52.8±2.5**
CNAPs	3.5	51.9±2.5	20.8±2.3	60.8±3.2	41.6±1.9	30.7±2.1	43.0±2.5
MAML	95.3	42.5±2.7	37.3±3.0	47.0±3.2	24.3±1.8	62.3±2.3	25.7±2.2
FineTuner	317.7	**61.0±2.2**	**11.5±1.8**	**72.6±2.9**	**48.4±1.9**	**19.1±1.7**	**54.1±2.5**
UpperCaSE	3.5	**63.0±2.2**	**8.8±1.6**	**74.4±2.8**	**48.1±1.8**	**18.2±1.7**	**54.5±2.5**

in Table 1 and Table 2. UpperCaSE outperforms all methods (including BiT) on MDv2 with an accuracy of 74.9% (ResNet50) and 76.1% (EfficientNetB0). On VTAB, UpperCaSE outperforms most methods, narrowing the gap with BiT. A closer look at the differences in performance on VTAB between UpperCaSE and BiT (see Table 1) shows that the gap is narrower on the natural and specialized splits ($+3.1\%$ and $+0.9\%$) but larger on structured ($+16.9\%$).

The breakdown by dataset reported in Appendix C.6 shows that the major performance drops are on tasks that require localization and counting (e.g. dSprites, SmallNORB). Similar issues are encountered by methods such as LITE (Bronskill et al., 2021) which are based on FiLM generators, suggesting that those tasks may introduce a strong domain shift w.r.t. the meta-training set that is difficult to compensate without fine-tuning the body. It is not clear whether transfer learning is beneficial on these datasets in the first place. The results in terms of adaptation cost (see Table 1) over a synthetic task (10-shot, 100 way) show that UpperCaSE is orders of magnitude more efficient (0.2×10^{12} MACs) than all fine-tuners, with BiT being the most expensive method overall (526.3×10^{12} MACs). The comparison against meta-learners in terms of number of adaptive parameters (see Table 2) shows that UpperCaSE requires a fraction of the parameters (0.4 vs 1.7 millions for an EfficientNetB0) compared to LITE (Bronskill et al., 2021) which is based on FiLM generators.

Comparison on ORBIT We compare UpperCaSE to other methods on ORBIT (Massiceti et al., 2021), a real-world dataset for teachable object recognizers. ORBIT consists of 3822 videos of 486 objects recorded by 77 blind/low-vision people on their mobile phones. The dataset is challenging because objects are poorly framed, occluded, blurred, and in a wide variation of backgrounds and lighting. The dataset includes two sets of target videos, one for clean video evaluation (CLE-VE) with well-centered objects, and another for clutter video evaluation (CLU-VE) with objects in complex, cluttered environments. We consider a hard transfer-learning condition where classifiers are meta-trained on MetaDataset and tested on ORBIT.

Results are reported in Table 3. UpperCaSE outperforms all other methods (on average) on most metrics, being within error bars with the two leading methods. Comparing UpperCaSE with FineTuner, the gap in favor of UpperCaSE is marginal on CLU-VE but substantial on CLE-VE (frame accuracy $+2\%$, video accuracy $+1.8\%$, and FTR -2.7). Comparison in terms of adaptation cost (average MACs over all tasks) shows that UpperCaSE is orders of magnitude more efficient than FineTuner and close to the leading method (ProtoNet).

6 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. Exploiting CaSE we have designed UpperCaSE, a hybrid method based on a Coordinate-Descent training protocol, that combines the performance of fine-tuners with the low adaptation cost of meta-learners. UpperCaSE achieves SOTA accuracy w.r.t. meta-learners on the 26 datasets of VTAB+MD and it compares favorably with leading methods in the ORBIT personalization benchmark.

Limitations There are two *limitations* that are worth mentioning: (i) UpperCaSE requires iterative gradient updates that are hardware-dependent and may be slow/unavailable in some portable devices; (ii) breakdown VTAB results per-dataset shows that the method falls short on structured datasets. This indicates that fine-tuning the body may be necessary for high accuracy when the shift w.r.t. the meta-training set is large.

Societal impact Applications based on CaSE and UpperCaSE could be deployed in few-shot classification settings that can have a positive impact such as: medical diagnosis, recommendation systems, object detection, etc. The efficiency of our method can reduce energy consumption and benefit the environment. Certain applications require careful consideration to avoid biases that can harm specific groups of people (e.g. surveillance, legal decision-making).

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

Antoniou, A., Edwards, H., and Storkey, A. (2018). How to train your maml. *arXiv preprint arXiv:1810.09502*.

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*.

Bauer, M., Rojas-Carulla, M., Świątkowski, J. B., Schölkopf, B., and Turner, R. E. (2017). Discriminative k-shot learning using probabilistic models. *arXiv preprint arXiv:1706.00326*.

Bennett, J., Lanning, S., et al. (2007). The netflix prize. In *Proceedings of KDD cup and workshop*.

Bertinetto, L., Henriques, J. F., Torr, P. H., and Vedaldi, A. (2018). Meta-learning with differentiable closed-form solvers. *arXiv preprint arXiv:1805.08136*.

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*.

Chen, W., Tripp, A., and Hernández-Lobato, J. M. (2022). Meta-learning feature representations for adaptive gaussian processes via implicit differentiation. *arXiv preprint arXiv:2205.02708*.

Chen, W.-Y., Liu, Y.-C., Kira, Z., Wang, Y.-C. F., and Huang, J.-B. (2019). A closer look at few-shot classification. *arXiv preprint arXiv:1904.04232*.

Chen, Y., Friesen, A. L., Behbahani, F., Doucet, A., Budden, D., Hoffman, M., and de Freitas, N. (2020). Modular meta-learning with shrinkage. *Advances in Neural Information Processing Systems*.

Dumoulin, V., Houlsby, N., Evci, U., Zhai, X., Goroshin, R., Gelly, S., and Larochelle, H. (2021). Comparing transfer and meta learning approaches on a unified few-shot classification benchmark. *arXiv preprint arXiv:2104.02638*.

Dumoulin, V., Shlens, J., and Kudlur, M. (2016). A learned representation for artistic style. *arXiv preprint arXiv:1610.07629*.

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*.

Guo, Y., Shi, H., Kumar, A., Grauman, K., Rosing, T., and Feris, R. (2019). Spottune: transfer learning through adaptive fine-tuning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Han, Y., Huang, G., Song, S., Yang, L., Wang, H., and Wang, Y. (2021). Dynamic neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

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*.

Huang, X. and Belongie, S. (2017). Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*.

Kolesnikov, A., Beyer, L., Zhai, X., Puigcerver, J., Yung, J., Gelly, S., and Houlsby, N. (2020). Big transfer (bit): General visual representation learning. In *European conference on computer vision*.

Lee, K., Maji, S., Ravichandran, A., and Soatto, S. (2019). Meta-learning with differentiable convex optimization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Li, W.-H., Liu, X., and Bilen, H. (2022). Cross-domain few-shot learning with task-specific adapters. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Massiceti, D., Zintgraf, L., Bronskill, J., Theodorou, L., Harris, M. T., Cutrell, E., Morrison, C., Hofmann, K., and Stumpf, S. (2021). Orbit: A real-world few-shot dataset for teachable object recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*.

Nichol, A., Achiam, J., and Schulman, J. (2018). On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*.

Ochal, M., Patacchiola, M., Storkey, A., Vazquez, J., and Wang, S. (2021a). Few-shot learning with class imbalance. *arXiv preprint arXiv:2101.02523*.

Ochal, M., Patacchiola, M., Storkey, A., Vazquez, J., and Wang, S. (2021b). How sensitive are meta-learners to dataset imbalance? *arXiv preprint arXiv:2104.05344*.

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*.

Raghu, A., Raghu, M., Bengio, S., and Vinyals, O. (2019). Rapid learning or feature reuse? towards understanding the effectiveness of maml. *arXiv preprint arXiv:1909.09157*.

Rajeswaran, A., Finn, C., Kakade, S. M., and Levine, S. (2019). Meta-learning with implicit gradients. *Advances in Neural Information Processing Systems*.

Rebuffi, S.-A., Bilen, H., and Vedaldi, A. (2017). Learning multiple visual domains with residual adapters. In *Advances in Neural Information Processing Systems*.

Rebuffi, S.-A., Bilen, H., and Vedaldi, A. (2018). Efficient parametrization of multi-domain deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

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*.

Sun, Q., Liu, Y., Chua, T.-S., and Schiele, B. (2019). Meta-transfer learning for few-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Tian, Y., Wang, Y., Krishnan, D., Tenenbaum, J. B., and Isola, P. (2020). Rethinking few-shot image classification: a good embedding is all you need? In *European Conference on Computer Vision*.

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*.

Veit, A. and Belongie, S. (2018). Convolutional networks with adaptive inference graphs. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

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

Wright, S. J. (2015). Coordinate descent algorithms. *Mathematical Programming*, 151.

Wu, Z., Nagarajan, T., Kumar, A., Rennie, S., Davis, L. S., Grauman, K., and Feris, R. (2018). Blockdrop: Dynamic inference paths in residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Xiong, W., Wu, L., Alleva, F., Droppo, J., Huang, X., and Stolcke, A. (2018). The microsoft 2017 conversational speech recognition system. In *IEEE international conference on acoustics, speech and signal processing (ICASSP)*.

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

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 6.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See dedicated paragraph in Section 6.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Included in the supplementary material.

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 5

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Error bars have been reported for ORBIT and for the breakdonwn of the results on MetaDataset (MD) in the supplementary material. The aggregated results on MD in the paper do not report error bars as they are the average across many datasets. On VTAB error bars are not provided as the benchmark is only used for evaluation (this is in line with the original protocol).

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See setup description at beginning of Section 5.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes]

(b) Did you mention the license of the assets? [Yes]

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

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 $\overline{\mathbf{h}}^{(l)} \in \mathbb{R}^C$, 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. An ablation over the activation function of SE and CaSE is provided in Appendix C.4 (Table 6).

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 Section 5 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 (Section 5), local pooling improves performances and uses less parameters, as no convolutional encoder is needed. Additional details about the differences between CaSE and FiLM generators is also provided in the paper (Section 4).

A.3 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)
```

B UppereCaSE: additional details

B.1 Algorithm of UpperCaSE

Algorithm 1 UpperCaSE: training function for the few-shot classification setting.

Require: $\mathcal{D} = \{\tau_1, \ldots, \tau_D\}$ training dataset
Require: $b_{\boldsymbol{\phi}}()$ pretrained feature extractor (body) with CaSE blocks parameterized by $\boldsymbol{\phi}$.
Require: `step()`: gradient-step function; $\mathcal{L}$ loss; α, β: step-size hyperparameters for the optimizer.

Set $\boldsymbol{\phi}$ to random values ▷ optional: set $\boldsymbol{\phi}$ to enforce identity in CaSE output
while not done **do**
  Sample task $\tau = (\mathcal{C}, \mathcal{T}) \sim \mathcal{D}$
  Forward pass over context set $b_{\boldsymbol{\phi}}(\mathcal{C}^x) \rightarrow \mathbf{z}_1, \ldots, \mathbf{z}_N$ ▷ CaSE in adaptive mode
  Store context embeddings and associated labels $\mathcal{M} = \{(\mathbf{z}_n, y_n)\}_{n=1}^N$ ▷ temporary memory buffer
  Define a linear model for the head $h_{\boldsymbol{\psi}_\tau}()$ and set $\boldsymbol{\psi}_\tau$ to zero
  for `total inner-steps` **do** ▷ loop to estimate head params
    Sample (with replacement) mini-batch of training pairs $\mathcal{B} \sim \mathcal{M}$
    Update the head parameters $\boldsymbol{\psi}_\tau \leftarrow \texttt{step}(\alpha, \mathcal{L}, \mathcal{B}, h_{\boldsymbol{\psi}_\tau})$
  end for
  Update the CaSE parameters $\boldsymbol{\phi} \leftarrow \texttt{step}(\beta, \mathcal{L}, \mathcal{C}, \mathcal{T}, b_{\boldsymbol{\phi}}, h_{\boldsymbol{\psi}_\tau})$ ▷ CaSE in adaptive mode
end while

Algorithm 2 UpperCaSE: test function for the few-shot classification setting.

Require: $\tau_* = (\mathcal{C}_*, \mathbf{x}_*)$ unseen test task with target input $\mathbf{x}_*$ an context $\mathcal{C}_*$.
Require: $b_{\boldsymbol{\phi}}()$ pretrained feature extractor (body) with meta-learned CaSE blocks parameterized by $\boldsymbol{\phi}$.
Require: `step()`: gradient-step function; $\mathcal{L}$ loss; α: step-size hyperparameter for the optimizer.

Forward pass over context set $b_{\boldsymbol{\phi}}(\mathcal{C}_*^x) \rightarrow \mathbf{z}_1, \ldots, \mathbf{z}_N$ ▷ CaSE in adaptive mode
Store context embeddings and associated labels $\mathcal{M}_* = \{(\mathbf{z}_n, y_n)\}_{n=1}^N$ ▷ temporary memory buffer
Define a linear model for the head $h_{\boldsymbol{\psi}_{\tau_*}}()$ and set $\boldsymbol{\psi}_{\tau_*}$ to zero
for `total inner-steps` **do** ▷ loop to estimate head params
  Sample (with replacement) mini-batch of training pairs $\mathcal{B}_* \sim \mathcal{M}_*$
  Update the head parameters $\boldsymbol{\psi}_{\tau_*} \leftarrow \texttt{step}(\alpha, \mathcal{L}, \mathcal{B}_*, h_{\boldsymbol{\psi}_{\tau_*}})$
end for
Return Prediction $\hat{y}_* = h_{\boldsymbol{\psi}_{\tau_*}}(b_{\boldsymbol{\phi}}(\mathbf{x}_*))$ ▷ CaSE in inference mode

C Additional experimental details and results

C.1 Additional details

MACs counting MACs are proportional to the size of the task, size of the images, and number of classes. We can count MACs using synthetic tasks. In our case we used a synthetic task of 100-way, 10-shot with input images of size $224 \times 224 \times 3$ generated via Gaussian noise ($\mu = 0, \sigma = 1$), and labels generated as random integers. We used a mini-batch of size 128 and 500 update steps for UpperCaSE and BiT with an EfficientNetB0 backbone for the first and a ResNet50-S for the second. For MD-Transfer we used the same parameters reported in Dumoulin et al. (2021) with images of size $126 \times 126 \times 3$ and ResNet18 backbone. For the ORBIT experiments we counted MACs by using the code in the original repository [2] and reporting the average MACs over all test tasks for both CLE-VE and CLU-VE using a ResNet18 backbone.

VTAB+MD training We follow the protocol reported in the original papers (Triantafillou et al., 2019; Dumoulin et al., 2021) training UpperCaSE for 10K tasks on the training datasets and evaluating on the MD test set and on the VTAB datasets. At evaluation time we sample 1200 tasks from the MD test set, and report the mean and confidence intervals. On VTAB we report the results of a single run on the test data (data points are given in advance and do not change across seeds). In all experiments we used the MetaDataset-v2 (MDv2) which does not include ImageNet in the test set. We used a pretrained EfficientNetB0 from the official Torchvision repository [3], and a pretrained ResNet50-S

[2] `https://github.com/microsoft/ORBIT-Dataset`
[3] `https://pytorch.org/vision`

from the BiT repository [4]. We normalized the inputs using the values reported in the Torchvision documentation (mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]), for ResNet50-S we use the BiT normalization values (mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5]).

ORBIT training For the ORBIT experiments we trained UpperCaSE on MDv2 using a pretrained ResNet18 taken from the official Torchvision repository. We normalized the inputs using the values reported in the Torchvision documentation (mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]). For the evaluation phase we followed the instructions reported in Massiceti et al. (2021).

C.2 CaSE vs SE

Table 4: Comparing CaSE against standard Squeeze-and-Excitation (SE) on VTAB+MD using different adaptation heads. MD: Mahalanobis distance head (Bronskill et al., 2021). Linear: linear head trained with UpperCaSE. All adaptive blocks use a reduction of 32. Best results in bold.

Model	SE	CaSE	SE	CaSE
Contextual pooling	No	Yes	No	Yes
Adaptation head	MD	MD	Linear	Linear
Image size	84	84	224	224
MetaDataset (all)	67.8	**69.6**	74.6	**76.2**
VTAB (all)	43.6	**45.3**	56.6	**58.2**
VTAB (natural)	47.5	**50.2**	65.3	**68.1**
VTAB (specialized)	63.6	**64.9**	**79.8**	79.6
VTAB (structured)	30.6	**31.8**	38.6	**40.1**

C.3 CaSE vs other adapters

Table 5: Comparing CaSE adaptive blocks (with reduction 64, 32, 16) on VTAB+MD against the FiLM generators used in Bronskill et al. (2021), 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	CaSE64	CaSE32	CaSE16
Adaptive Params (M)	n/a	0.02	**0.01**	**0.01**	**0.01**
Amortiz. Params (M)	n/a	1.7	**0.4**	0.8	1.6
MetaDataset (all)	53.4	68.4	69.8	69.6	**70.4**
VTAB (all)	43.5	44.7	46.2	45.3	**46.4**
VTAB (natural)	45.4	49.5	52.1	50.2	**52.6**
VTAB (specialized)	**69.4**	63.8	66.3	64.9	65.5
VTAB (structured)	29.1	31.7	31.8	31.8	**32.1**

C.4 Ablation studies

In this section we provide additional experimental results focusing on ablation studies of the CaSE adaptive block. The results can be summarized as follows:

- Ablation of the activation function for the output layer for both CaSE and SE. We have tested three activation funcitons: linear, sigmoid, sigmoid with multiplier. The sigmoid with multiplier uses a constant value set to 2 to center the sigmoid at 1 (this enforces the identity function). The empirical results reported in Table 6 show that the sigmoid with multiplier and the linear layer provide the best results.

[4] `https://github.com/google-research/big_transfer`

- Ablation of the number of hidden units in the hidden layers of CaSE. The number of hidden units is controlled by the reduction and min-units parameters in the code and it depends on the number of inputs. See the paper for more details. The results reported in Table 8 show that blocks with more units provide marginal gains or no gains at all. This is probably due to overfitting issues affecting the models with more units.

- Ablation of the number of hidden layers of CaSE. The results reported in Table 7 show that the best performance is obtained with 1 and 2 layers. The performance worsen when there are 3 or more layers which is likely due to overfitting issues affecting the models with more parameters.

- Ablation of the activation function for the hidden layers. Results reported in Table 9 show that CaSE is quite robust against this factor when activations like ReLU and SiLU are used but the performance worsen with Tanh. We have chosen SiLU for the experiments as this is the same activation typically used in Squeeze-and-Excitation layers (e.g. in EfficientNet backbones).

Table 6: Performance on VTAB+MD for various activation functions used in the last layer of SE and CaSE. Sigmoid-2 indicates that the output of a standard Sigmoid is multiplied by 2. Both SE and CaSE use a reduction factor of 32 with min-clipping of 32. All model have been trained using an EfficientNetB0 backbone with a linear head on images of size 224×224. Results for SE with linear activation have not been reported because the training was unstable (loss rapidly diverging at the first iterations). Best results in bold.

Adaptive block Activation (output)	SE Sigmoid	SE Sigmoid-2	CaSE Linear	CaSE Sigmoid	CaSE Sigmoid-2
MetaDataset (all)	74.2	74.6	75.8	74.9	**76.2**
VTAB (all)	56.8	56.6	**58.4**	56.8	58.2
VTAB (natural)	67.0	65.3	**68.3**	67.1	68.1
VTAB (specialized)	**81.1**	79.8	79.5	80.8	79.6
VTAB (structured)	36.9	38.6	**40.3**	37.1	40.1

Table 7: Comparing CaSE adaptive blocks with different number of hidden layers on VTAB+MD. All models have been trained and tested on 224×224 images, using CaSE with reduction 64 and clip factor (min-units) 16, using UpperCaSE and EfficientNetB0 backbone. Best results in bold.

# Hidden layers Amortiz. Params (M)	1 **0.420**	2 0.426	3 0.432	4 0.438
MetaDataset (all)	76.0	**76.1**	75.5	75.2
VTAB (all)	58.2	**58.4**	58.2	58.0
VTAB (natural)	68.3	**69.1**	68.0	67.4
VTAB (specialized)	79.7	80.3	**80.5**	80.3
VTAB (structured)	**40.0**	39.4	39.7	39.7

Table 8: Comparing CaSE adaptive blocks with different number of hidden units on VTAB+MD. The number of hidden units depends on the input size and is defined by the reduction and the clip factor (min-units). All models have been trained and tested on 224×224 images, using UpperCaSE and EfficientNetB0 backbone. Best results in bold.

Reduction factor	64	32	16	8
Clip factor	16	32	48	64
Amortiz. Params (M)	**0.4**	0.8	1.6	3.0
MetaDataset (all)	76.1	**76.2**	75.8	**76.2**
VTAB (all)	58.4	58.2	57.9	**58.5**
VTAB (natural)	**69.1**	68.1	67.9	68.3
VTAB (specialized)	**80.3**	79.6	79.4	79.0
VTAB (structured)	39.4	40.1	39.7	**40.9**

Table 9: Comparing CaSE adaptive blocks with different activation functions for the hidden layers on VTAB+MD. All models are based on a reduction factor of 64 and a clip factor of 16 (0.4M amortization parameters) and they have been trained and tested on 224×224 images, using UpperCaSE and EfficientNetB0 backbone. Best results in bold.

Activation (hidden)	SiLU	ReLU	Tanh
MetaDataset (all)	**76.1**	75.8	74.8
VTAB (all)	**58.4**	57.8	48.2
VTAB (natural)	69.1	**69.8**	67.0
VTAB (specialized)	**80.3**	79.7	80.8
VTAB (structured)	**39.4**	**39.4**	36.4

C.5 Role of CaSE blocks

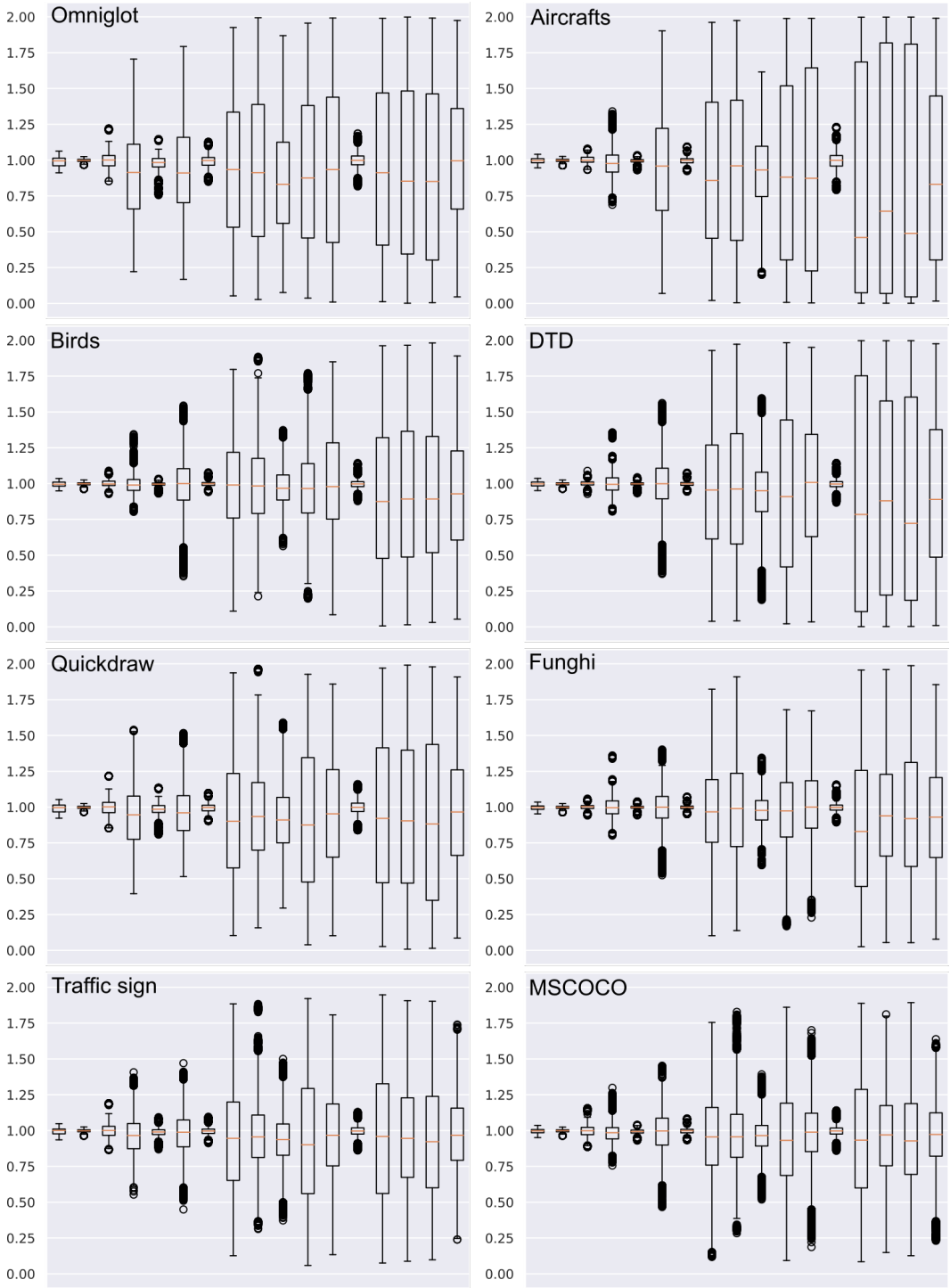

Figure 4: Boxplots for all the MDv2 test datasets (100 tasks per dataset) reporting the CaSE activation (vertical axis) at different stages of an EfficientNetB0 (horizontal axis, with early stages on the left). The box encloses first to third quartile, with the median represented by the orange line. 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.

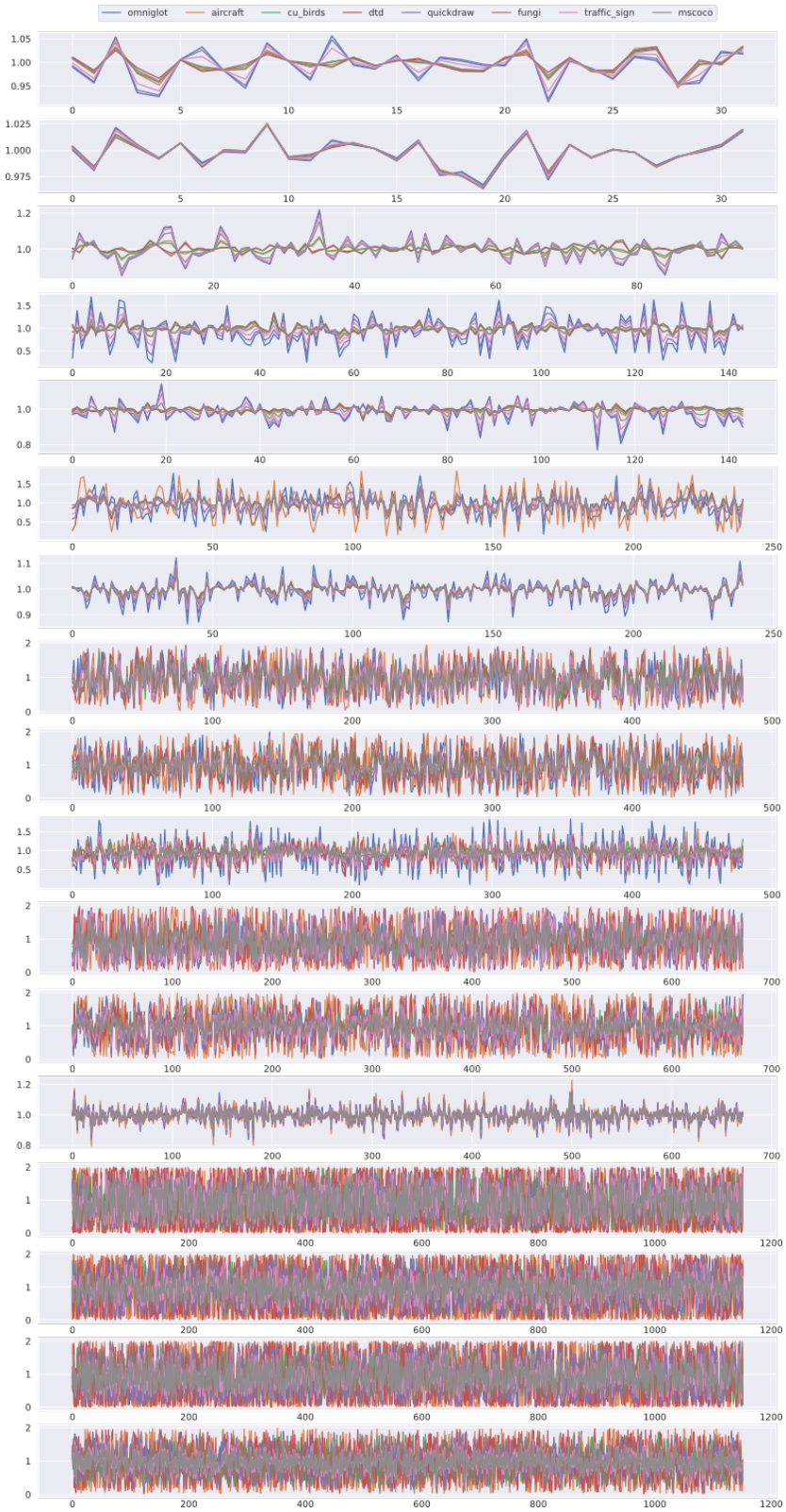

Figure 5: CaSE activation values (vertical axis) for all channels (horizontal axis) at different stages (top plots are early stages) in EfficientNetB0 for the MDv2 test dataset (one task per dataset). Values are similar and closer to one in the first stages but diverge in the latest. The magnitude tends to increase with depth.

C.6 UpperCaSE: results on VTAB+MD

In this section we provide a full breakdown of the results for UpperCaSE vs. other methods on the VTAB+MD benchmark. Results for other methods are taken from Bronskill et al. (2021) and Dumoulin et al. (2021). UpperCaSE uses CaSE with reduction 64 (min-clip 16) for EfficientNetB0 and reduction 32 (min-clip 32) for ResNet50-S. Results for UpperCaSE on MD are the average over 1200 test tasks.

In Table 10 we report the results for UpperCaSE against fine-tuning methods (BiT, MD-Trasnfer, SUR) and in Table 11 the results for UpperCaSE against meta-learning and hybrid methods (ProtoNet, ProtoMAML, Cross Transformer CTX, LITE). Overall UpperCaSE performs well on MD and the natural split of VTAB, this may be due to the fact that transfer learning is more beneficial on those datasets as they are more similar to those used during meta-training. The largest difference in performance between UpperCaSE and fine-tuning methods is on the structured split of VTAB, which includes tasks that require counting and pose estimation. This is likely due to the difference w.r.t. the meta-training set. In this case, fine-tuning the entire network is more effective than body adaptation as the knowledge gap is wider and it requires more adjustments to the parameters.

Table 10: Comparing UpperCaSE against fine-tuning methods. Best result in bold.

Model	BiT	MD-Transfer	SUR	UpperCaSE	UpperCaSE
Image Size	224	126	224	224	224
Network	RN50-S	RN18	RN50×7	ENB0	RN50-S
Params (M)	23.5	11.2	164.5	4.0	23.5
Omniglot	68.0±4.5	82.0±1.3	**92.8±0.5**	90.7±0.4	89.1±0.5
Aircraft	77.4±3.5	76.8±1.2	84.4±0.6	**89.4±0.4**	87.5±0.4
Birds	**90.8±1.5**	61.2±1.3	75.8±1.0	**90.4±0.4**	**89.6±0.4**
DTD	**85.0±2.5**	66.0±1.1	74.3±0.7	**83.4±0.4**	**84.8±0.5**
QuickDraw	66.6±3.7	61.3±1.1	70.3±0.7	**76.8±0.5**	73.7±0.6
Fungi	59.4±4.2	35.5±1.1	**81.7±0.6**	59.3±0.8	56.8±0.8
Traffic Sign	73.5±4.7	**84.7±0.9**	50.0±1.1	68.5±0.8	70.6±0.8
MSCOCO	**65.7±2.7**	39.6±1.0	49.4±1.1	50.8±0.7	46.7±0.8
Caltech101	87.2	70.6	82.3	**88.3**	86.2
CIFAR100	**54.4**	31.3	33.7	52.7	47.0
Flowers102	83.3	66.1	55.7	**85.3**	83.0
Pets	87.9	49.1	76.3	89.9	89.3
Sun397	33.3	13.9	27.5	**35.8**	32.5
SVHN	**70.4**	83.2	18.7	62.7	59.8
EuroSAT	**94.4**	88.7	78.9	92.2	91.6
Resics45	**76.1**	63.7	62.4	75.5	74.4
Patch Camelyon	**83.1**	81.5	75.6	79.3	80.9
Retinopathy	70.2	57.6	27.9	**74.3**	73.7
CLEVR-count	**74.0**	40.3	30.0	40.3	42.0
CLEVR-dist	51.5	**52.9**	37.1	38.9	37.3
dSprites-loc	82.7	**85.9**	30.0	45.3	38.1
dSprites-ori	**55.1**	46.4	19.8	42.5	41.4
SmallNORB-azi	17.8	**36.5**	12.9	15.7	15.1
SmallNORB-elev	**32.1**	31.2	18.1	22.7	21.0
DMLab	**43.2**	37.9	33.3	38.7	36.1
KITTI-dist	**79.9**	58.7	52.3	71.0	69.6
MetaDataset (all)	73.3	63.4	71.0	**76.1**	74.9
VTAB (all)	**65.4**	55.6	42.9	58.4	56.6
VTAB (natural)	**69.4**	52.4	49.0	69.1	66.3
VTAB (specialized)	**81.0**	72.9	61.2	80.3	80.1
VTAB (structured)	**54.5**	49.4	29.2	39.4	37.6

Table 11: Comparing UpperCaSE against meta-learning and hybrid methods. Best result in bold.

Model	ProtoNet	ProtoMAML	CTX	LITE	UpperCaSE	UpperCaSE
Image Size	224	126	224	224	224	224
Network	ENB0	RN18	RN34	ENB0	ENB0	RN50-S
Params (M)	4.0	11.2	21.3	4.0	4.0	23.5
Omniglot	88.3±0.8	**90.2±0.7**	84.6±0.9	86.5±0.8	**90.7±0.4**	89.1±0.5
Aircraft	85.0±0.7	82.1±0.6	85.3±0.8	83.6±0.7	**89.4±0.4**	87.5±0.4
Birds	**90.2±0.5**	73.4±0.9	72.9±1.1	88.6±0.7	**90.4±0.4**	89.6±0.4
DTD	81.4±0.6	66.3±0.8	77.3±0.7	**84.1±0.7**	83.4±0.4	**84.8±0.5**
QuickDraw	**76.0±0.7**	66.4±1.0	73.3±0.8	**75.7±0.8**	59.3±0.8	56.8±0.8
Fungi	57.4±1.1	46.3±1.1	48.0±1.2	56.9±1.2	**59.3±0.8**	56.8±0.8
Traffic Sign	53.5±1.1	50.3±1.1	**80.1±1.0**	65.8±1.1	68.5±0.8	70.6±0.8
MSCOCO	**49.8±1.1**	39.0±1.0	**51.4±1.1**	**50.0±1.0**	**50.8±0.7**	46.7±0.8
Caltech101	87.4	73.1	84.2	87.7	**88.3**	86.2
CIFAR100	43.1	29.7	37.5	48.8	**52.7**	47.0
Flowers102	78.2	60.2	81.8	83.5	**85.3**	83.0
Pets	88.6	56.6	70.9	89.3	**89.9**	89.3
Sun397	32.9	8.1	24.8	30.9	**35.8**	32.5
SVHN	35.2	46.8	**67.2**	51.0	62.7	59.8
EuroSAT	83.3	80.1	86.4	89.3	**92.2**	91.6
Resics45	68.8	53.5	67.7	**76.4**	75.5	74.4
Patch Camelyon	73.3	75.9	79.8	**81.4**	79.3	80.9
Retinopathy	31.3	73.2	35.5	40.3	**74.3**	73.7
CLEVR-count	27.2	32.7	27.9	31.4	40.3	**42.0**
CLEVR-dist	28.5	35.4	29.6	32.8	**38.9**	37.3
dSprites-loc	13.4	42.0	23.2	12.3	**45.3**	38.1
dSprites-ori	19.6	23.0	**46.9**	31.1	42.5	41.4
SmallNORB-azi	9.4	13.4	**37.0**	14.5	15.7	15.1
SmallNORB-elev	17.0	18.8	21.6	21.0	**22.7**	21.0
DMLab	35.8	32.5	31.9	**39.4**	38.7	36.1
KITTI-dist	56.5	54.4	54.3	63.9	**71.0**	69.6
MetaDataset (all)	72.7	64.2	71.6	73.9	**76.1**	74.9
VTAB (all)	46.1	45.0	50.5	51.4	**58.4**	56.6
VTAB (natural)	60.9	45.7	61.1	65.2	**69.1**	66.3
VTAB (specialized)	64.2	70.7	67.3	71.9	**80.3**	80.1
VTAB (structured)	25.9	31.5	34.1	30.8	**39.4**	37.6