AdaBet: Gradient-free Layer Selection for Efficient Training of Deep Neural Networks

Irene Tenison*
Massachusetts Institute of Technology
Cambridge, MA
itenison@mit.edu

Soumyajit Chatterjee
Nokia Bell Labs
Cambridge, UK
soumyajit.chatterjee@nokia-bell-labs.com

Fahim Kawsar
Nokia Bell Labs, Cambridge, UK
University of Glasgow, UK
fahim.kawsar@glasgow.ac.uk

Mohammad Malekzadeh
Nokia Bell Labs
Cambridge, UK
mohammad.malekzadeh@nokia-bell-labs.com

Abstract

To utilize pre-trained neural networks on edge and mobile devices, we often require efficient adaptation to user-specific runtime data distributions while operating under limited compute and memory resources. On-device retraining with a target dataset can facilitate such adaptations; however, it remains impractical due to the increasing depth of modern neural nets, as well as the computational overhead associated with gradient-based optimization across all layers. Current approaches reduce training cost by selecting a subset of layers for retraining, however, they rely on labeled data, at least one full-model backpropagation, or server-side meta-training; limiting their suitability for constrained devices. We introduce *AdaBet*, a gradient-free layer selection approach to rank important layers by analyzing topological features of their activation spaces through Betti Numbers and using forward passes alone. *AdaBet* allows selecting layers with high learning capacity, which are important for retraining and adaptation, without requiring labels or gradients. Evaluating *AdaBet* on sixteen pairs of benchmark models and datasets, shows *AdaBet* achieves an average gain of 5% more classification accuracy over gradient-based baselines while reducing average peak memory consumption by 40%. We open-source our code at https://github.com/Nokia-Bell-Labs/efficient_layer_selection

1 Introduction

Consider waking up to our mobile or wearable device that has fine-tuned, overnight and entirely on-device, a machine learning (ML) model to our unique skin tone and texture, personalizing melanoma screening without transmitting a single image to the cloud [4, 44]. By locally retraining a pre-trained ML model, we create specialized apps that preserve user privacy while providing personalized services for individual needs. Such an approach can extend from real-time mood prediction based on subtle facial micro-expressions to adaptive glucose-monitoring algorithms on insulin pumps, each model continuously refining itself at the user's side. Since on-device training still faces key challenges in resource consumption and computational efficiency, maintaining the right balance between performance and resource use is essential to enabling the next generation of efficient, privacy-preserving, and user-centric ML technologies [21].

In particular, deep neural networks (DNNs) deployed on edge or personal devices are typically pre-trained on large, generic datasets. For many apps, it is often necessary to re-train or fine-tune the DNN using a new, small dataset to enhance performance on a desired task. Such a retraining, when done locally on the device, enables personalization and preserves user privacy by avoiding interactions with a server. A plethora of applications benefit from such local training: perception systems in autonomous vehicles [42], wildlife monitoring via camera traps [32], personalized speech recognition [34], and health monitoring via wearables [50].

The problem remains the intensive computational requirements, substantial memory usage, and high power consumption associated with gradient-based training; making it impractical for many resource-constrained scenarios. Unlike cloud servers with dedicated GPUs and abundant memory, edge devices operate under strict limitations on memory, compute throughput, and battery life [1, 15, 16, 49]. To train DNNs, we require both forward and backward passes through all layers, which significantly increases memory usage and compute time compared to inference-only tasks. The backward pass typically consumes 3× more memory than the forward pass due to gradient and activation storage [6], often making full backpropagation infeasible on some devices. Therefore, enabling practical on-device learning demands novel approaches that prioritize memory efficiency and computational reduction while maintaining the model's accuracy.

An approach to these challenges is to adopt a shallow neural network architecture. This offers a straightforward

*Work is mainly done during the author's internship at Nokia Bell Labs.

and memory-efficient solution, but suffers from limited capacity, which can hinder performance on some tasks. Another approach is retraining a DNN using techniques such as gradient checkpointing [7, 10], layer-wise training [5], split-learning [30], or Transfer Learning-based methods [17]. While memory-efficient, these solutions introduce trade-offs: gradient checkpointing reduces memory usage at the cost of compute overhead and increased running time [40], and other methods like layer-wise training, split-learning, and Transfer Learning often suffer from suboptimal performance when there are significant changes in data distribution or downstream tasks [22, 37]. Moreover, some require communication overhead if parts of the model need to be offloaded to external compute platforms [30]. A third approach focuses on hardware-specific design strategies, such as quantization-aware training [2] and hardware-aware neural architecture search [23], which are primarily optimized for inference tasks rather than for flexible or on-device training.

A recent approach involves using sparse updates, where only the most important layers are updated to save memory, an idea presented by the Tiny Training Engine [24]. However, it performs layer selection at the server side and does not support dynamic, on-device adaptation when the data distribution shifts. TinyTrain [20] and ElasticTrainer [15] extend sparse updates to support on-device, task-aware retraining. TinyTrain achieves this by computing Fisher Information to select important parameters, but requires at least one full round of backpropagation through the entire model and relies on server-side meta-training. ElasticTrainer uses a dynamic programming approach to select important parameters. While offering more adaptive training, ElasticTrainer requires labeled data and involves multiple selection steps.

Our Contribution. We propose *AdaBet*, a memory- and compute-efficient framework for retraining of DNNs on edge and mobile devices. The main novelty of *AdaBet* is *gradient-free* layer selection, eliminating the need for either full-model backpropagation or server-side meta training. To achieve this, *AdaBet* selects trainable layers based on their learning capacity, quantified via topological features derived from the first Betti Number of each layer's activation outputs [8]. Betti Numbers are achieved during a forward pass and are normalized by the size of each layer to align with memory efficiency objectives. As this process is independent of label availability, *AdaBet* is suitable for even unsupervised scenarios. We benchmark *AdaBet* across several benchmarks using pre-trained DNNs: ResNet, VGG16, MobileNet, and ViT. Results show that *AdaBet* achieves comparable or better accuracy than gradient-based methods while reducing average peak memory usage by 40% (shown in Figure 1) and obtaining an average accuracy gain of +5% (detailed in §6). In summary, the key contributions of *AdaBet* are:

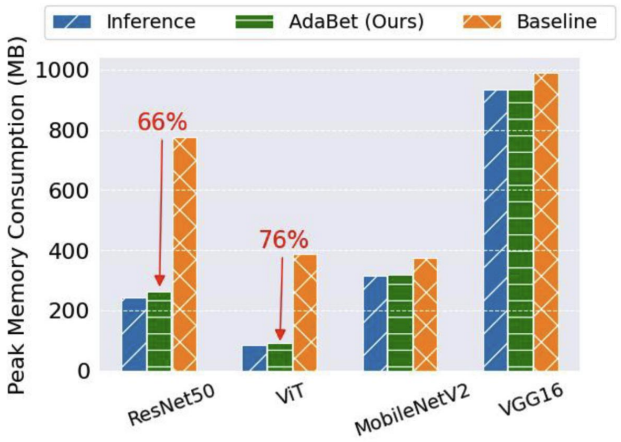


Figure 1: *Peak memory efficiency. AdaBet* reduces peak memory consumption of retraining by up to 76% (on average by 40% for different pre-trained DNNs retrained on Oxford-IIIT Pets), compared to peak memory of inference-only and full-layer training as the baseline.

(1) A memory- and compute-efficient on-device training that enables gradient-free layer selection without requiring labeled data or server-side meta training.
(2) Leveraging topological features based on well-studied Betti Numbers of activations, for selecting a subset of layers based on their learning capacity and resource constraints.
(3) Providing better balance between resource efficiency and accuracy, benchmarked on popular pre-trained DNNs across a range of image classification tasks.

2 Related Work: On-device Layer Selection

The growing need for personalization, data privacy, and post-deployment adaptability highlights the importance of on-device training for DNNs, which requires computational resources to perform backpropagation locally, including the calculation and storage of intermediate gradients for updating DNNs. To minimize resource consumption, existing approaches include techniques like gradient checkpointing [7, 10], layer-wise training [5], split-learning [30], and transfer learning [17]. Despite their resource efficiency, these approaches often suffer from reduced accuracy when faced with changes in data or downstream tasks.

Another approach is to co-optimize resource usage alongside the accuracy of downstream tasks. However, such approaches [6, 23, 24, 36, 47] either (1) rely on server-side architecture search to select layers before deployment on the device, or (2) require at least one full backpropagation through the entire model (computing all gradients), which creates a significant bottleneck for resource-constrained devices.

Table 1: Compared to related work, our *AdaBet* is the only gradient-free (i.e., label-free) selection method that requires no server-side meta-training.

	Unlabeled data?	Has Layer selection?	One-step selection	Server independent?	Selection approach
Transfer Learning [17]	×	×	×	✓	N/A
Quant-Aware Training [2]	×	×	×	×	N/A
TinyTrainingEngine [24]	×	✓	×	×	Evolutionary
TinyTrain [20]	×	✓	×	×	Fisher Info
PruneTrain [28]	×	✓	×	✓	Lasso Penalty
ElasticTrainer [15]	×	✓	×	✓	Dynamic Prog.
***AdaBet* (Our)**	✓	✓	✓	✓	Betti Numbers

PruneTrain [28] reduces training parameters through channel-level sparsity and structured pruning. It uses group lasso regularization to shrink the model while maintaining dense computation, enabling dynamic architecture adaptation during training. TinyTrain [20] performs an adaptive parameter selection method based on Fisher Information (FI) of activations to assess a layer's importance. This allows TinyTrain to be both compute- and memory-efficient on the device. However, to perform the online selection of layers, TinyTrain performs one round of backpropagation through the entire model. Since relying on a single round of backpropagation may result in suboptimal parameter selection [13, 18], TinyTrain performs multiple rounds of backpropagation during a server-side meta-training stage. Meta-training with data from similar distributions might not be realistic, as the distribution of the target private data could change significantly, which negatively impacts the effectiveness of the FI-based layer selection [41].

ElasticTrainer [15] dynamically selects parameters using a dynamic programming approach that minimizes resource consumption of on-device training. Unlike TinyTrain [20], ElasticTrainer eliminates the need for a meta-training phase by directly selecting important layers on the device. Such a selection process with dynamic programming involves multiple rounds of parameter tuning, each requiring backpropagation through the entire model to ultimately identify the optimal subset of layers. This makes the overall process of selection and training memory-intensive, at times even comparable to that of full training. A comparison between existing on-device layer selection approaches for DNNs and our proposed system, *AdaBet*, is summarized in Table 1.

3 Backgrounds and Motivations

3.1 Betti Numbers

In mathematics, **topology** examines the invariant properties of spaces under continuous deformations, focusing on connectedness and compactness. In ML, topology allows us to study the robustness of learned representations to transformations such as scaling, rotation, and noise addition [8]. In algebraic topology, **homology** analyzes the structure of *holes* in a space across various dimensions to understand the shape and connectivity of high-dimensional spaces. Instead of relying on visualizations, homology captures topological features as algebraic invariants, enabling broader and more efficient analysis of complex spaces.

Formally, the n^{th} homology group of a topological space $\mathcal{X}$, denoted as $H_n(\mathcal{X})$, quantifies *n-dimensional holes* in $\mathcal{X}$; such that 0-dimensional holes indicate connected components, 1-dimensional holes indicate loops, and higher-dimensional holes indicate structures such as voids and cavities.

To calculate $H_n(\mathcal{X})$, we turn the geometric problem (e.g., holes in a space) into an algebraic one (i.e., computing groups) by building a chain complex $C(\mathcal{X})$: a sequence of abelian groups linked by boundary operators δ_n such that

$$\cdots \xrightarrow{\delta_{n+1}} C_n \xrightarrow{\delta_n} C_{n-1} \xrightarrow{\delta_{n-1}} \cdots \xrightarrow{\delta_2} C_1 \xrightarrow{\delta_1} C_0 \xrightarrow{\delta_0} 0.$$

Here, each δ_n satisfies the key property that consecutive boundaries vanish, i.e., $\delta_n \circ \delta_{n+1} = 0$, meaning that the boundary of a boundary is always zero to make sure that every boundary encloses a well-defined cycle. Having $C(\mathcal{X})$, the n^{th} homology group is:

$$H_n(\mathcal{X}) = Ker(\delta_n)/Im(\delta_{n+1}),$$

where $\mathrm{Ker}(\delta_n)$ indicates the number of cycles (i.e., elements that map to zero under δ_n), and $\mathrm{Im}(\delta_{n+1})$ indicates the boundaries (i.e., elements that originate from a higher-dimensional space). *Thus, $H_n(\mathcal{X})$ captures nontrivial cycles that are not boundaries, effectively identifying the holes in $\mathcal{X}$.* Thanks to the computational technique called *persistent homology*, $H_n(\mathcal{X})$ is calculated to indicate how topological features, such as connected components, loops, and voids, emerge and persist across multiple scales in ML and data analysis.

An informative measure that quantifies the number of *independent n-dimensional holes* in a topological space $\mathcal{X}$ is the n^{th} **Betti Number** defined as the rank of $H_n(\mathcal{X})$:

$$b_n = \mathrm{rank}(H_n(\mathcal{X})). \tag{1}$$

For example, if our topological space $\mathcal{X}$ is a circle, we have $b_0 = 1$, because the space is connected, and $b_1 = 1$, because the space contains a single one-dimensional hole (i.e., a loop). In this case, all higher Betti Numbers vanish: $b_n = 0$ for all $n >= 2$, as a circle has no higher-dimensional holes. For an n-dimensional sphere, $b_0 = 1$ indicates a single connected component, and $b_n = 1$, indicates the n-dimensional void, while all intermediate Betti Numbers are zero (see Table 2). Such a multi-scale perspective, measured through Betti Numbers, helps in identifying topological structures while filtering out noise-induced artifacts. When $\mathcal{X}$ is a torus, persistent homology effectively identifies its two distinct one-dimensional holes and its two-dimensional void; regardless of transformations such as scaling, rotation, or translation.

Table 2: Illustration of Betti Numbers for some common geometrical shapes across different dimensions.

Betti Number	.			
b_0	**1**	**1**	**1**	**1**
b_1	0	**1**	0	**2**
b_2	0	0	**1**	**1**
b_3	0	0	0	0
:	:	:	:	:
b_n	0	0	0	0

Building on such topological insights, *AdaBet* leverages *the first Betti Numbers of each layer's activation outputs* in a DNN to quantify its complexity, thereby guiding the selection of layers for targeted retraining (as detailed in §4).

3.2 Betti Number vs. Fisher Information

Given a batch of N samples, and based on the gradients g of the loss function with respect to the activations a of the model, Fisher Information (FI) is defined as:

$$\Delta = \frac{1}{2N} \sum_{n=1}^{N} (a_n g_n)^2 \tag{2}$$

TinyTrain [20] uses FI to assess the importance of layers. We outline the drawbacks of FI approach to highlight how Betti Numbers can offer a more efficient and yet more effective alternative for layer selection.

Drawbacks of FI. Computing FI requires at least one backpropagation and access to labeled data, which limits its practicality in constrained devices, unsupervised settings, or label-scarce scenarios. As shown in Figure 2, FI-based layer selection is unstable: the chosen subset of layers changes with the number of backpropagations (Figure 2(a)) and across different batches of data due to random seeds (Figure 2(b)). This instability arises from its reliance on early-stage gradients, which are not yet aligned with the target data distribution. As a result, FI-based selection can bias toward the pre-training or meta-training dataset, leading to inconsistent or suboptimal retraining subsets. Moreover, there is no principled way to know when FI-based selection has converged, since peak accuracy may occur at intermediate stages rather than after extended updates. In contrast, our Betti Number–based method yields consistent layer rankings across batches without backpropagation, achieving stable accuracy with negligible variance (Figure 2(c))

Advantages of Betti Numbers Betti Numbers are calculated on the activation outputs of each layer, without backpropagation or requiring labeled data. Betti Numbers of each layer remain consistent across random sampling of a batch, allowing activations to be aggregated over multiple forward passes, even with small batch sizes. This provides a stable yet lightweight alternative for on-device layer selection. In Figure 2(c), we observe that Betti Numbers computed for each layer of a pre-trained DNN tend to peak in the early and late layers. To further investigate this observation, using the DBSCAN clustering algorithm, we estimated the number of clusters for each layer's activation outputs in Figure 3. The similarities between the two figures suggest that Betti Numbers effectively capture layers with higher representational capacity. This observation aligns with seminal works [8, 43] linking cluster separability in intermediate representations to model expressivity and generalization; as we discussed them further in §4.

4 Gradient-free Layer Selection

4.1 Layer Selection

Let us consider on-device retraining of a deep neural net, e.g., for model adaptation or personalization. Let $\mathcal{M}$ be a DNN *pre-trained* on a global dataset using any type of objective function. For example, a pre-trained model published on HuggingFace, comprising multiple *layers*, each containing a varying number of parameters. We assume $\mathcal{M}$ is deployed on a constrained device hosting a local training dataset $\mathbb{D}$, without imposing any restrictions on the characteristics of $\mathbb{D}$. Specifically, $\mathbb{D}$ can be unlabeled or from a different domain than the pre-training dataset. Our objective is to facilitate on-device re-training of $\mathcal{M}$ with $\mathbb{D}$, taking into account resource efficiency and target task accuracy. To achieve our objective, we should efficiently *select* the most important layers from $\mathcal{M}$, and only *selected layers* are later retrained on $\mathbb{D}$ using the target task objective function.

4.2 *AdaBet* Overview

We introduce a novel strategy for selecting *most important layers* to retrain, while satisfying three key properties: (a) *AdaBet* is gradient-free, reducing on-device memory and computational requirements; (b) *AdaBet* is label-free, enhancing its generalizability across diverse ML tasks; and (c) *AdaBet* is server-free, making it well-suited for privacy-preserving and fully on-device learning scenarios.

As illustrated in Figure 4, *AdaBet* comprises three steps: (1) the estimation of each layer's learning capacity via Betti Numbers computed on a subset of $\mathbb{D}$, (2) the selection of important layers based on learning capacity and the size of activations, considering the available computation budget on the device, and (3) the retraining of selected layers in $\mathcal{M}$ using the local dataset $\mathbb{D}$.

In particular, since $\mathcal{M}$ is pre-trained, we first evaluate all layers' learning capacity by forward propagating samples drawn from $\mathbb{D}$ through the DNN, and we capture the corresponding activations of each layer (see Figure 5). A topological analysis of these activations, via Betti Number calculation, is performed on each layer to assess the learning

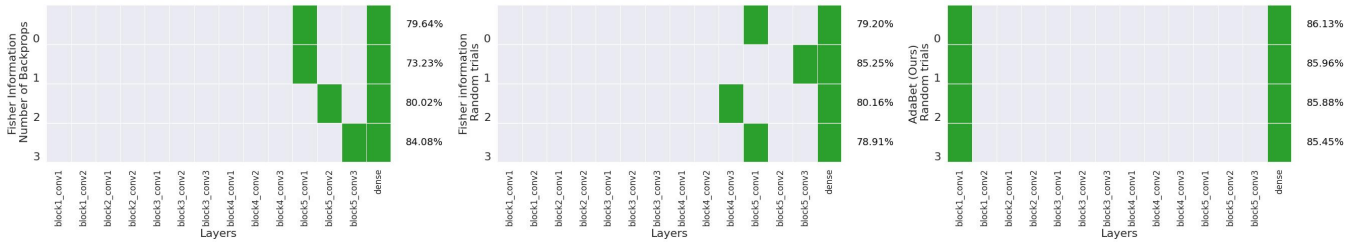


Figure 2: Fisher Information vs. Betti Numbers. Pre-trained VGG16 adapted to Stanford Dogs dataset. Green/dark rectangles show selected layers. (a) FI ranking depends on the number of backpropagations over the entire model. (b) FI ranking depends on the batch of data selected (due to changes in the random seed), with considerable changes in accuracy in all the cases. (c) Our gradient-free Betti Number ranking of *AdaBet*, is more consistent across different batches of data used to select the layers with negligible changes to the overall accuracy.

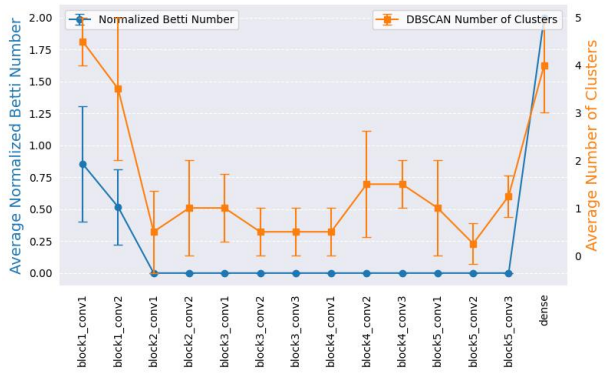


Figure 3: Layer-wise Betti Numbers computed on activations of a pre-trained VGG-16 model for Oxford-IIIT Pets, along with the number of clusters obtained via DBSCAN on UMAP reduced layer embeddings; both averaged across 5 independent runs. DBSCAN of the embeddings shows a similar ranking pattern, but Betti Numbers offer a more granular ranking of the layers.

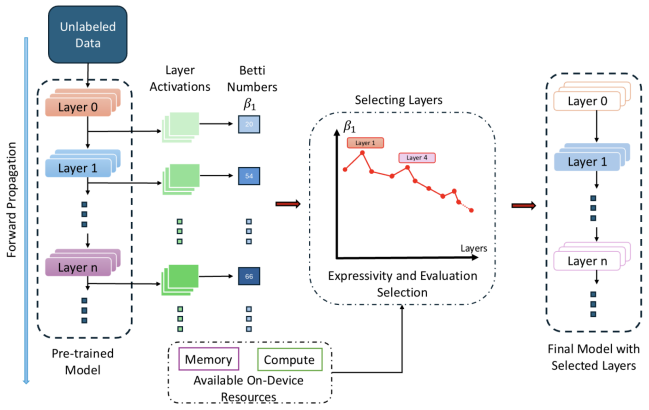


Figure 4: An overview of *AdaBet*. First, we estimate each layer's learning capacity via Betti Numbers. Second, we select the most important layers based on available compute budget. Third, the re-training of selected layers in $\mathcal{M}$ using the local dataset $\mathbb{D}$.

capacity of each layer. The important layers are then selected based on their ranking: layers with higher Betti Numbers are favored, while those with a larger number of parameters are penalized in the selection process. Finally, the re-training of $\mathcal{M}$ only updated the selected layers using $\mathbb{D}$.

4.2.1 Betti Numbers & Learning Capacity. Unlike existing layer selection methods [15, 20] that require at least one round of backpropagation through all layers, our solution only looks into the topological features obtained from layers' activations in the forward pass. Specifically, we choose the *first Betti Number*, b_1, of the activations from all layers in any pre-trained model on a batch of local $\mathbb{D}$ as part of our *selection metric*. In our observations, the zeroth Betti Number, b_0, has consistently matched the batch size used during the forward pass. This is because b_0 represents the number of connected components in a topological space, and typically, each sample in a batch forms its own connected component in the output activation space. On the other hand, the first Betti Number has been shown to quantify the learning capacity of DNN layers [8, 43].

Specifically, the first Betti Number, $b1$, measures the number of one-dimensional holes (loops or tunnels) in the output activation space, and is directly linked to the Rademacher Average capacity of a layer [9]. Since higher capacity is generally associated with an increased risk of overfitting, such layers may be more suitable candidates for fine-tuning. Moreover, Betti Numbers are more robust to small perturbations and noise applied to the topological space [12]. Since the calculation of Betti Numbers can be done using activations from

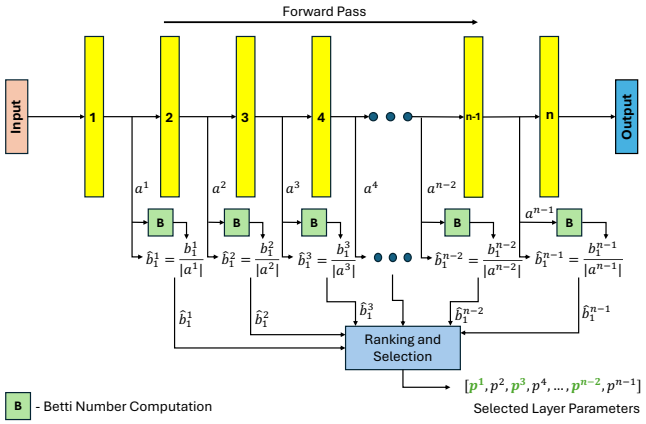


Figure 5: Detailed pipeline of *AdaBet* during its selection phase. Here a^i denotes the activations, b_1^i denotes the Betti Number b_1 from the activations, $\hat{b}_1^i$ is the normalized Betti Number and p^i is the parameters (weights and biases) of the i^{th} layer, respectively.

forward propagation of any randomly selected batch from $\mathbb{D}$, it does not require resource-intensive gradient computations for backpropagation, and also does not need labels.

4.2.2 Batch size flexibility. Small batch sizes (e.g., 4 samples at a time) can make Betti Numbers unreliable, because each small batch gives only a few activation samples. Gradient-based methods like ElasticTrainer [15] and TinyTrain [20] also face a similar issue: small batch sizes produce noisier or less reliable gradient estimates, which can hurt the accuracy of their selection criteria. However, since *AdaBet* only relies on the forward pass, we can get around this more easily, as we can run several forward passes, each with a different mini-batch, and collect all their activations before computing Betti Numbers. For example, with a batch size 8, we might do 5 passes to gather 40 activation vectors. Since Betti Numbers only use forward activations (no gradients or backpropagation), such"activation pooling" is inexpensive and manageable with available memory budget. After we choose which layers to train, we switch back to the original small batch size for the actual retraining. This way, we get a more stable layer choice without needing extra memory or compute.

4.2.3 Adjustment with Layer Types. In most pre-trained architectures, the base model with pre-trained weights can be broken into additional layers or blocks. These models typically include nested modules that combine both trainable and non-trainable components. *AdaBet* iterates through the model hierarchy, retrieving the activations only from sublayers that contain trainable parameters. Non-trainable layers such as dropout, pooling, or reshaping layers are excluded from consideration. Furthermore, in transformer architectures like ViT, all query, key, value, and output linear sublayers within a multi-head self-attention block are treated as a single layer to reduce the overall complexity and simplify the selection process for *AdaBet*.

4.2.4 Balancing Importance and Computation Cost. Although this is not a universal topological law, it is observed empirically that the first Betti Number, b_1 (which measures the number of independent one-dimensional holes in the activation space), tends to be slightly higher in layers with a larger number of activations [48]. Additionally, in convolutional layers, the memory needed to store activation values is often significantly greater than that required for storing the layer's parameters. For example, by a factor of about 14× in typical MobileNets [6, 24]. To balance the importance selection with the associated computational cost, we normalize the computed first Betti Number, b_1^i, for each layer i, by the total number of elements in that layer's activations $|a^i|$:

$$\hat{b}_1^i = \frac{b_1^i}{|a^i|} \tag{3}$$

This normalized first Betti Number $\hat{b}_1^i$ provides a more effective and cost-aware criterion for selection. We note that TinyTrain [20] also normalizes each layer's Fisher score with the number of parameters and number of multiply-accumulate (MAC) operations of that layer, respectively. The summary of the selection process is also presented in Figure 5.

We define a design parameter $\rho \in [0, 1]$ representing the proportion of layers in $\mathcal{M}$ selected for training on the target dataset $\mathbb{D}$. For instance, $\rho = 0.3$ means that 30% of the layers, ranked according to their $\hat{b}_1$ values, are chosen for retraining. A value of $\rho = 0$ corresponds to inference-only use, while $\rho = 1$ indicates full model retraining. This flexible approach allows *AdaBet* to take into account the capabilities of the target device, balancing performance gains with computational and memory constraints.

4.2.5 Selective Training on Local Dataset. *AdaBet* performs on-device fine-tuning of the selected subset of layers using the local dataset $\mathbb{D}$. Notice that *AdaBet* can adopt a smaller batch size during such retraining, even if a larger batch size was used earlier during the layer selection process via activation aggregation (see §4.2.2). Moreover, the selective training approach is agnostic to the learning paradigm and can be applied in supervised, semi-supervised, or self-supervised settings. While we make no assumptions about the availability of labeled data during retraining, in this paper, we focus on showcasing the benefits of *AdaBet* in standard supervised retraining scenarios used by our baselines. We leave the investigation of other paradigms for future work.

5 Evaluation Setup

Baselines. We compare *AdaBet* against five baselines:
(1) Full Training: all layers of the DNN parameters are updated using the local dataset.
(2) Vanilla Transfer Learning: only the final layer is updated, following standard transfer learning approaches [17, 35].
(3) ElasticTrainer: selected layers are updated during retraining, where selection is done using dynamic programming across multiple rounds of backpropagation [15].
(4) PruneTrain: structured pruning is applied to the DNN during on-device training to reduce training cost [28].
(5) Fisher Information: layers are selected based on their Fisher Information scores, which are computed from activations and gradients obtained in the first backpropagation round [20]. We select $\rho = 0.1$ (i.e., top 10%) of layers based on FI scores for re-training; the same default for *AdaBet*.

Note that baseline (5) differs from TinyTrain [20], which relies on meta-training. To ensure a fair comparison, we do not compare with server-based meta-training approaches [6, 20] or compute-intensive selection methods that need neural architecture search [24]. We believe these approaches, while valuable, are not practical for constrained devices requiring efficient and timely model retraining. Furthermore, the code for TinyTrain [20] is not available, making it challenging to reproduce their methodology.

DNN Architectures. We evaluate *AdaBet* across four widely-used DNNs: (i) Vision Transformers (ViT-B16) [11], which employs an attention mechanism adopted to vision tasks and consists of 12 Transformer Encoder blocks, (ii) ResNet50 [14], a deep model comprising 50 convolutional blocks with residual connections, (iii) VGG16 [39], which includes 13 convolutional blocks with small 3×3 convolution filters, and (iv) MobileNetV2 [38], a lightweight architecture with inverted residual blocks designed for mobile deployment. All DNNs are initialized with pre-trained weights from the ImageNet dataset. These four architectures were chosen to represent a broad range of design principles. For example, while ViT requires less memory than MobileNetV2, it suffers from slower on-device inference, which makes it more challenging for on-device ML [29].

Datasets. As the local dataset $\mathbb{D}$, we use three datasets: (i) Stanford Dogs [19] with 12,000 training and 8,580 testing images of 120 dog breeds, (ii) Oxford-IIIT Pets [33] with 3,680 training and 3,669 testing images from 37 categories of pets, and (iii) CUB [46] with 5,994 training and 5,794 testing images from 200 bird species. (iv) Flowers102 [31] with 1020 training and 6149 test images from 102 flower categories. To maintain consistency with our primary baseline ElasticTrainer [15], prior to selection and training, we resize all the input images to 224×224 and apply default pre-processing steps such as centering and random flipping.

Hyperparameters. For MobileNetV2 model on all datasets we use a learning rate of $1 \times e^{-4}$, for all models on CUB dataset we use a learning rate of $1 \times e^{-2}$, for all other models and datasets we use a learning rate of $1 \times e^{-3}$ and with the SGDW optimizer [27], a cosine decay momentum of 0.9, and a weight decay of 5×10^{-4}. For all *AdaBet* experiments, we set $\rho = 0.1$, unless otherwise specified. The batch size is set to 8 for all experiments, and the impact of batch size is separately analyzed in Section 6.3.5. The codebase is available here: https://github.com/Nokia-Bell-Labs/efficient_layer_selection.

Libraries and Hardware. We use TensorFlow 2.15 and TensorFlow Addons 0.23, building on the codebase of ElasticTrainer [15]. For topological computations, we employ Ripser 0.6.10 [3, 45], which provides fast and memory-efficient calculation of Betti Numbers. Memory computation and profiling are conducted using benchmarking code adapted from [6, 24]. Our training and evaluation experiments (§6.2–6.3), unless otherwise specified, are conducted on an NVIDIA Tesla V100 GPU with 16 GB of VRAM.

6 Experimental Results

In evaluating *AdaBet* against our baselines, we focus on three key aspects: (i) computational efficiency during retraining, (ii) classification performance on the target task, and (iii) the impact of various design choices, including batch size, normalization strategy, and layer proportion ratio to assess its real-world applicability at runtime on-device.

6.1 Classification Accuracy

Table 3 presents classification accuracy of *AdaBet* across four target datasets and four DNN architectures, comparing our *AdaBet* with the SOTA baselines. For each configuration, classification accuracy is averaged over three independent runs, totaling 288 runs. Note that all DNNs are pre-trained on the ImageNet dataset, and in this work, the layers or parameters selected by each method are further retrained on the target dataset.

We observe that *AdaBet* outperforms almost all the baselines across model architectures and datasets. With $\rho = 0.1$, *AdaBet* achieves the highest average accuracy (76.26%), demonstrating its ability to balance performance while achieving training efficiency (as we report in the next subsection). FI-based approaches suffer significantly as they are often heavily dependent on a separate resource-intensive meta-training phase to boost the accuracy at runtime [20]. While ElasticTrainer [15] performs competitively, *AdaBet* surpasses it across most configurations. Although the CUB dataset shows lower overall performance across methods due to its fine-grained nature, *AdaBet* maintains a competitive accuracy for this dataset as well.

Table 3: Top-1 accuracy of *AdaBet* with $\rho = 0.1$ and baselines.

		Full Training	Transfer Learning	Fisher Information	PruneTrain	Elastic Trainer	AdaBet (Ours)
Stanford Dogs (120 classes)	**ResNet50**	70.87	70.01	73.76	66.09	72.55	**75.52**
	VGG16	60.82	61.66	70.17	50.2	66.8	**72.09**
	MobileNetV2	80.47	80.4	80.47	73.31	77.02	**81.76**
	ViT	86.42	86.16	14.37	82.6	**87.79**	85.12
Oxford IIIT Pets (37 classes)	**ResNet50**	86.18	84.63	85.94	86.48	87.54	**88.74**
	VGG16	81.18	85.37	**86.56**	78.44	34.9	86.14
	MobileNetV2	86.95	87.78	87.16	27.3	86.70	**89.93**
	ViT	93.15	90.21	26.23	90.57	91.96	**92.23**
CUB (200 classes)	**ResNet50**	34.76	27.13	39.85	28.85	24.37	**46.37**
	VGG16	57.17	58.37	58.77	52.77	15.75	**60.30**
	MobileNetV2	42.35	**50.37**	42.26	50.05	15.95	46.36
	ViT	**85.73**	80.76	62.34	80.22	82.93	79.47
Flowers102 (102 classes)	**ResNet50**	80.56	77.18	79.05	80.45	80.6	**81.39**
	VGG16	61.79	**75.83**	74.21	33.68	74.86	74.71
	MobileNetV2	**61.72**	30.96	56.22	37.6	37.6	61.5
	ViT	78.66	95.45	13.10	98.12	97.71	**98.55**
Average		71.79	71.39	59.41	63.35	64.68	**76.26**

Table 4: Peak memory consumption of baselines and *AdaBet* during different stages of the process - layer selection and training. *AdaBet* consumes upto 75.77% less overall peak memory compared to full training. *AdaBet* also consumes lesser peak memory in comparison to other baselines that require backpropagation for layer selection. Overall peak memory during training for *AdaBet* is comparable to that of inference and transfer learning.

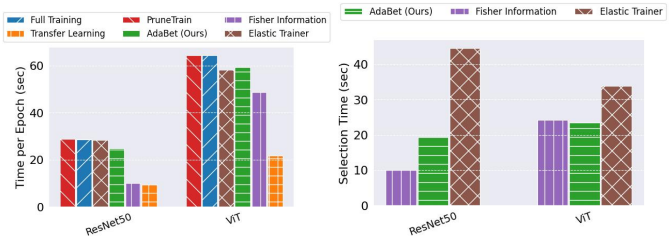
	Peak Selection Memory (MB)				Peak Training Memory (MB)			
	ResNet50	**ViT**	**MobileNetV2**	**VGG16**	**ResNet50**	**ViT**	**MobileNetV2**	**VGG16**
Inference	N/A	N/A	N/A	N/A	243.04	86.26	315.86	932.21
Full Training	N/A	N/A	N/A	N/A	777.12	386.60	373.58	988.77
Transfer Learning	N/A	N/A	N/A	N/A	243.30	86.46	316.02	932.27
ElasticTrainer	777.12	386.60	373.58	988.77	299.66	138.05	322.13	969.44
PruneTrain	777.12	386.60	373.58	988.77	777.12	386.60	373.58	988.77
Fisher Information	777.12	386.60	373.58	988.77	270.98	86.27	316.02	932.28
AdaBet (Ours)	243.04	86.26	315.86	932.21	261.78 (↓**66.31%**)	93.64 (↓**75.77%**)	318.5 (↓**14.74%**)	932.45 (↓**5.72%**)

In summary, *AdaBet* offers a more robust and accurate alternative to existing on-device retraining paradigms, consistently improving performance without the need for full model retraining or compute-expensive gradient-based layer selection. On average, across all 16 dataset–model pairs, *AdaBet* achieves +5% improvement over the second-best baseline.

6.2 Resource Efficiency

6.2.1 Memory Efficiency. Table 4 reports peak memory consumption in MB by the selection step and training step separately for all models and baselines. Note that Full Training, Transfer Learning, and inference do not involve a selection step, and PruneTrain [28] selects layers regularly throughout training at frequent intervals. Specifically, for ViT, setting $\rho = 0.1$ reduces peak memory usage by nearly 76% relative to Full Training and ElasticTrainer [15]. For ResNet50, the memory reduction is approximately 66%. Note that these results are independent of the dataset, since all input images are resized to 224×224.

These improvements come from the fact that *AdaBet* bypasses backpropagation through the entire network, unlike methods like ElasticTrainer and Fisher Information [6], which must perform at least one complete backward pass, which leads to memory requirements comparable to Full Training. Notably, *AdaBet* operates with memory usage comparable to inference and lightweight Transfer Learning yet delivers higher accuracy (as shown in Table 3).



(a) Training time per epoch **(b) Selection time**

Figure 6: (a) Total retraining time for 1 epoch over all samples in the on Oxford-IIIT Pets dataset. (b) Layer selection time on Oxford-IIIT Pets dataset.

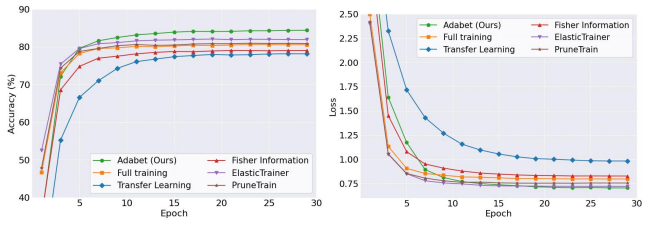


Figure 7: (Left) classification accuracy on test set and (right) training loss of ResNet50 with Flowers102 dataset across baselines and *AdaBet* with $\rho = 0.1$.

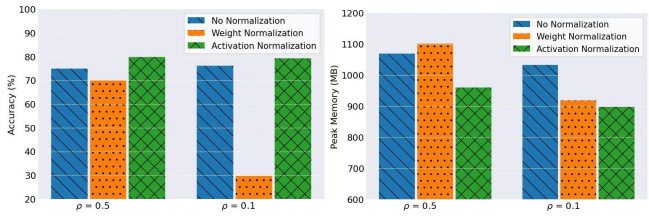


Figure 8: (Left) accuracy and (right) peak memory consumption of *AdaBet* under three normalization strategies applied to Betti Numbers. Activation normalization (our choice) offers the best trade-off between accuracy and memory usage, particularly under tighter selection budgets ($\rho = 0.1$).

In general, *AdaBet* enables more memory-efficient retraining, achieving nearly a 40% reduction in peak memory usage on average across four DNNs.

6.2.2 Training-Time Efficiency. ElasticTrainer [15] fixes the number of training epochs to ensure timely convergence. However, this strategy may lead to underfitting depending on the dataset and model. In Figure 6(a), we report the time taken per epoch for a fair comparison of training-time efficiency. Specifically, the per epoch training time (in seconds) is reported on Oxford-IIIT Pets with ResNet50 (left) and ViT (right). *AdaBet* consistently reduces per epoch time compared to Full Training and PruneTrain. For example, on the Oxford-IIIT Pets dataset with ResNet50 and ViT architectures, *AdaBet* achieves more than a 15% and 7% reduction relative to Full Training, respectively.

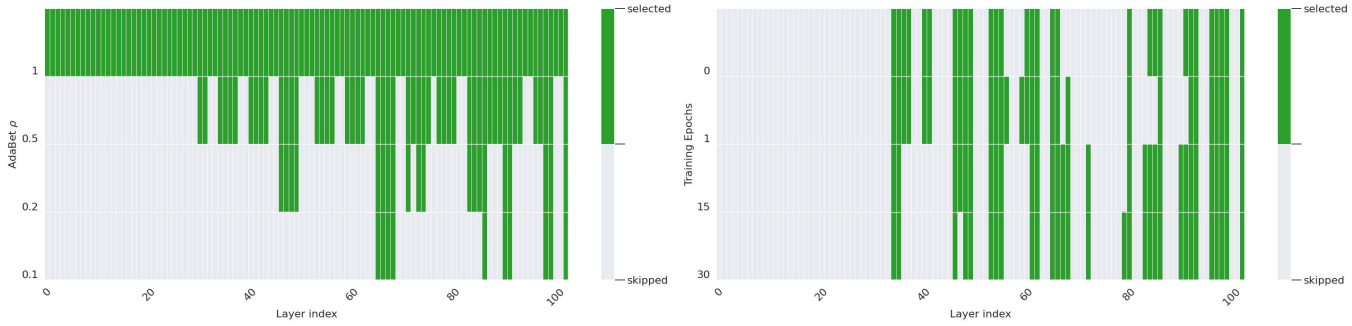
It is worth noting that Fisher Information shows shorter training times than *AdaBet*, approaching those of Transfer Learning. We observe that Fisher Information often selects the last layers (see Figure 2(a)-(b)), thereby reducing gradient flow and consequently decreasing overall training time [26]. This results in Fisher Information achieving lower accuracy than *AdaBet*, as shown earlier in Table 3.

Overall, *AdaBet* enables retraining on the local dataset to be nearly 11% faster than Full Training on average, while remaining comparable to other baselines, while offering higher accuracy with a lower memory footprint.

6.2.3 Selection Efficiency. As summarized in Table 1, layer selection approaches differ between the ElasticTrainer and our *AdaBet*'s strategy. To assess the overhead introduced by Betti Number computation compared to the importance calculation and dynamic programming-based selection in ElasticTrainer, we measure the time required for a single selection step in both methods.

Figure 6(b) illustrates the time taken for layer selection on ResNet50 and ViT models using the Oxford-IIIT Pets dataset. Across both architectures, *AdaBet* requires less time for the selection step than ElasticTrainer [15]. On ResNet50 *AdaBet* requires less time for the selection than Fisher Information [20], and on ViT *AdaBet* and Fisher Information [20] consume comparable time for a single selection step. In *AdaBet*, time consumption for the selection step is primarily due to the activation accumulation, which requires multiple forward passes along with Betti number computation, in contrast to a single forward and backward pass in the case of Fisher Information [20] and a single forward and backward pass with dynamic programming in the case of ElasticTrainer [15]. Notably, since ElasticTrainer requires periodic selection, the time shown will be scaled by the number of times this step is executed during training. Similarly, for Fisher Information, we perform only a single backpropagation compared to five forward passes for *AdaBet*, and as shown previously (in § 6.1), this single backpropagation leads to degradation in the overall performance.

In summary, *AdaBet* is on average 45% faster than ElasticTrainer[15] for a single layer selection step; however, since *AdaBet* performs selection in a single step, the overall retraining process benefits from improved performance.



(a) (b)

Figure 9: (a) Layers selected by *AdaBet* with varying ρ. (b) Layers selected by *AdaBet* from a model before any training updates (epoch 0), from a partially trained model (epochs 1 and 15), and a trained model (epoch 30)

6.2.4 Convergence. Training dynamics are shown in Figure 7, with test accuracy (left plot) and training loss curves (right plot) over 50 training epochs of ResNet50 on the Oxford-IIIT Pets dataset. ElasticTrainer shows the fastest convergence and ultimately matches the final accuracy achieved by *AdaBet*, while Full Training converges slowly and to a lower accuracy. Transfer Learning, which only updates the final layers, converges slowly and yields the lowest accuracy, highlighting the importance of informed layer selection. Fisher Information performs just better than Transfer Learning.

Although ElasticTrainer initially converges quickly for training loss, *AdaBet* achieves a lower final loss, indicating a better fit. In contrast, Full Training yields higher loss values, and Transfer Learning shows the slowest reduction with the highest final loss.

Overall, the training dynamics of *AdaBet* demonstrate a balance between fast convergence, lower loss, and competitive accuracy.

6.3 Ablation Studies on *AdaBet*

To examine our motivations and design choices, we conduct some ablation studies across factors influencing the efficiency and performance of *AdaBet*. In particular, we analyze the impact of normalization strategies, the impact of the parameter ρ, and batch size.

6.3.1 The Impact of Normalization. Betti numbers capture the topological characteristics of activations, reflecting a layer's learning capacity. However, without normalization, layers with a larger number of activations naturally yield higher Betti numbers, making direct comparisons misleading. As discussed before, we normalize the Betti Numbers before layer selection to address this bias (see §4.2.4). Here, we compare our normalization strategy with two other alternatives: (i) no normalization, and (ii) normalization by the number of trainable parameters (weight size).

Figure 8 compares the accuracy (left) and peak memory consumption (right) of ResNet50 across these normalization strategies for two values of ρ on Stanford Dogs dataset. Normalizing by activation size achieves the best trade-off, leading to higher accuracy and lower memory consumption. In contrast, weight normalization degrades accuracy, particularly at $\rho = 0.1$, while increasing peak memory usage. Activation normalization improves the performance, confirming its suitability for efficient layer selection.

6.3.2 Selection across Different Budgets. In Figure 9(a), we show a visualization of the layers selected by *AdaBet* as we vary the resource-budget parameter ρ across ResNet50 on Oxford-IIIT Pets. As ρ decreases, *AdaBet* consistently prioritizes a subset of layers, particularly those critical for maintaining performance under constrained budgets. In particular, the same core set of important layers (both early and late) remains selected even under tight budgets. In this way, *AdaBet* achieves a better balance: it identifies the most important layers across models and datasets, yet fine-tunes its selections to meet predefined resource constraints.

6.3.3 Selection across Training Epochs. Figure 9-b, illustrates the dynamics of layer selection by *AdaBet* across different training stages on ResNet50 and Oxford-IIIT Pets. At epoch 0 (before any training updates), *AdaBet* already shows a selection pattern similar to epoch 1 and epoch 15. The distribution of selected layers shows that certain layers are consistently selected while others are skipped, indicating that *AdaBet* consistently keeps the focus on layers that contribute more effectively to learning.

6.3.4 The Impact of ρ. We evaluate how varying the layer selection parameter ρ, which determines the proportion of

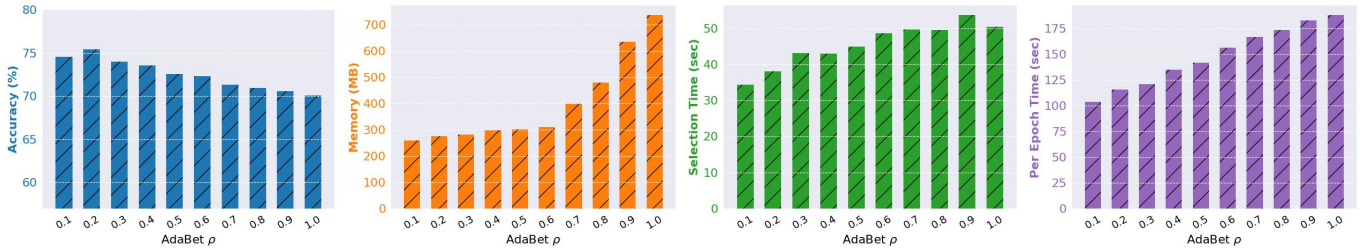


Figure 10: Accuracy, peak memory consumption, selection time, and training time per epoch of ResNet50 in *AdaBet* with varying ρ.

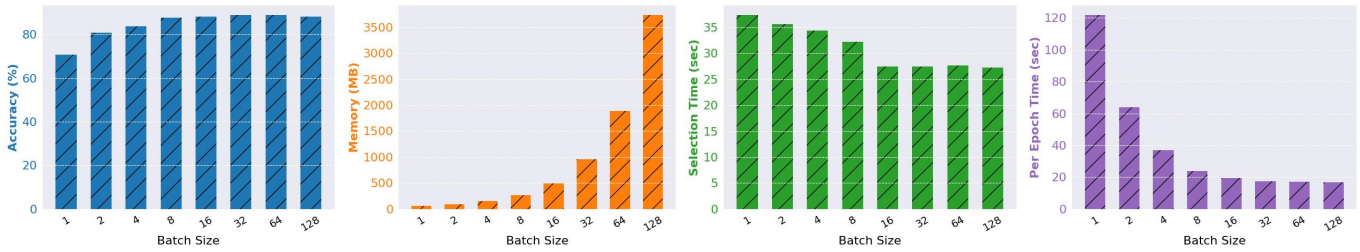


Figure 11: Accuracy, peak memory consumption, selection time, and training time per epoch of ResNet50 in *AdaBet* with varying batch size.

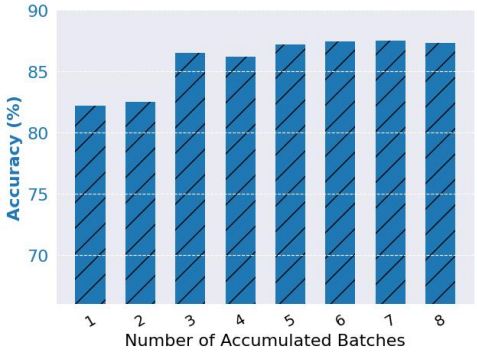


Figure 12: Performance of *AdaBet* using VGG16 with increasing number of data batches (each of size 8) accumulated during selection on Oxford-IIIT Pets dataset.

layers updated during retraining, affects the classification performance and peak memory usage of *AdaBet*. Figure 10 presents results for ResNet50 plotting classification accuracy on Stanford Dogs dataset, peak memory consumption, selection time, and time per epoch as a function of ρ. As ρ decreases, *AdaBet* trains a smaller subset of the model layers, leading to a substantial reduction in peak memory usage. For instance, at $\rho = 0.1$, memory consumption drops by approximately 65% for ResNet50, relative to full-model training ($\rho = 1.0$). This confirms that our layer selection is an effective strategy for lowering the computational overhead of on-device training. In terms of classification accuracy, the models exhibit stability as ρ decreases.

In terms of time, as ρ decreases, *AdaBet* selects and trains a smaller subset of the model layers, leading to an approximately 30% reduction in selection time and approximately 45% reduction in training time per epoch for ResNet50, relative to full-model training.

Overall, *AdaBet* enables an effective trade-off between memory efficiency, time consumption, and model accuracy through the tunable ρ parameter. These results highlight the potential of partial layer retraining for efficient learning.

6.3.5 Impact of Batch Size. Figure 11 presents a performance study of *AdaBet* under varying batch sizes, evaluating its effect on accuracy, memory usage, selection time, and per-epoch training time. We observe that *AdaBet* maintains high accuracy across all configurations, with only a slight drop as batch size decreases beyond 8. This stability underscores its robustness to batch size variation—an important trait for practical deployment. Memory consumption, as expected, grows significantly with batch size, with a sharp increase beyond 8. This reinforces the importance of using smaller batch sizes for on-device scenarios, where memory resources are typically constrained.

Interestingly, while Betti Number estimations are generally expected to benefit from larger batch sizes due to better topological representation, we do not observe such

a trend here. This is attributed to the accumulation strategy employed in *AdaBet* (see § 4.2.2), wherein activations from multiple smaller batches are aggregated to form an adequate batch size of 40 for layer selection. This allows us to retain the topological richness of larger batches while remaining within device constraints. Notably, the retraining phase continues to use the original smaller batch size.

Selection time across batch sizes decreases steadily until batch size 16, after which the accumulation strategy considers the input batch size as such to form an effective batch of size approximately 40 for selection. Beyond a batch size 16, the accumulation strategy fixes it at 16 to counter the memory consumption with increasing batch size. This effect can be observed at the constant selection time beyond batch size 16. Meanwhile, the time per training epoch decreases steadily with increasing batch size, following standard mini-batch processing behavior. Overall, these results validate that *AdaBet* is compatible with small-batch operation and performs reliably under varying resource budgets, with minimal accuracy degradation and strong efficiency characteristics.

6.3.6 The Impact of Batch Accumulation during Selection. To counter Betti Number's dependence on the volume of data used for its computation, *AdaBet* uses activation aggregation (see Section 4.2.2). To find the ideal number of data batches required for *AdaBet*, we run an ablation study as shown in Figure 12. We observe that as the number of batches increases, the performance increases until activations from 5 batches are accumulated for betti computation and layer selection. Beyond 5, the performance remained almost the same with consistent layer selection. This shows that *AdaBet* requires an aggregated equivalent of 5 batches with a batch size of 8 each to capture robust layer importance and select layers to achieve optimal performance.

7 Discussion and Future Works

We show the feasibility of gradient-free layer selection through topological analysis of activations, and our results show clear efficiency gains and competitive accuracy across diverse vision models. Yet, the current design makes some simplifying assumptions that limit its scope. This section discusses key areas where *AdaBet* can be extended and refined.

1. Hardware-aware Selection Mechanism: The current selection approach designed in *AdaBet* depends only on normalized Betti Numbers, which capture representational complexity but ignore device-level details. In practice, resource profiles can vary significantly across different hardware platforms. For example, some layers that appear lightweight on GPUs can be bottlenecks on microcontrollers or microprocessors due to activation storage or memory bandwidth [25]. Future work could integrate Betti-based ranking with profiling of FLOPs, latency, and energy costs, allowing selection policies to adapt to hardware-specific constraints. Such hardware-aware co-design would ensure that *AdaBet* not only identifies expressive layers but also aligns with the computational capabilities of the target platform.

2. Extension to Ubiquitous Sensing Applications: *AdaBet* is currently evaluated on vision benchmarks, yet its reliance on forward-pass activations makes it particularly suitable for domains where gradients are costly or labels are scarce. This, in particular, is useful for applications that heavily involve time-series data like audio, sensor data, or physiological data obtained from mobile and wearable devices, as these modalities often face severe annotation and computational challenges. *AdaBet* can provide lightweight adaptation without sacrificing privacy in these cases. We envision extending *AdaBet* to such modalities to demonstrate its versatility and practical relevance for edge scenarios in health monitoring and ubiquitous sensing.

3. Impact of Distribution Shifts: *AdaBet* performs a single selection step before retraining and shows highly accurate adaptation to diverse runtime datasets. Real deployments, however, can often involve gradual or abrupt distribution shifts. This can impact the computation of Betti Numbers and subsequently the ranking. In the following versions of *AdaBet*, we intend first to investigate the impact of distribution shifts on topological features like Betti Numbers and the final ranking mechanism. Such an in-depth study will help us understand the performance of *AdaBet* in real-world noisy environments.

4. Granularity of Selection: *AdaBet* currently operates at the granularity of whole layers. This is inherently due to layerwise activations for computing the Betti Numbers. While effective, this limits flexibility compared to tensor-level selection approaches such as ElasticTrainer [15]. Extending *AdaBet* to compute topological signals at finer levels would require carefully choosing a topological feature that can be computed at a tensor level. By acknowledging this limitation, we position *AdaBet* not as a final solution but as a foundation for future topology-guided methods that are highly efficient both during selection and training.

8 Conclusion

We introduce *AdaBet*, a gradient-free layer selection method for efficiently adapting pre-trained ML models to target local datasets. *AdaBet* utilizes topological features of activation outputs, specifically the first Betti number, to quantify the representational capacity of DNN layers and identify the most important layers for retraining. Evaluations on benchmark models and datasets demonstrate that *AdaBet* consistently outperforms state-of-the-art baselines. Across 16 dataset–model pairs, it achieves an average +5% gain in classification accuracy while reducing peak memory usage

by 40%. These results highlight *AdaBet* as a practical solution for efficient and privacy-preserving on-device retraining of ML models in resource-constrained environments.

While *AdaBet* demonstrates the feasibility of gradient-free layer selection via topological analysis of activations, several open directions remain. In particular, opportunities exist to incorporate hardware-aware selection policies, extend beyond vision benchmarks to ubiquitous sensing domains, account for distribution shifts in real-world deployments, and refine the granularity of selection beyond layers.

References

[1] Ali, M., Zain, J.M., Zolkipli, M.F., Badshah, G.: Battery efficiency of mobile devices through computational offloading: A review. In: IEEE Student Conference on Research and Development (2015)

[2] Ashkboos, S., Verhoef, B., Hoefler, T., Eleftheriou, E., Dazzi, M.: Efqat: An efficient framework for quantization-aware training. arXiv preprint arXiv:2411.11038 (2024)

[3] Bauer, U.: Ripser: efficient computation of Vietoris-Rips persistence barcodes. J. Appl. Comput. Topol. . https://doi.org/10.1007/s41468-021-00071-5

[4] Bdair, T., Navab, N., Albarqouni, S.: Fedperl: Semi-supervised peer learning for skin lesion classification (2021)

[5] Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H.: Greedy layer-wise training of deep networks. Advances in neural information processing systems (2006)

[6] Cai, H., Gan, C., Zhu, L., Han, S.: Tinytl: Reduce memory, not parameters for efficient on-device learning. In: Advances in Neural Information Processing Systems (NeurIPS) (2020)

[7] Chen, T., Xu, B., Zhang, C., Guestrin, C.: Training deep nets with sublinear memory cost. arXiv preprint arXiv:1604.06174 (2016)

[8] Corneanu, C.A., Madadi, M., Escalera, S.: Computing the testing error without a testing set. In: Proceedings of the IEEE/CVF CVPR (2020)

[9] Corneanu, C.A., Madadi, M., Escalera, S., Martinez, A.M.: What does it mean to learn in deep networks? and, how does one detect adversarial attacks? In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 4757–4766 (2019)

[10] CybertronAI: Gradient Checkpointing for PyTorch. https://github.com/cybertronai/gradient-checkpointing (2024)

[11] Dosovitskiy, A., et. al., L.B.: An image is worth 16x16 words: Transformers for image recognition at scale (2021), https://arxiv.org/abs/2010.11929

[12] Duarte, L.: Betti numbers under small perturbations. Journal of Algebra (2022)

[13] Hannun, A., Guo, C., van der Maaten, L.: Measuring data leakage in machine-learning models with fisher information (2021)

[14] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition (2015), https://arxiv.org/abs/1512.03385

[15] Huang, K., Yang, B., Gao, W.: Elastictrainer: Speeding up on-device training with runtime elastic tensor selection. In: ACM International Conference on Mobile Systems, Applications, and Services (MobiSys'23) (2023)

[16] Jacques, V., Alizadeh, N., Castor, F.: A study on the battery usage of deep learning frameworks on ios devices. In: 2024 IEEE/ACM 11th International Conference on Mobile Software Engineering and Systems (MOBILESoft) (2024)

[17] Jiang, Z., Chen, X., Huang, X., Du, X., Zhou, D., Wang, Z.: Back razor: Memory-efficient transfer learning by self-sparsified backpropagation. Advances in neural information processing systems (2022)

[18] Karakida, R., Osawa, K.: Understanding approximate fisher information for fast convergence of natural gradient descent in wide neural networks*. Journal of Statistical Mechanics: Theory and Experiment (2021)

[19] Khosla, A., Jayadevaprakash, N., Yao, B., Fei-Fei, L.: Novel dataset for fine-grained image categorization. In: First Workshop on Fine-Grained Visual Categorization, IEEE Conference on Computer Vision and Pattern Recognition

[20] Kwon, Y.D., Li, R., Venieris, S.I., Chauhan, J., Lane, N.D., Mascolo, C.: Tinytrain: Resource-aware task-adaptive sparse training of dnns at the data-scarce edge. In: Proceedings of the 41st International Conference on Machine Learning (PMLR)

[21] Laskaridis, S., Venieris, S.I., Kouris, A., Li, R., Lane, N.D.: The future of consumer edge-ai computing. IEEE Pervasive Computing (2024)

[22] Lee, Y., Chen, A.S., Tajwar, F., Kumar, A., Yao, H., Liang, P., Finn, C.: Surgical fine-tuning improves adaptation to distribution shifts. In: The Eleventh International Conference on Learning Representations (2023)

[23] Liberis, E., Dudziak, L., Lane, N.D.: µnas: Constrained neural architecture search for microcontrollers. In: Proceedings of the 1st Workshop on Machine Learning and Systems. Association for Computing Machinery (2021)

[24] Lin, J., Zhu, L., Chen, W.M., Wang, W.C., Gan, C., Han, S.: On-device training under 256kb memory. In: 36th Conference on Neural Information Processing Systems (NeurIPS) (2022)

[25] Lin, J., Zhu, L., Chen, W.M., Wang, W.C., Han, S.: Tiny machine learning: Progress and futures [feature]. IEEE Circuits and Systems Magazine **23**(3), 8–34 (2023). https://doi.org/10.1109/MCAS.2023.3302182

[26] Liu, Y., Agarwal, S., Venkataraman, S.: Autofreeze: Automatically freezing model blocks to accelerate fine-tuning. arXiv preprint arXiv:2102.01386 (2021)

[27] Loshchilov, I., Hutter, F.: Decoupled weight decay regularization. arXiv preprint arXiv:1711.05101 (2017)

[28] Lym, S., Choukse, E., Zangeneh, S., Wen, W., Sanghavi, S., Erez, M.: Prunetrain: fast neural network training by dynamic sparse model reconfiguration. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. https://doi.org/10.1145/3295500.3356156

[29] Marcos Rodrigo, Carlos Cuevas, N.G.: Comprehensive comparison between vision transformers and convolutional neural networks for face recognition tasks (2024)

[30] Nie, X., Miao, X., Yang, Z., Cui, B.: Tsplit: Fine-grained gpu memory management for efficient dnn training via tensor splitting. In: 2022 IEEE 38th International Conference on Data Engineering (ICDE) (2022)

[31] Nilsback, M.E., Zisserman, A.: Automated flower classification over a large number of classes. In: 2008 Sixth Indian conference on computer vision, graphics & image processing. pp. 722–729. IEEE (2008)

[32] Norouzzadeh, M.S., Nguyen, A., Kosmala, M., Swanson, A., Palmer, M.S., Packer, C., Clune, J.: Automatically identifying, counting, and describing wild animals in camera-trap images with deep learning. Proceedings of the National Academy of Sciences (25), E5716–E5725 (2018). https://doi.org/10.1073/pnas.1719367115, https://www.pnas.org/doi/abs/10.1073/pnas.1719367115

[33] Parkhi, O.M., Vedaldi, A., Zisserman, A., Jawahar, C.V.: Cats and dogs. In: 2012 IEEE Conference on Computer Vision and Pattern Recognition

[34] Radford, A., Kim, J.W., Xu, T., Brockman, G., McLeavey, C., Sutskever, I.: Robust speech recognition via large-scale weak supervision. In: Proceedings of the 40th International Conference on Machine Learning. JMLR.org (2023)

[35] Razavian, A.S., Azizpour, H., Sullivan, J., Carlsson, S.: Cnn features off-the-shelf: an astounding baseline for recognition (2014), https://arxiv.org/abs/1403.6382

[36] Rücklé, A., Geigle, G., et. al.: Adapterdrop: On the efficiency of adapters in transformers. In: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)
[37] Sakamoto, K., Sato, I.: End-to-end training induces information bottleneck through layer-role differentiation: A comparative analysis with layer-wise training
[38] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C.: Mobilenetv2: Inverted residuals and linear bottlenecks (2019), https://arxiv.org/abs/1801.04381
[39] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition (2015), https://arxiv.org/abs/1409.1556
[40] Singh, A., Pandey, N., Shirgaonkar, A., Manoj, P., Aski, V.: A study of optimizations for fine-tuning large language models. arXiv preprint arXiv:2406.02290
[41] Soen, A., Sun, K.: On the variance of the fisher information for deep learning (2021)
[42] Sun, P., Kretzschmar, H., Dotiwalla, X., Chouard, A., Patnaik, V., Tsui, P., Guo, J., Zhou, Y., Chai, Y., Caine, B., et al.: Scalability in perception for autonomous driving: Waymo open dataset. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 2446–2454 (2020)
[43] Sun, S., Chen, W., et. al.: On the depth of deep neural networks: A theoretical view. In: Proceedings of the AAAI Conference on Artificial Intelligence (2016)
[44] Swanson, K., Wu, E., Zhang, A., Alizadeh, A.A., Zou, J.: From patterns to patients: Advances in clinical machine learning for cancer diagnosis, prognosis, and treatment. Cell (8), 1772–1791 (2023)
[45] Tralie, C., Saul, N., Bar-On, R.: Ripser.py: A lean persistent homology library for python. The Journal of Open Source Software (2018)
[46] Wah, C., Branson, S., Welinder, P., Perona, P., Belongie, S.: The caltech-ucsd birds-200-2011 dataset. Tech. rep., California Institute of Technology (2011)
[47] Wang, Y., Jiang, Z., Chen, X., Xu, P., Zhao, Y., Lin, Y., Wang, Z.: E2-train: training state-of-the-art CNNs with over 80% energy savings (2019)
[48] Yang, J., Sang, L., Cremers, D.: Dive into layers: Neural network capacity bounding using algebraic geometry. arXiv preprint arXiv:2109.01461 (2021)
[49] Zhu, S., Voigt, T., Rahimian, F., Ko, J.: On-device training: A first overview on existing systems. ACM Transactions on Sensor Networks
[50] Zhu, S., Voigt, T., Rahimian, F., Ko, J.: On-device training: A first overview on existing systems. ACM Trans. Sen. Netw. (2024)