Appendix for: Pruning by Block Benefit (P3B)

This appendix is organized into several sections, each providing a focused look at a particular aspect of our research. The discussion begins with the formulations of Intra Block Importance in Section A and Soft Masking in Section B. Section C then details the experimental settings and training configurations, while Section D outlines the properties of the datasets used. Next, the inference speedup reached by *P3B* is examined in Section E. Descriptions of further experiments using Knowledge Distillation loss are found in Section F, and a deeper exploration of the importance of reordering in pruning is presented in Section G. Finally, Sections H and I present extended experimental results related to research questions Q1 and Q2 in the main paper.

A. Intra Block Importance

This section extends the information about the *Intra Block Formulation*, described in Sec. 3.2. Specifically, we further explain the architecture of the *Block Performance Indicator (BPI)* module $\Delta\Psi$ and define the smoothing function γ from Eq. 2.

The proposed method *Pruniny by Block Benefit* (*P3B*) assigns a depth dependent parameter budget as keep ratio κ_i^b to consecutively aligned Attention and Multi-Layer Perceptron (MLP) blocks. Thereby, *P3B* conserves the computational capacity of blocks, that mainly increase the class discriminance and penalizes ones with reduced impact. To measure this blockwise classification performance (*BP*), the Block Performance Indicator (*BPI*) $\Delta\Psi$ is introduced as a lightweight learnable function for classification token and patches. In Vision Transformers, the classification token is used to learn class distinguishable features, while patches contain semantic information. The architecture of classification and patch classifiers is defined as follows:

- **Class head** is realized as the origin ViT-classification head using a single linear layer [13].
- **Patch head** is based on the Resnet architecture [19]. We apply a single Resnet block and it's classification head to determine the logits for loss calculation.

The introduced *BPI* learns an individual classification and patch head (Ψ_i^c, Ψ_i^p) for each block, by using the feature map before and after the regarding block as input. For the classification task, *BPI* is optimized towards Cross-Entropy-Loss $\mathcal{L}_{CE}$. Individual heads for each block ensure that the resulting *BP*-score is not influenced by the feature maps of other network depths. Different to other soft classifier methods such as [53], we do not focus on the classification token alone, but also discriminate the patch tokens to identify beneficial semantic features. As defined in Eq. 1 in the main paper, *P3B* uses the relative loss to assign a depth dependent keep ratio κ_i^b to block i. Given the measured *BP*-score for classification tokens $\Delta\Psi_i^c$ and patches $\Delta\Psi_i^p$, the block importance is determined by

$$\mathcal{I}_i^{b,c} = \frac{\gamma(\Delta\Psi_i^c)}{|w_i| + \epsilon}. \tag{2}$$

The number of remaining parameters $|w_i|$ is determined with a mask value $M \geq 0.5$, while ϵ is a small value used for numerical stability. Additionally, a smoothing function γ is applied to restrain high peak values and those with negative BP. Therefore we normalize $\Delta\Psi^c$ and $\Delta\Psi^p$ by it's max value and apply:

$$\gamma(x) = SP(1.4 \cdot s \cdot x) - SP(1.4 \cdot s \cdot x - s). \tag{8}$$

The input value is scaled by factor $s = 10$ to adjust the smoothing amount. $SP(x)$ is the SoftPlus operator that approaches zero for values $x \leq 0$ and scales almost proportional to the input value for $x > 0$ [50]. While the first SP-term defines the lower bound to zero, the second SP-operator limits the upper bound for high values.

According to Eq. 3 in the main paper, we merge classification and patch importance $\mathcal{I}_i^{b,c}$ and $\mathcal{I}_i^{b,p}$ by a normalized weighted sum to block importance score $\mathcal{I}_i^b$. Following each block is assigned a keep ratio κ_i^b proportionally scaled to $\mathcal{I}_i^b$ until the overall model keep ratio $\kappa^m = \sum_{i=1}^{N} |w_i| \kappa_i^b$ has reached. During this scaling process individual block keep ratios can result in values $\kappa_i^b > 1$. Since this is an invalid solution, we clip this keep ratios to 1 and rescale the resulting blocks until a valid solution is found.

B. Soft Masking Formulation

In the previous section we defined the superior keep ratio κ_i^b for Attention and MLP blocks. Based on this parameter budget, *P3B* creates a soft mask M to prune the least important model structures. As extension to the block intrinsic mask definition in Sec. 3.3, this section explains more concretely how the goal sparsity is encoded in the soft mask and how the mask sharpness can be controlled to force a hard mask.

Our proposed method *P3B* uses a soft mask $M \in [0, 1]$ to smoothly reduce the influence of less important operations during training. Thereby, soft masking ensures already pruned elements to keep their numerical importance order, which is necessary for reactivation of already pruned channels. To clarify this point, hard masking prunes elements by setting the mask values to eather 0 or 1. Consequently the local importance order is lost for all pruned elements,

since the regarding importance scores $\mathcal{I}_j$ result in 0. In contrast, soft masking uses intermediate values between 0 and 1 which allows the gradient to roughly scale with the mask value. Consequently, already pruned elements can be reactivated using soft masking, without losing the numerical importance order. The updated mask M' is defined as

$$M' = \varphi(t(argsort(\mathcal{I}^i_{cat}), \kappa^b_i, \tau)), \tag{8}$$

where the values are assigned, according to the importance score $\mathcal{I}^i_{cat}$. High importance values will result in a mask value close to 1, while low importance values smoothly decrease to 0. Thereby, transformation function t in combination with sigmoid function φ ensures that the new mask satisfies keep ratio κ^b_i, while the mask sharpness can be controlled by factor τ. Function t is defined as:

$$t(x, \kappa^b_i, \tau)) = -log(\frac{1}{M_{ref}} - 1) \cdot \frac{x - x_{shift}(\kappa^b_i)}{\tau \cdot |x|}. \tag{9}$$

This formulation ensures two properties regarding pruning mask M:

1. Sparsity: The resulting mask is created by sigmoid function $\varphi \in [0, 1]$, based on the sorted list of importance scores $\mathcal{I}^i_{cat}$, where high scores get a higher mask value. To satisfy the sparsity condition of block keep ratio κ^b_i, we define index i_{shift} with value x_{shift} as the least important element that remains after pruning. By substracting x_{shift} to input vector x, we ensure element i_{shift} will be the least important remaining element with mask value $M' = 0.5$. Dependent on the importance index, higher values approaches 1, while unimportant indices converges to 0.

2. The **Sharpness** of mask M can be controlled by factor τ. Therefore we define a reference mask value $M_{ref} = 0.9$ at index $i_{ref} = i_{shift} + \tau \cdot |x|$. For higher values of τ the reference value M_{ref} is more distant form the fix value i_{shift}, resulting in a smooth mask. The sharpness can be increased by shifting parameter τ close to 0, where the lower bound of $\tau = 0$ represents a hard mask. During the pruning steps *warm up* and *sparsification*, we set factor $\tau = 0.1$. The following step *sharpening* we linearly decreases this value to 0 to force a hard mask.

C. Training Setting

The pruning process of *P3B* is divided into 4 steps: *warm up*, *sparsification*, *sharpening* and *fine-tuning*. During the first 3 steps *P3B* smoothly sparsify the model by masking out unimportant structures. Therefore, Table 5 describes the applied training parameters. Note that all pruning steps are trained for 50 epochs in sum.

Step *fine-tuning* is applied according to the standard training pipeline of Deit [45]. The model is trained for 300 epochs using batch size 512 with learning rate $0.0005 \frac{batchsize}{512}$. During all training steps, we adapt the augmentation strategies from Deit [45], including CutMix [56], Random Augmentation [10] and Mixup [57].

parameter	value	note
epochs warmup	3	
epochs sparsify	22	
epochs sharpening	25	
lr model	$5 \cdot 10^{-4}$	const value
lr *BPI*	$5 \cdot 10^{-4}$	const value
α	0.5	balance $\mathcal{I}^{b,c}$ vs. $\mathcal{I}^{b,p}$
M_{ref}	0.9	sharpening reference
τ	0.1	sharpening factor
mask update freq	1000	scale with data size

Table 5. Applied parameter for training Vision Transformer with *P3B*. This settings are used for dataset Imagenet-1K.

D. Datasets

The experiments to transfer learning tasks in Sec. 4.2 of the main paper show that our method *P3B* is highly performant on various downstream tasks. In this section, we further discuss the properties of the used datasets to show our experiments cover a high variety of domains and task complexities.

Table 6 provides a summary of the data size and number of classes for each applied datasets. The chosen downstream tasks INAT19, IFOOD and CIFAR100 show a diverse size of training samples and a high variation in their class complexity. Furthermore, the data sets encompass a wide domain range. While INAT19 focuses on the classification of plants, IFOOD classifies dishes. These two datasets can be classified as highly fine-grained. In comparison, the data distribution of CIFAR100 is more coarse.

For our transfer learning experiment we initialize the model with checkpoints from Deit [45] that are already tuned on dataset Imagenet-1K and train the model on another downstream task. For all experiments we apply the same settings as described in Sec. C, except the *mask update frequency*. Since this parameter scales with the data size we set this parameter to 50, 100 and 500 for dataset CIFAR100, IFOOD and INAT19, respectively.

dataset	train size	val size	number classes
Imagenet-1K [28]	1.281.167	50.000	1.000
INaturalist 2019 (INAT19) [22]	265.213	3.030	1.010
IFOOD 2019 (IFOOD) [25]	118.475	11.994	251
CIFAR100 [27]	50.000	10.000	100

Table 6. Overview of used datasets in this paper. We list the size of training and validation sets as well as the number of classes.

model	parameters remain (M) ↓	parameters pruned (%) ↑	GPU FPS ($\frac{1}{s}$) ↑	GPU FPS speedup ↑	CPU FPS ($\frac{1}{s}$) ↑	CPU FPS speedup ↑
Deit-Base	86.6	0%	536	1.0	3.8	1.0
	21.5	75%	1156	2.16	11.6	3.05
	8.7	90%	1674	3.12	22.9	6.03
Deit-Small	22.1	0%	1676	1.0	13.9	1.0
	5.4	75%	2473	1.48	35.0	2.52
	2.2	90%	3157	1.88	60.6	4.36

Table 7. Inference speed quantified in frames per second (FPS) measured on CPU and GPU. The models pruned by *P3B* exhibit a throughput increase of up to 3.12x on GPU and 6.03x on CPU.

E. Throughput

In this chapter investigates the throughput using *P3B* as pruning method. Therefore, we measure the processed images as frames per second (FPS). All results are obtained using an *Nvidia GeForce RTX 3090* GPU with a batch size of 1024 and an *Intel Core i9-12900K* CPU with a batch size of 1. Each result is averaged over 1000 runs.

Table 7 shows the resulting FPS-scores. Notably, a parameter reduction of 75% results in a speedup ratio of 2.16 on Deit-Base, while the pruning rate of 90% even increases the inference speed up to factor 3.12. This illustrates the potential using a reduced model size.

F. Knowledge Distillation

In this experiment we apply *P3B* in a Knowledge Distillation setting. Therefore the cross-entropy-loss $\mathcal{L}_{CE}$ is replaced by the Knowledge Distillation loss $\mathcal{L}_{KD}$ defined in [51]. To calculate distillation loss $\mathcal{L}_{KD}(p^c, p^d, p^T, y)$, the class logit for classification token p^c and distillation token p^d are needed from the student model. Here, p^T represents the class logit from the teacher model, and y denotes the ground truth label for the target class. Given the classification token x^c, the class logit is calculated by $p^c = h^c(x^c)$ and the distillation logit p^d, respectively. Thus the classification *Block Performance* for a Knowledge Distillation Problem is formulated as

$$\Delta\Psi_i^c = \mathcal{L}_{KD}(p_{i-1}^c, p_{i-1}^d, p^T, y) - \mathcal{L}_{KD}(p_i^c, p_i^d, p^T, y). \tag{10}$$

Similarly, we formulate the patch *Block Performance* given patch logit p^p as

$$\Delta\Psi_i^p = \mathcal{L}_{KD}(p_{i-1}^p, p_{i-1}^p, p^T, y) - \mathcal{L}_{KD}(p_i^p, p_i^p, p^T, y). \tag{11}$$

The pruning procedure is applied according to the settings described in Sec. 3 in the main paper. For comparison, we reduce the learning rate to $2 \cdot 10^{-4}$ according to the experiments in [51].

loss	models to train teacher	models to train student	req. GPU-memory ↓ Deit-Small	req. GPU-memory ↓ Deit-Tiny
KD	✓	✓	28.1 GiB	28.1 GiB
CE	✗	✓	**14.4 GiB**	**7.7 Gib**

Table 8. Trainings requirements using a non-teacher based loss (e.q. CE-loss) and Knowledge Distillation loss (KD), introduced in [51]. KD training strategies require the teacher model to also be adapted to the new target domain, which necessitates an additional training run.

Method	Param (M) ↓	Flops (G) ↓	Top-1 Acc. (%) ↑
Deit-B-Dist [45]	86.6	17.6	83.36
Deit-S-Dist [45]	22.4	4.6	81.20
NViT-S [51]	21.0	4.2	82.19
P3B-Dist-S (ours)	22.0	4.4	82.08
Deit-T-Dist [45]	5.9	1.3	74.50
NViT-T [51]	6.9	1.3	76.21
P3B-Dist-T (ours)	6.8	1.2	75.04

Table 9. Classification Results of distilled pruning methods on Imagenet-1K. All methods use Deit-Base-Distilled as prunable model [45]. The results show that using *P3B*-Distilled improves performance compared to Deit-S/T-Dist at similar sparsity levels.

To evaluate the Knowledge Distillation loss, we prune the on Imagenet-1K pretrained model Deit-Base-Distilled [45] to a size comparable with Deit-Tiny and Deit-Small. The classification results are shown in Tab. 9. *P3B*-Dist shows an improved performance compared to the the original distilled model of size small and tiny. For instance *P3B*-Dist-S improves the accuracy from Deit-S-Dist from 81.2% up to 82.08%. Compared to the results of NViT the accuracy of *P3B* slightly drops.

However, Knowledge Distillation typically relies on the teacher model being fully converged before training the student model. In real world scenarios the target domain is not encoded in the initial model, leading to a performance loss caused by a mismatch of weight importance, as investigated in Sec. 4.3 in the main paper. Training the model on the target task before pruning is feasible but requires more than twice the training time, as the large scaled teacher model must be trained on the target domain first. Table 8 illustrates the limitations of Knowledge Distillation, showing the GPU memory requirements for a dense training run. These measurements were performed at a batch size of 256, using DeiT-Base as the teacher. Cross-Entropy loss notably uses less GPU memory than KD-loss, reducing the memory demand by a factor of 2 for DeiT-Small and 4 for DeiT-Tiny. By focusing on non-teacher-based losses, such as CE-loss, our *P3B* pruning method efficiently saves GPU-memory and prunes the model in a single training run.

G. Necessity of Reordering

The experiments in Sec. 4.3 in the main paper emphasise that the changing importance order of mask elements, following described as *reordering*, has a significant effect on the resulting performance. We compared the pruning framework *P3B* as dynamic pruning approach to a one-shot pruning method by freezing the model during the pruning steps. The results in the main paper show, that the *frozen* model performs significantly worse than the *trainable* model. This performance drop increases with higher pruning rates and on new target domains. To further understand the reasons for this performance loss, this section analyses the measured Block Performance (*BP*) for the *frozen* case, in order to demonstrate the depth dependent information loss caused by pruning.

By freezing the model during the pruning steps of *P3B*, the model is not able to adapt to the new data domain, before finally getting pruned. Consequently, the decision about which elements to prune, is based only on the initial model configuration. Although the *frozen* model is not allowed to change its local structures, we train the *BPI*-function to evaluate the global block importance $\mathcal{I}_i^b$ and measure the local importance scores $\mathcal{I}_j$ to identify the least important channels that can be pruned. However, since the local importance score defined in Eq. 7 requires a gradient, we do not actually freeze the model, but set the pruning learning rate to $1 \cdot 10^{-8}$. This value is negligibly small, so we assume the model to be frozen. In comparison, *trainable* pruning uses an initial learning rate of $5 \cdot 10^{-4}$.

In addition to the accuracy results between the frozen and trainable model in the main paper, we present the relative contribution on each block for the *frozen* and *trainable* setting in Figure 5, measured by Block Performance (BP). The left column shows the class *BP* for Attention blocks $\Delta\Psi_i^c$, whereas the right column demonstrates the patch *BP* of MLP blocks $\Delta\Psi_i^p$. All graphs show that the *BP*-score drops significantly if the model is *frozen*. This illustrates the loss of class discriminative features, once the model is pruned or applied to another downstream task, as demonstrated by datasets CIFAR100, IFOOD and INAT19. The *frozen* model is not able to recover lost information. If we consider the classificaton *BP* of Attention blocks, the deeper layers with index ≥ 6, show an increased *BP* for the *trainable* model. Consequently, deeper layers are able to recover their discriminative ability, in case informations are lost by removed channels. Considering the MLP blocks in the right column, the *frozen* model exhibits a comparable information loss of semantic features, measured by patch *BP*. Interestingly, shallower layers with a smaller block index are able to better conserve the *BP*, compared to deeper layers.

To summarize this experiment, the layers performance drops drastically once model structures are pruned. Training the model while removing its structures allows Attention and MLP blocks to compensate lost information, and readjust the network topology. The accuracy results in Tab. 3 emphasize that this importance reordering during training improves the overall performance, especially in higher sparsity regimes and on new target domains.

H. Do Attention and MLP blocks create discriminative features equally good?

In Sec. 4.5 in the main paper we analyse the discriminative ability of individual Attention and *MLP* blocks in order to show the change of class-discriminance along the network depth. The relative discriminative improvement in the feature map is measured using the *BP*-metric introduced in Sec. 3.2 of the main paper. This metric learns individual classifiers from the input- and output-featurmap of each block to determine the relative gain measured by Cross-Entropy-Loss. We separate the BP-score for classification token and semantic patches.

Fig. 6 presents additional results to the experiments in the main paper. The column shows the results for models Deit-S and Deit-T while rows represent the datasets Imagenet-1k, INAT19, IFOOD, and CIFAR100. We observe the *BP* of MLP blocks is very close to 0 in all settings, while the Attention blocks reach an averaged *BP*-score of ≥ 0.15. This illustrates the small impact of MLP blocks on the classification token. Moreover we measure the semantic performance improvement within patches by $\Delta\Psi^p$. Attention and MLP blocks show a decreasing discriminative gain along the network depth. Layers 11 and 12 have a *BP*-score of almost 0, showing the low impact of these layers on the semantic feature representation. In comparison of Attention and MLP-blocks in terms of patch performance, the MLP-blocks show stronger semantic feature improvements in layers $1-6$, but both layer types contribute equally strong in the deeper layers $7-12$.

I. Which layers become discriminative first?

This section demonstrates further results to answer the question "Which layers become discriminative first?" from Sec. 4.6 in the main paper. Specifically, we extend the presented results on IFOOD [25] by dataset INAT19 [22].

In this experiment, we train the Vision Transformer model of size small on a defined downstream task and save certain intermediate checkpoints. To answer the question about the changing discriminative properties, we reload each checkpoint separately, freeze the model and train only the introduced Block Performance Indicator (*BPI*). The *BPI* consists of a separate classification head for each individual Attention and MLP block to accurately gauge the relative performance improvement of blocks. We tune the *BPI*-module on each checkpoint separately, to ensure this classifier to be fully converged. This convergence property is

not always given during the pruning procedure of *P3B*, but it is confident enough to deliver appropriate budget leverages. In this experiment we train our *BPI* for 50 epochs to convergence. To consider all data samples for the BP-score $\Delta\Psi$, the mask is updated only once per epoch.

Fig. 7 shows the Block Performance (*BP*) for IFOOD, which has already been discussed in the main paper. We extend this results for dataset INAT19 in Fig. 8. In comparison of this two datasets, we observe similar results. For instance, considering the class-*BP* $\Delta\Psi^c$ of the Attention blocks, the initial *BP* shows a peak-value at layer 6 and decreases down to 0 for layers at higher indices. We assume this decreasing performance is due to the property of deeper layers to contain more task specific features [33]. However, the performance of deeper layers can be recovered by fine tuning. Both datasets show that the deeper layers have more discriminative impact on the classification token in the converged state (epoch 300), compared to shallower layers.

Overall, we observe that shallower layers do not increase their feature contribution significantly by training. In contrast, all block performances ($\Delta\Psi^c$ and $\Delta\Psi^p$) show strong performance improvements in the deeper layers ($index > 6$).

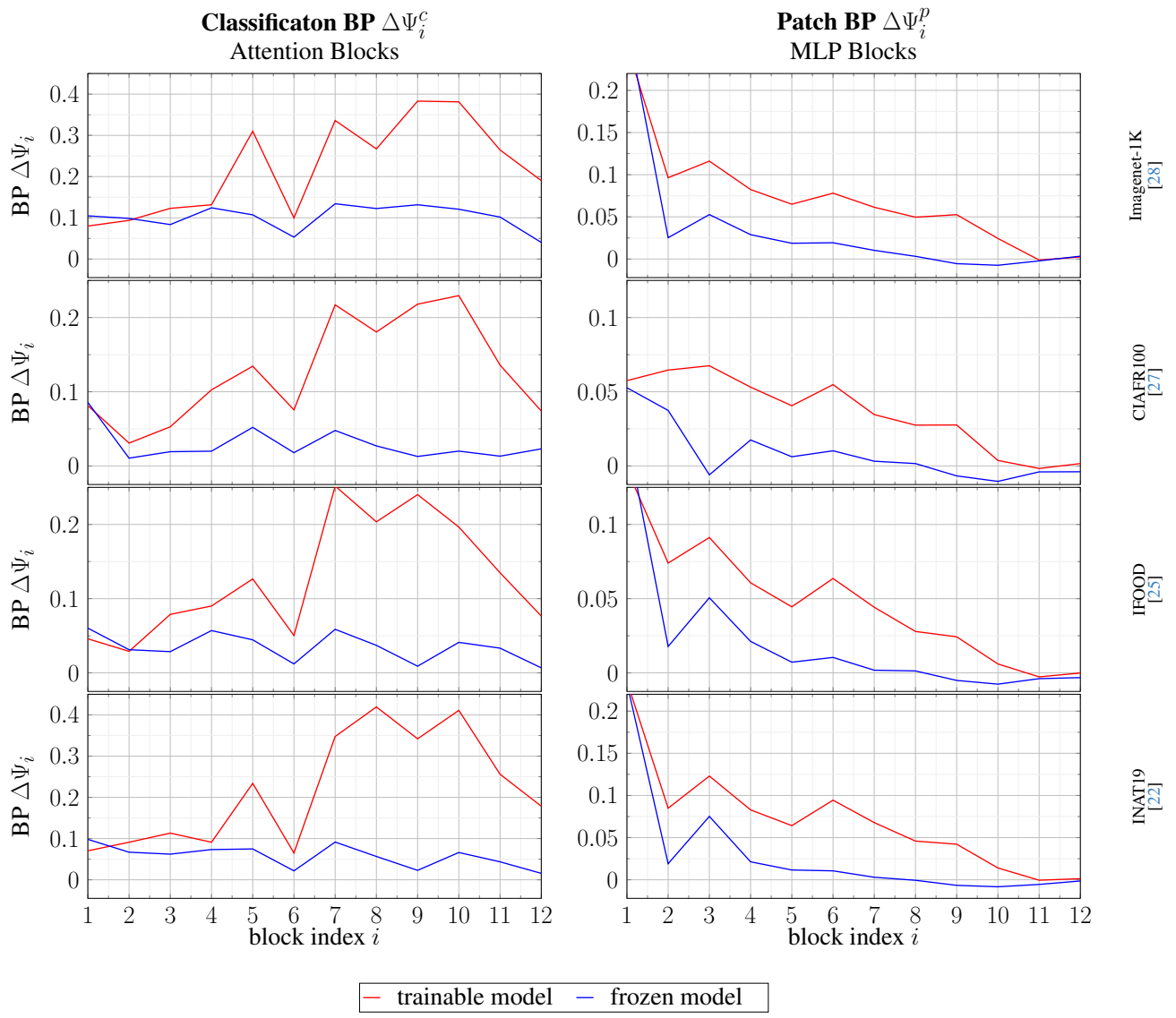

Figure 5. Block Performance $\Delta\Psi$ for *trainable* and *frozen* model during the pruning steps of *P3B* at pruning rate 50%. The left column shows the Classification BP $\Delta\Psi_i^c$ for Attention blocks, while the right columns demonstrates the patch BP $\Delta\Psi_i^p$ for MLP blocks. Both Attention and MLP blocks highly lose their discriminative impact in the *frozen* setting. This illustrates the necessity of reordering, to fairly measure the depth dependent importance using *P3B*.

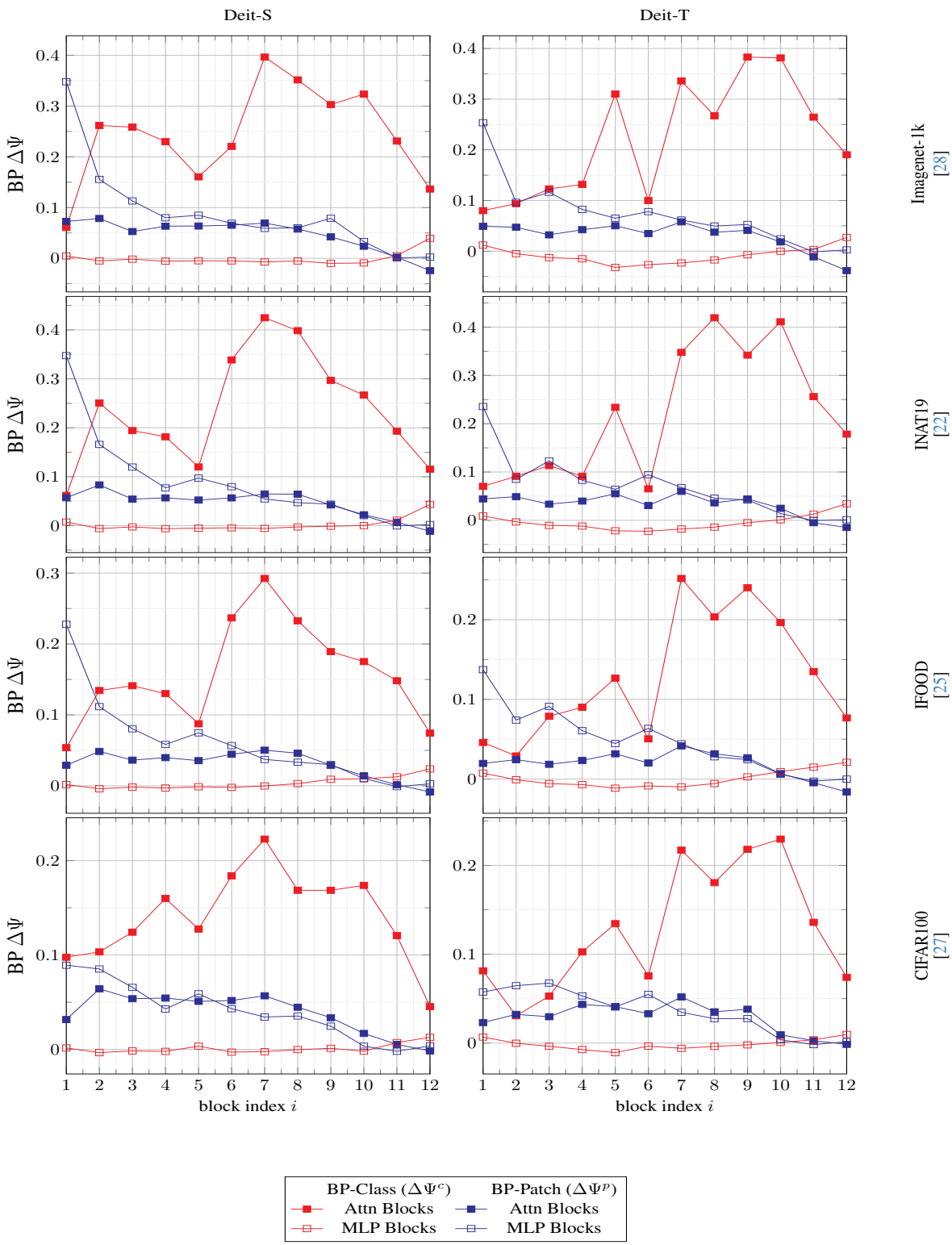

Figure 6. Relative performance gain of Attention and MLP blocks measured by Block Performance $\Delta\Psi$. We apply *P3B* to model Deit-S and Deit-T [45] with pruning rate 50%. Each row corresponds to one of the datatasets: Imagenet, INAT, IFOOD, CIFAR. The results show that only Attention blocks increase the classification tokens discriminance. MLP blocks mainly contribute to semantic patch tokens in a decreasing manner.

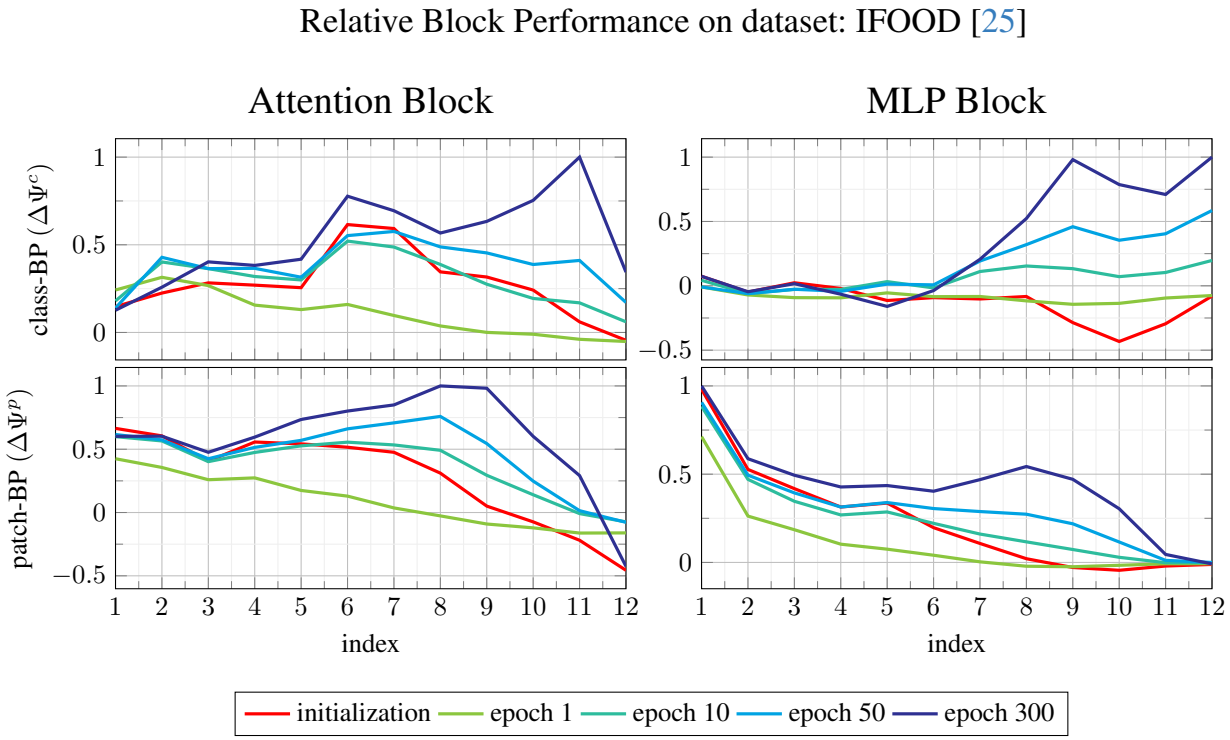

Figure 7. Block Performance $\Delta\Psi$ on intermediate checkpoints. We train model Deit-S on dataset IFOOD [25] **without** pruning the model. Each subplot is normalized by it's maximum value.

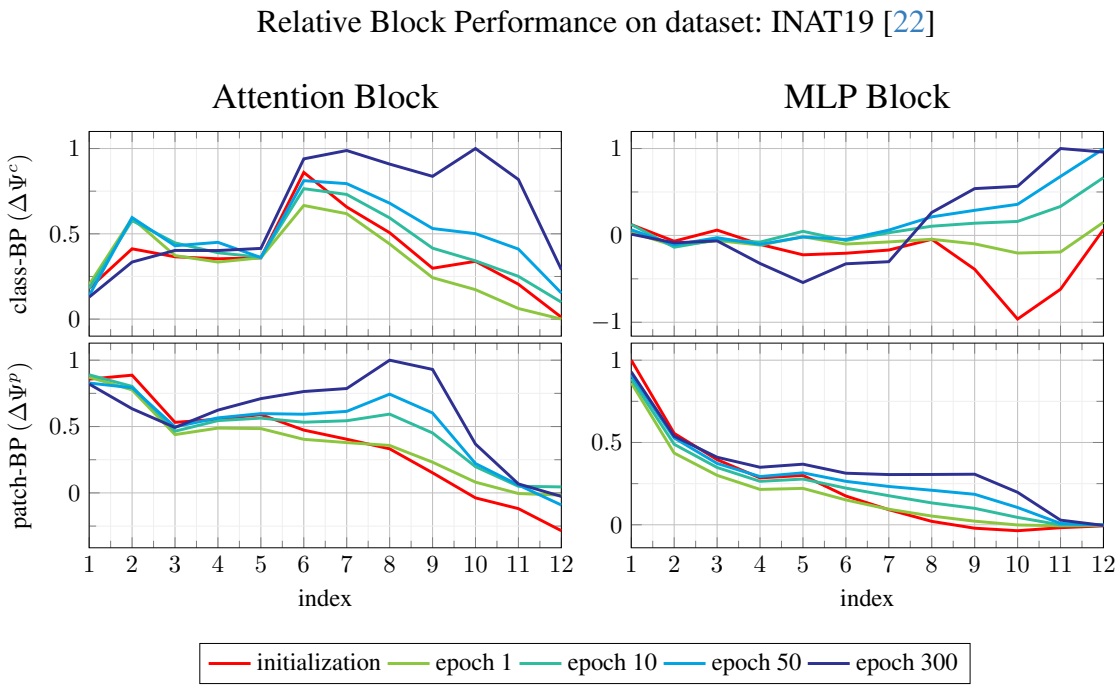

Figure 8. Block Performance $\Delta\Psi$ on intermediate checkpoints. We train model Deit-S on dataset INAT19 [22] **without** pruning the model. Each subplot is normalized by it's maximum value.